SEVENTH FRAMEWORK PROGRAMME

Information & Communication Technologies
ICT

Cooperation Programme

Nippon-European Cyberdefense-Oriented Multilayer threat Analysis[†]

Deliverable D5.4: EU workshop proceedings

Abstract: This document contains the proceedings of the *3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security* (BADGERS 2014). The workshop was organized by the NECOMA project and took place in Wrocław, Poland, on September 11th 2014. The event was collocated with the *19th European Symposium on Research in Computer Security* (ESORICS 2014).

Contractual Date of Delivery	October 2013
Actual Date of Delivery	October 2014
Deliverable Dissemination Level	Public
Editor	Thanasis Petsas
Contributors	All *NECOMA* partners BADGERS 2014 Publication Chairs
Quality Assurance	Christos Papachristos

[†] The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7-ICT-2013-EU-Japan) under grant agreement n° 608533.

The *NECOMA* consortium consists of:

Institut Mines-Telecom	Coordinator	France
ATOS SPAIN SA	Principal Contractor	Spain
FORTH-ICS	Principal Contractor	Greece
NASK	Principal Contractor	Poland
6CURE SAS	Principal Contractor	France
Nara Institute of Science and Technology	Coordinator	Japan
IIJ - Innovation Institute	Principal Contractor	Japan
National Institute of Informatics	Principal Contractor	Japan
Keio University	Principal Contractor	Japan
The University of Tokyo	Principal Contractor	Japan

Contents

1 Frontmatter

1.1 Preface

The BADGERS workshop is intended to encourage the development of large scale security-related data collection and analysis initiatives. It provides an environment to describe already existing real-world, large-scale datasets, and to share with the systems community the return on experiences acquired by analyzing such collected data. Furthermore, novel approaches to collect and study such data sets are presented at the third edition of this workshop. By giving visibility to existing solutions, we expect that the workshop will promote and encourage the better sharing of data and knowledge.

We are happy to report that the third BADGERS workshop received many interesting submissions, spanning three continents, and many aspects of data collection and analysis initiatives. In the end, the program committee accepted 8 papers (including three short papers) out of 16 submissions (50%) for publication and all of the papers received at least three reviews from our program committee. This workshop would never have taken place without the truly excellent program committee and external reviewers and we are grateful for all the hard work they put in.

The resulting program was quite interesting and resulted in lively discussions. In summary, the accepted papers address topics that range from testbeds that can be used to study current attacks, to large scale data collection systems and sharing. All very different papers and presentations, but all focussing on the problem of data collection and analysis initiatives. They were selected for their novelty, and their potential for interesting debate.

Wrocław, 11 September 2014

Mihai Christodorescu and Sotiris Ioannidis
Program Co-Chairs BADGERS 2014

1.2 Program Chairs

Mihai Christodorescu, Qualcomm Research, USA

Sotiris Ioannidis, FORTH, Greece

1.3 Publicity Chairs

Manolis Stamatogiannakis, VU University Amsterdam, The Netherlands

Stefano Zanero, Politecnico di Milano, Italy

1.4 Publication Chairs

Jason Polakis, Columbia University, USA

Manolis Stamatogiannakis, VU University Amsterdam, The Netherlands

Stefano Zanero, Politecnico di Milano, Italy

1.5 Steering Committee

Marc Dacier

Thorsten Holz, Ruhr University Bochum, Germany

Engin Kirda, Northeastern University, USA

1.6 Program Committee

Elias Athanasopoulos, FORTH, Greece

Davide Balzarotti, Institut Eurecom, France

Gregory Blanc, Telecom SudParis, France

Herbert Bos, VU University Amsterdam, The Netherlands

Rodrigo Diaz, Atos, Spain

Joaquin Garcia-Alfaro, Telecom SudParis, France

Guofei Gu, Texas A&M University, USA

Daisuke Inoue, NICT, Japan

Youki Kadobayashi, Nara Institute of Science and Technology, Japan

Piotr Kijewski, CERT Polska, Poland

Engin Kirda, Northeastern University, USA

Kanta Matsuura, University of Tokyo, Japan

Jose Nazario, Invincea Labs, USA

Moheeb Rajab, Johns Hopkins Univesity, USA

Marc Stoecklin, IBM Research Zurich, Switzerland

Jouni Viinikka, 6cure SAS, France

Vern Paxson, UC Berkeley, USA

Antonis Papadogiannakis, ProtectWise, USA

2 Full Papers

This chapter contains copies of the accepted research papers, as they will appear in the actual proceedings.

ANDRUBIS - 1,000,000 Apps Later: A View on Current Android Malware Behaviors

Martina Lindorfer*, Matthias Neugschwandtner*, Lukas Weichselbaum*,
Yanick Fratantonio†, Victor van der Veen‡, Christian Platzer*

*Secure Systems Lab
Vienna University of Technology
{mlindorfer,mneug,lweichselbaum,cplatzer}@iseclab.org

†Computer Security Lab
University of California, Santa Barbara
yanick@cs.ucsb.edu

‡The Network Institute
VU University Amsterdam
v.vander.veen@vu.nl

***Abstract*—Android is the most popular smartphone operating system with a market share of 80%, but as a consequence, also the platform most targeted by malware. To deal with the increasing number of malicious Android apps in the wild, malware analysts typically rely on analysis tools to extract characteristic information about an app in an automated fashion. While the importance of such tools has been addressed by the research community, the resulting prototypes remain limited in terms of analysis capabilities and availability.**

In this paper we present ANDRUBIS, a fully automated, publicly available and comprehensive analysis system for Android apps. ANDRUBIS combines static analysis with dynamic analysis on both Dalvik VM and system level, as well as several stimulation techniques to increase code coverage. With ANDRUBIS, we collected a dataset of over 1,000,000 Android apps, including 40% malicious apps. This dataset allows us to discuss trends in malware behavior observed from apps dating back as far as 2010, as well as to present insights gained from operating ANDRUBIS as a publicly available service for the past two years.

I. INTRODUCTION

Android is undoubtedly the most popular operating system for smartphones and tablets with a market share of almost 80% [1]. Its widespread distribution and wealth of application (app) distribution channels besides the official Google Play Store, however, also make it the undisputed market leader when it comes to mobile malware: according to a recent estimate, as many as 97% of mobile malware families target Android [2]. Estimations by anti-virus (AV) vendors as to the number of Android malware in the wild vary widely. McAfee reports about 68,000 distinct malicious Android apps [3] and Sophos collected a total of 650,000 unique Android malware samples to date, with 2,000 new samples being discovered every day [4].

Google reacted to the growing interest of miscreants in Android by introducing *Bouncer* [5], a service that transparently checks applications submitted to the Google Play Store for malware. Google reported that this service led to a decrease of the share of malware in the Play Store by nearly 40% since its deployment in February 2012. However, a common practice among malware authors is *repackaging* popular apps with malicious code and publishing them in alternative app markets that do not employ effective security measures. In fact, in line with findings from F-Secure [2], we found alternative markets hosting up to 5-8% malicious apps [6].

Consequently, a significant amount of research has focused on analyzing and detecting Android malware, with numerous tools and services being proposed and operated by researchers [7]–[11] and security companies [12]–[14]. Automated and reliable solutions are required to deal with the growing number of mobile malware samples. Analysis capabilities and availability of proposed research prototypes, however, remain limited. A recent study on state-of-the-art Android malware analysis techniques showed that among the 18 analysis tools surveyed, many systems were not available online or were no longer being maintained [15]. In an evaluation on the susceptibility of Android dynamic analysis sandboxes against evasion, Vidas et al. [16] only found three publicly accessibly systems (including the one presented in this paper).

In order to provide a large-scale analysis solution to the research community we propose ANDRUBIS, a hybrid Android malware analysis sandbox that generates detailed analysis reports of unknown Android apps based on features extracted during static analysis and behavior observed through dynamic analysis during runtime. Similar to the spirit of AndroTotal [17], a service that allows researchers to scan Android apps with a number of AV scanners, we operate ANDRUBIS as a publicly available service and data collection tool that allows us to collect and share a comprehensive and diverse dataset of both Android malware and benign apps.

We built ANDRUBIS as an extension to the dynamic Windows malware analysis sandbox ANUBIS [18,19]. ANUBIS has collected a dataset of Windows malware samples that represent a comprehensive and diverse mix of malware found in the wild since 2007 [20]. ANDRUBIS itself has been online since June 2012 and has analyzed over 1,000,000 unique Android apps so far. Based on AV labels collected from VirusTotal [21], we estimate 40% of those apps are malware (not including adware). We further assess the age of apps in our dataset and categorize them by year starting in 2010 allowing us to identify trends in Android malware behavior. Similar to the dataset of ANUBIS, our dataset represents apps from a variety of sources, with apps collected from crawls of the Google Play Store and alternative markets, sample exchange with other researchers, torrents and direct downloads, and anonymous user submissions.

The tight integration of our analysis with the existing ANUBIS infrastructure for analyzing Windows malware provides two main benefits: (a) we can take advantage of existing sample exchange agreements as malware feeds often contain both Windows and mobile samples, and (b) adapt existing analysis techniques for the use with Android apps. For example, experiments applying clustering [22] to Android apps yielded promising results and showed that the feature set produced by ANDRUBIS is rich enough to allow researchers to build various post-processing methods upon [23]. This last aspect is of particular importance as we envision ANDRUBIS to be integrated with other analysis tools to foster sample exchange and provide deeper insights into Android malware behavior. ANDRUBIS has already been integrated with different tools, such as AndroTotal to provide an additional analysis report to AV scanner results. Similarly, ANDRUBIS provides a seed of malicious apps to AndRadar [6], which it uses to scan the Google Play Store and 15 alternative markets and that in turn allows us to collect valuable meta information for our dataset. Besides shedding light on publishing habits of malicious app authors we can gain insights on an app's distribution across

markets and popularity according to user ratings and download numbers. In the future, we also hope to gain insights into the infection rates of user's devices by analyzing which apps are submitted through our mobile app interface from user's phones. Thereby we could verify reports of the small infection rates of less than 0.3% reported in related work [24]–[26].
In summary, we make the following contributions:

- We introduce ANDRUBIS, a fully automated analysis system that combines static and multi-layered dynamic approaches to analyze unknown Android apps.
- We provide ANDRUBIS as a large-scale analysis service to the research community, accepting public submissions at `https://anubis.iseclab.org` and through a mobile app [27].
- By collecting apps from a variety of sources we build a comprehensive and diverse dataset of over 1,000,000 Android apps, including over 400,000 malicious apps.
- We present insights gained from providing our service for the past two years and we discuss trends in malware behavior observed from apps dating back as far as 2010.

II. ANDRUBIS SYSTEM OVERVIEW

In this section we detail the building blocks of ANDRUBIS and how they contribute to forming a complete picture of an app's characteristics. ANDRUBIS follows the *hybrid analysis* approach and is based on both static and dynamic analysis complementing and guiding each other: results of the static analysis are used to perform more efficient dynamic analysis. Figure 1 shows an overview of the individual components of ANDRUBIS and how they relate to one another. Users can submit apps either through our web interface, automated batch submission scripts, or directly from their phone through a dedicated mobile app. We then subject each app to the following three analysis stages:

1) **Static Analysis.** During this stage we extract information from an app's manifest and its bytecode.
2) **Dynamic Analysis.** This core stage executes the app in a complete Android environment, and its actions are monitored at both the Dalvik and the system level.
3) **Auxiliary Analysis.** We capture the network traffic from outside the Android OS and perform a detailed network protocol analysis during post-processing.

A. STATIC ANALYSIS

Android apps are packaged in *Android Application Package* (APK) files, a ZIP archive based on the JAR file format. An APK file contains an app's bytecode stored in Dalvik Executable (DEX) format, resources, such as UI layouts, as well a manifest file (`AndroidManifest.xml`). The manifest is mandatory and without its information an app cannot be installed or executed. Thus, as a first step, we unpack the archive and parse meta information from the manifest, such as requested permissions, services, broadcast receivers, activities, package name, and SDK version. In addition we examine the actual bytecode to extract a complete list of available Java objects and methods.

We use the information gathered during static analysis to assist in automating the dynamic analysis, mainly during the stimulation of an app's components. Furthermore, an app requesting dangerous permissions can be indicative of malicious behavior. Therefore, we extract the permissions that are requested as well as the permissions that are actually used in the app's bytecode to later compare them to permissions used during runtime.

B. DYNAMIC ANALYSIS

Being designed for smartphones and tablets, Android is predominantly deployed on ARM-based devices. Since the underlying architecture should be of no difference to the apps, we decided to build our sandbox for the ARM platform, the typical environment for Android, and chose a QEMU-based emulation environment capable of running arbitrary Android OS versions. Since Android apps are based on Java, we instrument the underlying virtual machine (VM), called the Dalvik VM, and record activities happening within this environment. This allows us to monitor the file system and network, as well as phone events, such as outgoing SMS messages and phone calls, and the loading of additional DEX or native code during runtime. For a comprehensive analysis, however, these capabilities are not sufficient. Therefore, we implemented the following additional analysis facilities:

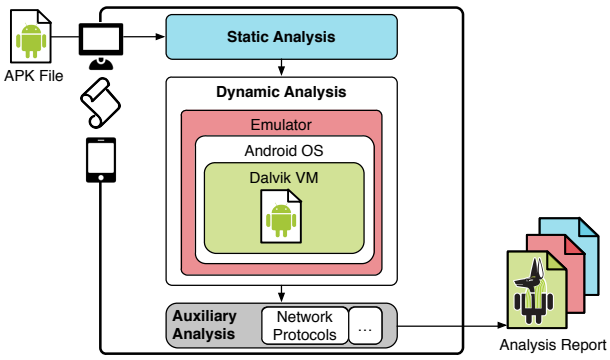


Fig. 1: System overview of ANDRUBIS.

- **Stimulation.** Due to the event-driven nature of Android, comprehensive input stimulation is invaluable for triggering interesting behavior from the app under analysis.
- **Taint Tracking.** To track privacy sensitive information ANDRUBIS uses taint tracking at the Dalvik level [28], which enables us to detect the leakage of sensitive information.
- **Method Tracing.** We record invoked Java methods, their parameters, and their return values. Combined with our static analysis, we can use method traces to measure the code covered during an analysis run, e.g., for evaluating and improving our stimulation engine.
- **System-Level Analysis.** To provide means for analysis beyond the scope of the Dalvik VM, we implemented an introspection-based solution at the emulator level. This enables us to monitor the system from outside the Android OS and to track system calls of native libraries and root exploits.

The output produced by the method tracer and the system-level analysis is not displayed in the public ANDRUBIS analysis report. As these tasks are quite resource-intensive and the log files are quite large, we only perform them on a subset of samples and provide them on an on-demand basis for researchers and analysts rather than ordinary users.

The remainder of the sandboxing system (network setup and traffic capturing, host environment, database, etc.) is comparable to conservative analysis systems. To mitigate potentially harmful effects of our analysis environment to the outside world while allowing apps under analysis to use the network, we took precautions to prevent apps from executing DoS attacks, sending spam e-mails or propagating themselves over the network. This part is based on our experience with Windows malware analysis and proved to be effective with ANUBIS in the past [19].

1) Stimulation: The purpose of stimulation is to exhaustively explore the functionality of an app. One major drawback of dynamic analysis in general is the fact that only a few of all possible execution paths are traversed within one analysis run. Furthermore, Android apps can have multiple entry points besides the main activity, which is displayed to the user when an app is launched, so that apps can react to system events or interact with each other. Luckily, since the app's manifest lists the

various components (activities, services, and broadcast receivers), we can stimulate them individually. Additionally, we can initiate common events that malicious apps are likely to react to.

Our stimulation approach includes the following sequence of events: after the initialization of the emulator, ANDRUBIS installs the app under analysis and starts the main activity. At this point, all predefined entry points are known from static analysis. During runtime ANDRUBIS keeps track of dynamically registered entry points, enabling it to perform the following stimulation events:

Activities. An activity provides a screen to interact with and defines the interaction sequences and UI layout presented to the user. Activities have to be registered in the manifest and cannot be added programmatically. Therefore, by parsing the manifest, ANDRUBIS has full knowledge about an app's activities and invokes each activity separately, effectively iterating all existing dialogs within an app.

Services. Background processes on the Android platform are usually implemented as services. In contrast to activities, they come without a graphical component and are designed to provide background functionality for an app. Naturally, they are also of interest to malware authors, as they can be used to implement communication with command and control (C&C) infrastructures of botnets, leak personal information, or forward intercepted text messages to an adversary. Again, all services used by an app must be listed in the manifest. Their existence, however, does not automatically mean the service is started: to save battery life and preserve memory, services have to be started on demand, with a lifetime defined by the programmer. For ANDRUBIS we utilize a customized *Activity Manager* to iterate and start all listed services of an app automatically after it has been installed.

Broadcast Receivers. Other possible entry points for Android apps are broadcast receivers. Broadcast receivers are basically event handlers used to receive events from the system or other apps on the Android platform. For example, a broadcast receiver for the `BOOT_COMPLETED` event can be registered to start an app after the phone has finished its boot sequence or a broadcast receiver for the `SMS_RECEIVED` event can be registered to intercept incoming SMS messages.

Just like services and activities, broadcast receivers can be registered in the manifest. However, for broadcast receivers this is not mandatory. In order to provide the possibility to react to certain events, or to provide communication with other apps dynamically, they can also be registered and deregistered at runtime. Therefore, we intercept the calls to `registerReceiver()` to obtain a list of dynamically registered event handlers that we can stimulate. Similar to the previous stimuli, ANDRUBIS uses the *Activity Manager* to invoke all statically registered broadcast receivers found in the manifest as well as the ones that have been dynamically registered.

Common Events. A far superior method compared to directly stimulating broadcast receivers with a targeted event is to emulate the events that apps might react to and especially malicious apps are likely to be interested in. Thus, we broadcast events such as boot completion, incoming SMS and phone calls, changes in the GPS lock, and changes in the WiFi and cellular connectivity. In contrast to directed stimuli, these events occur at the system level and thus also trigger receivers of the Android OS itself. That, in turn, avoids causing inconsistent states the OS would have to recover from when only invoking the event handler registered by an app.

Application Exerciser Monkey. The remaining elements that need to be stimulated are actions based on user input (button clicks, file upload, text input, etc.). For this purpose, we use the Application Exerciser Monkey, which is part of the Android SDK and generates semi-random user input. Originally designed for stress-testing Android apps, it randomly creates a stream of user interaction sequences that can be restricted to a single package name. While the triggered interaction sequences include any number of clicks, touches, and gestures, the monkey specifically tries to hit buttons. As some use cases might require repeatable analysis runs without any random behavior introduced by the monkey, we optionally provide a fixed seed in order to always trigger the same interaction sequences.

2) Taint Tracking: Data tainting is a double-edged sword when it comes to malware analysis. On one hand, it is the perfect tool to keep track of interesting data; on the other hand, it can be tricked quite easily if a malware author is aware of this mechanism within an analysis environment [29]. By leaking data through implicit flows, for instance, it would be possible to circumvent tainting. Furthermore, enabling data tainting always comes at the price of additional overhead to produce and track taint labels. Still, the possibility to track explicit flows of sensitive data sources, such as contacts, phone-specific identifiers, and the location, to the network is a valuable property of a dynamic analysis system. ANDRUBIS leverages *TaintDroid* [28] to track such sensitive information across application borders in the Android system. The introduced overhead in processing time of approximately 15% [28] is also acceptable for our purposes. As a result, ANDRUBIS can log tainted information as it leaves the system through three sinks: network, SMS, and files on disk.

3) Method Tracing: For an extensive analysis of Java-based operations, we extended the existing Dalvik VM profiler capabilities to incorporate a detailed method tracer. For a given app we log the executed Java methods on a per-thread basis. The method trace contains method names and their corresponding classes, the object's `this` value (if any), all provided parameters and their types, return values, constructors, exceptions and the current call depth. For non-primitive types, the tracer looks up and executes the object's `toString()` method, which is then used to represent the object.

Together with the output gained from system-level analysis (described in the next section), the fine-grained method traces can assist reverse engineering efforts, serve as input to machine learning algorithms, or they can be used to create behavioral signatures. Furthermore, by mapping the method trace to permissions utilizing a permission mapping, such as the ones provided by PScout [30] or Stowaway [31] we can determine the permissions an app actually used during runtime.

Our main incentive to integrate method tracing, however, is to measure the code covered during the individual phases of the stimulation engine. To this end, we first compile a list of executed method signatures. We then map this list against the list of functions extracted during static analysis based on their Java method signature excluding parameter types and modifiers, i.e., on their `<package>.<subpackage>.<class>.<method>` representation. Finally, we compute the code covered as the overall percentage of functions that were called during the dynamic analysis. However, apps may contain numerous functions that, during a normal execution, will never be invoked, such as localization and in-app settings or large portions of unused code from third-party libraries. Thus, for a less conservative and more realistic code coverage computation we can whitelist known third-party APIs or limit the computation to the main app package's code.

4) System-Level Analysis: In addition to monitoring the Dalvik VM, and in contrast to most related work on Android malware, ANDRUBIS also tracks native code execution. By default, Android apps are Java programs, being distributed as a DEX file within an APK file. Hence, the default way of programming for the Android platform and executing Android

apps is by running Dalvik bytecode within the Dalvik VM. However, Android apps are not limited to Dalvik bytecode and can also execute system-level code by loading native libraries via the Java Native Interface (JNI). While this functionality is mainly intended for performance-critical use cases, such as displaying 3D graphics, apps are not restricted to loading the native libraries shipped with the Android OS; instead they can also ship and load their own native libraries and, in turn, execute arbitrary system-level code. Naturally, the execution of this code takes place outside of the Dalvik VM and, thus, the behavior of this code is invisible to the analysis at Dalvik VM level. For malicious apps the use of native code is attractive as the possibilities to perform malicious activities, such as the usage of exploits to gain root privileges, are far greater than within the Dalvik VM – making system-level analysis indispensable for drawing a complete picture of an apps's behavior. In addition, Google recently introduced the new Android Runtime (ART) [32] that compiles Dalvik bytecode to native code at installation time. With the replacement of Dalvik with ART as the default runtime in upcoming Android OS releases [33], the capability to perform system-level analysis will gain further importance.

Being based on Linux, there are a couple of ways to implement system-level instrumentation in Android, such as using `LD_PRELOAD`, `ptrace` or a loadable kernel module. We decided to use the most transparent and non-intrusive way – virtual machine introspection (VMI). With VMI our analysis code is placed outside of the scope of the running Android OS, right in the emulator's codebase, and tracks the complete list of system calls performed by the emulator as a whole, including the OS. To capture the system-level behavior of the app under analysis, we ultimately need to extract the system calls executed by the library code that was loaded via JNI. To this end, we intercept the Android dynamic linker's actions in order to track shared object function invocations. System call tracking bundled with this information enables us to associate system calls with invocations of certain functions of loaded libraries. Android assigns a unique user ID (UID) to every app and runs the app as that user in a separate process – allowing us to associate system calls with apps based on the process UID. The result is a list of native code events caused by just the specific app under analysis.

C. Auxiliary Analysis

Network traffic is one of the most essential parts when establishing malware-detection metrics, with C&C communication being of particular importance. According to studies performed in production environments [34], more than 98% of Windows malware samples established a TCP/IP connection. Thus, in addition to tracking sensitive information to network sinks via taint tracking, we also capture all the network activity during analysis regardless of the performed action or the app causing it. This is necessary since apps not requesting and using the `INTERNET` permission themselves, can still use other installed apps like the browser, to send data over the network. Another way to transmit network data without requesting the appropriate permissions is by exploiting the Android OS and circumventing the permission system as a whole.

During post-processing we perform a detailed *Network Protocol Analysis* that extracts high-level network protocol features from the captured network traffic suitable for identifying interesting samples. Currently, we focus on the well-known and often used protocols DNS, HTTP, FTP, SMTP, and IRC.

III. Andrubis as a Service

In this section we present insights gained from offering Andrubis as a publicly available service for the past two years and the dataset of apps we collected along the way.

A. Submission Statistics

We base our analysis on a dataset collected over the span of exactly two years, between June 12, 2012 and June 12, 2014. We distinguish between *submissions* (all analysis requests Andrubis received), *tasks* (submissions for which the analysis was performed), and *samples* (unique apps based on their MD5 file hash). Overall, Andrubis received 1,778,997 unique submissions. Since Andrubis usually returns cached analysis reports in case an app is submitted multiple times (unless a user requests a re-analysis of a previous task), it performed analysis tasks for 1,073,078 (around 60%) of submissions. In total Andrubis received and analyzed 1,034,999 (58.18%) unique samples.

To put the number of Android samples into perspective we compare them to overall submissions to Anubis. During our observation period Anubis received a total of over 22 million samples, Android apps thus amount to close to 5% of overall samples. However, since the submission interface only assigns submissions of ZIP archives containing `classes.dex` and `AndroidManifest.xml` to Andrubis, we only report numbers on APK files and not submissions of related files such as stand-alone DEX classes. A large number of samples comes from malware feeds as part of exchange agreements. We receive feeds with Android apps from nine sources, most of them submitting both Windows executables and Android apps – with the exception of AndroTotal almost exclusively submitting APK files. Other malware feeds from security researchers and AV vendors contain from as little as 1% to up to 37% Android apps. The largest sample feed contributing more than five million samples in the observation period contains around 10% Android submissions.

Figure 2 shows the weekly number of total submissions, submissions through sample exchanges, i.e., semi-regular feeds of samples, new samples and analyzed samples. Submissions peaked in August 2012 and January 2013, when we received bulk submissions from Google Play crawls and in July 2013 when one feed submitted a higher than usual amount of samples. In November and December 2013 AndRadar [6] started submitting a backlog of apps before switching to a regular feed of apps. Besides a power outage in January 2014 Andrubis has been operating reliably and analyzed up to the current maximum capacity of 3,500 new apps per day.

In order to estimate the number of different users using our service, we distinguish them either by their username, or in case of anonymous submissions, by their IP address. Users can register for an account in order to gain special privileges, such as a higher priority for their tasks or the ability to force the re-analysis of an app. The account management is shared with Anubis, but 152 registered users submitted at least one Android app. With anonymous submissions coming from 8,123 unique IP addresses we estimate that 8,275 unique users from 130 different countries are using Andrubis. The majority of submissions come from registered users, with 15 individual users amounting to over 95% of total submissions, and only 38,905 (3.76%) of submissions coming from anonymous sources. Table I categorizes users by their number of submissions from single submitters with less than 10 submission to "power users" with more than 10,000 submissions. The maximum amount of 557,559 submissions for a single user stems from one of the aforementioned malware feeds.

Figure 3 shows the number of different sources, i.e., the number of distinct users that submitted a particular app: around 70% of apps were submitted by only one user and only 1.5% of apps were submitted by more than three distinct users. In general, malicious samples were submitted more frequently than goodware apps (with the exception of the top two apps): over 80% of apps submitted by more than five different users belong to the malware category. The popular game Flappy Bird was also

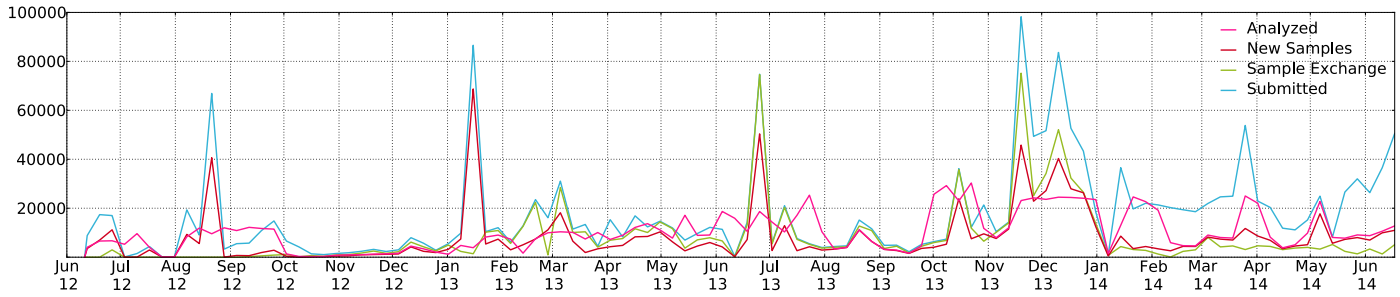


Fig. 2: Weekly number of total submissions, submissions through sample exchanges, new and analyzed samples.

TABLE I: Users categorized by their number of submissions and proportion of all submissions.

Category		# of Users	% of All Submissions
Bulk	(10,000+)	15	95.82%
Large	(1,000-10,000)	13	2.47%
Medium	(100-1,000)	34	0.59%
Small	(10-100)	247	0.41%
Single	(1-10)	7,966	0.72%

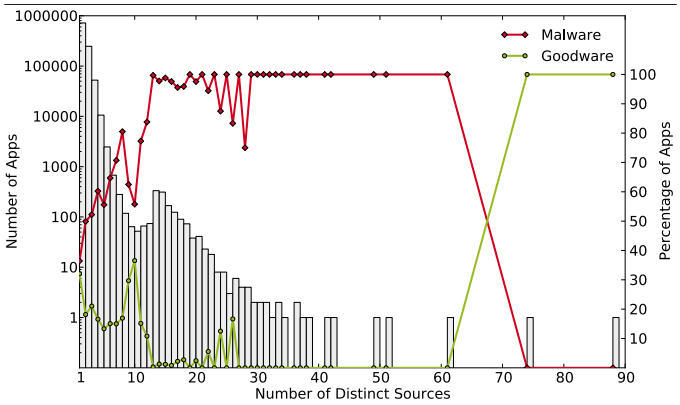


Fig. 3: Number of unique users submitting an app and percentage of goodware and malware submitted by N users.

the "most popular" app submitted to ANDRUBIS by 88 different users. The second most submitted app is the alpha version of our mobile interface to ANDRUBIS, with which users can submit apps directly from their phones (it is currently available for download on the ANDRUBIS's web interface). However, the remaining most popular apps (and all other apps submitted 26 times and more) are part of malware corpora, such as Contagio [35] and the Android Malware Genome Project [36].

B. Analysis Results and Limitations

Overall, ANDRUBIS successfully analyzed 91.67% of all apps. For the remaining samples, 0.34% failed due to bugs in our analysis environment and 7.99% of samples failed to install in our sandbox due to various reasons, such as the APK file being corrupt or the app exceeding the API level of the Android OS version installed in our sandbox. ANDRUBIS currently runs Android 2.3.4 Gingerbread and thus only supports apps with a minimum required API level ≤ 10. We know that 0.78% of apps require a newer OS version, and for 6.66% of samples we could either not parse the manifest or they did not specify an API level. However, this has no significant impact on malware analysis as of now. Instead, it is mainly a concern for goodware, of which 2.11% (6,099) require a higher API level, while only 0.10% (439) of malicious apps fail for this reason. Such a behavior by malware authors is expected: their malicious apps require a lower API level in order to maximize the potential user base for their apps, and, in turn, their profit. This is also confirmed by Figure 8 in the Appendix, which shows that malware authors are much slower in adopting new API levels than goodware authors.

C. Scalability

Currently, ANDRUBIS is capable of processing around 3,500 new apps per day, i.e., apps that have never been analyzed before and for which no cached report is available. The analysis of an app takes around 10 minutes, with 240 seconds analysis runtime in the sandbox plus an additional 387.27 seconds on average for pre- and post-processing. Pre-processing includes setting up the emulator and loading the Android OS snapshot, installing the app, parsing the manifest and performing static analysis on the APK. Post-processing includes extracting protocol information from the network traffic and preparing the final analysis report.

Judging from our experience running the Windows malware analysis service ANUBIS and similar to Andlantis [37], ANDRUBIS scales well by simply adding new workers to handle the analysis of new samples should submissions increase. However, already with the current throughput of over 100,000 apps per month, ANDRUBIS is capable of analyzing samples at market scale. For example, Google Play, the largest app store (by far), added, on average, 37,500 new apps per month in the last year, with peaks of up to 85,000 new apps in December [38]. When it comes to malware, Android still falls far behind the plethora of Windows samples circulating in the wild: Sophos estimates 2,000 new Android malware samples are being discovered each day [4], a number ANDRUBIS can handle in the current configuration and setup comfortably.

D. Sample Sources

One limitation of a public web interface allowing anonymous submissions is the lack of meta information associated with submitted apps. Since the majority of apps are submitted by registered users, however, we can associate them to sample exchanges, part of our own crawling efforts, or the integration of tools, such as AndRadar. Table II in the Appendix summarizes the number of apps from each source, as well as the proportion of benign and malicious apps (see the next section on how we separated goodware from malware). The apps in our dataset originate from the following eight sources:

Sample Exchange. These apps make up the majority of our dataset and come from sample sharing with other researchers. Most of the feeds are part of long-standing sample exchanges that started with Windows samples, but now also include Android samples, too.

Google Play. We initially crawled 100,000 apps from the Google Play US Store during May and June 2012. Additionally, since December 2013, we receive apps crawled from AndRadar that match a seed of malicious apps and are located in the Google Play Store. In April 2014, we started fetching the top apps overall, top new apps and top apps per category (limited to 500 entries each by Google Play) from the Google Play US and AT Store on a daily basis.

Alternative Markets. These apps are crawled by AndRadar from 15 alternative markets, including seven Chinese and one Russian market. This dataset is biased towards malware since AndRadar aims at locating malicious apps.

VirusTotal. We regularly download samples from VirusTotal. However, this dataset not only contains malware, but also a small percentage of samples labeled as adware as well as some samples not detected by any AV scanner.

Malware Corpora. This is a collection of manually gathered malware samples we encountered over time as well as samples from vetted malware corpora, such as the Contagio Mobile Malware Dump [35], the Android Malware Genome Project [36], and Drebin [39]. However, besides the relatively small Contagio set (470 samples) that is regularly updated, which, in turn, makes comparison hard, available malware corpora are already quite dated: the 1,200 samples (49 different families) from the Android Malware Genome Project were collected from August 2010 to October 2011, the 5,560 apps (179 families, including the Genome Project) from the Drebin dataset were collected in the period of August 2010 to October 2012.

Torrents. We downloaded apps from `isohunt.com`, `thepiratebay.se`, and `torrentz.eu` for which the torrent had at least ten seeders. To avoid distribution of copyright-protected content, our torrent client did not upload any data at all.

Direct Downloads. We downloaded a set of apps through direct downloads from various one-click hosters, including `filestube.com` and `iload.to`.

Unknown. These apps stem from anonymous user submissions and thus we do not have any information where they originate from.

E. Collected Dataset

The dataset gathered from samples submitted to ANDRUBIS allows us to perform a longitudinal analysis of Android app features in general and features specific to benign apps and malicious apps. First, however, we need to separate the dataset into subsets. Since the primary goal of ANDRUBIS is to provide researchers with a comprehensive static and dynamic analysis report of an app, not to automatically identify apps as goodware or malware, we have to rely on AV signatures as our ground truth:

Goodware. We classify apps as goodware if they do not match any AV signature from VirusTotal's AV scanners. Goodware apps make up 27.90% of our dataset.

Malware. We classify apps as malware if they match at least t AV signatures. We experimented with different settings for the threshold t and settled on at least 5 AV labels, ignoring all AV labels indicating adware. With thresholds $t > 5$ a large portion of apps exhibiting malicious behavior, such as exploiting the Master Key vulnerabilities (see Section IV-A7), would have been missed. Malware apps make up 41.15% of our dataset.

All. In addition to goodware and malware our complete dataset contains 30.95% other apps that are detected by 1 to 5 AVs or that are classified as adware.

Estimation of Release Date. In order to perform any kind of longitudinal analysis on our dataset and categorize apps by the year of their release, we need to estimate the age of each sample. Besides this yearly division of our dataset we also would like to have a more precise estimate to allow for a fine-grained evaluation, such as the time it takes for us to receive and analyze samples after they have been released. We estimate the age of an app from four data points: (1) the last modification date of the APK file (`zip_modification_date`), (2) the release date of the SDK indicated by the minimum required API level (`sdk_release`), (3) the date a sample was first published in any of the markets monitored by AndRadar (`market_release`), and (4) the date a sample was first submitted to ANDRUBIS (`first_seen`).

For (1), the last modification date of the APK file, we parse the timestamp for the archive member that was modified last, usually the app's certificate, from the ZIP central directory file header. Naturally, this date can be tampered with, as evidenced by 273 apps feigning a modification date in 1980, the first year the ZIP file format supports for timestamps, further 9,703 before the first Android version was released in 2008, as well as 86 apps dated in the future, up to the year 2107. For (2), we parse the minimum required API level from an app's manifest and map it against the Android version history [40]. For (3), we have information from AndRadar for 68,197 apps in our dataset, since not all markets specify the date an app was uploaded and we do not want the overall release date of an app but the date when a specific version (based on the MD5 file hash) was released.

In general, we trust the modification dates extracted from the ZIP header as we only encountered relatively few outliers exhibiting unrealistic modification dates. However, we sanitize the `zip_modification_date` by checking the `sdk_release` as a lower bound for when the app could have been released in case the ZIP timestamp was predated, and the `market_release` as an upper bound when the app was first seen in the wild in case the app was postdated. In the normal case the app requests a specific API level after the corresponding SDK was released and the app is built before it is released to the public, e.g., an application market. In this case we estimate the release date (`apk_date`) as the date the ZIP was created:

$$\texttt{sdk\_release} < \texttt{zip\_modified} < \texttt{market\_release}$$
$$\texttt{apk\_date} = \texttt{zip\_modified}$$

For 10,000 apps (1.04%) the ZIP file was created before the corresponding SDK was released. This could be either due to the ZIP file header being tampered with or the app being part of an alpha/beta test of an unreleased SDK. Since an app cannot be installed on devices if it requires a higher API level than the currently available Android OS version, we assign the date of SDK release to the release date:

$$\texttt{zip\_modified} < \texttt{sdk\_release}$$
$$\texttt{apk\_date} = \texttt{sdk\_release}$$

In only six cases the market release date indicates that the app was published before the requested SDK level was released. This could be due to an error on the developers side, unintentionally requesting a higher API level than required. In this case we choose the maximum of the SDK release and the ZIP creation date:

$$\texttt{market\_release} < \texttt{sdk\_release}$$
$$\texttt{apk\_date} = \max(\texttt{sdk\_release}, \texttt{zip\_modified})$$

Around 5,000 apps (0.50%) were published in a market before the ZIP was last modified. Since this means that the ZIP header obviously has been tampered with, we set the release date to the market release as the first date we saw the app in the wild:

$$\texttt{market\_release} < \texttt{zip\_modified}$$
$$\texttt{apk\_date} = \texttt{market\_release}$$

We now can use the `apk_date` to estimate the *analysis delay*, the time it takes between an app being released and the app being submitted to ANDRUBIS. Figure 4 shows the CDF of the analysis delay for the first and second year of operation. Within the first year we only saw 15% of samples within one week of their release for both malware and goodware. In the second year this number significantly increased to over 40% for goodware apps, in part due to our crawling of the popular new apps from the Play Store on a daily basis. In the first year ANDRUBIS analyzed 60%-70% of all samples within the first three months. This number increased for apps of all categories

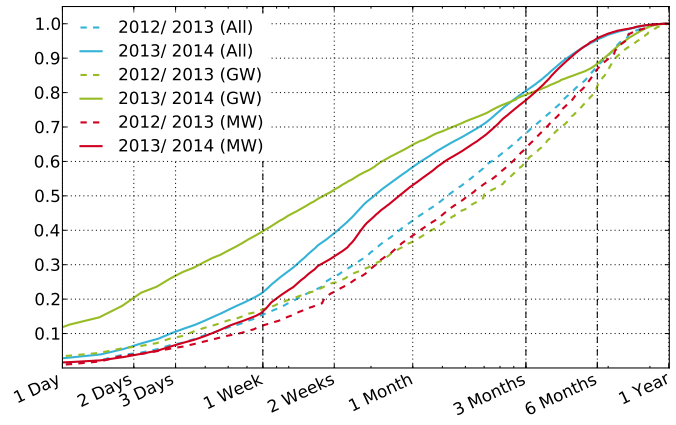


Fig. 4: CDF of time between APK creation and first submission to ANDRUBIS in the first (June 2012 - June 2013) and second year (June 2013 - June 2014) of operation.

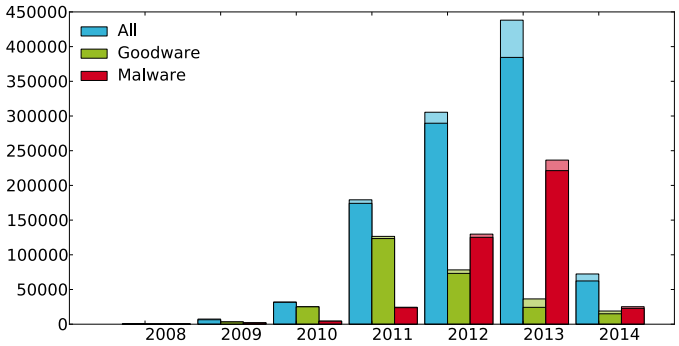


Fig. 5: Number of all, goodware and malware apps in our dataset per year. The share of successful analyses is highlighted in a darker shade.

to 80% in the second year. Finally, the number of apps analyzed within six months of their release increased from 2012 to 2013 by 10 percentage points for apps of all categories, to close to 90% of goodware and 95% of malware samples.

Finally, for categorizing our dataset by release year, we also include the date ANDRUBIS first saw an app in the estimation and assign $\min(\texttt{apk\_date},\texttt{first\_seen})$ as the final app release date. This results in the dataset depicted in Figure 5, separated by app release year and category (*All*, *Malware*, *Goodware*). Android was released in September 2008, however, malware first surfaced in 2010 [41] and apps released before 2010 amount to less than 0.76% of all apps in our dataset, thus, we focus in our following evaluation on apps released between 2010 and 2014.

IV. ANDROID MALWARE LANDSCAPE

We already used the dataset described in the previous section in prior work for exploring WebView-related vulnerabilities [42]. Based on apps collected between July 2012 and March 2013 we determined that 30% of apps were vulnerable to web-based attacks by exposing native Java objects via JavaScript.

In the following, we give a summary of apps' static analysis features and behavior during dynamic analysis and identify trends for *All*, *Goodware* and *Malware* samples over the past four years from 2010 to 2014. As our observations show, dynamic analysis is increasingly able to capture behavior otherwise missed by static analysis. This is in part due to the increasing use of dynamic code loading amongst malicious and benign apps and their use of obfuscation techniques and/or DRM protection. Additionally, while 57.08% of malware samples employ reflection with no significant change over the years, use of reflection amongst all apps has increased significantly from 43.87% in 2010 to 78.00% in 2014, and even more in goodware (from 39.55% to 93.00%). Therefore, it is essential for large-scale evaluations to include dynamic analysis systems.

A. OBSERVATIONS FROM STATIC ANALYSIS

While dynamic analysis is gaining importance in forming a complete picture about an app's functionality, for some features evaluation of static features already provides valuable insights. In this section we take a look at permission requests and their usage according to static analysis, application names, developer certificates, resources sharing between apps, registered broadcast receivers, the use of third-party libraries and the exploitation of Master Key vulnerabilities.

1) Requested Permissions: Android apps can define and request arbitrary permissions: in fact, we observed almost 30,000 unique permissions being requested overall. Here, we focus on permissions defined and safeguarded by the Android OS. In addition to parsing all requested permissions from the manifest, we statically extract the usage of permissions from the app's source. While this approach ignores permissions that are requested, but only used in code dynamically loaded at runtime, we could use ANDRUBIS's method tracer (Section II-B3), to determine the permission usage during dynamic analysis in future experiments. For now the dynamic extraction of used permissions is in an experimental state and results are only available for a subset of samples.

We statically extract the usage of 143 permissions, covering the most interesting and commonly requested permissions as shown in Table III (in the Appendix). In line with previous findings on permission usage amongst malware, malicious samples generally request more permissions than goodware, but use less of them: malicious apps request 12.99 (11.57 when only looking at the subset of permissions we can statically extract) permissions on average, but use only 5.31 of them, goodware apps on the other hand request 5.85 (5.56) permissions on average and use 4.50 of them. One explanation for this behavior is that malware samples request more permissions during installation than needed so that they have the possibility to load other code parts that use these permissions later on. Permission requests by malware have also increased from an average of 11.46 (10.19) in 2010 to 15.33 (13.93) in 2014, with the average number of used permissions increasing only from 5.51 to 5.86. The number of requested permissions for goodware has increased from 3.74 (3.58) in 2010 to 9.38 (8.45) in 2010, while the number of used permissions also increased from 3.13 to 5.62. For individual samples, the permission usage ratio has declined for both goodware and malware, however, more significantly for goodware: samples in this category from 2014 only use 13.38% of requested permissions in their code – a possible side effect of the increased use of dynamic code loading (see Section IV-B4). Figure 6 illustrates this development.

Table III shows an overview of the most frequently requested permissions for malware and goodware. While the most commonly requested permissions for both malware and goodware are related to accessing the Internet, checking the network connectivity and reading device specific identifiers from the phone state, the majority of malware samples also requests SMS-related permissions. Furthermore, the possibility to manipulate shortcuts on the home screen can be used for phishing attacks and is frequently requested by malware as well. Another critical permission requested by malware is `SYSTEM_ALERT_WINDOW`, which allows an app to show windows on top of all other apps, overlapping them completely. It is used to display aggressive ads and by ransomware that draws a window over all other apps to keep the user from accessing any other phone functionality.

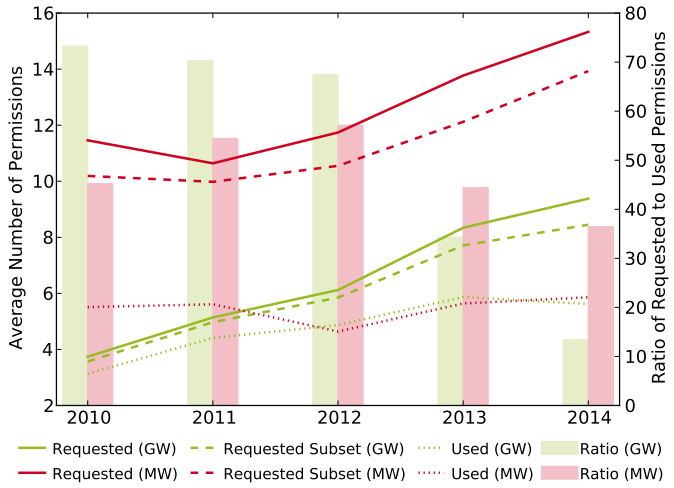


Fig. 6: Goodware (GW) and malware (MW) apps request an increasing number of permissions (overall as well as from the subset of permissions we statically extract), but permission usage stays constant – a side effect of the increasing use of dynamic code loading and obfuscation.

It is also important to note that not only individual but also combinations of permissions can be security-critical: while the `INSTALL_SHORTCUT` permission, which is requested by more than half of the malicious apps, is classified as dangerous, the same functionality can be achieved through the combination of the normal `READ_SETTINGS` and `WRITE_SETTINGS` permissions [43]. This is common practice amongst malware, with 10.88% of malware samples requesting both permissions, while only 0.20% of goodware samples do so.

During our evaluation of dynamic analysis features (see Section IV-B), we observed samples attempting to send SMS, connect to the Internet or accessing the SD card, without having the appropriate permissions – actions that will be prohibited by the Android OS. One explanation, besides a simple oversight, is developers mistyping the intended permission in some cases, for example as `andorid.permission.*`.

2) Application Names: The package name is the official identifier of an app, i.e., no two apps on a given device can share the same package name. Some markets, such as Google Play, also use it as a unique reference, but developers are not restricted from creating an app with an already existing package name. For malware authors reusing the package name of a legitimate app is also a way to masquerade as a benign app. Consequently, malware samples are far more likely to reuse package names than goodware samples: while 73.78% of goodware package names are unique, the same holds true for only 25.72% of malware's package names. Note, this number is likely to be slightly biased by submissions from AndRadar that explicitly locates apps in markets based on their package name to model and analyze how they spread [6].

A total of 8.50% of malware samples share their package name with legitimate apps from our goodware set – in total 4,059 distinct package names, half of which are currently available in the Google Play Store. Among the most frequently repackaged apps are Armor for Android Antivirus (`com.armorforandroid.security`, 387 samples), Steamy Window (`com.appspot.swisscodemonkeys.steam`, 93 samples), Opera (`com.opera.mini.android`, 68 samples), and Flappy Bird (`com.dotgears.flappybird`, 23 samples) – besides the paid Armor Antivirus all apps exceed 5 million downloads on the Google Play Store.

By far the most often shared package name, shared by 1,735 malicious apps with a single legitimate Google Play app, is `com.app.android`, however, more likely due to careless naming on the legitimate app's developers side. In general, authors of malicious apps tend to favor generic names and reuse them between samples, `com.software.app` and `com.software.application` being the most popular ones with 9,256 and 8,321 unique samples respectively. Starting in 2012, we observed malware authors adopting random looking package names, such as `ouepxayhr.efutel`, `ovbknnfm.xwscmnoi` and `rpyhwytfysl.uikbvktgwp`. F-Secure observed those package names being particularly popular amongst the *Android.Fakeinst* family [2]. However, contrary to the first impression, package names are not randomized on a per-app basis, as evidenced by up to 3,234 unique samples per name.

3) Certificates: Certificates are a corner-stone in Android security: each and every Android app has to be shipped with its developer's certificate and signed with his private key so that it can be installed. Android uses the certificate to enforce update integrity, i.e., it only allows updates signed with the same key, and it uses it to allow resource sharing and permission inheritance between apps from the same author [44].

Google does not impose any restrictions on the certificates used to sign Android apps and over 99% of all certificates are self-signed. We collected increasingly more apps signed by the same key, for goodware and malware alike. While, in 2010, 19.21% of all keys were used to sign more than one goodware app and 28.57% of the keys were used to sign more than one malicious one, this increased to 40% for both goodware and malware in 2014. This not only means that we are collecting more apps by the same developers, but also that blacklisting certificates used to sign malware is a viable option to keep malware from spreading. Especially widely used are four test keys distributed as part of the Android Open Source Project (AOSP): 8.92% of malicious samples are signed with one of these test key, however, the ratio significantly decreased from 65.29% of malicious apps in 2010 to 7.29% in 2014. Although those keys should not be used by legitimate apps, 2.26% of goodware apps are signed with a publicly available test key – making them vulnerable to attack: as we will show in the next section, if a user has such an app installed, malware signed with the same test key can potentially share permissions with the vulnerable app.

To our surprise we also found four samples, each labeled by more than 11 AV scanners as part of the *Android.Bgserv* malware family, that are signed with a valid Google certificate. These apps with the package name `com.android.vending.sectool.v1` are a malware removal tool by Google, mistakenly flagged by malware by numerous AV vendors [26].

4) Application Interdependencies: The Android system assigns, by default, a unique user ID (UID) to each app and runs it as that user in a separate process. Apps, however, can share their UID with other apps by specifying a `sharedUserId` in the manifest. This allows apps to share data, run in the same process, and even inherit each other's permissions [44], all under the prerequisite that apps are signed with the same key. Clearly, this feature also allows collusion amongst apps [45]: a malicious payload could be spread across multiple innocent looking apps. We saw this feature more commonly implemented in goodware than in malware: 1.14% of apps share their UID while only 0.29% of malicious apps do. This functionality becomes especially security critical when combined with an exploit for the powerful Master Key vulnerabilities (detailed in Section IV-A7). In theory, attackers could inject their code into apps not requesting any permissions at all but inheriting permissions from more privileged apps through a shared UID. Apps can even try to gain system privileges by exploiting an app signed with a platform certificate and sharing the UID with `android.uid.system`.

Furthermore, with numerous apps being signed with the test key from the AOSP, crafting a malicious app inheriting the permissions from other apps is possible even without having to utilize an exploit to circumvent the app signing process. In fact, 6.79% of benign and 17.57% of malicious samples that share a UID are signed with a public test key. This becomes especially critical when the Android OS itself is signed with a public key: according to DroidRay [46], a recent security evaluation of custom Android firmware, out of 250 firmware images, 56.80% were signed with a key pair from the AOSP. In our dataset, we identified 84 apps (4 of which were not detected by any AV scanners, the remainder was labeled as *Android.Fjcon*) that were capable of gaining system privileges through UID sharing with `android.uid.system`. All samples were signed with the same AOSP key pair used to generate the system signature for 134 (53.60%) of the firmware images evaluated by DroidRay.

5) Broadcast Receivers: Apps can register broadcast receivers for arbitrary custom events, however, we focus our analysis on broadcast receivers listening for system events. Broadcast receivers are by far more widely used in malicious apps than in benign apps: 82.18% of all malware samples register one or more broadcast receivers, while only 41.86% of goodware sample use this feature. Table IV (in the Appendix) lists the most frequently registered broadcast receivers for both categories. Goodware mainly watches for notifications to update their widgets, install referrers from the market and a user being present, probably to suspend idle mode quickly whenever a user unlocks the phone so that new data can be fetched and the app's status can be updated. Malware, on the other hand, often registers itself as a service, which is running in the background, and does not care for user input. More than half of all samples listen for the `BOOT_COMPLETED` event, which triggers as soon as the phone has booted the Android OS, and for the event that is published upon receipt of incoming messages, both text-based (`SMS_RECEIVED`) and data-based (`DATA_SMS_RECEIVED`). However, we only saw listeners for data-based SMS in 2012 and 2013 with 24.43% and 10.41% of malware samples listening for this event. We also see growing interest of malicious apps in the `CONNECTIVITY_CHANGE` and `AIRPLANE_MODE` receivers since 2012 with a peak of 17.93% and 14.99% in 2013 respectively. Furthermore, malicious apps started using Device Administrator Privileges, which makes them harder to uninstall. The latter are used by 11.94% of malware samples in 2014, which register for the `DEVICE_ADMIN_ENABLED` event.

6) Third-Party Libraries: We checked all apps in our dataset against a list of the 53 most popular *advertisement (ad) libraries* according to AppBrain [47]. Fewer malicious (17.45%) than benign (44.32%) apps come bundled with ad libraries, presumably in part because we excluded samples labeled as adware from our malware dataset. However, with ad fraud being one way to monetize malicious app installs, malicious samples include more ad libraries simultaneously: we saw a maximum of 13 ad libraries in a single goodware app and 14 ad libraries in a single malware app with 1.56 and 2.05 libraries on average respectively. Table V (in the Appendix) lists the most popular ad libraries for goodware and malware. Besides Google's AdMob being the most popular across both categories, albeit with diverging percentages of over 35% in goodware to only 5.7% in malware, there is little overlap. With mobile malware being particular prevalent in China [48], malicious apps mainly include Chinese ad networks. Malware also favors aggressive ad libraries, such as AirPush and Adwo, often classified by AV scanners as adware and banned from Google's Play Store [49] by policy because they push advertisements to the notification bar.

Social networking libraries are used in 11.14% of goodware apps (8.86% Facebook, 3.38% Twitter, 1.89% Google+), while the number of malicious apps including such libraries is a negligible 0.78% (0.66% Facebook, 0.13% Twitter, 0.09% Google+), possibly indicating those libraries are shipped with the original app that was targeted by repackaging to include malicious code.

The same as for social networking libraries holds true for the use of *billing libraries*: 3.58% of goodware and only 0.53% of malware apps make use of billing services (3.08% Google Billing, 0.57% Paypal, 0.17% Amazon Purchasing and 0.03% Authorize.net in goodware; 0.35% Google Billing, 0.19% Paypal and 0.05% Amazon Purchasing in malware). Billing services for in-app purchases are harder to monetize for malware since payment providers usually have refund policies.

7) Master Key Vulnerabilities: In 2013 researchers reported the Master Key vulnerability [50] in the Android app signing process, which allows an app's content, including its code, to be modified without breaking the signature – essentially allowing attackers to inject malicious code into any legitimate applications without repackaging them. This vulnerability stems from discrepancies between the handling of the ZIP file format between the signature verification and installation process in Android. Shortly after the original Master Key vulnerability was published, two similar vulnerabilities were discovered [51,52].

Bug 8219321, the original Master Key vulnerability, is based on the fact that the ZIP file format allows two files with the same file name, thus allowing attackers to hide an additional `classes.dex` file that is deployed by the installer instead of the original one that is checked by the signature verifier. We saw this vulnerability being exploited in 1,152 samples (0.11%), all from 2013 and 2014, and only in malware, possibly due to AV scanners automatically flagging apps as there is no legitimate reason for this behavior.

Bug 9695860 stems from a signed unsigned integer mismatch in the length of the extra field of the ZIP file header. In addition to allowing attackers to inject an app with a malicious `classes.dex`, the exploitation of this vulnerability also breaks analysis tools utilizing the unpatched version of the Python `zipfile` [53], such as Androguard in the default configuration. We saw 4,553 samples (0.44%) triggering the Python bug. However, we only found two samples with an extra field length triggering an integer overflow and thus the vulnerability, one of them being a proof of concept [54].

Bug 9950697 lies within the redundant storage of the length of the file name in both the central directory of the ZIP file as well as the local file header. Again this vulnerability allows attackers to specify a file name large enough for the installer to skip the original `classes.dex` file and install the injected one. However, we only observed this bug being exploited in 447 (0.05%) of all samples (starting already in 2011 with the majority of samples being from 2013), with 92 malware and 26 goodware samples respectively.

B. Observations from Dynamic Analysis

In contrast to static analysis, dynamic analysis lets us monitor an app's behavior during runtime – including behavior caused by dynamically loaded code. In addition, the obtained information is more comprehensive and includes full paths of file system accesses, called phone numbers, recipients and contents of SMS, leaks of sensitive information, as well as usage of cryptographic algorithms and a full profile of the app's network behavior.

1) File Activity: Apps can both read and write the internal storage as well as external storage from SD cards. Overall 72.49% of goodware and 95.99% of malware read files, and 83.11% of goodware and 94.70% of malware write to the

file system during dynamic analysis in ANDRUBIS. When distinguishing file system access to the primary storage and access to the secondary storage, i.e., the SD card, it becomes apparent that SD card access is far more prevalent amongst malware: 22.02% of malicious apps read and 27.82% write files to the SD card, while only 2.91% of benign apps read and 6.69% write to external storage. Starting in Android 3.2 (Honeycomb) Google restricted third-party apps from accessing the SD card by limiting the `WRITE_EXTERNAL_STORAGE` to the primary storage and requiring the `WRITE_MEDIA_STORAGE`, which is only granted to system apps, for write access to the SD card. However, this change was largely ignored by OEM and custom firmware developers [55]. In our dataset 93.08% of goodware and 97.69% of malware apps that write to the SD card request the first permission, while only 0.59% of goodware and 0.08% request both. Static analysis completely failed to determine the usage of the `WRITE_EXTERNAL_STORAGE` permission and thus the write access to the SD card in any app. Furthermore, despite Google's policy to restrict write access to SD cards, this behavior has been steadily increasing in goodware apps from 2.89% in 2010 to 16.64% in 2014. Writing to SD storage in malware has been a constant behavior in around 30% of malware. This is likely to increase even more in the future with new possibilities for monetization being explored: recently the Cryptolocker family started encrypting files stored on the SD card and demanding ransom for the decryption key [56].

2) Phone Activity: Concerning mobile-specific behavior, only very few applications initiated phone calls during dynamic analysis: 0.24% of goodware apps and only 0.04% of malware apps. For both malware and goodware, 98% of those apps requested the corresponding `CALL_PHONE` permission, however, static analysis failed to determine any usage of this permission from the apps' source.

While the percentage of apps sending SMS in the goodware dataset is as low as the percentage of apps initiating phone calls (only 0.26%), we observed 15.00% of malicious apps sending text messages. This comes as no surprise: sending SMS to premium numbers is a popular monetization vector of mobile malware [4]. Again, 98.57% (goodware) and 99.15% (malware) of those apps requested the necessary `SEND_SMS` permission, while static analysis revealed that 85.37% (goodware) and 81.79% (malware) of those apps actually use this permission in their source code – again showing the value and importance of dynamic analysis to uncover behavior from hidden or obfuscated function calls. Phone numbers tend to be shorter for malware, also indicating the use of premium numbers – goodware apps send SMS to 410 unique numbers with an average length of 7.18 digits, while the 1,943 distinct numbers malware sends SMS to is only 4.26 digits on average. Furthermore, we observed malware samples sending up to 120 SMS to premium numbers during four minutes of dynamic analysis.

3) Data Leakage: Data leakage is significantly more prevalent in malware than in goodware: overall, 14.28% of goodware apps leak information over the network, while 42.53% of malicious apps do so. When looking at the dataset as a whole, data leakage to the network overall occurred in 38.79% of all apps and significantly increased from 13.45% in 2010 to 49.78% in 2014. Both goodware and malware leak device specific identifiers, such as the International Mobile Station Equipment Identity (IMEI), International Mobile Subscriber Identity (IMSI), Integrated Circuit Card Identifier (ICCID) and the phone number. Goodware mainly leaks the IMEI, while a quarter of malware leaks the IMSI and almost 14% of malware leaks the user's phone number. Leakage of names and phone numbers from the user's address book is also more common amongst malware than it is amongst benign apps. Instead, goodware mainly leaks the location, an information source less commonly leaked by malware samples. Few samples in general leak information on installed packages, the contents of SMS, the call log, and browser bookmarks. Table VI (in the Appendix) summarizes the information sources most commonly leaked to the network by goodware and malware.

Data leakage via SMS occurred only in 0.04% of goodware and in 0.72% of malware samples. This number, however, has increased over the past years, with 1.87% of malware samples leaking identifiers such as the IMSI, IMEI, ICCID and the phone number, but also forwarding incoming SMS and the call log via SMS in 2014.

4) Dynamically Loaded Code: Android apps can load code at runtime to dynamically extend their functionality. However, this technique comes with severe security implications. While dynamic code loading is popular for legitimate reasons, such as loading external add-on code, shared library code from frameworks, or dynamically updating code during beta and/or A/B testing, it is especially interesting for malware. Since apps are typically inspected only once, either by an app market or by an AV scanner at installation time, malicious apps can download and load their malicious payload later at runtime to evade detection. Furthermore, the unsafe use of code loading techniques can also make legitimate apps vulnerable to code injection techniques, as shown by Poeplau et al. [57].

DEX Classes. One possibility to dynamically extend an app's functionality is to load modules at the Dalvik VM level through the DEX class loader. We observed this behavior for 2.97% of goodware and for 4.46% of malware apps, with a significant increase over the past two years. Static analysis successfully identifies the invocation of the `DexClassLoader` in 98.88% of goodware and 97.20% of malware respectively. On average, goodware loads 1.28 and malware loads 1.59 DEX classes. The maximum of different classes loaded is 37 for the Metasploit payload, 25 classes for samples from the *Android.SmsSpy* family and 9 classes for goodware in general.

Native Libraries. Overall, both goodware and malware apps load native libraries in equal proportions: we observed 8.60% and 8.50% of all benign and malicious apps loading native code during dynamic analysis, with a clear upward trend especially amongst goodware. The sources for the loaded native code and their impact differ: at a finer granularity, we distinguish between the number of system native libraries loaded and custom, non-system, libraries loaded. Custom libraries are far more dangerous than those provided by the Android system itself. The reason for system library usage is simple: games and graphically demanding apps make use of hardware-accelerated technologies found in modern graphics cards, like OpenGL or video decoding, for both performance reasons and increased battery life. Custom libraries, however, tend to be used by malware for a number of nefarious purposes, including the elevation of privileges through root exploits.

Goodware apps load 52.47% and 52.26% code from the system and the data directory respectively, contrary, only 19.46% of malware samples load native system libraries, while 84.19% load their own bundled native code or fetch it from remote servers. While for malware the percentage of system libraries loaded decreased from 2010 to 2014 by 13 percentage points and the usage of custom libraries increased by 20 percentage points, this trend is more severe for goodware: in 2010, 74.11% of goodware apps loaded native code from the system and only 29.57% loaded custom code; in 2014, 30.95% of apps loaded code from the system and 73.37% loaded it from the data directory.

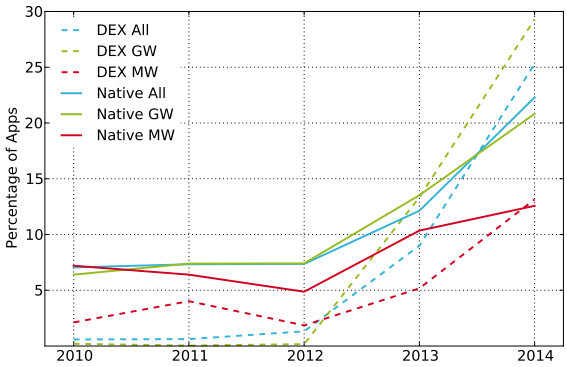


Fig. 7: Increasing use of DEX and native code loading overall, and in goodware (GW) and malware (MW).

Static analysis was far less successful in identifying native code loading compared to DEX class loading and only identified the `loadLibary()` call in 54.40% of goodware and 83.25% of malware. These numbers correspond to the number of apps shipping with unencrypted ELF libraries that can be identified based on their file signature: 54.29% in the case of goodware and 85.23% in the case of malware.

Dynamic code loading significantly increased during our observation period, especially for goodware over the past two years, as shown in Figure 7: in 2014, 29.29% of benign apps loaded Dalvik and 20.82% native code, while 13.15% of malicious apps loaded Dalvik and 12.57% native code. Furthermore, loading native libraries and DEX classes is not an either-or decision: 1.25% of all malware (5.43% in just 2014) and 0.45% of all goodware (4.92% in 2014) combine those techniques to load both native and Dalvik code.

5) Cryptographic API Usage: Another interesting case study is the use of cryptographic protocols. During dynamic analysis, we observed the usage of the Java crypto API in 5.63% of malicious apps, in contrast to only 1.10% of goodware apps. Interestingly, for those apps we could statically determine the use of cryptography in 99.21% of cases for goodware, but for only 43.24% of malware – either due to this part of the code being obfuscated and loaded dynamically at runtime. Overall, static analysis revealed the use of `javax/crypto/*` in 44.83% of goodware apps, increasing from 11.12% in 2010 to 79.18% in 2014. For malware, we did not see such a development with the Java crypto API being used by only 29.84% overall, likely due to malware shipping their own implementations in order to evade detection.

The most popular algorithms observed during dynamic analysis of goodware are AES (66.75%), PBEwithMD5andDES (15.03%), DES (11.98%), and RSA (5.08%). Malware, on the other hand, mainly used AES (74.82%), Blowfish (14.31%), DES (8.78%), and RSA (1.20%). We also observed a trend toward stronger cryptographic algorithms in malware: while DES was the predominantly used algorithm amongst malware in 2010 (98.44%), its usage declined significantly to 1.53% in 2013. Instead, in 2012 malware authors started adopting the stronger Blowfish algorithm, which is now being used by 31.58% of all malware apps from 2013, while we have not seen a single goodware app using Blowfish.

6) Network Activity: We observed network traffic in goodware and malware apps alike – 71.11% of goodware and 80.36% of malware, with almost 99% of those samples requesting but only 70.97% of benign and 61.43% of malicious samples using the `INTERNET` permission according to static analysis. This numbers decreased for malware in 2014 to only 94.40% requesting and 58.84% using the permission, indicating malware circumventing the permission system by performing network activity through other apps installed on the device, such as the browser, for example.

Almost all apps that use the Internet query domain names: 99.91% of malware and 97.34% of goodware perform DNS queries, but while one third (32.33%) of the queries by malicious samples fail and result in an invalid (NX) domain, only 10% of queries from goodware samples do.

UDP traffic is almost limited to DNS, with only a few samples using NTP. However, 55.33% of malware and 23.62% of goodware also establish TCP connections. This number increased for malware from 27.69% in 2010 to 58.65% in 2013, and decreased to 45.84% in 2014; for goodware it monotonically increased from 12.81% in 2010 to 43.50% in 2014. The most commonly observed network activity for malware occurred on port 443 (HTTPS, 44.09% of samples), port 80 (HTTP, 15.52%), and port 5224 (XMPP/Google Talk), 8245 (DynDNS), and 9001 (Tor) with less than 0.2% of samples each. For goodware we observed port 443 (HTTPS, 15.58%), port 80 (HTTP, 7.31%), and port 1130 (CASP, 0.46%).

Other protocols were hardly ever used: we only observed 77 apps in our whole dataset establishing FTP connections and 14 samples using IRC. We saw, however, 352 samples from 2013 and 2014 establishing SMTP connections and sending emails. The majority of those samples are classified by AV scanners as malware and they leak sensitive information such as the contents of the address book and incoming SMS via email to addresses from Chinese freemail providers, such as NetEase (`163.com`, `126.com`) and Tencent (`qq.com`).

7) Cross-Platform Malware: In 2013 Android malware started to download a malicious Windows payload (*Backdoor.MSIL.Ssucl*) and saving it together with an `autorun.inf` file in the root directory of the phone's SD card, hoping it would be automatically executed on Windows computers once the phone was connected to the PC via USB [58]. We only saw this behavior in 11 apps overall, nine of which were different versions of the goodware samples iSyncr and RealPlayer that placed their Windows installer together with `autorun.inf` on the SD card. Only 19 goodware samples embedded executables. The only malicious samples we saw exhibiting this behavior were from the *Android.UsbCleaver* [59] family. Overall, we detected 447 malware samples with a total of 27 different embedded executables that are flagged by at least one AV scanner.

There have been reports of Windows malware attempting to infect Android devices, and even installing the Android Debug Bridge (ADB) to do so [60]. We have only seen 119 Windows samples in ANUBIS attempting to drop APK files, 16 of which also tried to access the ADB (currently not installed in our Windows environment). The majority of those files, however, failed to download completely or seem to belong to rooting utilities. VirusTotal has labels for 56 out of the 99 dropped APKs, with 33 not being detected by any AV scanners, 20 detected, as root exploits and the remaining three belonging to *Android.AndroRat* and *Android.FakeAngry*.

V. RELATED WORK

For Windows malware, Bayer et al. [20] performed a similar analysis to ours on a dataset of 900,000 Windows samples ANUBIS received within its first two years of operation. Here, however, we focus on related work on the Android malware landscape.

Android security and the detection and characterization of Android malware in particular has been an extremely active field of research in the past years. Felt et al. [61] analyzed a total of 46 iOS, Symbian and Android malware samples collected between 2009 and 2011 to provide one of the first surveys on mobile malware and their author's incentives. The Android Malware

Genome Project [36] was a further attempt to systematize Android malware behavior and provided a publicly available dataset used in many following evaluations. The dataset, however, is now showing its age: the samples being collected between 2010 and 2011 behave significantly different than apps from 2012 to 2014, as we have shown in our evaluation (Section IV). Another available malware dataset is the one used by Drebin [39] for classifying Android malware. This dataset also includes the Genome Project and the most recent samples were collected in 2012. Further studies on malware behavior mainly focused on the practice of repackaging and the pervasiveness of repackaged apps in alternative app stores [62,63].

TaintDroid [28] was the first work to propose taint tracking for monitoring data flow dependencies and data leakage in Android apps and is now at the core of many sandboxes, such as ours, to track data leaks. DroidScope [9] is a dynamic analysis system solely based on VMI. While this approach has advantages, such as whole-system taint analysis, the delicate reconstruction of Java objects and the like from raw memory regions requires substantial adaption effort with each Android OS update.

SmartDroid [64] and AppsPlayground [65] aim at improving the stimulation of apps during dynamic analysis. They try to drive the app along paths that are likely to reveal interesting behavior through targeted stimulation of UI elements. Their approaches can be seen as intelligent enhancements of the Application Exerciser Monkey and our custom stimulation of activity screens. They are largely orthogonal to our work, which focuses on stimulating broadcast receivers, services and common events, instead of UI elements.

Concerning systems for the large-scale dynamic analysis of Android applications, Bläsing et al. [8] proposed AASandbox, the first dynamic analysis platform for Android based on system call monitoring. ANANAS [11], on the other hand, is a dynamic analysis framework focusing on extensibility through modules. DroidRanger [66] pre-filters applications based on a manually created permission-fingerprint before subjecting them to dynamic analysis. In contrast to this approach, we analyzed every app, yielding full behavioral profiles to base our evaluation on. Furthermore, DroidRanger performs monitoring through a kernel module instead of VMI and focuses only on system calls used by existing root exploits. Finally, DroidRanger does not employ stimulation techniques. None of the above tools are publicly available.

Dynamic analysis systems that are publicly available are CopperDroid [10,67], Tracedroid [68,69], SandDroid [70], and Mobile-Sandbox [7,71]. CopperDroid performs out-of-the-box system call monitoring through VMI and reconstructs Dalvik behavior by monitoring Binder communication. Tracedroid generates complete method traces by extending the Dalvik VM profiler and was subsequently integrated into ANDRUBIS, but it is also available as a standalone service. SandDroid performs monitoring of the Dalvik VM, but does not allow any network connections to the outside and therefore misses behavior in apps checking for Internet connectivity [16]. Mobile-Sandbox monitors native code through `ltrace` in addition to instrumenting the Dalvik VM. However, both SandDroid and Mobile-Sandbox seem to be unable to cope with their submission load: SandDroid has only analyzed around 25,000 samples to date and samples we submitted have been stuck in the input queue for almost nine months, while Mobile-Sandbox reports a backlog of over 300,000 samples with no samples seemingly being analyzed. We emphasize that, to the best of our knowledge, ANDRUBIS is the only dynamic analysis sandbox operating on a large-scale, providing a thorough analysis on both Dalvik and system level, and typically returning a report in ten minutes or less.

VI. LIMITATIONS

One limitation of any dynamic analysis approach is evasion. As long as a sandbox is not capable of perfectly emulating a system, a possibility to detect it exists. Petsas et al. [72] and Vidas et al. [16] recently explored the possibility to fingerprint Android sandboxes, and found that all, including ours, are susceptible to evasion. Sandbox detection techniques range from static characteristics of the specific Android OS installation to information from sensors, to the detection of the underlying virtualization technology. One proof of concept [73] is able to detect any QEMU-based environment based on binary translation: QEMU (and other emulators) usually take a basic block, translate it, and execute the whole resulting basic block on the host machine. Unfortunately, this property allows for an easy detection of emulated code, since the basic block cannot be interrupted by the guest operating system's scheduler. As a countermeasure, we enabled QEMU single-step mode, which makes ANDRUBIS undetectable by this evasion technique. However, this mode introduces an analysis overhead of 29% compared to 7% with Dalvik monitoring and 18% with QEMU VMI [23]. Generally, dealing with analysis evasion is a never-ending arms race between security researchers and malware authors.

A further limitation of dynamic analysis is code coverage. While we try to increase behavior seen during analysis through various stimulation techniques, a more intelligent user interface stimulation than the random input stream by the Android Exerciser Monkey could provide more complex and user-like input and, in turn, trigger much more behavior from the apps under analysis.

Currently public submissions to ANDRUBIS are limited to a file size of 8MB. This limit, however, is simply a limitation of our web interface and not a fundamental limitation of our analysis. We are currently evaluating to increase this limit, while keeping storage requirements at an acceptable level without having to discard apps after analysis.

Finally, a limitation of any analysis system allowing submissions from anonymous sources is the lack of metadata and ground truth. We have no indication when and where samples were found or how widespread they are in the wild. We tried to mitigate this in part by collecting metadata from markets with AndRadar [6]. Lacking ground truth, we have to rely on AV signatures to classify our dataset in goodware and malware, but we are experimenting with machine-learning approaches to automatically classify samples with higher accuracy than related work.

VII. CONCLUSION

In this paper we presented ANDRUBIS, a fully automated large-scale analysis system for Android apps that combines static analysis with dynamic analysis on both Dalvik VM and system level. ANDRUBIS accepts public submissions through a web interface and a mobile app and is currently capable of analyzing around 3,500 new samples per day. With ANDRUBIS, we provide malware analysts with the means to thoroughly analyze Android apps. Furthermore, we provide researchers with a solid platform to build post-processing methods upon based on an app's static features and dynamic behavior. For example, leveraging machine-learning approaches one can use our analysis results to tackle the problem of judging whether a previously unseen app is malware significantly more accurate than prior work.

ANDRUBIS has analyzed over 1,000,000 Android apps to date. On an evaluation of this dataset spanning samples from four years, we showed changes in the malware threat landscape and trends amongst goodware developers. Dynamic code loading, previously used as an indicator for malicious behavior, is especially gaining popularity amongst goodware, and, in turn,

loses significant information value when distinguishing between benign and malicious apps. Due to this development, static analysis tools alone are increasingly unable to completely capture an app's behavior, making dynamic analysis indispensable for a comprehensive analysis for a large number of apps.

In future work, we plan to explore the network behavior of Android malware further to identify C&C communication patterns and shared infrastructures with Windows malware. Furthermore, we are exploring the option of releasing a comprehensive malware dataset, once we sorted out legal and confidentiality issues.

ACKNOWLEDGMENTS

We would like to thank VirusTotal for the service they provided for our evaluation. The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n. 257007 (SysSec) and from the FFG – Austrian Research Promotion under grant COMET K1.

This work also has been carried out within the scope of u'smile, the Josef Ressel Center for User-Friendly Secure Mobile Environments. We gratefully acknowledge funding and support by the Christian Doppler Gesellschaft, A1 Telekom Austria AG, Drei-Banken-EDV GmbH, LG Nexera Business Solutions AG, and NXP Semiconductors Austria GmbH.

REFERENCES

[1] IDC, "Android and iOS Continue to Dominate the Worldwide Smartphone Market with Android Shipments Just Shy of 800 Million in 2013," http://www.idc.com/getdoc.jsp?containerId=prUS24676414, February 2014.

[2] F-Secure, "Threat Report H2 2013," http://www.f-secure.com/static/doc/labs_global/Research/Threat_Report_H2_2013.pdf, March 2014.

[3] McAfee Labs, "McAfee Threats Report: Second Quarter 2013," http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2013.pdf, August 2013.

[4] V. Svajcer, "Sophos Mobile Security Threat Report," http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-mobile-security-threat-report.ashx, 2014.

[5] H. Lockheimer, "Android and Security," http://googlemobile.blogspot.com/2012/02/android-and-security.html, February 2012.

[6] M. Lindorfer, S. Volanis, A. Sisto, M. Neugschwandtner, E. Athanasopoulos, F. Maggi, C. Platzer, S. Zanero, and S. Ioannidis, "AndRadar: fast discovery of android applications in alternative markets," in *Proceedings of the 11th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2014.

[7] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: Having a Deeper Look into Android Applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC)*, 2013.

[8] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. Camtepe, and S. Albayrak, "An Android Application Sandbox System for Suspicious Software Detection," in *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE)*, 2010.

[9] L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," in *Proceedings of the 21st USENIX Security Symposium*, 2012.

[10] A. Reina, A. Fattori, and L. Cavallaro, "A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors," in *Proceedings of the 6th European Workshop on System Security (EuroSec)*, 2013.

[11] T. Eder, M. Rodler, D. Vymazal, and M. Zeilinger, "ANANAS - A Framework For Analyzing Android Applications," in *Proceedings on the 1st International Workshop on Emerging Cyberthreats and Countermeasures (ECTCM)*, 2013.

[12] "ForeSafe Mobile Security," http://www.foresafe.com.

[13] "VisualThreat," http://www.visualthreat.com.

[14] "Joe Sandbox Mobile," http://www.joesecurity.org/joe-sandbox-mobile.

[15] S. Neuner, V. van der Veen, M. Lindorfer, M. Huber, G. Merzdovnik, M. Mulazzani, and E. Weippl, "Enter Sandbox: Android Sandbox Comparison," in *Proceedings of the 3rd IEEE Mobile Security Technologies Workshop (MoST)*, 2014.

[16] T. Vidas and N. Christin, "Evading Android Runtime Analysis via Sandbox Detection," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2014.

[17] F. Maggi, A. Valdi, and S. Zanero, "AndroTotal: A Flexible, Scalable Toolbox and Service for Testing Mobile Malware Detectors," in *Proceedings of the 3rd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2013.

[18] "Anubis," http://anubis.iseclab.org.

[19] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: A Tool for Analyzing Malware," in *Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR) Annual Conference*, 2006.

[20] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A View on Current Malware Behaviors," in *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats (LEET)*, 2009.

[21] "VirusTotal," http://www.virustotal.com.

[22] U. Bayer, P. Milani Comparetti, C. Hlauscheck, C. Kruegel, and E. Kirda, "Scalable, Behavior-Based Malware Clustering," in *Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS)*, 2009.

[23] L. Weichselbaum, M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. van der Veen, and C. Platzer, "Andrubis: Android Malware Under The Magnifying Glass," Vienna University of Technology, Tech. Rep. TR-ISECLAB-0414-001, 2014.

[24] C. Lever, M. Antonakakis, B. Reaves, P. Traynor, and W. Lee, "The Core of the Matter: Analyzing Malicious Traffic in Cellular Carriers," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS)*, 2013.

[25] A. Ludwig, E. Davis, and J. Larimer, "Android - Practical Security From the Ground Up," in *Virus Bulletin Conference*, 2013.

[26] H. T. T. Truong, E. Lagerspetz, P. Nurmi, A. J. Oliner, S. Tarkoma, N. Asokan, and S. Bhattacharya, "The Company You Keep: Mobile Malware Infection Rates and Inexpensive Risk Indicators," in *Proceedings of the 23rd International Conference on World Wide Web (WWW)*, 2014.

[27] "Andrubis Submission App," https://play.google.com/store/apps/details?id=org.iseclab.andrubis.

[28] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010.

[29] L. Cavallaro, P. Saxena, and R. Sekar, "Anti-Taint-Analysis: Practical Evasion Techniques Against Information Flow Based Malware Defense," Secure Systems Lab at Stony Brook University, Tech. Rep., 2007.

[30] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android Permission Specification," in *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS)*, 2012.

[31] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, 2011.

[32] Google, "Introducing ART," https://source.android.com/devices/tech/dalvik/art.html, 2014.

[33] C. Toombs, "Updates To AOSP Confirm Dalvik Runtime Will Be Removed From Android, ART Officially Takes Its Place," http://www.androidpolice.com/2014/06/19/updates-aosp-confirm-dalvik-runtime-will-removed-android-art-officially-takes-place, June 2014.

[34] J. Goebel, T. Holz, and C. Willems, "Measurement and Analysis of Autonomous Spreading Malware in a University Environment," in *Proceedings of the 4th International Conference on Detection of Intrusions & Malware, and Vulnerability Assessment (DIMVA)*, 2007.

[35] "Contagio," http://contagiominidump.blogspot.com.

[36] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[37] M. Bierma, E. Gustafson, J. Erickson, D. Fritz, and Y. R. Choe, "Andlantis: Large-scale Android Dynamic Analysis," in *Proceedings of the 3rd IEEE Mobile Security Technologies Workshop (MoST)*, 2014.

[38] "AppBrain Stats: Number of Android applications," http://www.appbrain.com/stats/number-of-android-apps.

[39] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS)*, 2014.

[40] "Android Version history by API level," http://en.wikipedia.org/wiki/Android_version_history#Version_history_by_API_level.

[41] B. Irinco, "First Android Trojan in the Wild," http://blog.trendmicro.com/trendlabs-security-intelligence/first-android-trojan-in-the-wild, August 2010.

[42] M. Neugschwandtner, M. Lindorfer, and C. Platzer, "A View To A Kill: WebView Exploitation," in *Proceedings of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2013.

[43] Y. Zhang, H. Xue, and T. Wei, "Occupy Your Icons Silently on Android," http://www.fireeye.com/blog/technical/2014/04/occupy_your_icons_silently_on_android.html, April 2014.

[44] D. Barrera, J. Clark, D. McCarney, and P. C. van Oorschot, "Understanding and Improving App Installation Security Mechanisms Through Empirical Analysis of Android," in *Proceedings of the 2nd ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2012.

[45] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun, "Analysis of the Communication Between Colluding Applications on Modern Smartphones," in *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC)*, 2012.

[46] M. Zheng, M. Sun, and J. C. Lui, "DroidRay: A Security Evaluation System for Customized Android Firmwares," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2014.

[47] "AppBrain Stats: Android Ad networks," http://www.appbrain.com/stats/libraries/ad.

[48] Lookout Mobile Security, "State of Mobile Security 2012," https://www.lookout.com/_downloads/lookout-state-of-mobile-security-2012.pdf, 2012.

[49] D. Ruddock, "Google Pushes Major Update To Play Developer Content Policy, Kills Notification Bar Ads For Real This Time, And A Lot More," http://www.androidpolice.com/2013/08/23/teardown-google-pushes-major-update-to-play-developer-content-policy-kills-notification-bar-ads-for-real-this-time-and-a-lot-more/, September 2013.

[50] J. Forristal, "Android: One Root to Own Them All," in *Black Hat USA*, 2013.

[51] P. Ducklin, "Anatomy of another Android hole - Chinese researchers claim new code verification bypass," http://nakedsecurity.sophos.com/2013/07/17/anatomy-of-another-android-hole-chinese-researchers-claim-new-code-verification-bypass, July 2013.

[52] ——, "Anatomy of a file format problem - yet another code verification bypass in Android," http://nakedsecurity.sophos.com/2013/11/06/anatomy-of-a-file-format-problem-yet-another-code-verification-bypass-in-android, November 2013.

[53] "Python Bug Tracker: Issue 14315," http://bugs.python.org/issue14315.

[54] "Fuzion24/Zip File Arbitrage: Exploit for Android Zip bugs: 8219321, 9695860, and 9950697," https://github.com/Fuzion24/AndroidZipArbitrage.

[55] C. Toombs, "External Blues: Google Has Brought Big Changes To SD Cards In KitKat, And Even Samsung Is Implementing Them," http://www.androidpolice.com/2014/02/17/external-blues-google-has-brought-big-changes-to-sd-cards-in-kitkat-and-even-samsung-may-be-implementing-them, February 2014.

[56] R. Lipovsky, "ESET Analyzes First Android File-Encrypting, TOR-enabled Ransomware," http://www.welivesecurity.com/2014/06/04/simplocker, June 2014.

[57] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, "Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS)*, 2014.

[58] V. Chebyshev, "Mobile attacks!" http://www.securelist.com/en/blog/805/Mobile_attacks, February 2013.

[59] F-Secure, "Android Hack-Tool Steals PC Info," http://www.f-secure.com/weblog/archives/00002573.html, July 2013.

[60] F. Liu, "Windows Malware Attempts to Infect Android Devices," http://www.symantec.com/connect/blogs/windows-malware-attempts-infect-android-devices, January 2014.

[61] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A Survey of Mobile Malware in the Wild," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2011.

[62] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces," in *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012.

[63] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, Scalable Detection of "Piggybacked" Mobile Applications," in *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2013.

[64] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, and W. Zou, "SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications," in *Proceedings of the 2nd ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2012.

[65] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: Automatic Security Analysis of Smartphone Applications," in *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2013.

[66] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*, 2012.

[67] "CopperDroid," http://copperdroid.isg.rhul.ac.uk.

[68] "Tracedroid," http://tracedroid.few.vu.nl.

[69] V. van der Veen, "Dynamic Analysis of Android Malware," *Internet & Web Technology Master thesis, VU University Amsterdam*, 2013.

[70] "SandDroid," http://sanddroid.xjtu.edu.cn.

[71] "Mobile Sandbox," http://mobilesandbox.org.

[72] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware," in *Proceedings of the Seventh European Workshop on System Security (EuroSec)*, 2014.

[73] F. Matenaar and P. Schulz, "Detecting Android Sandboxes," http://www.dexlabs.org/blog/btdetect, August 2012.

APPENDIX

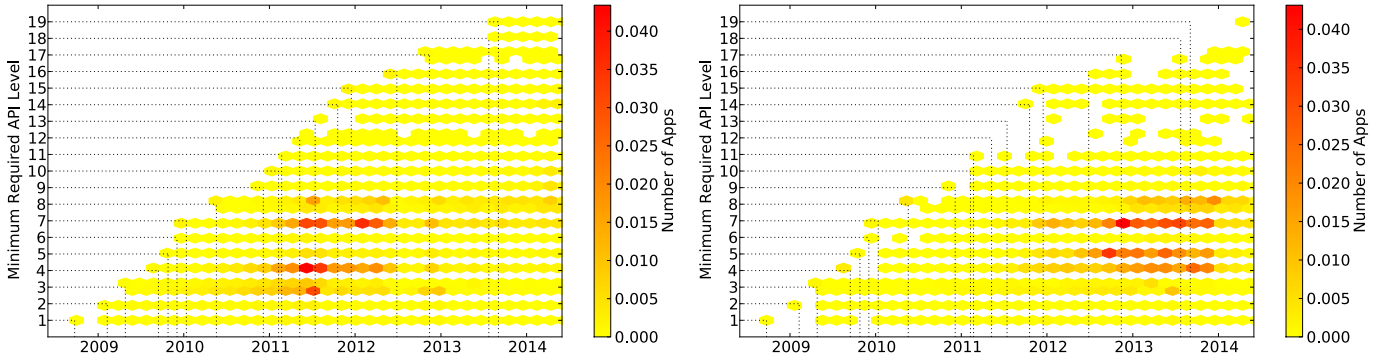


Fig. 8: Heatmap of API level adoption after Android SDK releases for goodware (left) and malware (right): goodware authors adopt new API levels much faster than malware authors, who try to maximize the potential user base for their apps.

TABLE II: Sources for apps in our dataset: sample exchange feeds have a high proportion of malware, while the Google Play Store and apps from torrents and direct downloads have low infection rates. Interestingly, not all samples from malware corpora are detected by AV scanners.

Category	Sample Exchange	Google Play	Alternative Markets	VirusTotal	Malware Corpora	Torrents	Direct Downloads	Unknown
All	683,842	125,602	60,951	37,499	5,997	17,916	1,704	159,040
Goodware	5.2%	88.73%	18.15%	0.20%	0.04%	88.50%	96.36%	78.65%
Malware	55.3%	1.60%	27.51%	98.65%	97.87%	1.60%	1.59%	7.56%

TABLE III: Most frequently requested permissions in goodware and malware by the percentage of apps in each set.

Goodware		Malware	
83.97%	INTERNET	95.37%	INTERNET
61.54%	ACCESS_NETWORK_STATE	91.42%	READ_PHONE_STATE
43.65%	WRITE_EXTERNAL_STORAGE	82.79%	WRITE_EXTERNAL_STORAGE
38.09%	READ_PHONE_STATE	71.99%	ACCESS_NETWORK_STATE
23.59%	ACCESS_COARSE_LOCATION	69.91%	SEND_SMS
22.51%	VIBRATE	60.67%	RECEIVE_SMS
21.56%	ACCESS_FINE_LOCATION	55.66%	INSTALL_SHORTCUT
19.32%	WAKE_LOCK	51.40%	WAKE_LOCK
18.05%	ACCESS_WIFI_STATE	48.73%	READ_SMS
12.11%	READ_CONTACTS	45.62%	RECEIVE_BOOT_COMPLETED
11.83%	RECEIVE_BOOT_COMPLETED	40.15%	ACCESS_WIFI_STATE
8.30%	CALL_PHONE	32.92%	WRITE_SETTINGS
8.15%	CAMERA	30.05%	READ_CONTACTS
7.66%	GET_TASKS	25.74%	CALL_PHONE
7.45%	SEND_SMS	24.70%	ACCESS_COARSE_LOCATION
6.72%	GET_ACCOUNTS	24.30%	ACCESS_FINE_LOCATION
6.31%	WRITE_SETTINGS	23.83%	VIBRATE
6.11%	WRITE_CONTACTS	23.04%	GET_TASKS
5.02%	SET_WALLPAPER	20.15%	WRITE_SMS
4.96%	CHANGE_WIFI_STATE	20.12%	CHANGE_WIFI_STATE
4.57%	INSTALL_SHORTCUT	19.21%	SYSTEM_ALERT_WINDOW
4.47%	RECEIVE_SMS	19.11%	CHANGE_NETWORK_STATE
4.03%	RECORD_AUDIO	17.81%	GET_ACCOUNTS
4.00%	READ_CALENDAR	13.93%	INSTALL_PACKAGES
3.79%	READ_LOGS	13.23%	UNINSTALL_SHORTCUT

TABLE IV: Most frequently registered broadcast receivers in goodware and malware by the percentage of apps in each set.

Goodware		Malware	
11.29%	BOOT_COMPLETED	56.32%	BOOT_COMPLETED
8.93%	APPWIDGET_UPDATE	41.73%	SMS_RECEIVED
8.74%	INSTALL_REFERRER	14.56%	CONNECTIVITY_CHANGE
6.74%	SCREEN_OFF	13.49%	DATA_SMS_RECEIVED
6.69%	USER_PRESENT	11.95%	AIRPLANE_MODE
4.17%	CONNECTIVITY_CHANGE	10.18%	PACKAGE_ADDED
2.40%	PACKAGE_ADDED	4.24%	NEW_OUTGOING_CALL
2.38%	IN_APP_NOTIFY	2.66%	USER_PRESENT
2.25%	SMS_RECEIVED	2.14%	BATTERY_CHANGED
1.43%	PHONE_STATE	1.72%	DEVICE_ADMIN_ENABLED
0.91%	MEDIA_BUTTON	1.60%	INSTALL_REFERRER
0.78%	PACKAGE_REMOVED	1.50%	APPWIDGET_UPDATE
0.70%	SERVICE_STATE	1.43%	PHONE_STATE
0.65%	SCREEN_ON	1.40%	BATTERY_CHANGED_ACTION
0.64%	MEDIA_MOUNTED	1.03%	PACKAGE_REMOVED
0.61%	NEW_OUTGOING_CALL	0.90%	UNINSTALL_SHORTCUT
0.60%	BATTERY_CHANGED	0.90%	INSTALL_SHORTCUT
0.47%	PACKAGE_REPLACED	0.90%	SIG_STR
0.40%	DEVICE_ADMIN_ENABLED	0.88%	ACTION_POWER_CONNECTED
0.36%	DEVICE_STORAGE_LOW	0.70%	SCREEN_OFF
0.32%	STATE_CHANGE	0.61%	PICK_WIFI_WORK
0.27%	TIME_SET	0.51%	TIME_SET
0.25%	WAP_PUSH_RECEIVED	0.41%	WAP_PUSH_RECEIVED
0.25%	ACTION_POWER_CONNECTED	0.39%	SCREEN_ON
0.24%	MEDIA_UNMOUNTED	0.36%	BATTERY_LOW

TABLE V: Most popular advertisement libraries in goodware and malware by the percentage of apps in each set.

Goodware		Malware	
36.76%	AdMob (Google)	5.74%	AdMob (Google)
5.61%	Flurry	3.90%	WAPS
4.00%	Millenial Media	2.94%	Kuogo
2.92%	MobClix	2.92%	domob
2.72%	AdWhirl	2.67%	Adwo
1.94%	InMobi	2.02%	AirPush
1.77%	MobFox	1.97%	YouMi
0.91%	MoPub	1.43%	Vpon
0.78%	Adlantis	1.27%	Wooboo
0.74%	Admarvel	1.15%	MobWIN
0.67%	Smaato	0.91%	Millenial Media
0.63%	YouMi	0.84%	Flurry

TABLE VI: Information most commonly leaked to the network by goodware and malware by the percentage of apps in each set.

Goodware		Malware	
12.86%	IMEI	39.68%	IMEI
1.70%	IMSI	25.88%	IMSI
1.51%	PHONE_NUMBER	13.89%	PHONE_NUMBER
1.12%	LOCATION	4.34%	ICCID
1.12%	LOCATION_GPS	1.40%	CONTACTS
0.60%	ICCID	0.40%	PACKAGE
0.08%	PACKAGE	0.11%	SMS
0.06%	CONTACTS	0.11%	CALL_LOG
0.05%	SMS	0.10%	LOCATION
0.02%	CALL_LOG	0.10%	LOCATION_GPS
0.01%	BROWSER	0.07%	BROWSER
0.01%	CALENDAR	0.00%	TAINT_CAMERA

Security and Privacy Measurements in Social Networks: Experiences and Lessons Learned

Iasonas Polakis*, Federico Maggi†, Stefano Zanero†, Angelos D. Keromytis*

*Network Security Lab
Columbia University, USA
{polakis,angelos}@cs.columbia.edu

†DEIB
Politecnico di Milano, Italy
{federico.maggi,stefano.zanero}@polimi.it

***Abstract*—We describe our experience gained while exploring practical security and privacy problems in a real-world, large-scale social network (i.e., Facebook), and summarize our conclusions in a series of "lessons learned". We first conclude that it is better to adequately describe the potential ethical concerns from the very beginning and plan ahead the institutional review board (IRB) request. Even though sometimes optional, the IRB approval is a valuable point from the reviewer's perspective. Another aspect that needs planning is getting in touch with the online social network security team, which takes a substantial amount of time. With their support, "bending the rules" (e.g., using scrapers) when the experimental goals require so, is easier. Clearly, in cases where critical technical vulnerabilities are found during the research, the general recommendations for responsible disclosure should be followed. Gaining the audience's engagement and trust was essential to the success of our user study. Participants felt more comfortable when subscribing to our experiments, and also responsibly reported bugs and glitches. We did not observe the same behavior in crowd-sourcing workers, who were instead more interested in obtaining their rewards. On a related point, our experience suggests that crowd sourcing should not be used alone: Setting up tasks is more time consuming than it seems, and researchers must insert some sentinel checks to ensure that workers are not submitting random answers.**

From a logistics point of view, we learned that having at least a high-level plan of the experiments pays back, especially when the IRB requires a detailed description of the work and the data to be collected. However, over planning can be dangerous because the measurement goals can change dynamically. From a technical point of view, partially connected to the logistics remarks, having a complex and large data-gathering and analysis framework may be counterproductive in terms of set-up and management overhead. From our experience we suggest to choose simple technologies that scale up if needed but, more importantly, can scale down. For example, launching a quick query should be straightforward, and the frameworks should not impose too much overhead for formulating it. We conclude with a series of practical recommendations on how to successfully collect data from online social networks (e.g., using techniques for network multi presence, mimicking user behavior, and other crawling "tricks"') and avoid abusing the online service, while gathering the data required by the experiments.

I. Introduction

Massive user participation has rendered online social networks (OSNs) a valuable target for attackers, and a lucrative platform for deploying various types of attacks, ranging from spam [1] to personalized phishing campaigns [2]. Ample research efforts have been dedicated to explore the potential ways in which OSNs can be exploited and attacked, and subsequently develop the appropriate defense mechanisms that will hinder actual incidents.

Research Challenges. Researching the security of online social networks presents a series of interesting challenges. On one hand, the large-scale nature of such services requires efficient and accurate experimentation methodologies as well as a sturdy infrastructure. Consider that miscreants are known to capitalize on popular events that are expressed through viral behavior on OSNs such as Facebook and Twitter. For example, during the night of the 2012 U.S. presidential election, 31 million Tweets were posted online at a peak rate of 372,452 Tweets per minute [3]. Had researchers wanted to study and analyze such content in search of SPAM or malware campaigns, it would have been a daunting task. Keeping up with the rate of user-generated content, also places significant burden on the network connection both in terms of bandwidth as well as latency. Moreover, maintaining such content for subsequent analysis mandates a large amount of storage space and processing power. On the other hand, the unique nature of security research presents both ethical and legal issues. From the standpoint of the OSN service, a researcher might seem like an attacker and from the standpoint of a researcher, probing the service to identify weaknesses might mean producing tools for the actual attackers. Despite this growing interest, we are not aware of any systematization nor retrospective work on OSN research with a focus on system security.

Our Experience. In this paper we present our experiences from our recent research on Facebook. In our use case, the goal was to build a system to identify Facebook's Social Authentication mechanism characteristics, analyze its behavior and point out the security weaknesses. Our work has been published in the proceedings of 2012's ACSAC [4]. We hereby present our experiences, and the mishaps we encountered and solved while interacting with a large-scale OSN service such as Facebook, with the goal of conducting a user-centered analysis.

We walk through the logistical and technical challenges that we had to take care of, and provide the reader with a series of practical recommendations in order to carry out a measurement experiment in the smoothest way possible. Empirical works on online social networks are probably the most representative example of user-centered measurements and, as such, require time and certain aspects to be taken into account. To this end, we provide a "meta workflow" that other researchers can adapt to their needs, in order to avoid mistakes that we committed when designing and developing our experiments. However, we also learned that a strict plan could sometime become counterproductive: We provide practical examples that explain when and how we needed to revise our plan, and adapt our measurement infrastructure to changing goals. Moreover, we discuss the delicate aspects related to terms of service in OSNs,

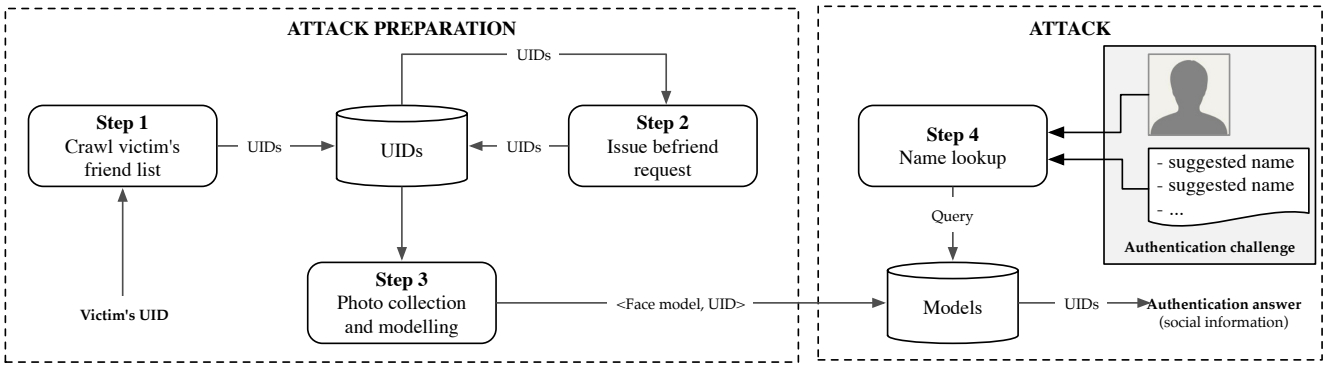


Fig. 1. Use case: An automated system that collects data from Facebook and exploit it to attempt to break its face-based social authentication system. The system operates in four steps. In **Step 1** we retrieve the victim's friend list using his or her UID. Then, in **Step 2** (optional), we send befriend requests, so that we have more photos to extract faces from and build face classifiers in **Step 3**. In **Step 4**, given a photo, we query the models to retrieve the corresponding UID and thus match a name to face. Step 4 takes place in real time while steps 1 through 3 at a previous time.

which often conflict with the goals of the security researcher, who needs to crawl corner regions of the network.

We summarize our conclusions on each aspect in a series of "lessons learned", which provide a starting point for future research in the same areas, or with similar measurement goals.

In summary, in this paper:

- We analyze the typical stages of large-scale experiments in OSNs and present them as a work flow, as we argue that the particularities of OSN security research demand a structured approach, and provide pointers for identifying non-technical challenges and requirements. We are not aware of prior work on the subject.
- We describe the peculiarities of OSN-related experiments, and stress the importance of considering non-technical issues when designing and deploying experiments. Ethical and legal aspects can greatly affect the outcome of an IRB protocol request. We identify the outdated nature of the IRB request procedure, and argue that it should be revised to reflect the requirements of current research.
- We discuss the benefits of cloud services for outsourcing data analysis in large-scale OSN experiments. We present the trade-offs we faced, and how a hybrid approach allowed us to conduct our various experiments efficiently.

II. Case Study

In this section we introduce our use case so as to provide the necessary context for understanding the various issues we came across during our research, and the decisions we had to make. In particular, we briefly describe Facebook's Social Authentication (SA) mechanism, and our experimental methodology and findings while evaluating its security properties. For an in-depth description, we encourage the reader to read our work [4].

A. Photo Based Social Authentication

Conceptually, the social authentication mechanism is an "instantiation" of the two-factor authentication scheme. Two-factor authentication offers additional security to traditional single-factor, password-based, approaches. Usually the second factor is something the user possesses, for example a hardware token, and needs to prove its physical presence in real-time during the authentication process. An attacker would have to steal both the password and the physical token to be able to impersonate the user and log into the Web service in the future.

In SA the idea is, essentially, to leverage a user's social knowledge as the second factor, so as to prevent attackers from compromising online profiles, after having stolen their credentials. Facebook's implementation of the social authentication, or SA in short, is activated only when certain security heuristics flag a login attempt as suspicious, for instance when taking place from a country or computer for the first time. In that case, right after the standard, password-based authentication, the user is prompted with a sequence of 7 pages featuring challenges, where each challenge is comprised of 3 photos of an online friend. For each page, the user must find the true name of the depicted friend, with the sole help of 6 suggested names, chosen from the user's social circle. The user is allowed to fail in 2 challenges, or skip them, but must correctly identify the depicted people in at least 5 in order to be granted access. The idea is that nobody but the actual user will possess the necessary social information to correctly pass the test.

B. Threat Model

The threat model initially covered all scenarios where the user's password had been compromised. However, Kim and collaborators in [5] showed that users with tightly connected social graphs, such as a university network on Facebook, share enough social knowledge to defeat the secrecy assumption made by Facebook. As such, the threat model was reduced to attacks made by complete strangers half-way around the world.

C. Research Goals and Findings

Our research was based on the hypothesis that any stranger (i.e., anyone not in a user's online social circle) can acquire enough knowledge to pass the photo-based challenges. For this, our system crawls the public portion of a user's online social

graph, collects a labeled dataset of photos depicting the user's friends, and trains face-recognition algorithms to automatically recognize faces presented during SA challenges. An overview of our system is summarized in Figure 1. Our work resulted in a system able to successfully break Facebook's SA under our original hypothesis.

We first implemented a crawler that measures the amount of sensitive data left publicly accessible by Facebook users. Our experiments showed that an attacker can obtain access to sensitive information for at least 42% of a user's friends that Facebook uses to generate the SA challenges. Next, we measured the efficiency of face-recognition algorithms against the SA mechanism. Using the aforementioned publicly-accessible information, our attacker trains a classifier and builds accurate facial models of the victim's friends. Our findings showed that by relying solely on publicly-accessible information, an attacker can automatically solve 22% of the SA tests, and gain a significant advantage for an additional 56% of the tests.

III. Approaching Online Social Network Research

Before researchers begin their work on a social networking service, they have to make ethical and legal decisions that will determine the guidelines under which they will have to operate.

A. Ethical Aspects

The first task that researchers must undergo is to consider the ethical issues that may arise from the type of work they wish to do. Designing a study or an experiment should take into consideration its impact on the participants, and the community in general. For instance, collecting user data, even when publicly available, may violate the privacy of those users who might not be aware of the ways their sensitive information is leaking on the Web. At the same time, even if users provide their explicit consent for data collection, it is the researchers' responsibility to provide assurances regarding the confidentiality and privacy of that information, a data retention time frame, as well as information on its secure disposal. Apart from privacy issues, one needs to consider how active experiments within social networking services will be, especially when interaction with actual users is involved (e.g., through dummy accounts). They must take into account that they might impact the users and the online service itself. Overall, researchers need to set a goal of minimizing any impact, and avoiding any permanent side effects of their actions.

Responsible Disclosure. A special category is research carried out for security reasons. For example, in our case we sought to evaluate the effectiveness of Facebook's SA mechanism. Our analysis of identified weaknesses and the documentation of our methodology could be misused by attackers to defeat this security mechanism. However, we believe that attackers might already be studying the weaknesses of SA and that it is better to point them out first, so as to raise the bar for the attackers. As a matter of fact, we have already detailed ways to enhance the security of this mechanism, and are continuing to explore improved countermeasures, and hope that our work has motivated other researches to do the same. It is crucial to improve the security mechanisms of a service with such a massive user base and amount of personal information. Note that the weaknesses that we identified were *conceptual* rather than *technical*. In other words, we did not find an exploitable technical vulnerability, but rather a flawed design. In case critical technical vulnerabilities are found, the researchers should follow responsible disclosure guidelines[1] and the social network's policy[2].

Moreover, as our goal was to determine the security provided by Facebook's SA against someone that has access to a victim's Facebook password, we had several options for testing our hypothesis. One option was to select real Facebook users at random and utilize dictionaries of common passwords to gain access, and actually test our experiments in the most realistic environment possible. Or we could employ lists of known compromised Facebook accounts available in the Internet underground. However, considering the ethical nature of such actions we decided to create, and attack, dummy accounts of our own. To be able to trigger the SA mechanism we needed to populate our accounts with friends and, thus, issued friend requests to random individuals using those accounts. We did not attack, or otherwise negatively affect those individuals. At the end of our experiments, we severed our links to them by un-friending them and deactivated our dummy accounts.

When in Doubt, Simulate. Another example is when we needed to repeat an experiment to test whether we were able to break the photo-based authentication. To obtain the most realistic conditions, we should have chosen to trigger the authentication mechanism and use our training base to try to break it, and repeat the experiment any time we needed to tune our algorithm's parameters. In addition to slowing down the experiment dramatically, it would have caused Facebook to ban our accounts. We had a better choice for performing an arbitrary number of experiments: We used the millions of photos in our data store to synthetically generate the authentication challenges, by randomly selecting an expected answer (i.e., a user's name), five wrong answers, and three random photos each. To the best of our knowledge, Facebook randomly selected candidate photos based on the likelihood that they contain a face. Thanks to the availability of an offline face-detection algorithm we were able to select—and index—only those photos that contained a face, and use them in our experiments. This allowed us to perform an arbitrarily large number of realistic experiments, while preserving the integrity of our dummy accounts, and avoid possible service abuse.

> **Lesson learned:** Ethical aspects must obviously be taken into account from the very beginning. Being familiar with, or at least making educated guesses about, the internals of the OSN web application helps avoiding their abuse. When feasible, try to replicate the settings offline, rather than polling/abusing the online service.

B. Institutional review board (IRB)

The role of an IRB is to supervise research involving human subjects that is carried out by members of that institution. Its mission is to ensure that researchers' actions adhere to the code of ethics, and protect the participants from

[1] https://www.schneier.com/essays/archives/2007/01/schneier_full_disclo.html

[2] In the case of Facebook: https://www.facebook.com/whitehat

physical or psychological harm. Traditionally, the focus of such committees has been biomedical and behavioral research. Nevertheless, an IRB needs to review all research involving human subjects and decide whether to allow it, as well as whether is should be exempt or not from supervision during its duration.

In our case, researchers at Columbia University are asked to provide information, in the form of an IRB protocol request, that seems to be oriented towards traditional forms of research involving humans in a physical location subject to tangible stimuli such as a specific substance, device or signal. Among the information asked to provide are the address of the building and room the research will take place and whether radiation or hazardous materials will be involved.

On the other hand, an IRB protocol request is not oriented towards research taking place online so it is up to the researchers to describe the various parameters in their own words. For example, the issue of collection of user information, including personally-identifying information (PII) [6], is not addressed apart from the case of social-security numbers (SSN). Besides the guarantees that a researcher should provide for strictly limiting the data collection to that needed for the purposes of the research, they should also state the measures taken to safeguard the privacy of the participants—both against first and third parties—when such data is stored and processed for the duration of the research. Moreover, there is no explicit mandate or procedure to properly dispose of collected data in the case of electronic information collected online. Also, the use of cloud services for storage and processing makes an interesting case of its own as the researchers' own privacy policy is tangled with the terms of service and privacy policy of the cloud services being used. In our case we made sure to honor such requirements and inform our participants when cloud services were used. We believe, however, that as researchers are explicitly instructed on how to handle biological material, the same should apply in electronic and online research involving human subjects.

We found that the IRB of Columbia University was knowledgeable and responsive regarding matters of online research. Nevertheless, we believe that future researchers could benefit from additional information, instructions and training.

> **Lesson learned:** Having a study approved by the IRB can be valuable and is generally considered a positive point by the technical program committee during the review process.

C. Terms of Service

Social networking services base their operation in the storage, processing and management of user-provided data. In an attempt to offer assurances to their users and safeguard their business model, they shape their terms of service accordingly, which in certain cases might prove to be counter-intuitive. For instance, Facebook clearly disapproves [7] accessing its pages and respective data (even the *public* portions) when done in an automated way (i.e., crawling). Large Web search engines are explicitly white-listed [8] and everybody else must apply for, and acquire, written permission by Facebook. In 2010, the social networking service did not hesitate to take legal action against an Internet entrepreneur [9] who collected publicly-available user data from Facebook and, subsequently, released an anonymized version for researchers to use. This was, obviously, a very extreme case, because the entrepreneur *released* the collected data, possibly affecting Facebook's business model.

Overall, we argue that the crawling of publicly-accessible data should not be hindered by social networking services as long as the data collection process does not directly impact the normal operation of the service. Moreover, researchers should also be able to collect private user-owned data as long as the respective users have given their explicit consent. A very encouraging step towards this direction has been taken by Twitter, which exposes API calls [10], not only facilitating the collection of publicly-available user-provided content (i.e., the tweets), but also offering specific API endpoints that perform data sampling.

Researcher: The Good, the Bad, the Ugly. A controversial aspect regards when researchers need to "bend the rules" to carry out their work that could benefit the scientific community, the users of the social networking service and even the service itself. For instance, security researchers might need to evaluate the behavior of users as well as privacy-preserving measures taken by the social networking service so as to propose improvements. This could require creating dummy or test accounts for interacting with the service and its users. Such actions are explicitly forbidden under Facebook's terms of service. However, real-world attackers are not bound by the terms of service. If researchers do not bend the rules, the security and privacy of the social networking services might go untested and vulnerabilities might remain hidden from the general community while being known to potential attackers. We argue that, in such cases, deviating from the terms of service is justified as long as the service itself or the users do not suffer irrevocable damage from such actions. In other words, in the case of dummy accounts, as long as they are destroyed by the end of the research project and any data collected or inflicted side effects are reversed, all sides are benefited. While conducting another research, we strove to obtain Facebook's consensus before proceeding, but it required a substantial amount of time that would have delayed our results. So, even though we were aware that we did not entirely adhere to the terms of service, it seemed that no better option existed—except, of course, that of not conducting our research at all.

> **Lesson learned:** Strive to get in touch with the OSN security team, but try to plan in advance. Be aware of what terms of services you are not adhering to, and include detailed recommendations alongside the discovered security vulnerabilities.

IV. Human Subjects

Having pointed out the security inefficiencies of Facebook's SA, we are continuing with research to improve it. For that matter, we carried out a user study to receive feedback from human participants on our efforts to enhance the quality of this authentication mechanism. Our results can be found here [11]. Since we were evaluating our idea of a modified, photo-based authentication scheme for something as popular as a social networking service, we opted for a diverse set

of participants that could not be found within an academic institution. Therefore, we explored reaching human subjects through crowd-sourcing services, namely Amazon Mechanical Turk (AMT)[3] and ResearchMatch [4].

Initially we designed the environment of our study and decided on its parameters. We felt it was important to create a respectable and trustworthy presence of our study online, in order to invite strangers to participate in it. For this reason we setup a site informing of our research[5]. In the home page, we adopted a layout that quickly and clearly conveys important information about our study. We identify ourselves to visitors of the page, briefly describe the purpose of the study and what information potential participants would have to release to us for the duration of the study. Finally we describe our privacy policy and include a point of contact.

Next, we developed a Facebook application to facilitate the efficient interaction between the experiments driving our study and the participants. We opted for a Facebook application because, first of all, they are deployed within a sandbox run by Facebook itself and are, thus, governed by a series of permissions that clearly state and, subsequently, enforce their capabilities and access to user data. This is important, as it inspires trust in users. Secondly, as we are using Facebook's SA as an example case for improving this type of security mechanism, it was important to integrate our work as close to the mechanics of the service itself as possible. Finally, as we require participants to grant us access to some of the data in their profile (e.g., their social graph), a Facebook application enables direct access. This is also in accordance with our efforts to respect user privacy and minimize collection of potentially sensitive information. In other words, having direct access to Facebook rather than having users upload that information to our own infrastructure means we are able to minimize the amount of a user's information kept within our infrastructure and operate on a best-effort basis to utilize pointers towards the actual Facebook source.

Once our study was ready to commence, we began the process of inviting Facebook users to participate. Through this process we gained experience on human subject involvement in OSN research, and hereby present the obstacles and issues that arose, and how we chose to deal with them.

> **Lesson learned:** Through a respectable website that transparently informs users about our ongoing research we were able to "engage" the community and gain trust among the participants.

A. Give good user incentives

To attract human subjects to participate in the experiments, researchers need to provide incentives. One possible way to entice users is to setup the experiment in such a way that it will appear as a game. Our first attempt was to organize the study as a series of challenges and provide users with a score, depending on how many challenges they solved correctly, and the ability to share that score with their Facebook friends. We thought that this could create a game-like feeling and competitiveness that would attract more users, and provide the incentives for completing multiple tests. This, however, was not the case, as most users found certain parts of our study tedious and stopped after a couple completed challenges. Ideally, the game-like approach would make the participation fun but it turns out that the received satisfaction from solving challenges related to their online friends was not enough to justify the effort required. Therefore, we investigated other incentives to attract a large set of users. The alternative is to provide users with monetary rewards for participating. A method for gaining access to a large set of potential test subjects is through a crowd-sourcing platform. Such platforms, like the Amazon Mechanical Turk service and ResearchMatch, allow one to express his interest for participants or "workers" (or "turks") for a specific task and it is up to the users to contact the initiator of the task. We did not consider active advertisement campaigns (e.g., via email), as that may give the wrong first impression.

B. Crowd-sourcing $\neq$ Easy workforce

Our first attempt was to leverage the Amazon Mechanical Turk (AMT) platform, where users receive monetary compensation for completing tasks. We created a task where users were asked to install our Facebook application and participate in the study. However, our task was rejected by Amazon, as it violated their terms of service due to the following reasons: a) we asked for the participants' (Facebook) identity, b) we required them to register at another website (the site we set up for the study), and c) we required them to install an application. AMT is oriented towards finding humans to carry out simple data processing jobs (e.g., image classification). They could be asked to use a search engine to locate some information, transcribe audio to text or answer statistics-gathering questionnaires. Even though we argue that we were very careful and responsible when designing our study and took measures to ensure the security and privacy of the participants, AMT's automated screening process was unable to realize that and, therefore, did not allow us to proceed. Even though research has been powered by AMT even in security-related projects [12], the service is geared towards more restricted tasks.

An encouraging example is that of ResearchMatch, which pairs researchers with potential participants. Researchers describe the type of study they are running as well as the profile of suitable participants. Users of the service identify interesting cases and apply to join. Unlike the AMT model, monetary compensation is not a part of the process. While users are called volunteers, this does not mean researchers cannot incentivize with monetary compensation. This, however, takes place outside the service. The site's orientation towards the research community is evident by the fact that it requires a valid IRB protocol for any initiated study, and identifies researchers by their affiliation with selected institutions. At the moment it has a little over 30,000 volunteers [13] compared to the over 500,000 workers of AMT [14]. Nevertheless, its environment seems significantly more research-friendly, although it is currently limited to the population of the United States.

In order to take advantage of AMT, we ended up modifying the type of tasks. Instead of requiring the workers to identify

[3] http://mturk.com
[4] http://researchmatch.org
[5] http://resoauth.necst.it/

their friends, we asked them to recognize well-known people (e.g., celebrities). This allowed us to remove the requirement of installing an application and asking for the workers' identity. However, the downside was that we had to had to translate a complex task into a simple image-classification task, which required additional time. Alongside this large-scale study conducted thanks to AMT, we set up a smaller-scale experiment in a controlled environment, which we used as a pilot.

> **Lesson learned:** The anecdotal belief that crowd-sourcing services allow researchers to carry out any type of batch work turned out to be misleading in our case. Once again, having a backup plan (i.e., standalone website and network of contacts) allowed us to finish our study.

C. Representativeness of Human Subjects Set

Another issue that arises when outsourcing tasks to human test subjects, is the representativeness of the set (i.e., how they will perform compared to the general population). A biased selection of test subjects might skew the experimental results, and this should be taken into account when designing the experiment. Although this aspect must be taken into account, there is no recipe for ensuring good representativeness, apart from ensuring that each task is solved by many different workers. Platforms such as AMT allow, to some extent, to select workers based on their reputation (e.g., percentage of correctly solved tasks), but there is currently no support to enforce uniform geographic or demographic distributions of tasks. We agree with previous studies that suggested prudent practices when using crowd-sourcing services [15].

D. Inspecting User Data

During the implementation process of our custom-built face recognition software, we manually inspected user photos for tweaking our parameter selection as well as for debugging purposes. While this might raise ethical concerns as it entails inspecting personal and potentially-private information, this is often unavoidable in the context of research experimentation. Before installing our application, users were informed that their data may be inspected by researchers.

E. Securing User Data

An important factor when storing user data, even if it is only for the duration of the experiments, is to secure it. Typical procedure includes anonymizing the data. In our case, this was not possible because the stored information (e.g., User ID, photo URLs) was used at runtime by our system. Nonetheless, to avoid the leakage of potentially private information, in [4] we conducted the majority of our experiments using publicly-available information and photos. In the cases where we collected photos from real SA tests, which could be private, we deleted all the data after we finished our experiments.

To minimize the chances of user data being obtained by malicious third parties, all data was stored on a single machine located in the proximity of the NECSTLab at Politecnico di Milano, and could only be accessed by a user account created specifically for conducting our experiments. No external (both from other users or from other machines) access to data was allowed, and access to the user account was restricted with our SSH public keys. Also, no files or photos were ever moved from our machine to portable drives or through the network.

V. Workflow vs. Flexibility

On the one hand, in measurement experiments, having a workflow is essential. On the other hand, every measurement must be treated differently. There is no recipe for an experimental workflow, and we are by no means proposing one.

A. Flexibility Pays Back

One of the most painful lessons that we learned is that flexibility is paramount: being able to quickly re-design an analysis task was essential for us to probe and measure. A rigid workflow would not have helped. Nevertheless, some high-level procedures can be depicted, in the hope that other researchers find it a good starting point. Honestly, we were able to draw Figure 2 only once the work was completed by 60–70%: Do not expect to meet with your co-authors and prepare the workflow for the next 3–6 months of measurement. The diagram shows how the design and implementation phases are decomposed into tasks. From our experience, and from the examination of previous work on OSNs, we believe that the resulting workflow is relatively generic, and can be used as a guideline by researchers first approaching this subject.

We divide the high-level workflow into two phases. In the **Design Phase** we sketched a high-level outline of the experiments. In this phase, it is very useful to elicit the phases starting from (1) the data that needs to be collected and (2) the questions that need an answer. In this phase, the researchers must strive to identify the details of their experiments and data, so as to identify the most important ethical and legal issues that may arise in the future. In the **Implementation Phase**, the researchers have already created an outline of their experiments, and are required to continue with the detailed design and implementation of their experiments, which is driven by the availability of resources and existing tools. Again, flexibility always pays back. So, our advice is to keep the plan as an indication, and modify it whenever the results require so.

B. Having a Plan Pays Back

Even if some degree of flexibility is important, measurements with no plans at all do not go very far. A concrete case is when collecting user data or conducting experiments with human subjects. Usually, an institutional review board (IRB) must approve such experiments and, without a plan, there is no formal way to talk to an IRB. The IRB protocol request must contain a thorough explanation of the type of user data that will be collected as well as the nature of the experiments involving human test subjects that will take place. Therefore, researchers have to undergo a preparation phase where experiments are designed in detail and their goals are clearly stated. This deviates from other types of security research, such as exploring methods of misusing a system [16] or searching for vulnerabilities, that may follow more of a "hit and miss" approach. As such, after conceptualizing the experiment and clarifying the desirable end results, both technical and non-technical issues have to be considered. During the design of the data collection and analysis process,

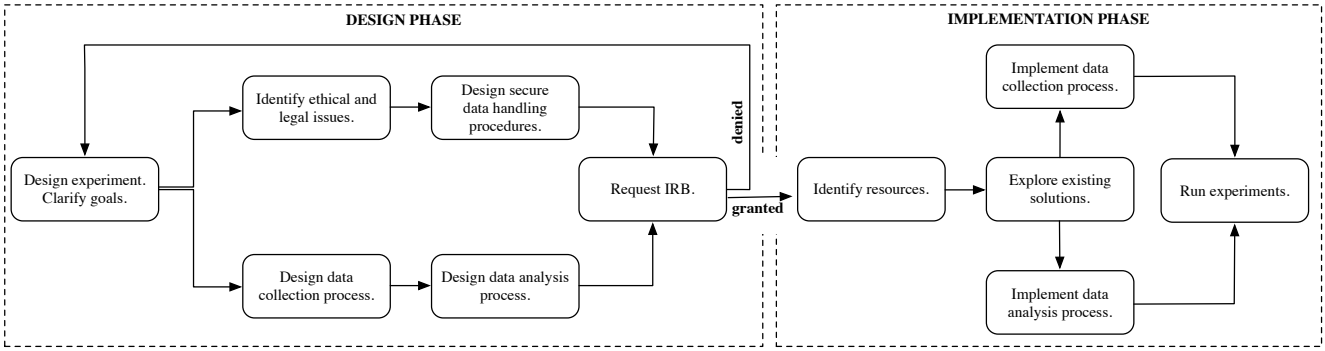


Fig. 2. Overview of the process followed when designing and implementing our research work on the security of Facebook's Social Authentication mechanism.

researchers must also identify the ethical and legal issues that arise. The handling of user data also mandates the design of the procedures that safeguard the data. When these processes have been designed in detail, the researchers can submit the information, and request an approval. If the IRB denies the request, the researchers will have to identify the reasons that led to this decision and re-design the experiments until they are approved.

C. Avoid Over Planning

If the IRB committee approves the request, the implementation phase begins. The first step is to identify the resources at the researchers' disposal. Resource availability is fundamental during this process, as it greatly influences many of the implementation decisions that will be made. Reviewing existing solutions and incorporating such systems can greatly reduce the implementation overhead. For instance, when implementing the data collection system, one may leverage distributed crawlers such as crawl-e[6], which supports crawling across multiple threads and machines. On the other hand, when resource availability is minimal, a centralized crawler with a custom, fine-grained allocation of the available resources may be a better option. As an example, we present the details of our crawler implementation in Section VII.

In regards to the data-analysis procedure, one of our goals was to demonstrate the severity of the security vulnerabilities of the SA mechanism, and the feasibility of our attack. In light of that, we wanted to show how the attack could be accomplished using off-the-shelf face-recognition algorithms as well as free cloud services, which are at any attacker's disposal. This affected the implementation phase of our data-analysis process, where we implemented a custom face recognition solution using an existing framework and also integrated a cloud based solution. We saw in practice, the importance of leveraging existing solutions that can provide much-needed functionality and, as it happened in our case, better results.

> **Lesson learned:** A high-level plan of the measurement experiments is necessary when presenting formal approvals or data-access requests. However, avoid over-planning and be ready to adapt quickly.

[6]http://code.google.com/p/crawl-e/

VI. Design Phase

In this section we detail the decisions when devising the design of the experiments. In our case, we needed to traverse public parts of Facebook's social graph, such that we could collect photos of users which we had previously befriended using a series of dummy accounts (see Section VII-B and VII-C for practical examples on how to create and maintain dummy accounts). We then needed to analyze the photos to produce a dataset of labeled faces, which we would supply to the face recognition algorithm to build models.

The aforementioned dummy accounts were treated as the victim accounts in our experimental scenario, where we assumed the role of the attacker. In this scenario, the attacker knows the password for the accounts, but lacks the social information to solve the SA challenges presented by Facebook. As a matter of fact, we did actually lack the social information even though we owned the victim accounts, as the friends were random strangers that we had befriended.

A. Data Storage: Simplicity Wins

When the experiments mandate the management of large—or, simply, unpredictably large—amounts of data, the selection of the appropriate type of database to be used is driven by two factors: *scalability* (both up and down) and *flexibility*. Scaling down is often under-estimated, although it is a crucial factor. For example, large and complex big-data frameworks pay back in term of scalability (up), but have a very steep setup and learning curve, such that a small modification is impractical.

Scaling Down. Scaling Up. We decided to implement our system upon a lightweight, non-relational database such as MongoDB or CouchDB. Relational databases such as MySQL and SQLite are suitable for small-scale pilots, but are not optimized for handling non-transactional (e.g., purely tabular), big-data repositories with evolving schema (e.g., new attributes or relationships). In addition, OSNs are well represented with graphs data structures, which do not map (natively) onto relational data structures. Last, and most importantly, queries in non-relational databases can scale up easily, if required, thanks to MapReduce. MapReduce has been extensively used in many researches with great benefits (e.g., [17]).

Normalization is Evil. During the phase of data modeling, one must take into account that the principal difference from relational, SQL-like databases is the ability to store and retrieve great quantities of data, and not the relationships between

the elements. Thus, JOIN operations are not supported, and data needs to be de-normalized using explicit references between documents for emulating relationships. Even though non-relational databases cannot, necessarily, give full ACID guarantees, at the same time they do offer a distributed, fault-tolerant architecture and the possibility to scale horizontally. These characteristics fit the prerequisites that stem from managing large amounts of data, where performance and real-time responses are more important than consistency.

Forget About Data Consistency. Apart from being suitable for the management of large volumes of data, non-relational databases are also very flexible as and there is no restriction for the mandatory application of a fixed schema. This results in the ability to change the structure of the collected data even after an experiment has started, without the need of rebuilding the entire database to make the old data consistent with the new structure. Most of the time, ensuring data consistency in an always-running experiment can be as easy and non-disruptive as adding proper "if" conditions in data processing routines.

ORMs are Evil: The Model is The Data. In practice, among all non-relational databases, we chose *MongoDB*[7], a document-oriented database, where data is handled in *collections* of *documents*. To draw a comparison between this concept and that of the SQL style, we could say that collections are like tables, and documents can be considered records. While every record in a table has the same sequence of fields, documents in a collection may have fields that are completely different. Additionally, the format of the responses (JSON) returned from the services in our experiments, perfectly matched the native data type of *dictionaries* in Python. Most importantly, having a data structure and model that maps directly in the chosen programming language is crucial because it removes the need for any object-relation or object-document mapping layer (ORM), which is often the source of bugs or bottlenecks. In addition, JSON is the data-exchange format adopted by many web-service APIs (including Facebook's). Also, in cases of multiple institutions collaborating on the same project, MongoDB offers two methods of cooperation: *replication* and *sharding*. The first one occurs through groups of servers, known as replica sets, and ensures redundancy, backup, and automatic failover. The latter distributes each database across a cluster of machines.

> **Lesson Learned**: Lightweight, flexible data-storage engines that easily scale both up and down, with low setup cost, pay back if compared to complex, big-data frameworks.

B. File Storage

The next design decision was about the type of file storage to be used. The available options in our case were a typical filesystem versus an embedded storage (e.g., GridFS), a system for storing an unlimited number of arbitrarily-large files directly into our data storage (i.e., MongoDB). The machine we chose for the experiments was already equipped with an ext3 formatted drive. The problem, however, is that the maximum number of i-nodes per directory is 32,000 in ext3, and that could pose serious limitations on the data we were about to gather and its organization on disk. Even though this limitation can be overcome using an ext4 filesystem or modifying some internal parameters by rebuilding the ext3 filesystem, this was not an optimal choice because folder indexing would have taken a considerably large time when the folder was accessed. While caching would have surely reduced this overhead, given the amount of data to be saved in files, it would not have solved the problem completely. Moreover, having a separate location for the files, creates an additional burden when setting up backup procedures. Therefore, we decided to rely on GridFS, which also allowed to easily reallocate the underlying database on a new, larger drive in case more space was needed. GridFS works by breaking large files into multiple chunks: It saves the chunks in one collection (`fs.chunks`) and metadata about the file in another collection (`fs.files`). When a query for a file is submitted, GridFS queries the chunk collections and returns the file one piece at a time.

In conclusion, although a filesystem can be seen as the simplest and fastest way, GridFS presents other advantages as well: data replication facilitates load balancing among distributed servers, millions of files can be co-located in a single logical directory without any performance impact and it has increased portability as file management is independent of the application technologies.

> **Lesson learned:** A file-storage solution integrated in the data-storage engine ensured zero setup time and high reliability, and avoids duplicating and maintaining multiple copies of the meta data (i.e., filesystem and data store).

C. Develop vs. Offload

When designing our face-recognition experiments, we identified two type of experiments that we needed to conduct. The first one demanded a more versatile approach towards the selection of algorithm parameters, while the second one was more demanding in terms of face-recognition accuracy. As such, we ended up building a custom solution as well as relying on an existing cloud-based service. On one hand, we did not want to become experts in another field of research. On the other hand, we needed to dig into the implementation details in order to modify some parameters. Given the level of maturity reached by face-detection and recognition technologies, and the availability of open-source libraries, we opted for investing some time to get to know the essential details, for understanding the consequences of our choices. However, we prepared a backup solution in case of unsatisfactory results.

Custom Solutions: Becoming Experts. The major advantage of a custom solution is the versatility in parameter tuning, as every aspect of the algorithm can be rigorously tested with different values. This was very important for specific experiments where we needed to measure the correlation between the size of the training dataset (i.e., number of faces), and the accuracy in recognizing a face. Furthermore, a custom solution allows to conduct multiple offline tests without incurring any limitations from the service provider. Finally, no network latency is present, which greatly affects the experiments' duration when dealing with large amounts of data.

Nonetheless, a custom solution also presents some disadvantages. First and foremost, it takes time and effort to

[7] http://www.mongodb.org

refine it so as to be comparable to state-of-the-art systems. Specifically, the custom solution was effective when simulating a scenario of an attacker that has obtained a fairly large set of photos by infiltrating the victim's social circle with a dummy account. However, in the scenario of an attacker that only relies on a small amount of publicly-available data, the accuracy of our custom solution was not as satisfactory. For this set of experiments, we needed a more accurate process, and explored the possibility of employing a cloud-based service that provides a more accurate face detection algorithm. Also, the computational power requirements are not negligible, as these algorithms are computationally intensive and have long execution times on commodity machines.

Cloud-based Services: Backup Option. Finding a backup option was very easy in our case, thanks to the availability of publicly accessible APIs for face detection and recognition. Ironically, the company offering the API of our choice, face.com, was acquired by Facebook itself, causing some delays to our experiments. However, API directories such as Mashape [8] allowed us access to alternative, bleeding-edge technology.

The advantage of a state-of-the-art solution is the far greater accuracy compared to a custom solution. Development of the face-recognition system is effectively being outsourced to the service, which offers a production-ready tool for researchers to use—although, as mentioned, with limited tweaking options. In terms of available resources, depending on the service and its usual load, researchers may be able to significantly increase their processing capabilities as opposed to utilizing only their local means. By building upon an existing service, no development time is needed for designing and implementing an algorithm that will be far less accurate (unless developed by computer-vision experts), and can be better allocated on other core tasks. Another advantage of a cloud-based solution is that the REST constraints, when present, ensures good scalability.

The most limiting disadvantage of using an existing service, is the restriction of API usage. As the number of API requests per hour is limited, conducting a large number of experiments will last longer than using a custom solution where an infinite number of experiments can be conducted without any restrictions.

> **Lesson learned:** Become familiar with the essential aspects of technologies borrowed from other research areas, and use cloud-based services only if not planning to repeat your experiments in the future, as the services may change or, worse, be shut down at some point.

VII. Implementation Phase

This section describes the practical aspects that we had to deal with during the implementation of our measurement study.

A. Crawling Online Social Networks

One of the most important aspects of research in OSNs is the collection of data. The massive user base mandates the retrieval of large amounts of data that will consist a large enough sample to accurately reflect the various properties of the network. As such, the implementation of the crawler was integral to our experimental process.

Scraping vs. API. The first dilemma we encountered was to decide whether our crawler would use Facebook's public APIs to collect the information we needed, or if we would build an entirely custom solution that did not rely on the API. Our first concern was whether a strict rate limit applies for the API usage. However, Facebook allows 100 million API calls per day [9], which is large enough for our intended experiments.

However, for our experiments we wanted to collect the data in the manner of an attacker, who collects any data left publicly available by Facebook users. We were aware that we should have avoided requests at high rates: Indeed, our choice was not dictated by the need of faster crawling speed, but simply of availability of certain segments of the social graph. Therefore, we implemented our solution so as to mimic the behavior of a "normal" user browsing the website. Thus, for every user, we retrieved the actual page that contains the list of friends, followed by the albums and photos. The entire solution was implemented in `Python` and every single web request was issued through the `urllib2` library, which impersonated the HTTP User Agent of a popular Web browser.

Think Asynchronously. The main problem with our tool was that some steps in the crawling procedures (i.e., album and photo retrieval) were much slower than others, which resulted in them becoming a severe bottleneck of the system. To overcome this obstacle, we built our crawler in a completely modular and asynchronous fashion. We built four standalone modules, each consuming data from an input queue and inserting tasks in an output queue that was in turn processed by the following component or saved into the database. Modern programming languages or libraries such as Akka[10] or Pykka[11] encourage this practice, and allow the developers to abstract the tedious low-level details of event-based or reactive-based programming.

The first module was the *Friend Collector*. It took a list of user IDs (UIDs) as input and browsed the Facebook profile of each one. It retrieved all the data of the user's contacts (name and UID) using regular expressions created specifically for that page's structure. If the number of friends was too large and they did not fit on one page, the module performed multiple requests (just like a browser would have done) to get all of them in multiple passes. Every bit of information was saved into our database and marked as *non-crawled*; at the same time the new UIDs were put in the input queue, so that they would eventually be processed and their friends would be retrieved as well. At the end of this step the initial UID was placed in the output queue, ready to be handled by the other modules.

The second module was the *Album Collector*. Each UID in the input queue was used to reconstruct the URLs of their photo album pages, which were subsequently scraped, and the exact URLs of all the albums were saved into the database and put in the output queue. The user with the respective UID was then marked as *crawled*.

[8] Mashape: https://www.mashape.com/search?query=face+detection

[9] https://developers.facebook.com/policy/

[10] http://akka.io

[11] http://www.pykka.org

This same structure was used by the *Photo Info Collector*. It took as input the queue of album URLs and crawled them one by one, saving into the database the real URL of the photo as well as all the tags each photo contained (coordinates and related UID) and placed the URLs in an output queue.

The last module was the *Photo Downloader*, which downloaded every image it found on its input queue and saved it into the database using the MD5 hash as the key.

Once we started the crawling procedure we understood two key factors. First, this crawling process was much faster than relying on the public APIs (although overcoming the speed limits was not our goal). Second, it was crucial to follow a pipeline design and also effectively distribute resources among the modules, as there was no way we could efficiently retrieve all the data following a sequential process of each user. Overcoming this issue was not trivial. In our initial experiments, our system was acquiring user information for a massive amount of users, while the number of downloaded albums and photos was, of course, significantly smaller. That resulted in our database being filled with potential targets for which we did not have any useful information for our experiments (i.e., photos and tags). For this reason we created an entire web application to keep the single queues monitored and be able to modify the behavior of the single modules at runtime: We could change the number of threads they used, change the request rate or even start and stop them at will, so as to de-allocate resources when necessary. Having a way to continuously monitor the experiments was essential. After a couple weeks of fine-tuning, we ended up putting as few resources as possible on the *Friend Collector* allocating no more than one thread and using a low request rate, while the greatest part of the resources was assigned to the downloading of album and photo information, with up to 32 threads per module.

> **Lesson learned:** Web service APIs are not always the fastest option to retrieve data, although abusing of screen scraping may be against the terms of service. Another crucial aspect for achieving an "always-running experiment": When measuring, having fresh data and receiving notifications when something changes or goes wrong is essential.

B. Mimicking User Behavior

The major asset of an OSN is the vast amount of data that OSNs have acquired. Consequently, they deploy various mechanisms for detecting and preventing automated crawlers from collecting that data. As aforementioned, during our experiments we conducted various actions on Facebook, such as creating test profiles, crawling the network to obtain friend lists and photo information (URLs and tags), and downloading photos. As these actions can lead to the account being suspended, any good crawling system should incorporate measures to avoid triggering such mechanisms. A very important measure is to refrain from "flooding" the OSN with a large amount of requests in short periods of time. In addition, by configuring the crawler to conduct other (automated) actions that resemble the behavior of a human user, we were able to perform our crawling experiments with a stealthier approach and avoid triggering the security mechanisms in most cases—triggering them occasionally is unavoidable. Specifically, we had a component that logged in as our dummy accounts, and mimicked user actions such as "liking" posts of other users and posting trivial status updates.

C. Network Multi Presence

During our experiments, we needed a mechanism for triggering Facebook's SA mechanism. During manual inspection we found that the mechanism was triggered when logging in from geographical locations that had not been associated with the account in the past (i.e., from an IP address belonging to a different country). To add this functionality to our system, we resorted to ToR [18]. By enabling our system to access Facebook through the ToR network, we were able to automatically trigger the SA mechanism. Unfortunately, after a number of logins from a specific location, Facebook stopped triggering the mechanism. However, to bypass that restriction we periodically changed the ToR circuits. This demonstrates that the ToR network can be effectively used for experiments that don't relate to privacy matters, but require a virtual presence at dispersed geographical locations. The downside when using ToR is that the bandwidth is reduced substantially. This factor should be accounted for when planning for the time needed to complete experiments, or by subscribing to VPN services.

D. Data-processing Software

The face-recognition software was the core component of our data analysis phase. As such, we had designed various experiments for evaluating the efficiency of our attack, and exploring whether it poses a realistic threat. We built a custom solution, which presented the advantage of versatility as we could fine tune all algorithm parameters. In addition, as this solution lacked the accuracy of state-of-the-art solutions, we also resorted to cloud based solutions, with higher accuracy. Similar design choices hold for other data-processing tasks, given the wide variety of services[12] that provide tempting opportunities for offloading the implementation.

Using existing systems can greatly reduce implementation time and yield better results. If they are not modifiable (as with cloud-based services) one can resort to hybrid solutions of existing and custom-built components. Depending on each experiment requirements, appropriate components can be used.

Custom Solution. In our case, we used a face-detection classifier part of the OpenCV[13] toolkit. Even though there are plenty of face-detection techniques available in the literature, which are more accurate than those implemented in OpenCV, our goal was to demonstrate that face-based social authentication offered only a weak protection. Even with simple, off-the-shelf solutions, an adversary can implement an automated attack that breaks it.

Existing Cloud-based Service. We investigated whether we could employ advanced face-recognition software offered as a cloud service. We selected face.com, which offered a face-recognition platform that allowed developers to build their own applications. The service exposed an API through which developers could supply a set of photos to use as training set,

[12] E.g., https://www.mashape.com/, programmableweb.com/apis/directory

[13] http://opencv.org/

and then query the service with new unknown photos for the recognition of individuals. The service allowed developers to use up to two different sets of training data, referred to as "namespaces". Each set could hold up to 1,000 users, and we found no restriction on the number of photos that could be used to train a user. A restriction was set on API usage, with 5,000 requests allowed per hour.

One major downside in using face.com was that the service was discontinued when Facebook acquired the company. Unfortunately, this happened while we were working on a follow-up experiment, which was postponed until we found alternative cloud-based, face-recognition APIs.

VIII. Related Work

This paper is a retrospective view on our previous and current research. For similar work, we refer the reader to systematization-of-knowledge (SoK) papers. Our work is also orthogonally related to previous research on privacy in OSNs and, partially, to crowd sourcing.

SoK and Experience Papers. Retrospective and systematization studies are becoming popular in the computer-science research community. We believe that this type of publications, which provide the reader with more than a mere survey, are of paramount importance in the field of system security, because they set the ground for good and prudent experimentation practices. Notable examples of retrospective studies include, for instance [19] or [20]. Notable examples of recent systematization efforts include Rossow's paper on how to design malware experiments [21], or Zhou's work on consolidating the common characteristics of Android malware [22]. Also, specific conference tracks (e.g., in IEEE Symposium on Security and Privacy) encourage the submission of so-called systematization-of-knowledge papers.

As briefly overviewed in the remainder of this section, researchers have conducted their studies by focusing on some of the aspects that we systematize in this paper. We are not aware, however, of any systematization or retrospective work on online social networks research with a focus on system security.

Mimicking User Behavior. Stringhini et al. [1] analyzed how spammers who target social networking sites operate. They created a set of fake, realistic-looking "honey profiles" which passively waited for friend requests from other accounts. The intuition behind this approach was that once the friendship request from a spammer had been accepted, the fake profile would start receiving messages from the spammer directly in his own feed. The authors analyzed some parameters to devise heuristics that could detect anomalies in the behavior of users who contacted the fake profiles while demonstrating some type of mis-behaviour. Even though this approach of populating fake accounts with friends was effective in this case where the goal was to befriend spammers, our experiments called for befriending legitimate users. Thus, we followed an active approach of sending friend requests to other accounts.

[23] piggybacked on existing functionality of OSNs, and leveraged human social behavior to augment the efficiency of their research. Instead of directly issuing friend requests to Facebook users, they simply visited their profiles and took advantage of the fact that Facebook subsequently presented the fake accounts as recommended connections to those users. A large fraction of those users would issue friend requests back to the fake accounts, thereby allowing the researchers to operate in a stealthy manner and quickly establish links to random users.

Security Analysis of OSNs. In two similar studies, Polakis et al. [24] and Balduzzi et al. [25] demonstrated how existing functionality of an online social service can be misused for actions other than those intended for. Specifically, they used the search utilities of social networking sites as an oracle; an attacker can search for an email address, and if a user profile has been registered with that address, the profile is returned. This process allows an attacker to map email addresses to social profiles. Kontaxis et al. [26] also used the search functionality of social-networking services, this time for accessing the social graph indexes which are computed by the service. This enabled the efficient identification of potentially cloned profiles within or across OSNs. A more naïve approach would require extensively crawling the social graph to acquire the same information.

Network Multi Presence. Researchers frequently need to perform network experiments that require multiple and distributed vantage points. Apart from us, other researchers have utilized Tor [18] as well for such experiments. Antoniades et al. [27] carried out distributed network measurements through its geographically disperse topology. By explicitly selecting and routing their traffic through select overlay nodes they evaluated replication strategies of content delivery networks and investigated network neutrality violations through port blocking and traffic shaping. Alicherry et al. [28], in an effort to combat man-in-the-middle attacks, validated self-signed SSL certificates and host keys by fetching them from the respective end host through multiple alternate paths realized by distinct exit nodes.

Crowd-sourcing Platforms. Wang at al. [29] conducted a series of online interviews and surveys that investigated the types of posts which Facebook users regretted having shared. During the survey the users were asked what and why did they regret about some of posts they made, as well as what the consequences caused by these posts were. Survey participants were recruited using the AMT service.

Privacy Implications of Face Recognition. Acquisti et al. [30] investigated the feasibility of combining publicly available OSN data with off-the-shelf face-recognition technology, so as to link online (and potentially sensitive) data to someone's face in the offline world. Three experiments were conducted. In the first experiment they mined publicly-available images from Facebook profiles and attempted to map them to user profiles on one of the most popular dating sites in the United States. In the second experiment they used publicly-available images from a Facebook college network to identify students walking around the campus of a North-American academic institution. In the third experiment they inferred personal information from a subject's OSN profile in real time, after recognizing her face through an application installed on a common mobile phone. Additional personal information was then found (or inferred through data mining) online, and displayed on the phone.

IX. Conclusive Remarks

Comprehensive measurement and data-analysis work is very difficult to achieve and time consuming, especially if conducted from the users' perspective. For example, Maggi et al. in [31] wanted to measure how short URLs are used and perceived by the users. The study required 2 years to complete, and more than 6 months just to have enough users spontaneously subscribe (non-spontaneous subscriptions could bias the measurement).

Empirical works on online social networks are probably the most representative example of user-centered measurements and, as such, require time and certain aspects to be considered.

We believe that this retrospective view on our work will be useful to other researchers working on similar problems. Also, we hope that this will inspire follow-up work and provide future extensions containing more insights and lessons learned while conducting large-scale security or privacy measurements on real Web services.

Acknowledgements

We thank the anonymous reviewers for their valuable comments. This work was supported in part by the FP7 project SysSec funded by the EU Commission under grant agreement no 257007, and by the MIUR under the FIRB2013 FACE grant. This work was also supported by the NSF Grant CNS-13-18415. Author Keromytis was also supported by (while working at) the NSF during the conduct of his work. Any opinions, fundings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or the NSF.

References

[1] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *ACSAC*. ACM, 2010, pp. 1–9.

[2] D. Jacoby, "Facebook Security Phishing Attack In The Wild," http://www.securelist.com/en/blog/208193325/Facebook_Security_Phishing_Attack_In_The_Wild.

[3] Twitter, "Twitter Blog - Election Night 2012," http://blog.twitter.com/2012/11/election-night-2012.html.

[4] I. Polakis, M. Lancini, G. Kontaxis, F. Maggi, S. Ioannidis, A. Keromytis, and S. Zanero, "All your face are belong to us: Breaking facebook's social authentication," in *ACSAC*. ACM, 2012.

[5] H. Kim, J. Tang, and R. Anderson, "Social authentication: harder than it looks," in *FC*. Springer, 2012.

[6] B. Krishnamurthy and C. E. Wills, "On the leakage of personally identifiable information via online social networks," in *WOSN*. ACM, 2009, pp. 7–12.

[7] Facebook, "Automated Data Collection Terms," http://www.facebook.com/apps/site_scraping_tos_terms.php.

[8] ——, "robots.txt," http://www.facebook.com/robots.txt.

[9] P. Warden, "How i got sued by Facebook," http://petewarden.typepad.com/searchbrowser/2010/04/how-i-got-sued-by-facebook.html.

[10] Twitter, "REST API v1.1 Resources," https://dev.twitter.com/docs/api/1.1.

[11] I. Polakis, P. Ilia, F. Maggi, M. Lancini, G. Kontaxis, S. Zanero, S. Ioannidis, and A. Keromytis, "Faces in the distorting mirror: Revisiting photo-based social authentication," in *CCS*. ACM, 2014.

[12] S. Sheng, M. Holbrook, P. Kumaraguru, L. F. Cranor, and J. Downs, "Who falls for phish?: a demographic analysis of phishing susceptibility and effectiveness of interventions," in *CHI*. ACM, 2010, pp. 373–382.

[13] ResearchMatch, "ResearchMatch Metrics," https://www.researchmatch.org/index_pubsitemetrics.php.

[14] Natala@AWS, "MTurk CENSUS: About how many workers were on Mechanical Turk in 2010?" https://forums.aws.amazon.com/thread.jspa?threadID=58891.

[15] J. Ross, L. Irani, M. S. Silberman, A. Zaldivar, and B. Tomlinson, "Who are the crowdworkers?: Shifting demographics in mechanical turk," in *CHI*. ACM, 2010, pp. 2863–2872.

[16] A. Kapravelos, I. Polakis, E. Athanasopoulos, S. Ioannidis, and E. P. Markatos, "D(e — i)aling with voip: Robust prevention of dial attacks," in *ESORICS*, 2010, pp. 663–678.

[17] S. Hanna, L. Huang, E. Wu, S. Li, and C. Chen, "Juxtapp: A Scalable System for Detecting Code Reuse Among Android Applications," in *DIMVA*. Springer, 2012.

[18] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*, 2004.

[19] A. Cui and S. J. Stolfo, "Reflections on the engineering and operation of a large-scale embedded device vulnerability scanner," in *BADGERS*. ACM, Apr. 2011.

[20] S. Li, "Juxtapp and DStruct: Detection of Similarity Among Android Applications," Master's thesis, EECS Department, University of California, Berkeley, May 2012.

[21] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. van Steen, "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook," in *SSP*. IEEE, 2012.

[22] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *SSP*. IEEE Computer Society, May 2012.

[23] D. Irani, M. Balduzzi, D. Balzarotti, E. Kirda, and C. Pu, "Reverse social engineering attacks in online social networks," in *DIMVA*. Springer, 2011, pp. 55–74.

[24] I. Polakis, G. Kontaxis, S. Antonatos, E. Gessiou, T. Petsas, and E. P. Markatos, "Using social networks to harvest email addresses," in *WPES*. ACM, 2010.

[25] M. Balduzzi, C. Platzer, T. Holz, E. Kirda, D. Balzarotti, and C. Kruegel, "Abusing social networks for automated user profiling," in *RAID*. Springer, 2010.

[26] G. Kontaxis, I. Polakis, S. Ioannidis, and E. P. Markatos, "Detecting social network profile cloning." in *SESOC*. IEEE, 2011, pp. 295–300.

[27] D. Antoniades, E. P. Markatos, and C. Dovrolis, "Mor: Monitoring and measurements through the onion router," in *CPACM*. Springer, 2010.

[28] M. Alicherry and A. D. Keromytis, "Doublecheck: Multi-path verification against man-in-the-middle attacks," in *SCC*. IEEE, 2009.

[29] Y. Wang, G. Norcie, S. Komanduri, A. Acquisti, P. G. Leon, and L. F. Cranor, ""I regretted the minute I pressed share": a qualitative study of regrets on Facebook," in *SOUPS*. ACM, 2011.

[30] A. Acquisti, R. Gross, and F. Stutzman, "Faces of Facebook: How the largest real ID database in the world came to be," BlackHat USA, 2011, http://www.heinz.cmu.edu/~acquisti/face-recognition-study-FAQ/acquisti-faces-BLACKHAT-draft.pdf.

[31] F. Maggi, A. Frossi, S. Zanero, G. Stringhini, B. Stone-Gross, C. Kruegel, and G. Vigna, "Two years of short URLs internet measurement: security threats and countermeasures," in *WWW*. International World Wide Web Conferences Steering Committee, 2013, p. 861872.

Classification of SSL Servers based on their SSL Handshake for Automated Security Assessment

Sirikarn Pukkawanna and Youki Kadobayashi
Nara Institute of Science and Technology
Nara, Japan
{sirikarn-p, youki-k}@is.naist.jp

Gregory Blanc, Joaquin Garcia-Alfaro, and Hervé Debar
Institut Mines-Télécom, Télécom SudParis
Evry, France
{gregory.blanc, joaquin.garcia_alfaro, herve.debar}@telecom-sudparis.eu

***Abstract*—The Secure Socket Layer (SSL) and Transport Layer Security (TLS) are the most widely deployed security protocols used in systems required to secure information such as online banking. In this paper, we propose three handshake-information-based methods for classifying SSL/TLS servers in terms of security: (1) Distinguished Names-based, (2) protocol version and encryption algorithm-based, and (3) combined vulnerability score-based methods. We also classified real-world SSL/TLS servers, active during July 2010 to May 2011, using the proposed methods. Finally, we propose 45 features, deemed relevant to security assessment, for future SSL/TLS data collection. The classification results showed that servers had bimodal distribution, with mostly good and bad levels of security. The results also showed that the majority of the SSL/TLS servers had seemingly risky certificates, and used both risky protocol versions and encryption algorithms.**

I. Introduction

In today's Internet, the Secure Socket Layer (SSL) and Transport Layer Security (TLS) are the most widely deployed security protocols used when a client and a server desire to securely exchange data over the Internet. SSL/TLS is used in several ways. Online businesses (e.g., online retails) use SSL/TLS to build customers' confidence that their sensitive information will not be compromised during online transactions. Enterprise mail servers utilize SSL/TLS to encrypt messages being transmitted over the Internet or within Intranets.

Unfortunately, as reported by Netcraft in 2012, SSL/TLS can also be used spitefully [2]. Netcraft found a significant number of phishing websites using valid SSL certificates issued by trusted Certificate Authorities (CA), such as Symantec and Comodo. These websites intended to employ HTTPS to convince victims to trust them. Even though they account for a small fraction of phishing attacks, they are eroding trust in SSL/TLS. In the rest of this paper, we will use the term SSL instead of SSL/TLS.

To establish an encrypted communication using SSL, a client and a server perform a handshake. The client requests the SSL certificate from the server. Upon receiving the server's certificate, the client performs certificate verification as follows. It uses the corresponding preloaded CA's public key to verify the authenticity of the digital signature in the server's certificate. It also validates the certificate by checking the certificate's issued and expiry dates. Finally, it generally verifies that the service for which the certificate has been issued matches the service to which it wishes to connect.

The most popular SSL clients and servers are web browsers and servers. Typically, when encountering a server's certificate issued by an untrusted CA, an expired certificate, or a mismatched domain, browsers issue a security warning to users. Browsers also provide more security assistance to their users. For example, by clicking on a padlock in the browser window, users can see information related to the website's certificate, the certificate's issuer, and the period of validity of the certificate. browsers issue a security warning. Furthermore, browsers show a green address bar when a user is connecting to a website using an Extended Validation (EV) certificate. The green bar implies that such a connection is more secure, because CAs use an audited and rigorous entity authentication to issue an EV certificate [3]. However, users must eventually assume the responsibility of trusting an HTTPS website.

The contributions of this paper are fourfold. First, we propose three methods for classifying SSL servers in terms of security: (1) Distinguished Names-based (DN-based), (2) protocol version and encryption algorithm-based, and (3) combined vulnerability score-based methods. Second, we used the proposed methods to classify a large dataset of real-world SSL servers, which were active from July 2010 to May 2011, and present the results. Third, we studied the correlation between the trustworthiness of the certificates and the security of the server-side security parameters. Fourth, we propose 45 features, deemed relevant to security assessment, for future SSL data collection and analysis.

More specifically, the DN-based method checks identity information, called Distinguished Names or DN in the server's certificate, and then uses that information to determine the security level of that server. The protocol version and encryption algorithm-based method directly takes into account known security flaws of the protocol version and the encryption algorithm chosen by the server. The combined vulnerability score-based method examines multiple criteria related to the server-side security parameter settings. It computes security vulnerability scores for each criteria and eventually combines those scores to assess the server. The usefulness of these methods is that they can be implemented as a supplementary client-side security module (e.g., a web browser plug-in) to aid users in assessing the risk of their SSL communications. For example, a web browser integrating our classifier could raise a security alarm when the user's encrypted data may easily be compromised due to a weak cipher, or when the user connects to a malicious server.

The rest of the paper is structured as follows: In Section II,

TABLE I. FEATURES TO DISTINGUISH MALICIOUS CERTIFICATES FROM LEGITIMATE CERTIFICATES [1]

#	Feature name	Type	Notes
1	md5	boolean	The signature algorithm of the certificate is `md5WithRSAEncryption`
2	bogus subject	boolean	The subject section of the certificate is clearly bogus
3	self-signed	boolean	The certificate is self-signed
4	host-common-name-sim	boolean	The subject's common name and domain name share the same domain root
5	issuer common	string	The issuer's common name
6	issuer organization	string	The issuer's organization name
7	issuer country	string	The issuer's country
8	subject country	string	The subject's country
9	validity duration	integer	The validity period (in days)

we describe related work and existing public SSL datasets. In Section III, we describe the SSL handshake along with which information our methods used and also describe the X.509 certificate. In Section IV, we describe the SSL dataset investigated in this paper. We describe the proposed SSL server classification methods in Section V. The classification and study results are shown in Section VI. In Section VII, we discuss the limitations of our methods, emerging SSL handshake-information-based features for security assessment, and present the list of 45 features for future SSL data collection. Finally, we discuss our conclusions in Section VIII.

II. RELATED WORK

A. SSL Server Assessment

There is much research on analyzing and studying trust networks driven by SSL. Coarfa et al. [4] comprehensively studied the performance costs of SSL, such as CPU and cryptographic operational cost. Eckersley and Burns [5], [6] from the Electronic Frontier Foundation (EFF) and iSEC Partners observed several characteristics of Internet X.509 Certificates such as their validity, issuers, and Distinguished Names (DN). Holz et al. [7] studied the X.509 Public Key Infrastructure (PKI), taking particular interest in what cipher and signature algorithms are most widely used, as well as other statistics such as the number of certificates per host and the most prolific certificate issuers between 2009 and 2011. Amann et al. [8] presented a large-scale study of Internet SSL traffic collected passively from five different operational networks. Vratonjic et al. [9] also studied the behavior of SSL certificates, measuring how many SSL servers have domain mismatches. All of them present analysis results that provide a deeper understanding about the current state of the SSL trust network.

In this paper, we try to assess the security level of an SSL server (e.g., HTTPS website) based on the information embedded in the server's handshake responses. This section describes emerging methods that are similar to our work.

As shown in Table I, Almishari et al. [1] proposed a method for detecting web-fraud domains based on nine certificate features. Firstly, they converted certificates to 29-feature vectors which breakdown as follows: features 5 to 8 account for 6 sub-features each, and the remaining features (features 1 to 4, and 9) account for one sub-feature. Secondly, they separately fed the 29-feature vector set to machine learning-based classifiers to train them. Finally, they selected the domains that were labeled as malicious by the best classifier. The best classifier was judged by precision-recall performance metrics. Pan and Ding [10] proposed an anomaly-based method for detecting phishing pages. They considered DN attributes in web certificates as one of several metrics to discriminate phishing pages from legitimate pages. Their assumption is that a phishing page has DN attribute values that are inconsistent with the claimed identity. Ivan Ristic et al. [11] introduced an SSL server rating guideline on behalf of SSL Labs. They proposed assigning a trustworthiness score to SSL servers based on four criteria which are: (1) the SSL protocol version, (2) the key exchange algorithm and the key size, (3) the encryption key size, and (4) the server's SSL certificate. A high score represents high trustworthiness and vice versa. In the guideline, a server will immediately get a score of zero when it has a self-signed, invalid, expired, revoked, or untrusted certificate. If a domain mismatch is found, the server gets a zero score as well. Finally, the scores from all criteria are combined for the final assessment. The OWASP Foundation also provides an SSL server security testing guideline as a part of the OWASP Testing Guide v3 [12]. They state several testing criteria like the SSL Labs criteria, however, they consider two additional features: the data compression and the hashing algorithms. For example, if a server has an X.509 certificate signed using `MD5`, the server is assessed as a vulnerable server due to known collision attacks on this hashing algorithm [13].

B. SSL Data Collection

To the best of our knowledge, the Crossbear project [14] scanned popular websites in the Alexa list [15] from October 2009 to March 2011 from seven locations in the world, namely Germany, China, Russia, Brazil, Australia, Turkey, and the United States. Crossbear's dataset contains information of scanned SSL hosts (such as the host name and the security setting) and their X.509 certificates. The Electronic Frontier Foundation (EFF) Observatory project [16] provides SSL server responses derived by scanning all allocated IPv4 space on the Internet in August and December 2010. The Zmap Team [17] at the University of Michigan [18], [19] and Rapid 7 [20], [21] provide two datasets: SSL certificate and HTTPS Ecosystem. The SSL certificate dataset includes X.509 certificates of servers scanned from October through December 2013. The HTTPS Ecosystem dataset is another comprehensive X.509 dataset collected by performing 110 Internet-wide scans over 14 months between June 2012 and August 2013. Like Crossbear, École Polytechnique Fédérale de Lausanne (EPFL) researchers [9] also provide SSL certificates of the top one million websites ranked by Alexa [15]. During our survey, we noted that most researchers use publicly available datasets but some also create their own, as is the case in [8] where long-term SSL data was collected passively from SSL traffic flowing through operational networks. Active web server scanning is also common as performed by Almishari et al. [1] and for the data used in [22]. SSL datasets from several sources were rarely combined for analysis [23].

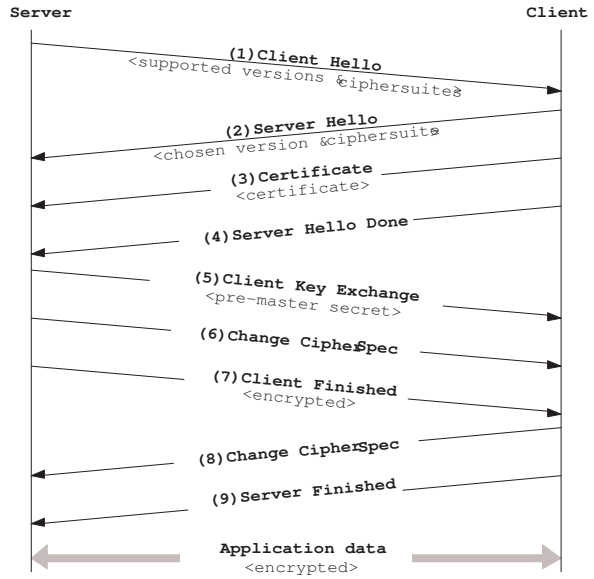


Fig. 1. Steps of the SSL handshake and messages

III. Overview

A. SSL Handshake

SSL helps a client-server application protect the application data during transfer by creating an encrypted channel for private communication over the public Internet. Before an encrypted communication begins, an SSL handshake between the client and server is performed for security parameter negotiation and authentication. This section describes the SSL handshake steps and the information exchanged during the handshake that are used in our work as key information to assess an SSL server. Below we describe the SSL handshake step by step, as shown in Fig. 1.

- The client sends (1) a `Client Hello` message to the server to negotiate security parameters that will be used for the encrypted channel, namely protocol versions, ciphersuites describing key exchange, encryption, hashing algorithms, and compression methods.
- The server chooses an acceptable type for each parameter and replies with (2) a `Server Hello` message.
- The server sends its own SSL certificate, usually an X.509 certificate [24], with (3) a `Certificate` message to the client.
- The server sends (4) a `Server Hello Done` message to notify the client that the server is awaiting a response.
- The client uses the server's certificate to authenticate the server and then sends (5) a `Client Key Exchange` message containing a generated pre-master secret key to the server. Now both the client and the server generate session keys based on the pre-master secret key.
- The client sends (6) `Change Cipher Spec` and (7) `Client Finished` messages to notify the server that the next messages will be encrypted using the session key.
- Finally, the server sends (8) `Change Cipher Spec` and (9) `Server Finished` messages to end the handshake. The client and the server can now exchange application data.

Because our aim was to assess the security level of the SSL server, we examined the information in the server's handshake messages instead of the information in the client's messages. The examined information includes the chosen encryption algorithm in the `Server Hello` message and the certificate in the `Certificate` message because we believe that the encryption algorithm which the server chose is a crucial indicator for assessing security. In addition, we believe that the server's certificate is representative of how secure that server may be.

B. X.509 Certificate

According to the X.509 standard [24], an X.509 certificate contains a public key, a certificate version, a validity period, a serial number, a signature algorithm, and a signature. Aside from those basic fields, the X.509 certificate must contain two Distinguished Names (DN) fields such as the subject DN and the issuer DN. The subject DN is the identity description of the subject who owns the certificate while the issuer DN describes the identity of the Certificate Authority (CA) who issued the certificate. These DNs are used when the client wants to perform name chaining for certification path validation. Technically, the DN is a set of attributes with values separated by comma. For example, a subject DN can be C=`US`,CN=`www.sample.com` in which the C is the country and the CN the common name which is strictly the domain name. For reasons of compatibility and implementations, as shown in Table II, the certificate must contain six standard DN attributes with qualified values, namely C (Country), O (Organization), OU (Organizational Unit), S (State or province), CN (Common Name), and SerialNumber (Serial number).

IV. Collection of Server Responses

The SSL dataset that was used in this work, consists of five SSL surveys: three private surveys launched in July 2010, April 2011, and May 2011 carried out by Levillain et al. [22]; and two publicly available surveys launched in August and December 2011 by the Electronic Frontier Foundation (EFF) [16]. All surveys consist of response messages derived by probing Internet SSL servers during those periods of time. Table III describes the specification of the `Client Hello` messages that were sent out, the total number of SSL hosts in each survey that answered the `Client Hello` messages, and the total number of SSL hosts categorized by the chosen protocol version. Note that the total number of hosts in the #Total SSL hosts column excludes the number of SSL hosts that replied with `Alert` messages to refuse SSL negotiations. For the Jul-2010, Apr-2011, and May-2011 surveys, to gather the SSL handshake data, Levillain et al. first searched for active Internet IPv4 hosts that were listening on port 443. Second, they sent an SSLv2-compatible `Client Hello` message to each active host and terminated the connection after receiving a `Server Hello Done` message. The `Client Hello`

TABLE II. STANDARD DN ATTRIBUTES AND QUALIFIED VALUES [24]

DN attribute name	Description	Qualified value
C	Country	Any ISO numeric or ISO ALPHA-2 country code that its size does not exceed two or three bytes respectively
O	Organization	Any organization name that its size does not exceed 64 bytes
OU	Organizational Unit	Any organization unit name that its size does not exceed 64 bytes
S	State or province	Any state or province name that its size does not exceed 128 bytes
CN	Common Name	Any absolute domain [25] or Certificate Authority (CA) name that its size does not exceed 64 bytes
SerialNumber	Serial number	Any integer that its size does not exceed 64 bytes

TABLE III. SPECIFICATION OF THE CLIENT HELLO MESSAGES AND THE TOTAL NUMBER OF SSL HOSTS CATEGORIZED BY PROTOCOL VERSION IN EACH SURVEY

Survey name	Client Hello message specification		The total number of SSL hosts in survey				
	Offered highest protocol version	Offered ciphersuites	#Total SSL hosts	#SSLv2 hosts	#SSLv3 hosts	#TLSv1.0 hosts	#TLSv1.1 hosts
Jul-2010	TLSv1.0	Standard Firefox suite	9,683,188	0 (0%)	430,973 (4%)	9,252,214 (96%)	1 (<.0%)
Aug-2010	TLSv1.0	SSLv2 and TLSv1.0 suites	11,045,233	11,226 (0.1%)	504,400 (5%)	10,529,606 (95%)	1 (<.0%)
Dec-2010	TLSv1.0	SSLv2 and TLSv1.0 suites	7,705,536	51 (<.0%)	316,933 (4%)	7,388,551 (96%)	1 (<.0%)
Apr-2011	TLSv1.0	Standard Firefox suite	7,134,873	0 (0%)	156,890 (2%)	6,977,983 (98%)	0 (0%)
May-2011	TLSv1.0	Standard Firefox suite	3,796,437	0 (0%)	52,404 (1%)	3,744,133 (99%)	0 (0%)

message for these three surveys offered TLSv1.0 as the highest supported protocol version and offered standard Firefox ciphersuites. For the Aug-2010 and Dec-2010 surveys, EFF sent `Client Hello` messages proposing SSLv2 and TLSv1.0 ciphersuites, and specified TLSv1.0 as the highest supported protocol version. In summary, there were about 9.6 million active SSL servers in the Jul-2010 survey and 11 million SSL servers in the Aug-2010 survey. About seven million SSL servers responded to the Dec-2010 and the Apr-2011 surveys. The May-2011 survey contained about 3.7 million SSL servers. More specifically, most probed SSL hosts (more than 94%) in every survey chose TLSv1.0. The next most popular protocol versions which were chosen were SSLv3 and SSLv2.

V. CLASSIFICATION METHOD

In this paper, we focus on the classification of SSL servers in terms of security using the information embedded in the `Server Hello` and `Certificate` messages sent by the servers during the SSL handshake. This section describes the three proposed classification methods based on that information.

A. Certificate Information

In this work, we make the following assumptions. First, a reliable certificate (e.g., a certificate signed by a trusted CA) tends to include all standard DN attributes with qualified values. Conversely, an unreliable certificate tends to have sloppy DNs. Second, a certificate issued by a known compromised CA or a CA offering incredibly-cheap/free certificates is likely to be a risky certificate. Third, a self-signed certificate is not inherently trusted. Finally, we assume that most malicious servers (e.g., a phishing host) tend to hold unreliable or risky certificates due to their negligence. Based on the information in [24] and these assumptions, we propose the following certificate-based indicators can be used to discriminate risky SSL servers from seemingly harmless SSL servers.

- Indicator 1: at least one standard DN attribute is not presented in the certificate.
- Indicator 2: at least one standard DN attribute value is not a qualified value specified according to Table II.
- Indicator 3: the value of the O/OU attribute contains `self-signed`, `127.0.0.1`, any compromised CA name, or any name of CAs/resellers issuing low-priced or free certificates.

To find compromised CAs, we searched for disclosures regarding CAs that have been compromised and alleged risky CAs. On March 15, 2011, the US CA Comodo reported that its registration authorities had been hacked [26]. In the same year, DigiNotar, a big Dutch CA, was hacked, which resulted in the fraudulent issuing of certificates for a number of domains [27]. GoDaddy was allegedly hacked in 2012, and that resulted in several hours of downtime for millions of websites [28]. However, GoDaddy claimed later that it was only an internal network problem. Below we list the string keywords containing names of the compromised CAs and some CAs/resellers offering low-priced/free SSL certificates identified so far.

- Compromised CAs: `Comodo`, `DigiNotar`, and `GoDaddy`.
- CAs/resellers offering certificates with low-priced costs: `Namecheap`, `RapidSSL`, `fxdomain`, `hostingdude`, and `cheap-domainnames`.
- CAs/resellers offering free certificates: `StartSSL`, `StartCom`, and `CAcert`.

For a real implementation, indicator 2 could be replaced with the following two representative indicators in case that the user's application must reduce memory usage and matching time for analysis. With these representative indicators, the application could use a checklist that is smaller than the size of the full whitelist shown in Table II.

- Representative indicator 1: the CN's value is not in domain name format.
- Representative indicator 2: at least one standard DN attribute (not including SerialNumber) contains one of the following values.
  - an empty or an implied empty value (e.g., "", " ", `NONE`, `None`, `none`, `BLANK`, `blank`, X(s), ?(s), or −(s)),
  - a default value (e.g., S="`SomeState`") or a seemingly meaningless value (see Table IV).

TABLE IV. SEEMINGLY MEANINGLESS VALUES OF EACH STANDARD DN ATTRIBUTE

C (Country)	O (Organization)	OU (Organizational Unit)	S (State/province)	CN (Common Name)
XY, NON-STRING-VALUE, single double quotation	SomeState, Someprovince, SomeOrganization, MyCompany	single double quotation, Single dot, SomeState, Someprovince, SomeOrganizationUnit, Division, section	SomeState, Someprovince, Some_State, Select one, Default, default	localhost.localdomain, 127.0.0.1

TABLE V. PROTOCOL VERSION AND ENCRYPTION ALGORITHM PAIRS FOR SSL SERVER ASSESSMENT

Protocol version	Secure algorithm	Risky algorithm	Insecure algorithm
SSLv3	None	3DES_CBC, IDEA_CBC	DES_CBC, RC2_CBC, RC4
TLSv1.0	None	3DES_CBC, AES_CBC, IDEA_CBC	DES_CBC, RC2_CBC, RC4
TLSv1.1	3DES_CBC, AES_CBC, IDEA_CBC	None	DES_CBC, RC2_CBC, RC4
TLSv1.2	3DES_CBC, AES_CBC, AES_CCM, AES_GCM, Camellia_CBC, Camellia_GCM	None	RC4

B. Protocol Version and Encryption Algorithm

A key assumption we make is that the protocol version and the encryption algorithm chosen by an SSL server are suitable parameters to assess the security level of the communication as well as the SSL server. From a security standpoint, application data sent over a weak protocol version (e.g., SSLv2) or encrypted with a weak encryption algorithm (e.g., RC4 [29]) is in danger due to their security flaws. On the other hand, if the data was sent using a known-good protocol version or a strong encryption algorithm that does not suffer from any known security vulnerabilities, the client can assume that the communication is safer. To assess an SSL server, we analyzed the encryption algorithms supported by each SSL protocol version and assessed their efficiency based on publicly discovered flaws. Table V categorizes the implementation of different encryption algorithms by different versions of SSL with respect to their security. A secure server is one that selects a known-strong protocol version and encryption algorithm. The strongest protocol version at the moment is TLSv1.2. Some known-secure encryption algorithms are the Advanced Encryption Standard (AES) standardized by the US National Institute of Standards and Technology (NIST) in 2001 and Camellia developed later by Mitsubishi and Nippon Telegraph and Telephone (NTT). We consider AES and Camellia as secure encryption algorithms because AES has yet to be compromised by any attacker and Camellia has been proven as strong as AES [30]. Furthermore, to break these algorithms, technically an attacker requires an enormous number of years, e.g., 5×10^{21} years for 128-bit AES encryption [31]. On the other hand, a risky server is one that selects a risky encryption algorithm and SSL protocol version that may be vulnerable to attacks. CBC ciphers are considered risky because they can be broken by the BEAST (Browser Exploit Against SSL) attack [32]. However, if the user's application is designed to defeat the BEAST attack, the attack can be mitigated. For example, Firefox and Chrome using Network Security Services (NSS) libraries for BEAST-like attack mitigation are able to reduce the effect of the BEAST attack. Table V also indicates that if a server uses a newer version of SSL, the security of that server increases because the number of implementation flaws has decreased. For example, a server that chooses 3DES_CBC with TLSv1.1 is described as secure because TLSv1.1 has fixed issues regarding the BEAST attack. Finally, an insecure server is one that selects any publicly known weak encryption algorithm. For example, DES has been defeated by brute-force and differential [33] attacks, and RC2 or RC4 have been defeated by related-key attack [34]. Most importantly, a server that selects old-fashioned SSLv2 with any encryption algorithm is immediately considered as insecure because SSLv2 is flawed in a variety of ways [35]. For example, it has a weak Message Authentication Code (MAC) construction that uses MD5 with a secret prefix, making it vulnerable to length extension attacks [36].

C. Security Assessment Score

Instead of using a single criterion to assess the security of an SSL server, we combine two criteria, still based on the server response: (1) the SSL protocol version and the encryption algorithm that the server chooses and (2) the certificate's DN. Table VI shows the assessment criteria with cases and the assigned scores, which are vulnerability scores, of each case. Criteria 1 consists of nine cases ranked from high to low vulnerability based on the known flaws of those protocol versions and encryption algorithms similar to the proposed method in Section V-B. In Table VI, the set of protocol version and encryption algorithm pairs (secure, risky, and insecure) is the same set shown in Table V. For the scores of criteria 1, we assigned high scores for high vulnerability cases and assigns low scores for low vulnerability cases. As a result, we initially assigned the highest score, which is 1.5, to case 1.1 associated with SSLv2. Due to the smaller number of security flaws of SSLv3 compared to SSLv2, cases 1.2 to 1.4 were assigned lower scores which are in a range from 1.25 to 1. The scoring gap among those SSLv3 cases is 0.125. For the same reason, cases 1.5 to 1.7 associated with TLSv1.0 were assigned a range with lower scores that are between 0.625 to 0.5. The scoring gap between the TLSv1.0 cases is smaller than the scoring gap of SSLv3 cases. It is 0.625, which is one half of the scoring gap of SSLv3 cases (0.125/2). For case 1.8 associated with TLSv1.1, we gave a score of 0.0625, which is smaller than case 1.7's score. We assigned the lowest score to case 1.9 associated with TLSv1.2, which is the most secure SSL protocol at the moment. As a result, case 1.9 has a score of 0.03125, which is one half of case 1.8's score. Next, criteria 2 consists of two cases associated with the certificate's DN. Case 2.1 conforms to indicator 1 described in Section V-A. Case 2.2 conforms to indicators 2 and 3. We assigned a score of 1 to case 2.1, which is similar to the score of case 1.4 because we assumed that a server with a certificate not conforming to the X.509 standard has a moderate vulnerability. Finally, case 2.2 was assigned

TABLE VI. CRITERION AND VULNERABILITY SCORES FOR SSL SERVER ASSESSMENT

Criteria 1: vulnerability of protocol version & encryption algorithm	**Vulnerability score**
1.1: SSLv2 with any encryption algorithm	`1.5`
1.2: SSLv3 with a weak encryption algorithm	`1.25`
1.3: SSLv3 with a risky encryption algorithm	`1.125`
1.4: SSLv3 with a secure encryption algorithm	`1`
1.5: TLSv1.0 with a weak encryption algorithm	`0.625`
1.6: TLSv1.0 with a risky encryption algorithm	`0.5625`
1.7: TLSv1.0 with a secure encryption algorithm	`0.5`
1.8: TLSv1.1 with a weak encryption algorithm	`0.0625`
1.9: TLSv1.2 with a weak encryption algorithm	`0.03125`
Criteria 2: trustworthiness of certificate's DN	**Score**
2.1: At least one standard DN attribute does not presented in the certificate	`1`
2.2: A standard DN attribute with invalid value	`0.2`/instance

TABLE VII. CLASSIFICATION RESULTS BASED ON CERTIFICATE INFORMATION

Survey name	#Seemingly harmless servers	#Risky servers
Jul-2010	3,291,377 (34%)	6,391,811 (66%)
Aug-2010	3,749,160 (34%)	7,296,073 (66%)
Dec-2010	2,957,136 (38%)	4,748,400 (62%)
Apr-2011	2,193,934 (31%)	4,940,939 (69%)
May-2011	1,149,757 (30%)	2,646,680 (70%)

a score of `0.2` for each found invalid value. For example, if the C (Country) attribute is not a country code, and the OU (Organizational Unit) attribute contains `self-signed`, the total score of case 2.2 is `0.4`. Note that these scores are adjustable as long as those scores are in the same order with the score in Table VI or each score is scaled accordingly. For example, if a score of `10` is assigned to case 1.1, then case 1.2's score is supposed to be less than `10`. In addition, case 1.3's score is supposed to be less than case 1.2's score.

To assess the security level of an SSL server, the total vulnerability score ($Tscore$) derived from combining criteria 1's score and criteria 2's score is calculated. Then, the $Tscore$ is used to determine the security level of that server. More specifically, the $Tscore$ is calculated using the following equation.

$$Tscore = \omega_1(criteria1Score) + \omega_2(criteria2Score) \quad (1)$$

where $criteria1Score$ is the score of criteria 1 and $criteria2Score$ is calculated by summing the scores of case 2.1 and case 2.2. The ω_1 and ω_2 are weights for criteria 1 and 2 respectively.

VI. EXPERIMENTAL RESULTS

We performed three experiments to classify real-world SSL servers based on the three proposed methods described in Section V. We used the five SSL surveys from Levillain et al. and the Electronic Frontier Foundation (EFF) described in Section IV. Table III shows their details.. We also studied the behavior of the SSL servers in those surveys, particularly focusing on the relationships between: (1) the certificate quality and the protocol version and (2) the certificate quality, the cipher strength, and the trustworthiness of key exchange algorithm. This section describes the classification results of each method and the study results in detail.

A. Certificate Information

In this experiment, we clustered the SSL servers in the surveys based on the certificate-based classification method that we proposed in Section V-A. We used indicators 1 and 3, and the representative indicators 1 and 2 to separate the SSL servers into two classes: risky and seemingly harmless. We used the same keyword set of compromised CAs and CAs/resellers described in Section V-A for indicator 3. Note that in this experiment, we inspected only the certificate's subject DN because we lacked information of the issuer DN for the Jul-2010, Apr-2011, and May-2011 surveys. If a server's certificate matches at least one indicator, the server is classified as risky immediately. Table VII shows the total numbers of seemingly harmless and risky SSL servers in each survey. The results indicate that more than 61% of the SSL servers in every survey fell into the risky class and the rest of the servers are in the seemingly harmless class. This means that most SSL servers, active during July 2010 to May 2011, had subject DNs that did not conform to the X.509 standard and/or contained seemingly risky values.

TABLE VIII. CLASSIFICATION RESULTS BASED ON PROTOCOL VERSION AND ENCRYPTION ALGORITHM

Survey name	#Secure servers	#Risky servers	#Insecure servers	#Unknown servers
Jul-2010	0 (0%)	5,972,246 (62%)	3,400,572 (35%)	310,370 (3%)
Aug-2010	1 (<0.5%)	7,068,241 (64%)	3,970,174 (36%)	6,817 (<0.5%)
Dec-2010	1 (<0.5%)	4,938,107 (64%)	2,762,196 (36%)	5,232 (<0.1%)
Apr-2011	0 (0%)	4,444,114 (62%)	2,350,419 (33%)	340,340 (5%)
May-2011	0 (0%)	3,675,969 (97%)	109,409 (3%)	11,059 (<0.5%)

B. Protocol Version and Encryption Algorithm

The purpose of this experiment was to classify the SSL servers in the surveys based on the SSL protocol versions and the encryption algorithms that the servers chose during handshakes. We used Table V to classify the servers into three classes: secure, risky, and insecure. If a server chose an encryption algorithm that is not contain in Table V, we classified it as an unknown server. Table VIII shows the total numbers of SSL servers classified into each class for each survey. The results indicate that more than 61% of the SSL servers in every survey appear to be risky. Interestingly, up to 97% of the servers in the May-2011 survey chose risky protocol version and encryption algorithm pairs. In this work, we did not check to see if those risky servers that we found in the May-2011 survey existed or not in the older surveys (the Jul-2010, Aug-2010, Dec-2010, and Apr-2011 surveys). Thus, we could not confirm whether the servers tried to downgrade the security. The results also indicate that there was only one secure server in the Aug-2010 and Dec-2010 surveys. Except for the May-2011 survey, about 35% of all servers chose insecure protocol version and encryption algorithm pairs. Finally, we encountered a few servers that used unknown encryption algorithms.

C. Security Assessment Score

Next, we measured the security levels of the SSL servers using the score-based method proposed in Section V-C. For case 2.2, we used the representative indicators 1 and 2. In each survey, by using equation 1 and the assigned scores in Table VI, we calculated the $Tscore$ of each server. In this

TABLE IX. VULNERABILITY SCORE RANGES WITH SECURITY LEVELS TO ASSESS AN SSL SERVER

Total vulnerability score ($Tscore$) range	Security level
$Tscore < 0.5$	Best
$0.5 \leq Tscore < 1$	Good
$1 \leq Tscore < 1.5$	Average
$1.5 \leq Tscore < 2$	Bad
$Tscore \geq 2$	Worst

experiment, ω_1 and ω_2 were set to be one, which means that we have weighted the two criteria equally. As a result, the $Tscore$ was in the range from 0 to 3.5 ($1(1.5)+1(1+(5\times0.2))$). Note that the values assigned to ω_1 and ω_2 can be adjusted, depending on how much importance is given to each criteria. For example, if using a weak protocol version/encryption algorithm is considered more critical than having an untrusted DN, then $\omega_1 > \omega_2$. Instead of directly representing the security level of a server by its $Tscore$, we defined five intuitive terms to qualitatively describe the security levels: (1) best, (2) good, (3) average, (4) bad, and (5) worst. Table IX shows the total vulnerability score ($Tscore$) ranges for the five security levels. The classification results based on their total vulnerability scores are shown in Table X. The results reveal that there are no servers that appear to be the best server in terms of security in the surveys. About 43% of the servers are good servers and about 50% of the servers are bad servers, a bimodal distribution. The results also show that less than 7% of the servers have either average or worst security levels.

D. SSL Server Behavior

To gain more knowledge from the SSL dataset, we also studied the SSL server behavior. Our aim was to investigate the relationships between (1) the certificate quality and the protocol version; and (2) the certificate quality, the cipher strength, and the security level of the key exchange algorithm of the SSL servers in the surveys.

1) Certificate quality and protocol version: There are three grades of SSL certificates in general, namely Extended Validation (EV), Organization Validation (OV), and Domain Validation (DV). The EV certificate is widely considered to be the most trusted SSL certificate nowadays because obtaining it requires extensive entity verification of the certificate requester by a Certificate Authority (CA) [3]. Thus, an SSL server holding an EV certificate is fairly reliable. The OV certificate requires less entity verification steps than the EV certificate. The DV certificate provides the lowest level of entity verification because a DV certificate can be issued online and often is offered at a much lower price than the OV and EV certificates. This makes the DV certificate less trustworthy than the OV and EV certificates. Self-signed certificates are another type of SSL certificates which are issued by the server themselves with no entity verification. Thus, a self-signed certificate is unreliable.

In this study, we assessed the quality of the SSL certificates in the surveys based on their certificate type and subject DN. We found that most SSL servers in the surveys held OV certificates and provided imperfect DNs. Furthermore, surprisingly we found that some servers chose TLSv1.1 and had self-signed certificates. This means that an SSL server that chooses a secure protocol version does not necessarily also use a trusted certificate.

TABLE X. CLASSIFICATION RESULTS BASED ON SECURITY ASSESSMENT SCORE

Survey name	#Good servers	#Average servers	#Bad servers	#Worst servers
Jul-2010	4,462,945 (46%)	472,745 (5%)	4,494,127 (46%)	253,371 (3%)
Aug-2010	4,933,374 (42%)	521,230 (5%)	5,307,106 (48%)	283,523 (3%)
Dec-2010	3,720,084 (48%)	308,675 (4%)	3,482,081 (45%)	194,696 (3%)
Apr-2011	2,764,267 (39%)	258,491 (4%)	4,027,915 (56%)	84,200 (1%)
May-2011	1,526,521 (40%)	212,406 (6%)	2,053,710 (54%)	3,800 (<0.5%)

2) Certificate quality, cipher strength and trustworthiness of key exchange algorithm: Another aim was to study the relationship between the certificate quality, the cipher strength, and the trustworthiness of the key exchange algorithm that SSL servers in the surveys chose during a handshake. To study this, the certificate quality of a server was measured based on the certificate type and the subject DN similar to the previous study. For cipher strength, we used the information shown in Table V as well as the key size that the server used to encrypt the data. Fundamentally, a large key size will have more strength than a small key size because an attacker requires more time to compromise such key. For example, using the same encryption algorithm, if data A is encrypted with a 256-bit key and data B is encrypted with a 128-bit key, then data A is safer than data B. The last feature that we studied was the trustworthiness of the key exchange algorithm identified in the chosen ciphersuite in the `Server Hello` message. This algorithm is used in the authentication process between a client and a server when they do a handshake. To assess the trustworthiness of the key exchange algorithm, we simply assumes that an anonymous key exchange algorithm is likely to be less robust. On the other hand, a well-known key exchange algorithm is likely to be more trustworthy. Our study found that the majority of SSL servers in every survey chose well-known key exchange algorithms, such as `RSA`, `DHE_RSA`, `DHE_DSS`, `DH_RSA`, and `DH_DSS`. Furthermore, we found that some SSL servers in the surveys had unreliable SSL certificates although they also chose very strong ciphers and vice versa.

VII. DISCUSSION

By analyzing the collection surveys, we found that most SSL servers chose risky encryption algorithms (e.g., `RC4`) with well-known key exchange algorithms. This confirms what Holz et al. [7] and Amann et al. [8] stated who discovered that the most frequent cipher used is `RC4` with `RSA`. We also found that most servers in the surveys were holding Organization Validation (OV) certificates, which are in practice more trustworthy than Domain Validation (DV) and self-signed certificates.

Our results were based solely on known flaws which have been disclosed, among the current implementations on the server side. Therefore, for the moment, our methods may provide effective and precise assessments until a new flaw/attack on cryptographic protocol is revealed or its protocol implementation is refined. This is one limitation of our methods. To preserve the efficiency and accuracy of assessment of our methods, the classification models must be updated regularly.

TABLE XI. PROPOSED 45 FEATURES FOR FUTURE SSL SERVER ASSESSMENT

#	Feature name	Type	Feature description
1	Country	string	The county code where the server is located
2	City	string	The city name where the server is located
3	Region	string	The region code where the server is located
4	Area	integer	The area code where the server is located
5	Time zone	string	The time zone of the region where the server is located
6	DMA	integer	The Designated Market Area code where the server is located
7	Metro	integer	The metropolitan area code where the server is located
8	Postal code	integer	The postal code where the server is located
9	Latitude	number	The latitude number where the server is located
10	Longitude	number	The longitude number where the server is located
11	Continent	string	The continent name where the server is located
12	Organization	string	The organization name who owns the server
13	AS	integer	The Autonomous System number of the zone where the server is located
14	ISP	string	The Internet Service Provider of the server
15*	Protocol version	number	The SSL protocol version that the server chooses during SSL handshake
16	Key exchange algorithm	string	The key exchange algorithm that the server chooses during SSL handshake
17	Hashing algorithm	string	The hashing algorithm that the server chooses during SSL handshake
18*	Encryption algorithm	string	The encryption algorithm that the server chooses during SSL handshake
19*	X.509 country	string	The country attribute's value specified in the SSL certificate's Distinguished Names (DN)
20*	X.509 state	string	The state attribute's value specified in the SSL certificate's DN
21	X.509 city	string	The city attribute's value specified in the SSL certificate's the SSL certificate's DN
22*	X.509 organization	string	The organization name attribute's value specified in the SSL certificate's DN
23*	X.509 organizational unit	string	The organizational unit attribute's value specified in the SSL certificate's DN
24*	X.509 common name	string	The common name attribute's value specified in the SSL certificate's DN
25	X.509 domain component	string	The domain component attribute's value specified in the SSL certificate's DN
26	X.509 surname	string	The surname attribute's value specified in the SSL certificate's DN
27	X.509 given name	string	The given name attribute's value specified in the SSL certificate's DN
28	X.509 email address	string	The email address attribute's value specified in the SSL certificate's DN
29	X.509 MAC	string	The Message Authentication Code attribute's value specified in the SSL certificate's DN
30	X.509 serial number	number	The serial number attribute's value specified in the SSL certificate's DN
31	X.509 title	string	The title attribute's value specified in the SSL certificate's DN
32	X.509 description	string	The description attribute's value specified in the SSL certificate's DN
33	X.509 business category	string	The business category attribute's value specified in the SSL certificate's DN
34	X.509 postal address	string	The postal address attribute's value specified in the SSL certificate's DN
35	X.509 postal code	string	The postal code attribute's value specified in the SSL certificate's DN
36	X.509 post office box	string	The poster office box number attribute's value specified in the SSL certificate's DN
37	X.509 street address	string	The street address attribute's value specified in the SSL certificate's DN
38	X.509 telephone number	number	The telephone number attribute's value specified in the SSL certificate's DN
39	X.509 initials	string	The initials attribute's value specified in the SSL certificate's DN
40	X.509 certificate type	string	The type of the server's certificate (e.g., Extended Validation)
41	Security degree of protocol version	number	The vulnerability score of the SSL protocol version that the server chooses during SSL handshake
42	Security degree of X.509 certificate	number	The vulnerability score of the server's SSL certificate
43	Security degree of cipher	number	The vulnerability score of the encryption algorithm and encryption key size that the server chooses
44	Security degree of key exchange mechanism	number	The vulnerability score of the key exchange mechanism that the server chooses
45*	Security degree of SSL server	number	The overall vulnerability score of the server

Aside from the SSL protocol version, our work considered the cryptographic parameters (encryption algorithm and its key size), the key exchange algorithm, and the server's certificate (type and its identity description). However, other entities also can be used as indicators to measure the security level of an SSL communication/server. SSL Labs [11] assumed that an SSL certificate that has been revoked (whatever the reason) should not be trusted, thus a server with a revoked certificate is penalized as an insecure server. Currently, some web browsers (e.g., Internet Explorer from version 7 and Firefox) optionally consider the revocation status of an X.509 certificate to help their users verify the certificate's validity by using the Online Certificate Status Protocol (OCSP). In [11], the authors also argue that a well-deployed SSL server should avoid the use of a wildcard certificate. They claim that although existing wildcard certificates are not any less secure from a strict technical point of view, the way in which these wildcard certificates are typically handled (especially in larger organizations) makes them less secure in practice. Furthermore, using a wildcard is not permitted for an EV certificate. Thus, the existence of a wildcard may be an good indicator for SSL security assessment. The hashing algorithm used to create the message digest can also be used for SSL server security rating [11], [12]. The server's public key can be used as well to determine the likelihood of exploitation of an SSL server using an old version of OpenSSL [37]. The cryptographic library/toolkit that the server uses can be used for assessment. For example, if an SSL server uses OpenSSL version 1.0.1 through 1.0.1f or 1.0.2 beta through 1.0.2-beta1, it is vulnerable to the Heartbleed attack [38]. The data compression algorithm for an SSL channel is also a relevant indicator [12] because if compression is enabled, the communication can be compromised by the CRIME and BREACH attacks [39], [40]. As a result, most web browsers currently either disable or permanently remove the compression feature to mitigate these attacks.

Finally, based on the experiences gained through this study, we propose 45 features shown in Table XI for future SSL data collection. These features consist of the features or indicators used in the research and practice described above as well as the geolocation features of SSL servers deemed relevant to security assessment. Note that the eight features marked with "*" were used in this work and the remaining features are the newly proposed features. More specifically, features 1 to 14 are the geolocation information of the server, such as the country and the city where the SSL server is located, and the Internet Service Provider (ISP) of the server. Feature 15 is

the protocol version chosen by the server. Features 16 to 18 are security algorithms described in the chosen ciphersuite. Features 19 to 40 are the certificate's DN attributes, which include standard and optional DN attributes. Features 41 and 42 are the vulnerability scores of the chosen SSL protocol version and the server's certificate respectively. Feature 43 is the vulnerability score of the cipher (encryption algorithm and encryption key size). Feature 44 is the key exchange mechanism's vulnerability score. Finally, feature 45 is the overall vulnerability score of the server

VIII. Conclusion

In this paper we proposed three methods to classify SSL servers in terms of security: (1) Distinguished Names-based (DN-based), (2) protocol version and encryption algorithm-based, and (3) combined vulnerability score-based methods. Then we classified Internet SSL servers active between July 2010 and May 2011 based on our proposed methods. We found that more than 61% of the SSL servers were classified as risky because they were using SSL certificates containing seemingly meaningless subject DNs and/or choosing risky SSL protocol versions and encryption algorithms for communications. By considering multiple criteria, we found servers had a bimodal distribution, with mostly good and bad levels of security. Furthermore, we studied a correlation between the trustworthiness of the certificates and the security of the security parameters that the servers chose. We did not find a correlation between them: a server with a trusted certificate may provide insecure communication and vice versa. Finally, we also found that the majority of the servers had Organization Validation (OV) certificates.

Acknowledgment

This research has been supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA). The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the Ministry of Internal Affairs and Communications, Japan, or of the European Commission.

References

[1] M. Almishari, E. De Cristofaro, K. El Defrawy, and G. Tsudik, "Harvesting SSL Certificate Data to Identify Web-Fraud," *I. J. Network Security*, vol. 14, no. 6, pp. 324–338, 2012.

[2] Netcraft, "Phishing Alerts for SSL Certificate Authorities," available at: http://news.netcraft.com/archives/2012/08/22/phishing-on-sites-using-ssl-certificates.html.

[3] The CA/Browser Forum, "Guidelines For The Issuance And Management Of Extended Validation Certificates," available at: https://cabforum.org/wp-content/uploads/EV-Code-Signing-v.1.2.pdf.

[4] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance Analysis of TLS Web Servers," *ACM Trans. Comput. Syst.*, vol. 24, no. 1, pp. 39–69, Feb. 2006. [Online]. Available: http://doi.acm.org/10.1145/1124153.1124155

[5] P. Eckersley and J. Burns, "An observatory for the SSLiverse," July 2010, talk at DEFCON '18. Available at: https://www.eff.org/files/DefconSSLiverse.pdf.

[6] ——, "Is the SSLiverse a safe place," 2010, talk at 27C3. Available at: https://www.eff.org/files/ccc2010.pdf.

[7] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The SSL Landscape: A Thorough Analysis of the x.509 PKI Using Active and Passive Measurements," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 427–444. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068856

[8] B. Amann, M. Vallentin, S. Hall, and R. Sommer, "Revisiting SSL: A Large-Scale Study of The Internet's Most Trusted Protocol," ICSI, Tech. Rep., December 2012.

[9] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J.-P. Hubaux, "The Inconvenient Truth About Web Certificates," in *Economics of Information Security and Privacy III*, B. Schneier, Ed. Springer New York, 2013, pp. 79–117. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-1981-5_5

[10] Y. Pan and X. Ding, "Anomaly Based Web Phishing Page Detection," in *Proceedings of the 22nd Annual Computer Security Applications Conference*, ser. ACSAC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 381–392. [Online]. Available: http://dx.doi.org/10.1109/ACSAC.2006.13

[11] SSL Labs, "SSL Server Rating Guide," SSL Labs, Tech. Rep., 2009, available at: https://www.ssllabs.com/downloads/SSL_Server_Rating_Guide_2009d.pdf.

[12] Open Web Application Security Project, "Testing for SSL-TLS (OWASP-CM-001)," available at: https://www.owasp.org/index.php/Testing_for_SSL-TLS_(OWASP-CM-001).

[13] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques*. Springer-Verlag, May 2005.

[14] Crossbear Project, "SSL Landscape - X.509 data sets," available at: https://pki.net.in.tum.de/node/8.

[15] Alexa, "The top 500 sites on the web," available at: http://www.alexa.com/topsites.

[16] Electronic Frontier Foundation, "The EFF SSL Observatory," available at: https://www.eff.org/observatory.

[17] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap - The Internet Scanner," available at: https://zmap.io.

[18] Internet-Wide Scan Data Repository, "University of Michigan - HTTPS Ecosystem Scans," available at: https://scans.io/study/umich-https.

[19] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS Certificate Ecosystem," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 291–304. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504755

[20] Internet-Wide Scan Data Repository, "Rapid 7 - SSL Certificates," available at: https://scans.io/study/sonar.ssl.

[21] M. Schloesser, B. Gamble, J. Nickel, C. Guarnieri, and H. Moore, "Project Sonar - IPv4 SSL Certificates," available at: https://community.rapid7.com/community/infosec/sonar.

[22] O. Levillain, A. Ébalard, B. Morin, and H. Debar, "One Year of SSL Internet Measurement," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 11–20. [Online]. Available: http://doi.acm.org/10.1145/2420950.2420953

[23] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX, 2012, pp. 205–220. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger

[24] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," rFC5280.

[25] P. Mockapetris, "Domain Names - Implementation and Specification," 2008, rFC1035.

[26] Comodo, "Report of incident on 15-mar-2011," Tech. Rep., March 2011, available at: https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html.

[27] Vasco, "Diginotar reports security incident," August 2011, available at: https://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx.

[28] E. Mills, "Go Daddy-serviced Web sites go down; hacker takes credit," *CNET*, September 2012, available at: http://www.cnet.com/news/go-daddy-serviced-web-sites-go-down-hacker-takes-credit.

[29] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," in *in Proceeding of the 4th Annual Workshop on Selected Areas of Cryptography*, ser. SAC'1, 2001, pp. 1–24.

[30] S. Moriai, A. Kata, and M. Kanda, "Addition of Camellia Cipher Suites to Transport Layer Security (TLS)," July 2005, rFC4132.

[31] H. O. Alanazi, B. B. Zaidan, A. A. Zaidan, H. A. Jalab, M. Shabbir and Y. Al-Nabhani, "New Comparative Study Between DES, 3DES and AES within Nine Factors," *Computing*, vol. 2, no. 3, pp. 152–157, March 2010.

[32] T. Duong and J. Rizzo, "Here Come The $\otimes$ Ninjas," May 2011, unpublished manuscript.

[33] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Cryptology*, vol. 2, no. 3, pp. 3–72, July 1990.

[34] B. Eli, "New types of cryptanalytic attacks using related keys," *Cryptology*, pp. 229–246, 1994.

[35] A. Freier, P. Karlton, P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0," August 2011, rFC6101.

[36] T. Duong and J. Rizzo, "Flickr's API Signature Forgery Vulnerability," Tech. Rep., September 2009, available at: http://netifera.com/research/flickr_api_signature_forgery.pdf.

[37] Debian, "DSA-1571-1 openssl – predictable random number generator," 2008, available at: http://www.debian.org/security/2008/dsa-1571.

[38] Common Vulnerabilities and Exposures, "CVE-2014-0160," available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160.

[39] D. Goodin, "Crack in Internet's foundation of trust allows HTTPS session hijacking," *Ars Technica*, September 2012, available at: http://arstechnica.com/security/2012/09/crime-hijacks-https-sessions.

[40] J. Leyden, "Step into the BREACH: HTTPS encrypted web cracked in 30 seconds," *The Register*, August 2013, available at: www.theregister.co.uk/2013/08/02/breach_crypto_attack.

Are We Missing Labels? A Study of the Availability of Ground-Truth in Network Security Research

Sebastian Abt
da/sec – Biometrics and Internet Security Research Group
Hochschule Darmstadt, Darmstadt, Germany
sebastian.abt@h-da.de

Harald Baier
da/sec – Biometrics and Internet Security Research Group
Hochschule Darmstadt, Darmstadt, Germany
harald.baier@h-da.de

***Abstract*—Network security is a long-lasting field of research constantly encountering new challenges. Inherently, research in this field is highly data-driven. Specifically, many approaches employ a supervised machine learning approach requiring labelled input data. While different publicly available data sets exist, labelling information is sparse. In order to understand how our community deals with this lack of labels, we perform a systematic study of network security research accepted at top IT security conferences in 2009 – 2013. Our analysis reveals that 70% of the papers reviewed rely on manually compiled data sets. Furthermore, only 10% of the studied papers release the data sets after compilation. This manifests that our community is facing a missing labelled data problem. In order to be able to address this problem, we give a definition and discuss crucial characteristics of the problem. Furthermore, we reflect and discuss roads towards overcoming this problem by establishing ground-truth and fostering data sharing.**

I. Introduction

Network security is a highly active field of research. Especially, development of effective and efficient network anomaly detection systems is constantly challenging academia and industry. For anomaly detection, the majority of contemporary research (e.g. [1]–[5]) follows a supervised machine learning or statistical approach and, consequently, requires *a-priori labelled* input data for training and evaluation. Unfortunately, such data is rare. Most public data repositories offering network traffic samples provide only anonymised data and do not contain labels. Hence, data sets available in these repositories can not be linked to other databases (e.g. blacklists) in order to derive labels. As a consequence, researchers often individually collect data sets in environments where expert knowledge of network traffic is available and, because of that, labels can be assigned automatically or semi-automatically. These environments include, but are not limited to working group, campus or industry networks. Data won in such environments typically contains sensitive information, i.e. personally identifiable information, such as IP addresses or login credentials and, unfortunately, cannot be widely shared.

In order to understand how our community handles this limitation in available data sets and which data sets our community utilises, we review in this paper 106 network security papers accepted at top IT security conferences in the years 2009 – 2013 according to the data sets used for training and evaluation. Additionally, we analyse and discuss existing publicly accessible data repositories and the data sets provided therein. Based on these analyses, we identify two main weaknesses in our community:

1) Researchers in our community tend to manually compile data sets for system design. External data sets are typically included for later evaluation. However, both data sets are typically not publicly released. We speculate about reasons for this *data sharing shortcoming* in Sect. III-D1.
2) The absence of a-priori labelled data sets combined with the previously mentioned data sharing shortcoming leads to a lack of ground-truth data. As argued in Sect. III-D2, this *missing labelled data problem* - as we are tempted to call it - affects repeatability and comparability of research.

Furthermore, we reflect the results of our analysis in context of related work in our community. Specifically, we discuss work in three complementary directions that our community may follow in order to foster data sharing and increase repeatability and comparability of research.

Our work is motivated by own experiences when performing data-driven network security experiments. Furthermore, we recognised that absence of adequate data sets and difficulties in compiling such data sets is often incidentally remarked in papers. In doing this analysis, we hope that our paper contributes to and stimulates an ongoing active discussion on availability and quality of labelled data in our community by quantifying and defining the problem we are facing. To the best of our knowledge, we are the first to perform a systematic and comparative analysis of data sets utilised in contemporary network security research.

The remainder of this paper is structured as follows: Section II gives an overview of existing public data repositories and discusses general issues and limitations of these repositories. Section III presents the results of our analysis of recent research and concludes that our community is facing a missing labelled data problem. In Section IV, we discuss and reflect possibilities to overcome this problem. Section V gives an overview and discussion of related work. Section VI summarises and concludes.

II. Analysis of Public Data Repositories

Currently, different public data repositories comprising varying network traffic traces exist. A listing of these repositories is given in Table I. Probably the most notable data repositories (simply by size) are CAIDA and PREDICT. The Cooperative Association for Internet Data Analysis (CAIDA) continuously performs Internet traffic measurements at varying scales and with varying granularity. The CAIDA repository

contains public and semi-public data sets that can be freely downloaded or requested by researchers. The CAIDA repository contains, amongst others, Internet traffic statistics, as well as Internet topology data sets, backscatter traces and real-world Internet traffic captures. In the latter data sets, IP addresses are typically anonymised using Crypto-PAn [6] and packet payload is removed. Unfortunately, traffic captures provided by CAIDA are unlabelled.

The Protected Repository for the Defense of Infrastructure Against Cyber Threats (PREDICT) is an effort to provide a distributed data repository together with centrally managed access processes. PREDICT is funded by the US Department of Homeland Security (DHS) and data sets are contributed by different data providers, one of which is CAIDA. PREDICT offers three classes of data: unrestricted data, quasi-restricted data and restricted data. Unrestricted data is available to every PREDICT user that completed the formal sign-up process. Quasi-restricted and restricted data are only accessible after completing sign-up and after request is granted by the data provider or the PREDICT application review board, respectively. Data sets indexed by PREDICT contain, amongst others, BGP routing data, DNS data, darknet and sinkhole data, Netflow data, topology data as well as packet header captures and synthetically generated data. Most data sets provided via PREDICT do not contain packet payload and many data sets contain anonymised IP addresses. PREDICT indexes 430 data sets in total, from which 30 data sets belong to class unrestricted, 284 to class quasi-restricted and 116 to class restricted. The only unrestricted packet level data source have been collected in 2003, contain anonymised IP addresses and do not contain payload. Furthermore, the traces do not contain explicit per-record class labels. However, an implicit labelling via data categories might be possible.

The other data repositories are smaller in size and mostly resulted from specific research questions. Thus, these repositories serve as good examples of how data sets can be published together with research papers. The Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD) stores data sources containing wireless network data. The DARPA Intrusion Detection Evaluation Datasets (DARPA IDEVAL) [7], [8] have been collected in 1999 and 1998 and are well known and heavily criticised [9] in our community. Despite all criticism on the data sets, both data sets are outdated and do not reflect state-of-the-art attacks seen in contemporary networks. Hence, using these data sets is not recommended for contemporary research. The Internet Traffic Archive (ITA) of the Network Research Group (NRG) at Lawrence Berkley National Laboratory (LBNL) contains anonymised traffic captures and derivatives thereof as well as tools developed for trace recording and anonymisation. The latest update contributed to the repository was in April 2008. The Monitoring and Measurement database (MOME) is a repository of tools and data of different data providers, comparable to PREDICT. The Simpleweb Traffic Traces Data Repository indexes data sets created by the Design and Analysis of Communication Systems (DACS) group of the University of Twente. The repository contains anonymised packet header traces, Netflow records, a Dropbox traffic data set as well as a labelled data set for intrusion detection. The data sets listed there seem to be single-effort data sets related to a particular study performed by the group and, hence, unfortunately do not provide continuous captures. The labelled data set has been collected using an active honeypot [10] in 2008. The UMass Trace Repository of the Laboratory for Advanced System Software of University of Massachusetts Amherst contains different network related data sets which are typically anonymised. Finally, the Waikato Internet Traffic Storage (WITS) project offers packet traces which typically have IP addresses anonymised using Crypto-PAn [6] and payload being removed.

As the above discussion of the data repositories listed in Table I shows, nearly all data sources found in these repositories show at least one of the following three characteristics that impact the sources' utility for network security research:

1) Data sets are anonymised, i.e. sequences of data are removed or modified in order to eliminate personally identifiable information (PII).
2) Data sets are static, i.e. they are compiled for a fixed period of time and may be outdated rather soon.
3) Data sets are unlabelled, i.e. records contained within the data sets are not attributed according to a-priori expert knowledge.

In the following subsections, we briefly argue why these characteristics impact the utility of data sources for network security research.

A. Anonymisation

Anonymisation approaches are comprehensively discussed in literature (e.g. [6], [11]–[14]). Typically, anonymisation is applied to captures of real-world network traffic in order to remove PII from the traces. This is a necessary pre-condition for collection and publication of data in most countries and typically set by current law. Common anonymisation strategies include modification of IP addresses as well as removal of payload information. While such anonymised data may be valuable for specific measurements and statistics, it is typically of less utility to the network security research community. In fact, if anonymised data sets do not provide a-priori labels they typically render themselves useless for network anomaly detection. Specifically, modification of IP addresses in different data sources leads to data that cannot be linked across data sources in order to assign labels. Removal of payload leads to application layer attacks not being detectable.

B. Timeliness

Data sets collected at one specific point in time will be aged in later months or years as the attack landscapes constantly evolve. For instance, the DARPA IDEVAL data sets [7], [15], two data sets heavily utilised from 1999 to 2005, do not contain any command and control (CnC) traffic typically found in today's malware communication. Hence, these data sets are of no utility when it comes to design and evaluation of, for instance, botnet detection systems – solutions countering a highly recognised contemporary threat. Moreover, even the statistical value of a one-time data set may be highly limited, especially when the data set is heavily anonymised, as traffic patterns constantly change [16].

Data repository	URL
CAIDA	http://www.caida.org/data/overview/
CRAWDAD	http://crawdad.cs.dartmouth.edu/index.html
DARPA IDEVAL	http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/
Internet Traffic Archive	http://ita.ee.lbl.gov/
MAWI Working Group Traffic Archive	http://mawi.wide.ad.jp/mawi/
MOME	http://www.ist-mome.org/database/index.html
PREDICT	https://www.predict.org/
Simpleweb Traffic Traces Data Repository	http://www.simpleweb.org/wiki/Traces
UMass Trace Repository	http://traces.cs.umass.edu/index.php/Network/Network
WITS	http://wand.net.nz/wits/

TABLE I. DATA REPOSITORIES OFFERING NETWORK TRAFFIC TRACES.

C. Missing Labels

The absence of labels in data sets requires researchers to manually analyse and attribute data according to phenomena they try to model and detect, if a supervised approach is chosen. This has two fundamental consequences for research: *1)* Depending on the a-priori knowledge available in different research groups, outcome of manual labelling may differ among groups, even when working towards approaches having the same goal. As a consequence, ground-truth available to develop and evaluate different approaches may vary and, consequently, results are not directly comparable. *2)* If data sets are not only missing labels, but also are heavily anonymised, then a-posteriori assignment of labels is very difficult and in most cases impossible. Especially the latter phenomenon effectively diminishes a data sets utility for network security research.

As a consequence, we assume that the data sets available and described above suffering from the characteristics detailed above are currently not heavily used for system design and evaluation. Indeed, this assumption is reflected by our analysis of contemporary network security research given in the next section.

III. ANALYSIS OF CONTEMPORARY RESEARCH

This section presents the results of our review of contemporary network security research accepted at top IT security conferences in the time period 2009 – 2013. In the remainder of this section we describe our paper and conference selection strategy (Sect. III-A) as well as our analysis criteria (Sect. III-B), discuss results of our analysis (Sect. III-C), draw conclusions from these results (Sect. III-D) and reflect these conclusions with responses we received from authors (Sect. III-E).

A. Paper and Conference Selection

In order to understand how our community performs data-driven analysis, design and evaluation, we analyse work accepted at highly rated IT security conferences in the years 2009 – 2013. We limit our analysis to those conferences focussing explicitly on network security or having at least a dedicated network security track. We orient our selection on conference rankings provided by Gu et al.[1], Microsoft Academic Research[2], the conference impact factor proposed by Zhou[3] as well as Google Scholar[4]. As a result, we analyse papers accepted at:

- ACM Conference on Computer and Communications Security (CCS)
- IEEE Symposium on Security and Privacy (S&P)
- International Symposium on Research in Attacks, Intrusions and Defenses (RAID)
- ISOC Network and Distributed System Security Symposium (NDSS)

We would like to underline that we neither aim at giving any particular ranking of the above listed conferences nor that we aim at discriminating any particular other conference not listed above. Specifically, we are aware of other high-quality conferences (e.g. USENIX Security, ESORICS), but at some point we had to make a cut-off in order for the analysis to be feasible. We believe that the conferences listed above constitute a representative sample of top IT security conferences applying the highest standards in peer-review and quality assurance and, thus, serve well for an analysis of contemporary network security research. Furthermore, we limit our analysis to conference papers and do not review journal articles, as we have the impression that in our community new results are preferably published via conferences and that journal articles are typically extended versions of results already published in one or more conference papers. Therefore, we believe that the bias possibly introduced to our analysis by selecting only conference papers is negligible.

In total, 793 papers have been accepted at these conferences in the time period under investigation. From these papers, we select a subset for review and analysis according to the following two criteria:

1) Papers have to focus on network security. More specifically, we do not analyse papers that focus on host-based or software security (e.g. return oriented programming, code analysis, etc.) and cryptography.
2) Papers have to utilise network traffic traces for learning or evaluation. Any paper not relying on captures of network traffic is disregarded.

As a result of this filtering, we analyse 106 papers, constituting approximately 13% of papers accepted at the conferences CCS, S&P, RAID, and NDSS within the time frame 2009 – 2013.

[1] http://faculty.cs.tamu.edu/guofei/sec_conf_stat.htm

[2] http://academic.research.microsoft.com/RankList?entitytype=3&topdomainid=2&subdomainid=2&last=5

[3] http://icsd.i2r.a-star.edu.sg/staff/jianying/conference-ranking.html

[4] http://scholar.google.com/citations?view_op=top_venues&hl=en&vq=eng_computersecuritycryptography

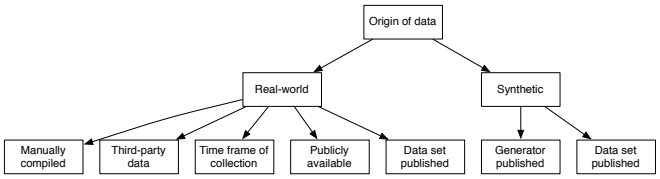


Fig. 1. Illustration and interdependence of criteria used to assess the selected papers.

The papers we analysed are listed in Table II. Again, we believe that this selection of papers and conferences constitutes a representative sample of contemporary work and, consequently, our derived results are sound.

B. Analysis Criteria

As focus of our analysis is on understanding which data sets researchers utilise in order to conduct their work, we perform our analysis according to different data set related criteria. The analysis criteria we apply are structured as illustrated in Fig. 1. As top-most criterion, we analyse whether work is based on real-world or synthetically generated network traffic. Based on the outcome of this analysis, we analyse additional result specific criteria. For real-world data, we assess whether the data set used is manually compiled, provided by third-parties or is publicly available. Furthermore, we assess whether the time span of data collection is provided in the paper and if the data set has been published after work. For synthetically generated data, we assess whether data generators or synthetically generated data sets have been published after work. These criteria are discussed in more detail as follows. Specifically, we provide and discuss a hypothesis reflecting our expectations on the outcome of each criterion. Before, however, we would like to notice that the evaluation criteria are not necessarily mutually exclusive. For instance, research work may incorporate both, synthetically generated data as well as real-world data captures, which both may either be manually compiled or third-party sponsored.

1) Origin of data: We try to understand where data sets used in papers are stemming from. More specifically, as top-most criterion, we analyse whether authors rely on *real-world* (C.1) captures of network traffic or *synthetically generated* (C.2) data. We define real-world captures as any captures that contain network traffic emitted by software that has not specially been crafted to generate traffic for the sake of the traffic itself. Along this line, we define synthetically generated data as any data set containing traffic which has been generated by a computer program that has been developed for the sake of the traffic itself. Specifically, we regard any data set created by capturing packets transmitted in a network or by running malware samples in a sandbox (e.g. [17]–[19]) environment as real-world captures. In contrast, we define data sets generated using simulators (e.g. [20], [21]) as synthetically generated.

Hypothesis: A prevalent impression in network security research seems to be that research results achieved using synthetically generated network traffic are less predictive of the utility of a system in real-world environments than results achieved using real-world traffic captures. We assume that this impression basically results from the difficulty in simulating Internet traffic [16]. On the other hand, Ringberg et al. [22] argue that simulation, and hence data synthesis, is a requirement for sound validation of experiments. Yet, to the best of our knowledge, no studies exist that systematically explore capabilities and limitations of synthetically generated data in network security research. As a consequence, we expect most of the work accepted at the venues above to be based on real-world evaluation.

2) Real-world data: (C.1) In our analysis, we aim at assessing where real-world data sets utilised by researchers are stemming from. As we expect the most work to rely on real-world captures, we would like to understand whether researchers leverage publicly available sources, share data amongst each other or industry, or take the expenses of manually compiling data sets. Hence, we assess papers relying on real-world data sets based on the following criteria:

a) Manually compiled: (C.1a) We define manually compiled data sets as any data set that is collected by researchers in their own premises or third-party premises. Furthermore, we define any publicly available or third-party data set lacking class labels as manually compiled, if and only if researchers manually generate class labels in order to annotate the sponsored data.

Hypothesis: Manually compiling and, especially, labelling data sets is a labour intensive process. Thus, we would expect researchers to rely on third-party or publicly available data sets wherever possible in order to be able to focus on the actual problem at hand instead of data collection. On the other hand, manually compiling data sets allows researchers to assure quality of the input data they use for system design and evaluation. Hence, outcome of research may be more predictive when manually compiled data sets are used.

b) Third-party: (C.1b) Third-party data sets are defined as any real-world data set that has not been manually compiled by the researchers itself, but has been provided by any third-party. For instance, data sets provided by network operators or other researchers are regarded as third-party.

Hypothesis: We expect research to heavily depend on especially industry-sponsored third-party data in order to evaluate own work. By evaluating own work using industry-sponsored third-party data, researchers satisfy the prevalent community belief that real-world data is essential to demonstrate real-world utility and, thus, validate contribution and impact.

c) Publicly available: (C.1c) Any data set that can potentially be publicly accessed by researchers is defined as publicly available. Specifically, we do not require the data-sponsoring entity to publish a specific data set without registration or access restriction. However, we require the data-sponsoring entity to publicly announce the availability of the data (e.g. in conference papers or on websites) and, if required, to provide a publicly accessible registration process.

Hypothesis: As mentioned in Section II, different public data repositories (e.g. CAIDA, PREDICT) exist. Unfortunately, these repositories usually offer anonymised and unlabelled data sets. As argued above, utility of such data sets for network security research may be limited as manual post-processing of the data is still required, if possible. On the other hand, using these data sets as starting ground potentially eliminates tedious

data collection. Balancing these aspects, we hypothesise that researchers heavily utilise publicly available data as starting point.

d) Time frame of publication: (C.1d) For all real-world data sets we analyse whether the time frame of collection is specified in the papers. By definition, this characteristic can not be evaluated for synthetically generated data.

Hypothesis: As not only the attack and threat landscapes are constantly evolving, but also user behaviour is heavily driven by technical advancement [16], we expect this to be a commonly provided information.

3) Synthetically generated data: (C.2) The use of synthetically generated data sets allows researchers to easily craft different data sets during research. Specifically, data can be tailored to model specific aspects and to evaluate corner-cases in order to estimate the boundaries of a proposed solution. If synthetically generated data sets are used in specific work, we specifically assess whether the process of data synthesis and parameters of the underlying models are discussed.

Hypothesis: Use of synthetically generated data gives great flexibility and many degrees of freedom. On the other hand, as mentioned earlier, we recognise a common mindset in the network security research community that questions results achieved using synthetically generated data sets. More specifically, two arguments against the use of synthetically generated data commonly encountered are:

- *Incompleteness* Synthetic data is usually generated by simulation. Simulation relies on specific models of real-world. As these models hide specific aspects of reality in order to be able to terminate simulation in finite time, synthetic data inherently cannot contain the variety of subtleties found in real-world.
- *Artefacts* Synthetic data is typically generated using simulators and specific models of reality. Consequently, synthetic data shows potential to contain artefacts, such as periodicity or determinism, that may not be found in real-world.

Both characteristics of synthetically generated data may negatively impact research. As machine learning (ML) techniques, which are commonly employed for network anomaly detection, perform well in learning and recognising similarities in data, but perform worse in learning and recognising irregularities [23], ML-based approaches fed with synthetically generated data sets may be tempted to specifically learn artefacts during training. As, by definition, these artefacts are not found in real-world captures, approaches may not perform well in real-world environments. Similarly, when models are built with an incomplete representation of reality, approaches may be confronted with unknown patterns in real-world. Hence, high false alarm rates may be expected and approaches deduced from synthetic data may have little utility in real-world. Consequently, we expect the majority of studies accepted at the conferences under investigation to not rely on synthetically generated data sets without incorporating additional real-world data sets.

4) Publication of data: As mentioned earlier, we regard availability of labelled data sets as prerequisite for comparability and repeatability of experiments. Hence, we specifically analyse if data used for design and evaluation of published research is published as well. To assess this property, we analyse papers with regard to paragraphs that indicate publication of data sets or describe a processes how to access data sets used. Furthermore, we use Google search to find data sets using the title of the papers as search query. We denote *publication of real-world* data sets as criterion (C.1e). If a paper relies on synthetically generated data, we not only analyse the *publication of the synthetic data corpus* (C.2a) itself, but also for *publication of the data generator* (C.2b).

Hypothesis: We suppose that importance of data to conduct research is obvious to any active researcher. We furthermore argue that well-processed and labelled data sets are a product of every data-driven research. Hence, publication of data should be effortless after research work has been accepted for publication. On the other hand, we recognise constraints that prohibit public sharing of data. For instance, non-disclosure agreements (NDAs) may especially prohibit publication of industry-sponsored data sets due to fear of loss of customers or reputation. Additionally, data protection law may prohibit publication of data sets containing sensitive information that are vital for research (e.g. in case research focusses exactly on that part of data). Consequently, we expect researchers to publish data sets after research has been performed. In the case this is not possible, we expect researchers to discuss circumstances prohibiting data sharing.

C. Analysis Results

In this section, we present and discuss the results of our analysis of 106 network security research papers. An overview of the results of our empirical study is given in Table II. Specifically, Table II lists the papers reviewed as well as the cumulative numbers of papers categorised according to the criteria defined in Section III-B per conference and year. The last row shows the sum of papers per category for all categories and over all conferences and years. This summary given in the last row is the basis of our statistics. From our analysis we derive four key observations and some curiosities that we will discuss in the following subsections.

1) Real-world data sets are preferred: As a primary result, our analysis reveals that research in the network security area preferably choses real-world captures of network traffic instead of synthetically generated data. Specifically, 88% (93 of 106 papers) of the investigated papers accepted at the conferences mentioned above used real-world captures for learning or evaluation. In contrast, only 16% (17 of 106 papers) of the papers we analysed leveraged synthetically generated data. Interestingly, 10 of the 17 papers utilising synthetic data relied on synthetic data only, i.e. did not use additional real-world data sets. Hence, only 9% (10 of 106) of all papers under investigation did not use real-world data to conduct their work. This statistic follows our initial hypothesis that we expect research to be based on real-world data. We are convinced that this figure underlines our speculation of the current mindset of our community, that research based on synthetic data does not guarantee utility in real-world.

2) Researchers tend to manually compile data sets: From the work based on real-world captures, 44% (41 of 93) utilised third-party sponsored data sets. In contrast, 70% (65 of 93)

Conf.	Year	C.1	C.2	C.1a	C.1b	C.1c	C.1d	C.1e	C.2a	C.2b	Papers
CCS	2013	5	2	4	4	2	3	0	0	0	[24]–[29]
	2012	3	1	2	1	0	2	0	0	0	[30]–[33]
	2011	4	0	4	1	0	2	0	0	0	[34]–[37]
	2010	1	1	1	0	0	0	0	0	0	[38], [39]
	2009	3	0	3	0	1	1	1	0	0	[40]–[43]
S&P	2013	1	1	1	1	0	0	0	1	0	[44], [45]
	2012	2	0	2	0	0	0	0	0	0	[46], [47]
	2011	3	0	2	2	1	2	0	0	0	[48]–[50]
	2010	8	3	7	5	4	4	0	1	0	[51]–[59]
	2009	8	1	3	2	2	6	0	1	0	[60]–[66]
RAID	2013	6	0	2	3	5	3	1	0	0	[67]–[73]
	2012	5	0	4	2	0	1	0	0	0	[74]–[78]
	2011	4	0	4	0	0	4	1	0	0	[79]–[83]
	2010	8	3	7	5	4	4	1	1	0	[84]–[93]
	2009	8	1	3	2	2	6	0	1	0	[94]–[102]
NDSS	2013	9	1	8	5	1	5	0	0	0	[103]–[112]
	2012	3	0	2	1	0	2	0	0	0	[113]–[115]
	2011	5	1	4	2	1	2	1	0	0	[116]–[120]
	2010	4	1	1	3	3	4	0	0	0	[121]–[124]
	2009	3	1	1	2	1	1	0	0	1	[125]–[129]
	Σ	93	17	65	41	27	52	5	5	1	106

TABLE II. RESULTS, IN NUMBER OF PAPERS, OF THE ANALYSIS WE CONDUCTED ON 106 RESEARCH PAPERS ACCORDING TO CRITERIA DEFINED IN SECT. III-B. THE PAPERS WE ANALYSED PER CONFERENCE AND YEAR ARE LISTED IN THE RIGHTMOST COLUMN.

of papers relying on real-world captures utilised manually compiled data sets for learning or evaluation, leading to the conclusion that researchers preferably compile data sets themselves. This result is especially interesting as compiling real-world data sets is a time-consuming task. On the other hand, it underlines the difficulty of obtaining real-world data sets from industry. Non-surprisingly, the most commonly referenced sources for manually compiled data sets are sandboxes or sandnets and the university or working-group network.

3) Publicly available data sets are not leveraged: Our analysis shows that publicly available data sources are leveraged by only 29% (27 of 93) of the papers we studied. This is a very interesting result contradicting our initial hypothesis. We assumed that research would heavily make use of publicly available data sets as these data sets enable rapid start of research. From the results of our analysis we conclude that the lack of class labels for publicly available data sets is an even bigger show stopper than expected. This is presumably amplified by anonymisation of public data, leading to missing sequences or sequences that cannot be linked with other data sources. Thus, post-processing publicly available data (e.g. assigning class labels) is apparently more expensive than compiling an entirely new data set.

4) Data sets are not published: One astonishing result of our survey is that the network security research community seems to be particular reserved when it comes to data publication and sharing. Our analysis of papers accepted at top IT security conferences reveals that only 5% (5 of 93) of real-world data sets used to conduct research were released after acceptance of the work. Results are slightly better for synthetically generated data sets. In 35% (6 of 17) of papers utilising synthetic data, the data set itself or the data generator was published after work has been accepted. In total, however, only for 10% (11 of 106) of the papers we analysed data has been published. This strongly contradicts our initial hypothesis that researchers usually publish their data after the corresponding work was accepted for publication. Even more surprisingly, only a negligible fraction of the papers explained why data sets could not be published.

5) Curiosities: In addition to our four observations that our analysis of contemporary network security research reveals, we found two interesting curiosities which we discuss next.

a) Unknown origin: We found that 6 of 106, i.e. almost 6%, of the papers we analysed did not reveal any information on the underlying data set. During our analysis, it was either unclear whether synthetic or real-world data had been used or where real-world data was stemming from, i.e. whether it was manually compiled, third-party sponsored or publicly available. We regard this as a very serious curiosity if we remember that the conferences under investigation are commonly regarded as top venues.

b) Reporting of time frame: Along that line, our analysis reveals that only 56% (52 of 93) of the papers relying on real-world data published the time frame during which data had been acquired. We are puzzled by this result as on one hand, reporting the period of collection is neither expensive in terms of numbers of lines to be devoted, nor in terms of time required to report. On the other hand, specifying the time frame during which data was acquired effectively helps to assess research results at a later point in time. As vulnerabilities, attacks and tools as well as human Internet behaviour are constantly changing, we regard it as a fundamental requirement of any data-driven research to reveal the time span of data acquisition. As this number is rather high, our only explanation for this observation is that publication of time frames is currently not mandated by reviewers and, hence, seems not to be a prevalent requirement for sound experiments in our community.

D. Analysis Conclusions

From the analysis results presented above, we draw two main conclusions:

1) Data sharing shortcoming: Section III-C2 shows that researchers in our community tend to manually compile their data sets for system design. External data sets are typically

included for later evaluation. However, data sets are typically not publicly released together with the publication as showed in section III-C4. We are particularly astonished by this result, as any active researcher should understand our community's demand on available data sets. When speculating about this issue, we come down to two possible reasons related to a researcher's mindset. We continue to describe these reasons in a notion of stereotype researchers:

a) The restricted researcher: We regard current law of most jurisdictions as one fundamental driver of scarce data sharing amongst researchers. This holds especially true if data utilised for research is third-party sponsored or based on any real-world captures. In such cases, data privacy law usually restricts the researcher's capability of sharing data. If in such cases data sharing is possible at all, researchers are typically required to additionally process data in order to make it conform to law or contractual requirements (e.g. extensive anonymisation of data). We assume that this additional effort is typically not recognised as added value or, more specifically, that a researcher cannot predict the added value of releasing a data set, usually measured in citation count of a paper, at the time of taking the additional efforts.

b) The competing researcher: As mentioned earlier, manually compiling data sets is a very time-consuming process which may require several months or even several years of active work. For instance, if data has to be captured in real-world networks, technical details have to be discussed with operationally responsible peers, contractual (especially NDA-related) details have to be negotiated, data collectors have to be placed, and data has to be manually post-processed in order to remove noise and assign class labels. And most importantly, collection has to be conducted for a sufficiently sized period of time in order to compile a representative data set. Once finished, however, the resulting data source potentially reflects as substrate of very detailed and focussed research, contributing to the reputation of the data owner. As research is a competition on novel ideas and solutions and our community faces intensive career pressure, researchers having access to data sets with limited public accessibility have a clear competitive advantage.

Any of the two cases lead to a shortcoming of data sharing. In combination with a lack of publicly available labelled data or non-linkable publicly available data, as discussed in Section II, this shortcoming consequently leads to a problem we call the *missing labelled data problem*.

2) Missing labelled data problem: The shortcomings of public data repositories (cf. Section II) and the lack of published data sets, as confirmed by our analysis in Section III-C4, combined with the previously mentioned data sharing shortcoming leads to a lack of publicly available ground-truth data, i.e. labelled data. This absence of ground-truth data negatively affects comparability and repeatability of results and, as such, contradicts fundamental principles of science. Furthermore, as for every researcher digging into a new problem domain the expensive acquisition of data sets becomes a prerequisite for successful work, the absence of ground-truth data specifically hinders rapid research and, thus, scientific progress in our community. Hence, our community faces an intrinsic challenge. In order to be able to frame the dimensions of this challenge, we aim at explicitly defining the missing labelled data problem as follows:

Definition. The missing labelled data problem is *the problem of not having access to labelled data sets of adequate quality and utility with respect to the problem to solve at time of problem solving*.

Our definition reflects the following four dimensions that we derive from our analysis of public data repositories and research work:

a) Access to data: As mentioned earlier, one issue lies in the availability of ground-truth data. As publicly available data sets typically do not contain labelling information and only a small fraction of researchers is able or willing to publish data sets, the availability of ground-truth data is limited. However, publicly available ground-truth data sets are required in order to fulfil scientific principles, such as comparability and repeatability of research. If no common ground for analysis and evaluation is available, research results are in fact not comparable and work can not be repeated. And if research is not repeatable, new approaches can not effectively build upon previously published results. Consequently, research is self-contained within research groups and work of different groups is performed in parallel instead of being sequential.

b) Quality of data: Quality of data is commonly expected to be predictive of an approach's utility in real-world. Hence, quality of data is interchangeable with reality of data. If the data at hand is expected to be a representative sample of reality, i.e. if data is expected to be realistic, results achieved when using the data for evaluation are expected to be achieved in real-world environments as well. As consequence, quality of data serves as a measure of transferability of research results. Hence, quality is an important aspect of ground-truth data.

c) Utility of data: By intuition, we expect that ground-truth data sets can not reflect every single aspect of reality. As such, different ground-truth data sets for different problem domains, or even within one and the same problem domain, are required. In order to be able to assess impact of research conducted on a specific ground-truth data set, it is important to understand the coverage of the data set. Hence, we see a specific requirement of ground-truth data sets in the proof of utility of data for a given problem domain. Proof of utility of data for a specific problem to solve is a prerequisite for any further conclusions.

d) Timeliness: One limiting characteristic of existing publicly available data sets is that data sets are usually static, i.e. data sets are typically captured for a specific period of time and afterwards released. However, reality constantly moves and patterns change. For instance, as attacks and threats constantly evolve, network attack patterns change over time. Botnets, as one example, are a consequence of such evolution. While botnets pose a prevalent threat to our today's infrastructure, botnet CnC traffic was not present 20 years ago. Hence, data sets collected at that time are useless for research of botnet countermeasures. Consequently, one challenge and requirement of ground-truth data is its continuous development. In order to address this, next to a ground-truth data set, methodology of data collection has to be discussed in publications and tools required for data collection have to be published.

E. What the authors say

In order to gain more insight into the problem and to reflect our conclusions, we considered surveying a sample of the authors whose paper we reviewed during our analysis. Specifically, we were interested in surveying why authors deliberately decided to release data sets and why not. However, we were particularly unsure how to survey those authors that not released data. Especially, we expected those answers, if received at all, to refer to sensitivity of information collected in a specific restricted-access context and, particular, to NDAs and legal requirements. Indeed, we got similar answers in prior work when we performed experiments that we wanted to compare. Unfortunately, we would not have been able to assess these answers. Particularly, we assume that we would not have been able to judge whether the answering author is of restricted researcher or of competitive researcher stereotype, which would have given interesting insight into our community. We are currently unsure on how to frame such survey best and leave that part for future work.

Nevertheless, we decided to reach out by email to all authors of those papers that published data sets. We presented the authors a brief summary of key results of our analysis and asked to briefly explain why they chose to publicly release the data set. Actually, two authors responded as follows:

- *Overall, we thought that while many malware repositories are available, there was a real need for (largely) labeled malware datasets. Hopefully, other groups can use it to evaluate their malware classification techniques.*
- *We shared the data because: 1) There aren't enough security datasets available so security research is not very repeatable. We felt that this gap needed to be bridged. 2) The privacy laws in [...] were relaxed so it was easier to share the datasets after anonymization.*

Interestingly, the responses exactly stress that our community, while having various publicly accessible data repositories, is missing *labelled* data sets and that, without such data sets, research is not *repeatable*. Both aspects are in line with our argumentation and analysis results. Hence, even if the sample size is small, this result fully supports our conclusions. Also, we find it particular interesting that one of the respondents explicitly mentions the causality between privacy law and ability to release anonymised data. While an analysis of this causality, and especially implications thereof, is not covered in this paper, we regard it as highly interesting research question for future work.

IV. Overcoming the Problem

From Sect. III we conclude that our community inherently faces the missing labelled data problem. If data sets for research are unavailable, experiments can not be repeated and results or claims can not be verified. Additionally, future work can not be evaluated using the same data set. Hence, different results achieved in work with similar objectives are not comparable. While we recognise this as an inherent property of our domain, it fundamentally contradicts principles of science. Consequently, our community has to develop solutions in order to not loose credibility over time. In the remainder of this section, we present and discuss three complementary approaches as a step towards this direction.

The approaches we discuss here have as well been proposed in different work by others (cf. discussion of related work in Section V). However, from our analysis we conclude that our community has not significantly changed since. We can only speculate why this is the case, but we believe that it is due to the intrinsic difficulty of the problem we describe in this paper.

Additionally, we would like to note that we are aware that the problem we discuss in this paper and the possible approaches towards overcoming it are not necessarily unique to network security research or computer science in general. However, we regard ourselves as experts in network security only and hence hesitate to generalise from our observations in this community. Nevertheless, we are convinced that the missing labelled data problem is especially prevalent in network security research as, from our experience, people are increasingly becoming aware of sensitivity of network data; which is a good development demonstrating some success in our field on one hand, on the other hand making sound data-driven network experiments even harder.

A. Establishing ground-truth

In order to address the lack of ground-truth, research has to focus not only on solving prominent problems, but also on generating common ground-truth data sets. That is, research has to accept missing labelled data sets as a problem of itself. For the conferences under investigation in this paper, compilation of ground-truth data had not been listed in the latest calls for papers. From this observation we conclude that working towards this direction is not heavily recognised in our community and, consequently, less attractive for researchers. From a scientific and, specifically, methodological point of view, however, that kind of research is of great value to the community. Hence, work on ground-truth should become one central topic of interest for relevant IT security conferences in order to stimulate research.

Work towards compilation of ground-truth can comprise the following aspects:

1) Real-world captures: Capturing, post-processing and publishing real-world traffic is a challenging effort. In order to assure that traffic is not biased by behaviour of a specific user group (e.g. behaviour of IT security specialists being connected to the working group network), traffic has usually to be collected on more central points in the network [23]. This essentially requires much communication with operationally responsible peers within the own or within other organisations until formal requirements are met and technical issues can be tackled.

Probably most challenging in that direction is finding an anonymisation tradeoff between legal or contractual requirements and utility of data sets. As shown in Section II, heavily anonymised data sets which are not labelled are available. Our analysis results in section III-C3, however, show us that such data sets can hardly be leveraged by our community as the data can not easily be linked to other sources and, hence, labels can not be easily derived. Finding appropriate

anonymisation techniques that satisfy both, the requirements of our community and those of the data sponsoring party is challenging. Especially, data providers' trust in such techniques may be undermined by publications demonstrating effective data leakage due to attacks on the anonymisation technique [130].

Additionally, such approaches should ideally work towards a continuous data capturing platform in order to be able to continuously track changes of network traffic patterns and to be able to access a representative sample at every point in time. As discussed in Section II, this is a fundamental issue of most data sets that have been crafted for one specific research project. Moreover, being able to provide a constant stream of labelled data would effectively stop overstudy of data sets or publication of irrelevant results on outdated data sets, as seen in case of the DARPA IDEVAL data sets.

By definition, a continuous data capturing system deployed at representative sites in real-world would theoretically address all dimensions of the missing labelled data problem given in Section III-D2 and, consequently, would solve the problem. However, we are aware that it is a long road towards this direction, if possible at all. Nevertheless, research towards this direction is valuable and should especially focus on *methodology*. The more we can learn on *how* to securely design such systems, *how* to technically bridge the gap between anonymisation and utility and *how* to sociologically solve privacy concerns, the faster we can proceed. Literally, at the time of writing, we are convinced that the emphasis is on *'how'* to sensibly capture and provide data, i.e. methodology, and not on the data capture itself.

2) Synthesis software: As mentioned above, utility of synthetically generated data is often challenged in our community. Consequently, our analysis in Section III-C1 shows that only 16% of papers under review utilised synthetically generated data. However, to the best of our knowledge, it is yet unproven that synthetically generated data cannot be used to draw valid conclusions. We are convinced that it is possible to build efficient and effective anomaly detection systems that perform well in real-world. Hence, developing a data synthesis toolchain and assessing utility of data generated using these tools reveals as important future work. In fact, Ringberg et al. [22] and Sonchak et al. [131] argue that synthetically generated data is indeed a requirement for performing repeatable network security experiments. In any case, if in an ideal world a data synthesis tool can be generated that is capable of generating traffic samples of high quality and utility, the missing labelled data problem can effectively be solved for the domain addressed by this tool.

However, the challenge of assessing the quality of synthetically generated data remains. One straightforward approach is to perform statistical tests. If synthetically generated data equals real-world captures with regard to statistical distribution of key aspects of the problem domain, probability is high that (statistical) learners being trained on synthetically generated data sets work well on real-world data sets as well. As an alternative, existing learners published in the problem domain of interest (e.g. classifiers) can be used to assess quality of synthetically generated data. If learners showing high performance on real-world data are capable of detecting events in synthetically generated data and vice versa, we can conclude that the synthetically generated data reflects our current understanding of reality. Obviously, however, the disadvantage of these two approaches is the dependency on real-world data, leading to a recursive problem. The advantage, on the other hand, is that those having access to real-world data would be able to derive synthetically generated data sets that can freely be shared without restrictions, increasing the availability of data in our community.

Nonetheless, we are aware and specifically want to highlight that utilising synthetically generated data sets is just the next best approach compared to real-world data. However, we are convinced that publicly accessible synthetically generated data sets and synthesis toolchains can not only greatly increase comparability and repeatability of network security research, but also foster our understanding of network traffic patterns. In any case, we would like to remind and encourage our community to study capabilities and limitations of synthetically generated data as well as methodology of data synthesis. If, after intensive research, our community comes to the conclusion that we can not establish protocols that support effective and efficient sharing of real-world data, we have to live with the second best approach and accept synthetically generated data as ground-truth.

3) Labelling public data: As our analysis and discussion of data repositories in Section II shows, different publicly available data sources exist. However, class labels describing specific characteristics of the data records are usually missing. This correlates with our observation in Section III-C3 that publicly available data sources are rarely utilised in network security research. Specifically, to the best of our knowledge, the only contemporary data sets providing labelled data records with emphasis on intrusion detection evaluation are provided by Sperotto et al. [10] and Song et al. [132]. However, these data sets have both been collected utilising active honeypots.

One valuable approach in compiling a common ground-truth, thus, would be to focus on exactly filling this gap of missing labels. Hence, researchers should focus on generating and publishing class labels for already existing data sources as this would unleash the full utility of already ongoing data collection efforts. Additionally, doing so would release from the burden associated with manual collection of data and allows for rapid advancement and supports comparability of research.

Furthermore, we would like to note that labelling publicly available data sets also comes without costs when data sets are utilised for research anyway. As shown in Section III-C3, 29% of the papers we reviewed utilised publicly available data sets. If labels would have been released afterwards, these studies would have contributed to solve the missing labelled data problem our community is facing. At that time, we can only speculate about the reasons not to release such labels and come to the conclusion that the lack of labelled data and the contribution the authors could have made to the community is virtually not present to the authors as our community as a whole has not fully internalised the problem. Again, from this we conclude that formalising and discussing the missing labelled data problem, as we do throughout this paper and particularly in Section III-D2, is essential in order to overcome it.

B. Indexing data

In order to solve the missing labelled data problem, fulfilling the requirement of access to data, as described in Sect. III-D2, is essential. One step towards that direction is the establishment of a common data sharing platform which can be used to uniquely index data sets, comparable to the digital object identifier (DOI) system. Such data indexing serves two goods:

1) *Referencing of data.* By assigning unique identifiers to data sets published in a data sharing platform, data sets can uniquely be referenced. Hence, data sets can easily be integrated in literature and it will be trivial to look up specific characteristics of data sets.
2) *Availability of data.* Alongside with indexing, data sets should be reliably stored in the data sharing platform. Thus, data sets will be available and accessible for long time spans, making not only research more comparable and transparent, but also supports other research areas, such as the systematical analysis of evolution in our community.

From the data repositories listed in Section II, especially CAIDA, PREDICT and MOME aim at providing such a platform. Both data repositories, PREDICT and MOME, list various data sets (and, for MOME, even tools) of different data providers while CAIDA basically provides access to own data of affiliated institutes and universities. However, there's a significant overlap between the data repositories. Especially, PREDICT lists a significant proportion of data sets also available in the CAIDA repository. The problem we see here is that neither of these repositories aims at developing a unique and standardised naming scheme. Even worse, neither procedures to access restricted data, nor naming of data classes and data sets is identical in all cases. This variability is confusing, especially to new researchers in our community, and should be removed by defining and agreeing on a community standard in data set naming, attributing and indexing.

We are aware that having a data indexing platform is worthless if we are missing appropriate data to index. Hence, we regard establishment of such a data indexing platform as complementary to establishment of ground-truth, as discussed in Section IV-A. On the other hand, as described previously, we already have a significant amount of (unlabelled) data sets available in our industry that would highly benefit from being uniquely indexed by and accessible via such a platform.

C. Incentivising the researcher

Probably the most important, and even most challenging step towards overcoming the missing labelled data problem is incentivising the researcher. As derived in Section III-D1, we consider two stereotype researchers describing the intrinsic motivation and mindset found in our community. While we have no formal proof for these stereotypes to correctly reflect all individuals within our community, we nevertheless believe that it broadly characterises the majority of researchers. When discussing these stereotypes with colleagues, they invariably were able to agree.

The stereotype *restricted researcher*, as discussed in Section III-D1, may be willing to publish his data sets but may be restricted due to outer constraints, the *competing researcher* in contrary may be able to share, but is unwilling to do so. Hence, the restricted researcher may be intrinsically motivated, while the competing researcher may not. One approach to motivate both researchers even stronger is to incentivise publication of data, i.e. to extrinsically motivate researchers until publication becomes a matter of course. One way of achieving this would be to mandatorily demand release or specification of at least one data set or, if synthetically generated data has been used, parameters required to generate data for validation of research alongside with paper submission for all top-ranked publication platforms. This proceeding effectively enforces comparability and repeatability of research and, hence, essential principles of scientific work. Furthermore, it enables the community to incorporate insight from previous work into new work much stronger, even across different research groups, and, thus, allows us to systematically and sequentially solve problems instead of working in parallel, as discussed in Section III-D2.

Ideally, in case of release of new data sets, data sets should be submitted to data sharing and indexing platforms (cf. Section IV-B) and linked to the paper under submission. We believe that such requirement would initiate reconsideration of paper design, especially of data sets used for evaluation of work. In the long term, this proceeding effectively eliminates the missing labelled data problem we are facing thus far. As the time of writing, however, we are not aware of any conference or journal in network security research mandating researchers to specify one publicly accessible reference for repetition of experiments and comparison of results or otherwise incentivises researchers to publish data. On a step towards this direction, ACM Internet Measurement Conference (IMC) is offering a dedicated award for papers contributing novel data sets. Similarly, USENIX Symposium on Networked Systems Design and Implementation (NSDI) offers a community award for the best paper publishing its data and/or code. We propose to adopt similar awards for key network security venues.

We are aware that forcing researchers to specify a publicly accessible data source in order to repeat research and compare results in conjunction with an accepted paper would severely affect our community. Nevertheless, we are convinced that this proceeding is effective in overcoming the issues arising from the missing labelled data problem ware are facing today. Specifically, we would like to note that we do not insist in mandating researchers to publicly release private/restricted data sets in general. As mentioned throughout this paper, we are well aware and understand constraints that prohibit such data release. However, we propose to mandate researchers to give reference to one publicly accessible data set that, in addition to the private/restricted data set, has been used to evaluate the system proposed in a research paper in order to give fellows the possibility to repeat experiments and compare results. Such mandate simply causes the researcher to take the burden of additional efforts of labelling publicly available data sets, crafting a synthetically generated data set or specifying parameters used by an established data generator to synthesise data sets. We believe that this burden is feasible and especially is outweighed by the long-term benefit to the community. Moreover, we predict that the burden of such mandate monotonically decreases when time elapses, as, after a while, a significant amount of reference data would be publicly accessible by definition. In a significantly lowered

version of the above, major conferences could introduce special *data sponsoring papers* sessions. To these sessions, the above requirements should be applied and only papers fulfilling the criteria mentioned above should be considered for acceptance. Hence, such sessions would specifically incentivise papers that focus on contributing data sets and data collection methodology and would explicitly raise broad awareness for the problem, which, as described in Section IV-A, seems currently not to be the case.

On a different location in the continuum, data sharing can practically be incentivised by the data providers. Specifically, we propose data providers to release data sets together with a well-crafted usage codex which especially enforces that labelling information is fed back to the data providers and linked to the data set. When analysing the data repositories mentioned in Section II, we find that data providers typically restrict usage of data (e.g. data sets may not be used to perform research targeting at breaking anonymisation strategies employed) and require researchers to link to a specific paper describing the data collection process. Furthermore, some data providers regularly ask researchers for published work utilising data sets found in their data repositories in order to have that work listed on the data provider's websites (e.g. CAIDA, PREDICT). However, for the repositories we analysed, we did not find any data usage codex requiring researchers to submit information that enrich the publicly accessible data sets. In prior work [133], we propose such codex our community should adhere to. Citing one rule of that codex, we ask that *researchers should publish the results they achieved when utilising a specific set of data. Specifically, the results should be re-submitted to the data providing organisation and should be linked, together with a reference to the research work, online together with the data set [133].* Requiring such codex effectively contributes to establishing ground-truth by labelling public data as proposed in Section IV-A3.

V. Related Work

To the best of our knowledge, no similar comparative study of data sets utilised in network security research has been conducted so far. As data is highly relevant to our community, we therefore believe that the epistemological work we present here is justified. Comparable to our work in spirit is a comparative analysis of malware samples utilised in malware research by Rossow et al. [134]. In this study, 36 academic publications on malware analysis in the time frame 2006 – 2011 have been analysed. Amongst others, the paper identifies shortcomings in transparency, realism and methodology for a significant amount of analysed publications. While this paper analyses malware data sets used, whereas we focus on network traffic captures, the results of [134] are comparable to our results and indicate that our community is facing issues in performing scientific sound experiments.

Recent work supporting our line of argumentation and conclusions in earlier sections is presented in [130], [131], [135]–[140]: Ethics and issues of sharing measurement data have been discussed by Allman and Paxson [130]. Specifically, [130] provides considerations for data providers as well as data receivers. However, the usage codex proposed in [130] gives no recommendation for returning supplementary information to the data providers. We especially regard this as an easy and effective way of data sharing. A discussion and classification of different data available to and required by our community is given by Heidemann and Papdopoulos in [135]. Back in 2009, the authors formulated our community's requirement on annotations and metadata as future research topic. Our work underlines this requirement and quantifies the demand and degree of data sharing. Also, our analysis demonstrates that our community has not significantly evolved with regard to data sharing and availability of labelled data within the last 5 years. This is also underlined by work of Sonchack et al. [131] and Ringberg et al. [22]. In [131], the authors discuss the need of labelled data for evaluation of large scale collaborative intrusion detection systems in order to perform repeatable experiments. To bridge this data gap, Sonchack et al. propose a data synthesis approach called parametrised trace scaling, which aims at expanding small real-world traffic samples to generate large and realistic data sets. In [22], the authors argue that synthetically generated data is required in order gain experimental control and to be able to repeatedly evaluate intrusion detection systems. Specifically, the authors argue that synthetic data should be used for training and evaluation and, afterwards, systems should be verified in real-world. However, the use of static data sets for intrusion detection system evaluation - especially if data is synthetically generated - is also challenged in our community. In [9], McHugh intensively criticises methodology and results of the DARPA IDEVAL data sets [7], [8]. In fact, for these data sets we have seen how research has been tuned to the data sets, data sets have been overstudied and systematic deficiencies of data sets can render research results useless. Nevertheless, this approach has heavily stimulated research activity in our community and contributed a lot to the evolution of our community. For future work, we have to incorporate lessons learned from the DARPA approach and especially have to make sure that labelled data sets are continuously compiled, as proposed in Section IV-A. If we achieve to continuously compile realistic data sets, the necessity of relying on old and overstudied data sets vanishes and technical program committee members have a profound argument to withdraw work that is tailored to data sets or based on arbitrarily old data. In [136], Kenneally and Claffy discuss privacy issues in data sharing and propose a privacy-sensitive sharing (PS2) framework. Specifically, the authors emphasise the need of network traffic data for empiric studies and attribute especially industry hesitation to the challenge in balancing advantages and disadvantages of data sharing due to different legal regimes and flawed technology models. With the PS2 framework, [136] provides a viable guideline and demonstrates its utility in the CAIDA use case. Further approaches describing data acquisition and PII-removal methodologies and challenges are described in [137]–[140]. One of the most prominent and heavily used IP address anonymisation schemes called Crypto-PAn is described in [6]. This scheme proposed by Xu et al. is prefix-preserving, i.e. if two anonymised IP addresses $j' = f(j)$ and $k' = f(k)$ coincide with the first n bits, then the first n bits of the original IP addresses j and k are equal, too. Crypto-PAn is based on cryptographic hash functions. Indeed, the author demonstrates that the scheme is cryptographically strong.

VI. Summary and Conclusion

Research in the area of network security is heavily data-driven. Especially, in our daily work we observed that our community heavily relies on the availability of a-priori labelled data. As we experienced in own work, such data is hard to find. Indeed, an analysis of data repositories we performed shows that publicly accessible labelled data sets is a rare good. For inherently empirical studies, this observation is quite idiosyncratic. On one hand, data is a necessity in order to perform empirical studies and to be able to publish results. Given the number of empirical studies our community publishes per year, we conclude that such data indeed exists. On the other hand, finding publicly accessible labelled data sets is nearly impossible. From that observation, we hypothesise that our community does not share data sets. In order to be able to accept or reject this hypothesis, we perform a systematic study of 106 network security research papers accepted at CCS, S&P, RAID and NDSS conferences in the years 2009 – 2013. As a result of our analysis, we find that the majority, i.e. 70%, of the papers we review relies on manually compiled data sets. Furthermore, our analysis reveals that only a very small fraction, i.e. 10%, of the papers we analyse release data sets or data generators after compilation. We also notice that a significant amount of work we review, i.e. 44%, tend to utilise external data sets from industry, which are not released either. Interestingly, a surprisingly small fraction of the investigated papers, i.e. 29%, utilise existing public sources containing network traffic. From that analysis, we have to accept our hypothesis and conclude that our community is facing a missing labelled data problem. To the best of our knowledge, we are the first to quantify this severe issue of our community by empirical analysis of contemporary research. In order to be able to frame the problem our community is facing, we derive a definition of the missing labelled data problem and discuss its crucial dimensions. Furthermore, we propose different research areas and challenges towards establishment of ground-truth and propose to establish a common data sharing and indexing platform. Furthermore, we propose how to incentivise researchers to publish and share data.

While we are aware that some of our proposals are rigorous and would severely affect methodology in our community, we deliberately chose to bluntly formulate them. We are convinced that these proposals have the potential to contribute to and stimulate an active discussion in our community. We are aware that similar issues exist in other academic sciences. As computer science is a particular young discipline, we propose to learn from more matured disciplines. Specifically, we are aware that approaches to overcome the missing labelled data problem comparable to those raised by us are currently established in other disciplines of science. For instance, the Nature journal[5] requires authors to share and submit their complementary data. Similar requirements can be found for publishing in Science[6] and the Oxford Journal of Heredity[7]. From that observation, we conclude that an elaborate discussion of this phenomenon in our community is satisfied. Especially, we recognise and want to point out that absence of data, on which empirical analysis is based on, contradicts basic principles of science. Specifically, unavailability of data hinders repeatability of research and comparability of results. While we are well aware and understand constraints that limit general availability of data, as a community we nevertheless have to take care of maintaining a scientific approach in order to not loose credibility over time. Especially, notwithstanding healthy competition and career pressure, we have to make sure that the competing researcher mindset we discussed in this paper does not become prevalent. We are tempted to claim that this is as important to our community as solving the everyday security issues we're facing.

[5] http://www.nature.com/authors/policies/availability.html

[6] http://www.sciencemag.org/site/feature/contribinfo/prep/gen_info.xhtml

[7] http://www.oxfordjournals.org/our_journals/jhered/for_authors/msprep_submission.html

Acknowledgments

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under grant number 03FH005PB2 (INSAIN) and has been supported by CASED. We would like to thank Vern Paxson for his inspiration to coin the problem missing *labelled* data problem instead of missing data problem.

References

[1] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 129–138.

[2] S. Abt, C. Dietz, H. Baier, and S. Petrović, "Passive remote source nat detection using behavior statistics derived from netflow," in *Emerging Management Mechanisms for the Future Internet*. Springer, 2013, pp. 148–159.

[3] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: finding bots in network traffic without deep packet inspection," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '12, ACM. New York, NY, USA: ACM, 2012, pp. 349–360. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413217

[4] W. He, G. Hu, and Y. Zhou, "Large-scale ip network behavior anomaly detection and identification using substructure-based approach and multivariate time series mining," *Telecommunication Systems*, vol. 50, no. 1, pp. 1–13, 2012.

[5] J. François, S. Wang, W. Bronzi, T. Engel *et al.*, "Botcloud: Detecting botnets using mapreduce," in *Information Forensics and Security (WIFS), 2011 IEEE International Workshop on*. IEEE, 2011, pp. 1–6.

[6] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon, "Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme," in *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*. IEEE, 2002, pp. 280–289.

[7] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000.

[8] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, vol. 2. IEEE, 2000, pp. 12–26.

[9] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000. [Online]. Available: http://doi.acm.org/10.1145/382912.382923

[10] A. Sperotto, R. Sadre, F. Vliet, and A. Pras, "A labeled data set for flow-based intrusion detection," in *Proceedings of the 9th IEEE International Workshop on IP Operations and Management*, ser. IPOM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 39–50. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04968-2_4

[11] J. Xu, J. Fan, M. Ammar, and S. B. Moon, "On the design and performance of prefix-preserving ip traffic trace anonymization," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 2001, pp. 263–266.

[12] D. Koukis, S. Antonatos, D. Antoniades, E. P. Markatos, and P. Trimintzios, "A generic anonymization framework for network traffic," in *Communications, 2006. ICC'06. IEEE International Conference on*, vol. 5. IEEE, 2006, pp. 2302–2309.

[13] T. Brekne, A. Årnes, and A. Øslebø, "Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies," in *Privacy Enhancing Technologies*. Springer, 2006, pp. 179–196.

[14] R. Pang, M. Allman, V. Paxson, and J. Lee, "The devil and packet trace anonymization," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 29–38, 2006.

[15] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *Recent Advances in Intrusion Detection*. Springer, 2003, pp. 220–237.

[16] S. Floyd and V. Paxson, "Difficulties in simulating the internet," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 392–403, 2001.

[17] C. Gorecki, F. C. Freiling, M. Kührer, and T. Holz, "Trumanbox: improving dynamic malware analysis by emulating the internet," in *Stabilization, Safety, and Security of Distributed Systems*. Springer, 2011, pp. 208–222.

[18] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti, "Detecting environment-sensitive malware," in *Recent Advances in Intrusion Detection*. Springer, 2011, pp. 338–357.

[19] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *Security & Privacy, IEEE*, vol. 5, no. 2, pp. 32–39, 2007.

[20] G. Carneiro, "Ns-3: Network simulator 3," in *UTM Lab Meeting April*, vol. 20, 2010.

[21] A. Varga *et al.*, "The omnet++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'2001)*, vol. 9. sn, 2001, p. 185.

[22] H. Ringberg, M. Roughan, and J. Rexford, "The need for simulation in evaluating anomaly detectors," *Computer Communication Review*, vol. 38, no. 1, pp. 55–59, 2008.

[23] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 305–316.

[24] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee, "Beheading hydras: performing effective botnet takedowns," in *ACM Conference on Computer and Communications Security*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 121–132.

[25] K. Borgolte, C. Kruegel, and G. Vigna, "Delta: automatic identification of unknown web-based infection campaigns," in *ACM Conference on Computer and Communications Security*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 109–120.

[26] M. Javed and V. Paxson, "Detecting stealthy, distributed ssh bruteforcing," in *ACM Conference on Computer and Communications Security*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 85–96.

[27] V. Dave, S. Guha, and Y. Zhang, "Viceroi: catching click-spam in search ad networks," in *ACM Conference on Computer and Communications Security*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 765–776.

[28] S. Shin, V. Yegneswaran, P. A. Porras, and G. Gu, "Avant-guard: scalable and vigilant switch flow management in software-defined networks," in *ACM Conference on Computer and Communications Security*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 413–424.

[29] G. Stringhini, C. Kruegel, and G. Vigna, "Shady paths: leveraging surfing crowds to detect malicious web pages," in *ACM Conference on Computer and Communications Security*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 133–144.

[30] M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park, "Kargus: a highly-scalable software-based intrusion detection system," in *ACM Conference on Computer and Communications Security*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 317–328.

[31] A. Bianchi, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Blacksheep: detecting compromised hosts in homogeneous crowds," in *ACM Conference on Computer and Communications Security*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 341–352.

[32] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing your enemy: understanding and detecting malicious web advertising," in *ACM Conference on Computer and Communications Security*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 674–686.

[33] C.-Y. Hong, F. Yu, and Y. Xie, "Populated ip addresses: classification and applications," in *ACM Conference on Computer and Communications Security*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 329–340.

[34] M. Heiderich, T. Frosch, M. Jensen, and T. Holz, "Crouching tiger - hidden payload: security risks of scalable vectors graphics," in *ACM Conference on Computer and Communications Security*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011, pp. 239–250.

[35] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: feature hashing malware for scalable triage and semantic analysis," in *ACM Conference on Computer and Communications Security*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011, pp. 309–320.

[36] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "Midea: a multi-parallel intrusion detection architecture," in *ACM Conference on Computer and Communications Security*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011, pp. 297–308.

[37] L. Lu, R. Perdisci, and W. Lee, "Surf: detecting and measuring search poisoning," in *ACM Conference on Computer and Communications Security*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011, pp. 467–476.

[38] T. Limmer and F. Dressler, "Dialog-based payload aggregation for intrusion detection," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. ACM, 2010, pp. 708–710.

[39] K. Zhang, Z. Li, R. Wang, X. Wang, and S. Chen, "Sidebuster: automated detection and quantification of side-channel leaks in web application development," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. ACM, 2010, pp. 595–606.

[40] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. A. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: analysis of a botnet takeover," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 635–647.

[41] J. Caballero, P. Poosankam, C. Kreibich, and D. X. Song, "Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 621–634.

[42] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia, "A new cell counter based attack against tor," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 578–589.

[43] M. Q. Ali, H. Khan, A. Sajjad, and S. A. Khayam, "On achieving good operating points on an roc plane using stochastic anomaly score prediction," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 314–323.

[44] Z. Li, S. A. Alrwais, Y. Xie, F. Yu, and X. Wang, "Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2013, pp. 112–126.

[45] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2013, pp. 65–79.

[46] C. Kolbitsch, B. Livshits, B. G. Zorn, and C. Seifert, "Rozzle: Decloaking internet malware," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012, pp. 443–457.

[47] L. Invernizzi and P. M. Comparetti, "Evilseed: A guided approach to finding malicious web pages," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012, pp. 428–442.

[48] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time url spam filtering service," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2011, pp. 447–462.

[49] K. Levchenko, A. Pitsillidis, N. Chachra, B. Enright, M. Félegyházi, C. Grier, T. Halvorson, C. Kanich, C. Kreibich, H. Liu, D. McCoy, N. Weaver, V. Paxson, G. M. Voelker, and S. Savage, "Click trajectories: End-to-end analysis of the spam value chain," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2011, pp. 431–446.

[50] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose, "Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2011, pp. 3–18.

[51] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, "Synthesizing near-optimal malware specifications from suspicious behaviors," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 45–60.

[52] K. Singh, A. Moshchuk, H. J. Wang, and W. Lee, "On the incoherencies in web browser access control policies," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 463–478.

[53] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu, "Investigation of triangular spamming: A stealthy and efficient spamming technique," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 207–222.

[54] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda, "Inspector gadget: Automated extraction of proprietary gadgets from malware binaries," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 29–44.

[55] P. M. Comparetti, G. Salvaneschi, E. Kirda, C. Kolbitsch, C. Kruegel, and S. Zanero, "Identifying dormant functionality in malware programs," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 61–76.

[56] H. Chan and A. Perrig, "Round-efficient broadcast authentication protocols for fixed topology classes," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 257–272.

[57] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to de-anonymize social network users," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 223–238.

[58] A. B. Lewko, A. Sahai, and B. Waters, "Revocation systems with very small private keys," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 273–285.

[59] Y. Liu, P. Ning, and H. Dai, "Authenticating primary users' signals in cognitive radio networks via integrated cryptographic and wireless link signatures," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 286–301.

[60] M. I. Sharif, A. Lanzi, J. T. Giffin, and W. Lee, "Automatic reverse engineering of malware emulators," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009, pp. 94–109.

[61] S. Chen, Z. Mao, Y.-M. Wang, and M. Zhang, "Pretty-bad-proxy: An overlooked adversary in browsers' https deployments," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009, pp. 347–359.

[62] P. M. Comparetti, G. Wondracek, C. Krügel, and E. Kirda, "Prospex: Protocol specification extraction," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009, pp. 110–125.

[63] K. Borders and A. Prakash, "Quantifying information leaks in outbound web traffic," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009, pp. 129–140.

[64] H. Yu, C. Shi, M. Kaminsky, P. B. Gibbons, and F. Xiao, "Dsybil: Optimal sybil-resistance for recommendation systems," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009, pp. 283–298.

[65] M. T. Louw and V. N. Venkatakrishnan, "Blueprint: Robust prevention of cross-site scripting attacks for existing browsers," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009, pp. 331–346.

[66] M. Backes, B. Köpf, and A. Rybalchenko, "Automatic discovery and quantification of information leaks," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009, pp. 141–153.

[67] S. J. Stolfo, A. Stavrou, and C. V. Wright, Eds., *Research in Attacks, Intrusions, and Defenses - 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings*, ser. Lecture Notes in Computer Science, vol. 8145. Springer, 2013.

[68] J. Fritz, C. Leita, and M. Polychronakis, "Server-side code injection attacks: A historical perspective," in *RAID*, ser. Lecture Notes in Computer Science, S. J. Stolfo, A. Stavrou, and C. V. Wright, Eds., vol. 8145. Springer, 2013, pp. 41–61.

[69] M. Z. Rafique and J. Caballero, "Firma: Malware clustering and network signature generation with mixed network behaviors," in *RAID*, ser. Lecture Notes in Computer Science, S. J. Stolfo, A. Stavrou, and C. V. Wright, Eds., vol. 8145. Springer, 2013, pp. 144–163.

[70] C. Wressnegger, F. Boldewin, and K. Rieck, "Deobfuscating embedded malware using probable-plaintext attacks," in *RAID*, ser. Lecture Notes in Computer Science, S. J. Stolfo, A. Stavrou, and C. V. Wright, Eds., vol. 8145. Springer, 2013, pp. 164–183.

[71] M. Akiyama, T. Yagi, K. Aoki, T. Hariu, and Y. Kadobayashi, "Active credential leakage for observing web-based attack cycle," in *RAID*, ser. Lecture Notes in Computer Science, S. J. Stolfo, A. Stavrou, and C. V. Wright, Eds., vol. 8145. Springer, 2013, pp. 223–243.

[72] N. Jiang, Y. Jin, A. Skudlark, and Z.-L. Zhang, "Understanding sms spam in a large cellular network: Characteristics, strategies and defenses," in *RAID*, ser. Lecture Notes in Computer Science, S. J. Stolfo, A. Stavrou, and C. V. Wright, Eds., vol. 8145. Springer, 2013, pp. 328–347.

[73] Y. Nadji, M. Antonakakis, R. Perdisci, and W. Lee, "Connected colors: Unveiling the structure of criminal networks," in *RAID*, ser. Lecture Notes in Computer Science, S. J. Stolfo, A. Stavrou, and C. V. Wright, Eds., vol. 8145. Springer, 2013, pp. 390–410.

[74] D. Hadziosmanovic, L. Simionato, D. Bolzoni, E. Zambon, and S. Etalle, "N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols," in *RAID*, ser. Lecture Notes in Computer Science, D. Balzarotti, S. J. Stolfo, and M. Cova, Eds., vol. 7462. Springer, 2012, pp. 354–373.

[75] B. Amann, R. Sommer, A. Sharma, and S. Hall, "A lone wolf no more: Supporting network intrusion detection with real-time intelligence," in *RAID*, ser. Lecture Notes in Computer Science, D. Balzarotti, S. J. Stolfo, and M. Cova, Eds., vol. 7462. Springer, 2012, pp. 314–333.

[76] J. Chu, Z. Ge, R. Huber, P. Ji, J. Yates, and Y.-C. Yu, "Alert-id: Analyze logs of the network element in real time for intrusion detection," in *RAID*, ser. Lecture Notes in Computer Science, D. Balzarotti, S. J. Stolfo, and M. Cova, Eds., vol. 7462. Springer, 2012, pp. 294–313.

[77] J. Zhang, C. Yang, Z. Xu, and G. Gu, "Poisonamplifier: A guided approach of discovering compromised websites through reversing search poisoning attacks," in *RAID*, ser. Lecture Notes in Computer Science, D. Balzarotti, S. J. Stolfo, and M. Cova, Eds., vol. 7462. Springer, 2012, pp. 230–253.

[78] S. Marchal, J. François, R. State, and T. Engel, "Proactive discovery of phishing related domain names," in *RAID*, ser. Lecture Notes in Computer Science, D. Balzarotti, S. J. Stolfo, and M. Cova, Eds., vol. 7462. Springer, 2012, pp. 190–209.

[79] S. Shin, R. Lin, and G. Gu, "Cross-analysis of botnet victims: New insights and implications," in *RAID*, ser. Lecture Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier, Eds., vol. 6961. Springer, 2011, pp. 242–261.

[80] M. B. Salem and S. J. Stolfo, "Modeling user search behavior for masquerade detection," in *RAID*, ser. Lecture Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier, Eds., vol. 6961. Springer, 2011, pp. 181–200.

[81] M. Heiderich, T. Frosch, and T. Holz, "Iceshield: Detection and mitigation of malicious websites with a frozen dom," in *RAID*, ser. Lecture Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier, Eds., vol. 6961. Springer, 2011, pp. 281–300.

[82] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *RAID*, ser. Lecture

Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier, Eds., vol. 6961. Springer, 2011, pp. 161–180.

[83] N. Boggs, S. Hiremagalore, A. Stavrou, and S. J. Stolfo, "Cross-domain collaborative anomaly detection: So far yet so close," in *RAID*, ser. Lecture Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier, Eds., vol. 6961. Springer, 2011, pp. 142–160.

[84] C.-H. Hsu, C.-Y. Huang, and K.-T. Chen, "Fast-flux bot detection in real time," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 464–483.

[85] S. Mathew, M. Petropoulos, H. Q. Ngo, and S. J. Upadhyaya, "A data-centric approach to insider attack detection in database systems," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 382–401.

[86] A. J. Oliner, A. V. Kulkarni, and A. Aiken, "Community epidemic detection using time-correlated anomalies," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 360–381.

[87] J. Cucurull, M. Asplund, and S. Nadjm-Tehrani, "Anomaly detection and mitigation for disaster area networks," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 339–359.

[88] P. Li, L. Liu, D. Gao, and M. K. Reiter, "On challenges in evaluating malware clustering," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 238–255.

[89] A. Srivastava and J. T. Giffin, "Automatic discovery of parasitic malware," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 97–117.

[90] S. Stafford and J. Li, "Behavior-based worm detectors compared," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 38–57.

[91] M. Antonakakis, D. Dagon, X. Luo, R. Perdisci, W. Lee, and J. Bellmor, "A centralized monitoring infrastructure for improving dns security," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 18–37.

[92] C. V. Wright, C. Connelly, T. Braje, J. C. Rabek, L. M. Rossey, and R. K. Cunningham, "Generating client workloads and high-fidelity network traffic for controllable, repeatable experiments in computer security," in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 218–237.

[93] I. U. Haq, S. Ali, H. Khan, and S. A. Khayam, "What is the impact of p2p traffic on anomaly detection?" in *RAID*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 1–17.

[94] F. Giroire, J. Chandrashekar, N. Taft, E. M. Schooler, and D. Papagiannaki, "Exploiting temporal persistence to detect covert botnet channels," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 326–345.

[95] G. Yan, S. Eidenbenz, and E. Galli, "Sms-watchdog: Profiling social behaviors of sms users for anomaly detection," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 202–223.

[96] J. François, H. J. Abdelnur, R. State, and O. Festor, "Automated behavioral fingerprinting," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 182–201.

[97] P. Li, D. Gao, and M. K. Reiter, "Automatically adapting a trained anomaly detector to software patches," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 142–160.

[98] K. Griffin, S. Schneider, X. Hu, and T. cker Chiueh, "Automatic generation of string signatures for malware detection," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 101–120.

[99] M. Rehák, E. Staab, V. Fusenig, M. Pechoucek, M. Grill, J. Stiborek, K. Bartos, and T. Engel, "Runtime monitoring and dynamic reconfiguration for intrusion detection systems," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 61–80.

[100] G. F. Cretu-Ciocarlie, A. Stavrou, M. E. Locasto, and S. J. Stolfo, "Adaptive anomaly detection via self-calibration and dynamic updating," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 41–60.

[101] L. Liu, G. Yan, X. Zhang, and S. Chen, "Virusmeter: Preventing your cellphone from spies," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 244–264.

[102] D. Bolzoni, S. Etalle, and P. H. Hartel, "Panacea: Automating attack classification for anomaly-based network intrusion detection systems," in *RAID*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds., vol. 5758. Springer, 2009, pp. 1–20.

[103] D. Canali and D. Balzarotti, "Behind the scenes of online attacks: an analysis of exploitation behaviors on the web," in *NDSS*. The Internet Society, 2013.

[104] S. Venkataraman, D. Brumley, S. Sen, and O. Spatscheck, "Automatically inferring the evolution of malicious activity on the internet," in *NDSS*. The Internet Society, 2013.

[105] A. M. White, S. Krishnan, M. Bailey, F. Monrose, and P. A. Porras, "Clear and present data: Opaque traffic and its security implications for the future," in *NDSS*. The Internet Society, 2013.

[106] D. Y. Wang, S. Savage, and G. M. Voelker, "Juice: A longitudinal study of an seo botnet," in *NDSS*. The Internet Society, 2013.

[107] J. Zhang, Y. Xie, F. Yu, D. Soukal, and W. Lee, "Intention and origination: An inside look at large-scale bot queries," in *NDSS*. The Internet Society, 2013.

[108] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in *NDSS*. The Internet Society, 2013.

[109] T. Lauinger, M. Szydlowski, K. Onarlioglu, G. Wondracek, E. Kirda, and C. Krügel, "Clickonomics: Determining the effect of anti-piracy measures for one-click hosting," in *NDSS*. The Internet Society, 2013.

[110] G. Bai, J. Lei, G. Meng, S. S. Venkatraman, P. Saxena, J. Sun, Y. Liu, and J. S. Dong, "Authscan: Automatic extraction of web authentication protocols from implementations," in *NDSS*. The Internet Society, 2013.

[111] J. Zhang and G. Gu, "Neighborwatcher: A content-agnostic comment spam inference system," in *NDSS*. The Internet Society, 2013.

[112] C. Lever, M. Antonakakis, B. Reaves, P. Traynor, and W. Lee, "The core of the matter: Analyzing malicious traffic in cellular carriers," in *NDSS*. The Internet Society, 2013.

[113] *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[114] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi, "Host fingerprinting and tracking on the web: Privacy and security implications," in *NDSS*. The Internet Society, 2012.

[115] S. Lee and J. Kim, "Warningbird: Detecting suspicious urls in twitter stream," in *NDSS*. The Internet Society, 2012.

[116] M. Schuchard, A. Mohaisen, D. F. Kune, N. Hopper, Y. Kim, and E. Y. Vasserman, "Losing control of the internet: Using the data plane to attack the control plane," in *NDSS*. The Internet Society, 2011.

[117] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis," in *NDSS*. The Internet Society, 2011.

[118] M. Balduzzi, C. T. Gimenez, D. Balzarotti, and E. Kirda, "Automated discovery of parameter pollution vulnerabilities in web applications," in *NDSS*. The Internet Society, 2011.

[119] S. E. Coull, F. Monrose, and M. Bailey, "On measuring the similarity of network hosts: Pitfalls, new metrics, and empirical analyses," in *NDSS*. The Internet Society, 2011.

[120] A. Houmansadr and N. Borisov, "Swirl: A scalable watermark to detect correlated network flows," in *NDSS*. The Internet Society, 2011.

[121] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. M. Voelker,

V. Paxson, N. Weaver, and S. Savage, "Botnet judo: Fighting spam with itself," in *NDSS*. The Internet Society, 2010.

[122] S. Sinha, M. Bailey, and F. Jahanian, "Improving spam blacklisting through dynamic thresholding and speculative aggregation," in *NDSS*. The Internet Society, 2010.

[123] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu, "On network-level clusters for spam detection," in *NDSS*. The Internet Society, 2010.

[124] W. K. Robertson, F. Maggi, C. Kruegel, and G. Vigna, "Effective anomaly detection with scarce training data," in *NDSS*. The Internet Society, 2010.

[125] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *NDSS*. The Internet Society, 2009.

[126] A. Houmansadr, N. Kiyavash, and N. Borisov, "Rainbow: A robust and invisible non-blind watermark for network flows," in *NDSS*. The Internet Society, 2009.

[127] X. Hu, M. Knysz, and K. G. Shin, "Rb-seeker: Auto-detection of redirection botnets," in *NDSS*. The Internet Society, 2009.

[128] C. Gates, "Coordinated scan detection," in *NDSS*. The Internet Society, 2009.

[129] Y. Song, A. D. Keromytis, and S. J. Stolfo, "Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic," in *NDSS*. The Internet Society, 2009.

[130] M. Allman and V. Paxson, "Issues and etiquette concerning use of shared measurement data," in *Internet Measurement Conference*, C. Dovrolis and M. Roughan, Eds. ACM, 2007, pp. 135–140.

[131] J. Sonchack, A. J. Aviv, and J. M. Smith, "Bridging the data gap: Data related challenges in evaluating large scale collaborative security systems," in *Presented as part of the 6th Workshop on Cyber Security Experimentation and Test*. Berkeley, CA: USENIX, 2013. [Online]. Available: https://www.usenix.org/conference/cset13/workshop-program/presentation/Sonchack

[132] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, ser. BADGERS '11. New York, NY, USA: ACM, 2011, pp. 29–36. [Online]. Available: http://doi.acm.org/10.1145/1978672.1978676

[133] S. Abt and H. Baier, "A darknet-driven approach to compilation of hostile network traffic samples," in *Proceedings of Sicherheit in vernetzten Systemen: 20. DFN-Workshop*, C. Paulsen, Ed. DFN-Cert Services GmbH, 2013.

[134] C. Rossow, C. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. van Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Security and Privacy (SP), 2012 IEEE Symposium on*, 2012, pp. 65–79.

[135] J. Heidemann and C. Papdopoulos, "Uses and challenges for network datasets," in *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, March 2009, pp. 73–82.

[136] E. Kenneally and K. Claffy, "Dialing privacy and utility: A proposed data-sharing framework to advance internet research," *Security Privacy, IEEE*, vol. 8, no. 4, pp. 31–39, July 2010.

[137] P. Porras and V. Shmatikov, "Large-scale collection and sanitization of network security data: Risks and challenges," in *Proceedings of the 2006 Workshop on New Security Paradigms*, ser. NSPW '06. New York, NY, USA: ACM, 2007, pp. 57–64.

[138] D. Frankowski, D. Cosley, S. Sen, L. Terveen, and J. Riedl, "You are what you say: Privacy risks of public mentions," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '06. New York, NY, USA: ACM, 2006, pp. 565–572.

[139] B. Sangster, T. O'Connor, T. Cook, R. Fanelli, E. Dean, W. J. Adams, C. Morrell, and G. Conti, "Toward instrumenting network warfare competitions to generate labeled datasets," in *Proc. of the 2nd Workshop on Cyber Security Experimentation and Test (CSET'09)*, 2009.

[140] M. Zimmer, ""but the data is already public": On the ethics of research in facebook," *Ethics and Information Technology*, vol. 12, no. 4, pp. 313–325, 2010.

[141] D. Balzarotti, S. J. Stolfo, and M. Cova, Eds., *Research in Attacks, Intrusions, and Defenses - 15th International Symposium, RAID 2012, Amsterdam, The Netherlands, September 12-14, 2012. Proceedings*, ser. Lecture Notes in Computer Science, vol. 7462. Springer, 2012.

[142] R. Sommer, D. Balzarotti, and G. Maier, Eds., *Recent Advances in Intrusion Detection - 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011. Proceedings*, ser. Lecture Notes in Computer Science, vol. 6961. Springer, 2011.

[143] S. Jha, R. Sommer, and C. Kreibich, Eds., *Recent Advances in Intrusion Detection, 13th International Symposium, RAID 2010, Ottawa, Ontario, Canada, September 15-17, 2010. Proceedings*, ser. Lecture Notes in Computer Science, vol. 6307. Springer, 2010.

[144] E. Kirda, S. Jha, and D. Balzarotti, Eds., *Recent Advances in Intrusion Detection, 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5758. Springer, 2009.

[145] *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013.

[146] *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011.

[147] *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010.

[148] *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*. The Internet Society, 2009.

[149] *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013.

[150] *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012.

[151] *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. IEEE Computer Society, 2011.

[152] *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*. IEEE Computer Society, 2010.

[153] *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*. IEEE Computer Society, 2009.

[154] A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds., *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. ACM, 2013.

[155] T. Yu, G. Danezis, and V. D. Gligor, Eds., *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. ACM, 2012.

[156] Y. Chen, G. Danezis, and V. Shmatikov, Eds., *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*. ACM, 2011.

[157] E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds., *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. ACM, 2010.

[158] E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds., *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*. ACM, 2009.

EyeBit: Eye-Tracking Approach for Enforcing Phishing Prevention Habits

Daisuke Miyamoto*, Takuji Iimura*, Gregory Blanc†, Hajime Tazaki*, and Youki Kadobayashi‡

*Information Technology Center
The University of Tokyo
2-11-16 Yayoi, Bunkyo-ku, Tokyo, 113-8658 JAPAN
{daisu-mi, iimura, tazaki}@nc.u-tokyo.ac.jp

†Institut Mines-Télécom/Télécom SudParis
CNRS UMR 5157 SAMOVAR
9 rue Charles Fourier, 91011 Évry, FRANCE
gregory.blanc@telecom-sudparis.eu

†Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0192 JAPAN
youki-k@is.aist-nara.ac.jp

***Abstract*—This paper proposes a cognitive method with the goal to get end users into the habit of checking the address bar of the web browser. Earlier surveys of end user behavior emphasized that users become victims to phishing due to the lack of knowledge about the structure of URLs, domain names, and security information. Therefore, there exist many approaches to improve the knowledge of end users. However, the knowledge gained will not be applied unless end users are aware of the importance and develop a habit to check the browser's address bar for the URL structure and relevant security information.**

We assume that the habit of checking the bar will improve educational effect, user awareness of secure information, and detection accuracy even in the case of sophisticated phishing attacks. To assess this assumption, this paper conducts a participant-based experiment where 23 participants' eye movement records are analyzed, and observes that novices do not tend to have the said habit. We then consider a way for them to acquire these habits, and develop a system which requires them to look at the address bar before entering some information into web input forms. Our prototype named EyeBit is developed as a browser extension, which interacts with an eye-tracking device to check if the user looks at the browser's address bar. The system deactivates all input forms of the websites, and reactivates them only if the user has looked at the bar. This paper shows the preliminary results of our participant-based experiments, and discusses the effectiveness of our proposal, while considering the potential inconvenience caused by EyeBit.

***Keywords*—Phishing, Cognitive Psychology, Eye-Tracking**

I. Introduction

Phishing is a fraudulent activity defined as the acquisition of personal information by tricking an individual into believing the attacker is a trustworthy entity [1]. Phishing attackers lure people through the use of a phishing email, as if it were sent by a legitimate corporation. The attackers also attract the email recipients to a phishing site, which is the replica of an existing web page, to fool them into submitting personal, financial, and/or password data.

There have been many participants-based studies to understand decision patterns of end users while the fundamental problem in phishing is the fact that they are deceived. According to Dhamija [2], some participants do not look at browser-based information such as the address bar, the status bar or the security indicators, leading to incorrect choices 40% of the time. Instead, they consider various other criteria while assessing a website's credibility.

In our previous work [3], we asked 309 participants the reason of their decision. The participants browsed 14 simulated phishing sites and six legitimate sites, judging whether or not the site appeared to be a phishing site, and answered the reason for their decision via a questionnaire. The results showed that experts tended to evaluate a site's URL and/or browser's SSL indicator rather than the contents of a web page to judge the credibility of the sites. Conversely, novices, who often failed to decide rightly, received strong signals from web contents only while. Due to the nature of phishing, the web contents are quite similar to what is displayed by a legitimate site, leading novices to fall victims to the phishing trap.

It can be naturally assumed that checking the browser's address bar is beneficial for end users to be aware of phishing. The reader should note that modern web browsers do show the website's URL and security information in the address bar. Even the knowledge of URL and security are also important for phishing prevention and are the strong motivation for seeing there, the both of them could not work before the users did not see the bar. This paper assesses this assumption with a participant-based experiment in which 23 participants are shown with 20 websites, and asked to determine which ones are phishing while having their eye movement monitored. Based on our experiment, it might be reasonable to consider that novices do not have the habit of visually checking the address bar.

According to these results, this paper then explores new mechanisms for the users to acquire the habits of checking the address bar while assessing the credibility of a website. Our idea is to enforce them to look there first, before entering any information to web input forms. Our proposed system, named *EyeBit*, is implemented as a browser's extension, and interacts with an eye-tracking device. EyeBit deactivates all input forms at the beginning of browsing, and activates the forms when it confirmed that the user gazed the address bar. For our preliminary evaluation of EyeBit, we called ten participants to test if they got a habit in checking the address bar.

To this end, this paper makes the following contributions:

- We present an approach to counter phishing tactics,

that we argue significant benefit for getting into the habit of secure web browsing.

- We propose EyeBit which forms the habit of checking the address bar for safe browsing in section III-A. To the best of our knowledge, this is the first attempt at making end users to acquire habits for phishing prevention.
- We assess our assumption (*"checking the browser's address bar is beneficial to end users in making them aware of phishing"*) through a participant-based experiment in section III-B.
- We design EyeBit in consideration of cognitive aspects, and then implement a prototype of EyeBit as an extension for Chrome web browser in section III-C.
- We evaluate the effectiveness of seeing the address bar with an eye-tracking camera in a within-subject experiment. The implementation is demonstrated to show the effectiveness of getting the habit in section III-D.
- We observe that the inconvenience caused by EyeBit is negligible in section IV-A.

II. Related work

A. Behavior of end users

The targets of phishing attacks are end users, so there were various contributions to analyze end users and their activities. According to an analysis report of 2,684 people by Fogg et al. [4], 46.1% checked the design look and feel of a website and 28.5 % used website structure of information, when people assessed a real web site's credibility. Ye et al. [5] also stated that end users would convinced by the content of HTML and URL, regardless of checking SSL padlock icons.

According to Kumaraguru et al. [6], there were the difference in the model for making trust decision between novices and experts. In comparison to experts, novices were sensitive to superficial signals when they made trust decision. Novices also ignored some signals such as SSL, address bar, and so on, where experts received these signals. Dhamija et al. [2] reported their participant within tests for identifying phishing sites. They found that phishing caused of lack of knowledge. For example, participants thought www.ebay-members-security.com belongs to www.ebay.com due to the lack of system knowledge. Also, many participants did not understand security indicators. They did not know that a closed padlock icon in the browser indicates that the page they are viewing was delivered securely by SSL. Even if they understand the meaning of that icon, users can be fooled by its placement within the body of a web page. They also found that the best phishing websites fooled 90% of participants. The URL of the site is "www.bankofthevvest.com", with two "v"s instead of a "w" in the domain name.

Wu et al. also measured the effectiveness of security toolbars, which informs end users that they are visiting phishing sites [7]. They tested three types of security toolbars, namely, Neutral-information toolbar such as NetCraft Toolbar [8], SSL-Verification toolbar such as TrustBar [9], System-Decision toolbar such as SpoofGuard [10]. Each of the three security toolbars was tested with ten participants, and they browsed both phishing sites and legitimate sites with one security toolbar, and they also classified the site was phishing or not. Wu et al. concluded that all toolbars failed to prevent users from being spoofed by high-quality phishing attacks. Users failed to continuously check the browser's security indicators, since maintaining security was not the user's primary goal. Although users sometimes noticed suspicious signs coming from the indicators, they either did not know how to interpret the signs or they explained them away.

B. Failure analysis

The root cause of social engineering is human errors; the targets failed to behave or understand against attacks. Failure analysis is the process of investigating the reason of failure. Its process also collects and analyzes data, and develops methods and/or algorithms to eliminate the root causes of the failure. Zahran et al. [11] summarized the categorization techniques for such analysis and introduced the component-based categorization; the failure can be caused of the components of information systems, namely, hardware, software, communications networks, people, data resources and organization. The analysis of human error is also important part in cyber security. Especially, the interface studies investigated the reasons of users' misjudgments [12]. Based on their subjects experiments, they clarified the mental model of users and indicated the way for improving the user interfaces.

In the context of the people in enterprise, human factors were analyzed to mitigate risks in the organization. According to Hawkey et al. [13], [14], challenges of IT security managements were classified into technical, organizational, and human factors. To understand human behavioral model, Parkin et al. [15] showed five behavioral foundation, namely cultural, ethical, temporal, mindset, and capability difference. Based on the foundation, they developed ontology which aims at maintaining compliance with ISO27002 standard [16] while considering the security behaviors of individuals within the organization.

Alfawaz et al. [17] classified the characteristics of organizational subjects involved in these information security practices. They analyzed the participants' activities and categorized individual security behaviors into four modes, (i) Knowing-Doing mode, (ii) Knowing-Not doing mode, (iii) Not knowing-Doing mode and (iv) Not knowing-Not doing mode. Term "Knowing" means that the participants know the organization's requirements for information security of behavior and have security knowledge. "Doing" also means that they are doing the right behavior. The cases of (i) and (iv) said that the participants (do not) know the requirements and (do not) have the knowledge, therefore, they are (not) doing the right behavior. The example of the mode (ii) is that the participant is unaware of the requirements, but asks someone before taking certain actions. The mode (iii) is serious, that the participants do not perform the right behavior even they know the requirements. The root causes of the mode (iii) is regarded as stressful events. Basically, people have a limited capacity for information processing and routinely multitasks [18]. They tends to conserve mental resources; full attention is for few tasks and decisions.

The earlier researches can be summarized that understanding both the personal knowledge and his/her internal mental

processes are necessary for thwarting the impact of human error. There are many approaches to reduce the human errors, and this paper focuses on habits of security. Trustworthy computing habits can maximize opportunities that the knowledge works efficiently. It must be noted that the habitual action is often performed under unconscious. Regardless of the stress, habits have possibilities to improve the chance to exert the knowledge for end users.

C. Cognitive analysis

Cognitive psychology is the study of relationship between internal mental processes and observable behavior. To address the problem in the observation, this paper refers to the evaluation of cognitive methods for supporting operators studied by Groojten [19] in which the following criteria were formulated.

- **Sensitiveness to workload changes.**
  We need to employ the behavioral observation methods that can estimate the internal mental model. The methods might also leverage the collected information regardless of the Fear of Negative Evaluation (FNE) [20]; observations are often affected by FNE, in which some of people will conceal their human errors. In fact, disclosing mistakes often damage their own self-image and professional standing.
- **Obtrusiveness for the operator.**
  The observation should not take much effort to start collecting data or disturb the handling of people during the tasks performance. Furthermore, people will not carry implants or needles or other devices which may hurt them in any way.
- **Availability of equipment.**
  The observation should employ the method which is easily applicable to people. Within the context of phishing prevention, the methods should be available while users are browsing. Non-contact devices might be preferred.

Based on the requirements, this paper explores the suitable methods. Brain activity [21], heart measure, and blood pressure [22] are feasible due to the sensitivity to workload changes, but they tend to require much obtrusiveness for people. Contrastively, Facial expression [23] and Gesture recognition [24] were often affected by FNE.

We speculate that the following research domains that might be helpful for the observation.

- **Eye Movements.**
  Research on experimental psychology has evidenced a strong link between eye movements and mental disorders [25], [26]. Leigh et al. [27] classified the eye movements into four categories, namely Saccades, Fixations, Smooth pursuit movements, and Vestibulo-ocular reflexes. In the context of mental model, Irwin et al. showed that the mental rotation is suppressed during the movements [28], and Tokuda [29] showed that mental workload, the indicator of how mentally/cognitively busy a person is, can be estimated from saccadic intrusions. In addition to that, recent eye-tracking devices also support non-mounting monitoring as well as head-mount monitoring.
- **Facial Skin Temperature.**
  Variation of facial skin temperature has received some attention as a physiological measure of mental status [30]–[32]. According to Genno et al. [33], their experiments showed that temperature change in nose area when subjects experienced sensations like stress and fatigue. Furthermore, the thermography, when combined with other modes of measurement provides a highly automated and flexible means to objectively evaluate workload [30].

In this paper, we decided to employ eye movements-based observations by following reasons. At first, our motivation is to let end users to get the habits of investigating the browser's address bar; an eye-tracking is a straight forward way for observing users' behavior. The second is that monitoring eye movement will not significantly penalize users' convenience according to the above consideration. We also expect the eye-tracking for recognizing mental anomalies to reduce impact of human failure.

III. EyeBit: eye-tracking for phishing prevention

This section introduces EyeBit, a system for end users to get into the habit of checking the surrounding area of the browser's address bar while assessing a website's credibility. Section III-A summarizes the overview, and Section III-B assesses our assumption, that is, checking the browser's address bar is beneficial for end users. Section III-C presents the design and implementation of EyeBit , which is evaluated in Section III-D.

A. Overview

In this paper, we speculate that the habit of checking the address bar plays an important role in safe browsing. The key idea is to require end users to look at the browser's address bar before entering anything into the web input forms.

According to Dhamija [2] who studied the reason why novices fall victims to phishing, phishing is often successful when there is a lack of knowledge about domain names (in order to differentiate between URLs), about security information, or lack of attention to this information. However, modern social engineering attacks attempt at affecting victims' composure. For example, a phishing email states *"your account was locked because you violated the terms of service"* which will prompt the victim to immediately click an URL placed below and presented as a way of recovery. From a psychological aspect, the victims' primary concern is about their locked account, and not security, leading the authors to invalidate security education as not sufficient, in that case, to prevent phishing.

To improve the acquisition of security education and knowledge, the habit of looking at the bar might be reasonable. The advantage is that this habitual action is often performed unconsciously. Even if the primary concern of the end user is not security, the habit would work like a conditioned reflex action. The habit also improves the chance of being aware of security information. Since modern web browsers show the website's URL and related security information in the address bar, the surrounding area of the browser's address bar shows good signals for phishing detection.

TABLE I: Conditions of each site used for recording eye movement

#	Website	Phish	Lang	Description
1	Google	no	JP	SSL
2	Amazon	yes	JP	tigratami.com.br, once reported as a compromised host
3	Sumishin Net Bank	no	JP	EV-SSL
4	Yahoo	yes	JP	kazuki-j.com, once reported as a compromised host
5	Square Enix	yes	JP	secure.square-enlix.com, similar to legitimate URL secure.square-enix.com
6	Ameba	no	JP	non-SSL
7	Tokyo Mitsubishi UFJ Bank	yes	JP	bk.mufg.jp.iki.cn.com, similar to legitimate URL bk.mufg.jp
8	All Nippon Airways	yes	JP	IP address
9	Gree	no	JP	non-SSL
10	eBay	no	EN	EV-SSL
11	Japan Post Holdings	yes	JP	direct.yucho.org, SSL
12	Apple	yes	EN	apple.com.uk.sign.in...
13	DMM	no	JP	SSL
14	Twitter	yes	JP	twittelr.com
15	Facebook	yes	JP	IP address
16	Rakuten Bank	yes	JP	vrsimulations.com, once reported as a compromised host
17	Sumitomo Mitsui Card	yes	JP	www.smcb-card.com, SSL
18	Jetstar Airways	no	JP	SSL, non pad-lock icon by accessing non-SSL content
19	PayPal	yes	EN	paypal.com.0.security-c...
20	Tokyo-Tomin Bank	no	JP	3rd party URL www2.answer.or.jp, EV-SSL

We therefore develop EyeBit, a system for enforcing phishing prevention habits. Based on eye-tracking technologies, EyeBit monitors if users see a particular portion of the screen. Failing to look at the address bar will deactivate parts of web contents in which users can input their personal information.

B. Assumption

In this section, we want to examine the assumption that checking the browser's address bar is beneficial to end users in making them aware of phishing. In order to assess if gazing the address bar improves the accuracy, we performed a participant-based experiment to monitor an end-user's eye activity. It must be noted that our experiments must not collect and/or analyze personally identifiable information. The experimental design, concept and methodologies for recruiting participants are also explained below.

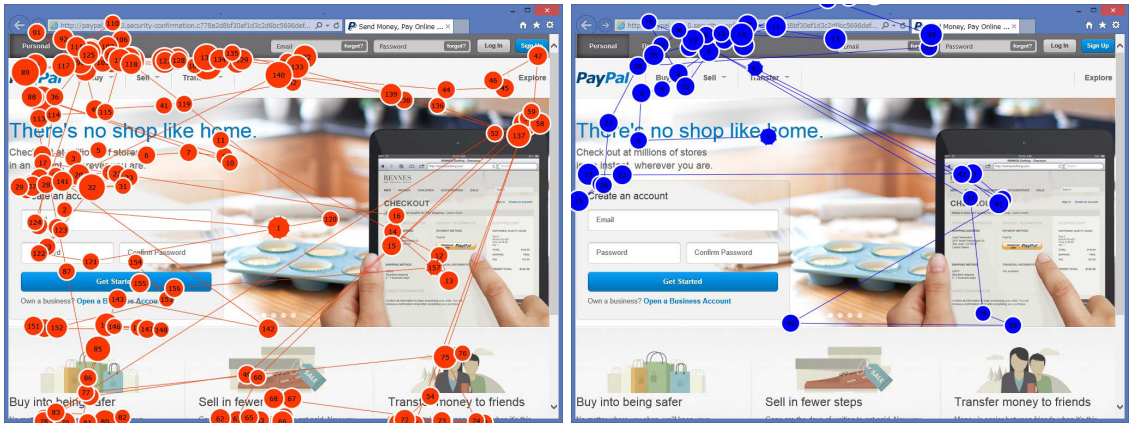
1) Recruitment of participants through a poster advertisement at a college campus.
2) Explanation of our experiment to the participant.
   - Our purpose is to observe the user's activity, in particular with respects to assessing the credibility of websites.
   - Our goal is to develop security mechanisms for protecting users from phishing.
   - Before the experiments, each participant will be asked his/her age and sex.
   - During the experiments, each participant will be monitored by an eye-tracking device, and be shown 20 websites. Their activity will be monitored, and they will be asked if each website seems to be phishing or not. They will also be asked the reason of their decision.
   - Collected data consists of the participants' age, sex, decision result, decision criteria, and eye-tracking data.
   - Collected data is shared with both European and Japanese research members.
3) Display of 20 website screenshots, including legitimate websites and pseudo phishing sites.
   In the experiment, the phishing sites are not real phishing sites to avoid information leakage. Instead, our participants are presented with 20 screenshots of a browser that rendered the websites. These screenshots were taken on Windows 7 equipped with IE 10.0.

As shown in Table I, we prepared twelve phishing sites and eight legitimate ones for the test. In comparison, a typical phishing IQ test [2] presented participants with 13 phishing sites and seven legitimate ones, so the ratio of phishing sites over legitimate ones is quite similar to ours.

In this experiment, we recruited 23 participants to observe their eye movement. The volunteers were mainly males in their twenties. With their consent, their eye movements were recorded by our prepared eye-tracking device, Tobii TX300 [34]. It needed calibration procedure for each participant.

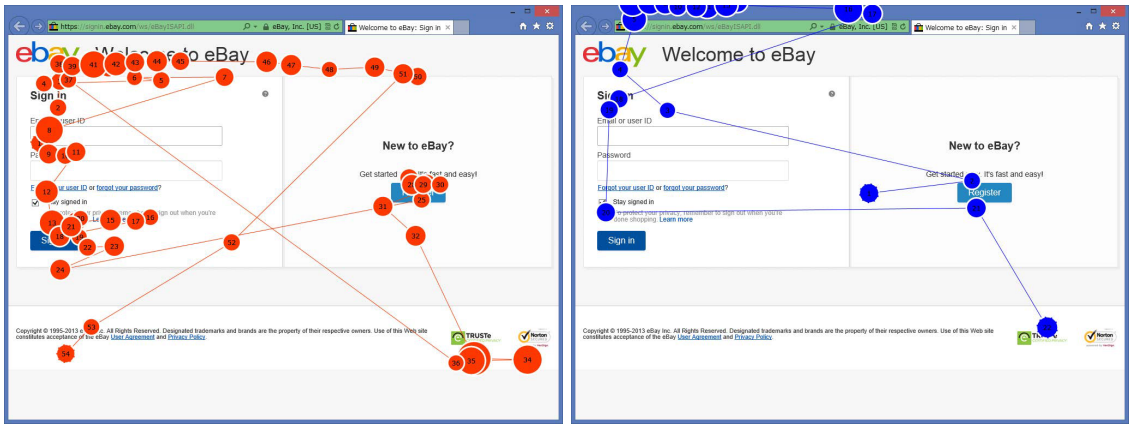
We observed that the participants who rely on the URL of the website would fail to flag websites 5, 14 and 17, since these sites had almost the same URL as the legitimate sites, except for one letter. The URLs of the websites 7, 12, and 19 contained a legitimate-sounding domain name. Website 20 was legitimate but the domain name of this site had no indication of its brand names. For participants who tended to rely on security information of browsers, websites 11 and 20 might be difficult to assess because although they were phishing sites, they presented participants with a valid SSL certificate. Conversely, websites 6 and 9 were legitimate but did not employ valid SSL certificates though they required users to login. Of course, since our prepared phishing websites were lookalikes of the legitimate ones, it might have been more difficult for the participants who relied on web contents.

Fig. 1 and 2 show typical eye-movement records on both phishing and legitimate website, for a novice and an expert respectively. Circles denote fixations, and the numbers in the circles denote the order of the fixation. In the phishing case, the novice looked at the web content but ignored the browser's address bar while assessing credibility, as shown in Fig. 1a. Since the text and visual in phishing sites are quite similar to the ones in legitimate sites, he failed to label the phishing site



(a) Novice (b) Expert

Fig. 1: Eye-tracking in phishing website



(a) Novice (b) Expert

Fig. 2: Eye-tracking in legitimate website

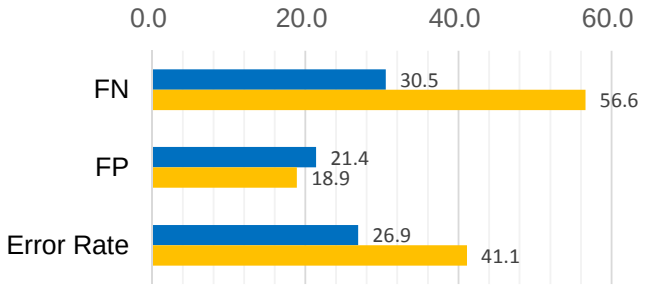


Fig. 3: The average false positive, false negative and error rate for users that looked (blue) and did not look (orange) at the address bar

correctly. In the legitimate case, he also only paid attention to the web content as shown in Fig. 2a. In contrast, an expert tends to evaluate the site's URL and/or the browser's SSL indicator rather than the contents of the web page to judge the credibility of the sites, as shown in Fig. 1b and 2b.

We then analyzed the detection accuracy of participants who looked at the address bar and those who did not look, respectively. The results were shown in Fig. 3, where he blue bar denotes the average rates for the participants looked at the address bar of the browser, and the orange bar denotes that for the participants did not look at the bar.

Out of the 331 times the bar was gazed, 89 (26.9%) misjudgments were observed. In the phishing websites case, the participants looked at the bar 200 times in total, which occurred 61 (30.5%) false negatives, i.e., labeling phishing as legitimate. In the legitimate websites case, they looked at the bar 131 times in total, which occurred 28 (21.4%) false positives, i.e., labeling legitimate as phishing. In contrast, the average error rate was 41.1% (53 out of 129), the false negative rate was 56.6% (43 out of 76), and the false positive rate was 18.9% (10 out of 53), when participants would ignore the address bar. The average error rate and false negative rate indeed decreased when the address bar was checked, although experimental errors might have occurred due to some possible offsets caused by the eye-tracking calibration procedure. The increase of the false positive rate seems to be marginal. We

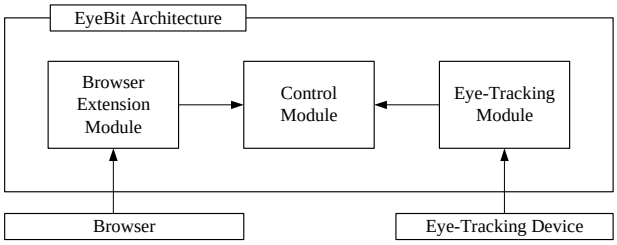


Fig. 4: The architecture of EyeBit

TABLE III: Decision results of participants

#	A_1	A_2	A_3	A_4	A_5	B_1	B_2	B_3	B_4	B_5
1	F				F		F	F		
2										
3	F				F		F	F	F	
4	F		F					F		
5									F	
6										
7			F							
8										
9					F	F				
10		F	F							
11	F	F	F					F		
12										

therefore considered that our assumption, i.e., *checking the browser's address bar is beneficial to end users in making them aware of phishing*, is reasonable.

C. Design and implementation

Based on the assumption described in Section III-B, we implemented EyeBit, a system which enables novices to get into the habit of checking the address bar. The requirements of EyeBit are as follows.

- **Web inputs control.**
  It must have functions to activate/deactivate web input forms. EyeBit deactivates all input forms, at first. When it detects that the user has checked the browser's address bar, all input forms are then activated.
- **Eye-tracking capabilities.**
  It must interact with eye-tracking devices, and identify that the user has looked at a particular portion in the web browser with certainty. It also should provide interfaces to obtain an end user's eye position from third-party developed application.
- **Address bar localization.**
  It should be able to locate the address bar within the screen.

The architecture of EyeBit is shown in Fig. 4. It consists of (i) an eye-tracking module, (ii) a browser extension module, and (iii) a control module. In order to meet the requirements, we implemented EyeBit as a browser extension. The module deactivates all input forms at first, and then activates them after the eye-tracking module has confirmed that the user looked at the address bar. The task of the eye-tracking module is to interact with an eye-tracking device. We selected an eye-tracking camera which could provide an interface to obtain an end user's eye position from our implementation.

Our prototype was implemented as an extension of Google Chrome, therefore written in JavaScript, and consisted of roughly 100 lines of code. We also selected Eye-Tribe-Tracker [35] as the eye-tracking device. Its software development kit (SDK) embeds the function of web server and provides the user's eye position in JavaScript Object Notation (JSON) format messages.

Due to the performance difference, this device could not correctly deal with eye-fixation, however, our implementation checked if the user looked at the area of the address bar and 50 pixels of margins on each side. It stores the 30 seconds of eye-tracking records, and inspected his/her gaze position in one second intervals, and reactivated the forms when the position of the gaze was in the area for at least one time interval.

The limitation of our prototype was the localization of the address bar. Instead, it measured the absolute position within the screen. Assuming the browser's window is maximized, the position of the bar can be easily estimated. We will discuss about methods for locating the bar in Section IV-D.

D. Evaluation

This section evaluates the effectiveness of EyeBit. As our pilot study, in May 2014, we invited ten participants and performed a within-subject experiment. The participants browsed six emulated phishing sites and the same number of legitimate sites, as listed in Table II. They checked whether the site appeared to be a phishing site or not. Among the ten participants, nine belonged to Nara Institute of Science and Technology, and the rest belonged to the University of Tokyo. All of them were male, two of them completed their M.Eng degree in the last five years, while the remaining were master's degree students. We explained to them our purpose, goal, and usage of the collected data as stated in Section III-B.

The experiment is composed of the following steps.

1) **First stage: labeling websites 1–4**
   All participants were shown four websites 1 – 4 in Table II and checked whether the sites were phishing or not. When the participant deemed the site legitimate, he/she would input the word "john" as a pseudo persona for the website's username input field.
2) **Educational break**
   Before employing EyeBit, we would explain about what the address bar indicates. The participants would be shown with our educational material. With reference to typical material [36], we convinced them to carefully check the website's URL, the presence of an SSL padlock icon, and the EV-SSL information.
3) **Second stage: labeling websites 5–8**
   After the educational break, five of the ten participants would equip with EyeBit and be explained about EyeBit; input forms would be deactivated until the browser's address bar was gazed upon. The rest of the participants were not equipped with EyeBit. This differentiation was made to comparatively study the effectiveness of EyeBit. All participants were shown four websites 5 – 8 in Table II and checked whether the sites were phishing or not. Similarly to the first stage, the participants inputted the

TABLE II: Conditions of each site used for evaluating EyeBit

#	Website	Phish	Lang	Description
1	Yahoo	yes	JP	`dmiurdrgs.cher-ish.net`, once reported as a phishing site
2	PayPal	no	EN	EV-SSL
3	eBay	yes	EN	`signin-ebay.com`, similar to legitimate URL `signin.ebay.com`
4	DMM	no	JP	SSL
5	Amazon	yes	EN	`www.importen.se`, once reported as a phishing site
6	Bank of America	no	EN	EV-SSL
7	Facebook	no	JP	SSL
8	Square Enix	yes	JP	`hiroba.dqx.jp...`, similar to legitimate URL `hiroba.dqx.jp`
9	Twitter	yes	JP	`twittelr.com`
10	Google	no	JP	SSL
11	Battle.net	no	EN	EV-SSL
12	Sumitomo Mitsui Card	yes	JP	`www.smcb.card.com....`, similar to legitimate URL `www.smbc-card.com`

word "john" to the website's input form when deeming a site legitimate. The five participants equipped with EyeBit would need to activate the forms by checking the address bar before labeling a website as legitimate.

4) **Interval for sanitizing**
   Basically, people tend to be sensitive to phishing after the education. We wait one hour for interval between the second and last stage.
5) **Last stage: labeling websites 9–12**
   Finally, we let all participants show the last four websites 9 – 12 in Table II. We intended to analyze the behavioral differences between five participants who used EyeBit and the rest participants who did not use EyeBit. We also planned to observe remaining effect of education, therefore, all participants did not equip EyeBit.

The detection results were shown in Table III, where $A_1 \cdots A_5$ denote the participant who used EyeBit, $B_1 \cdots B_5$ denote the participant who did not use EyeBit, the letter "F" denotes that a participant failed to judge the website, and the empty block denotes that a participant succeeded in judging correctly.

We assumed that participants A_1 and A_5 were novices. Since they had no criteria for making decisions, they often received strong signals from web content and hence, they answered all of websites in the first stage as legitimate. After the education, they were seemed to have criteria, so they could perfectly answered to the websites 5 – 8. During one hour, the effect would not be significantly attenuated. In the case of websites 9 – 12, they saw the browser' address bar at least ten seconds, even if they did not equip EyeBit.

The participants $B_1 \cdots B_5$ did not employ EyeBit, and we speculated that the participant B_2 and B_3 were also novices; they could not correctly identify phishing websites in the first stage. However, their eye movement were formed to see the address bar after the education.

There were some reasons that EyeBit worsened in the participants $A_1 \cdots A_5$ compared to when it was not used $B_1 \cdots B_5$. We assumed that the most predominant reason was the small sample size. In the cases of the participant A_2 and A_3, EyeBit had a potential for making them paranoid, since websites 9 and 10 were legitimate but were labeled as phishing. The another reason was that the educational effect did not dissolve in one hour, and hence, the participants $B_1 \cdots B_5$ performed better.

We then conducted a follow-up study in June 2014. The study was focused on four novices, namely the participants A_1, A_5, B_2, and B_3, and we observed the difference of the educational effect remains in four participant after one month. The participants were shown websites 1 to 20 in the table II. We observed that the participants A_1, A_5, and B_2 often looked the browser's address bar, although the participant B_3 did not. In regard to a difference from the pilot study in May 2014, the participants A_5 and B_2 could judge correctly the websites 1 – 4. Through the follow-up study, no false negative error was observed in the case of the participant A_1 since he often looked at the bar. In the case of the participant B_3, the false negatives increased in comparison of the pilot study. In particular, he answered that the websites 1 – 4 seemed to be legitimate, as same as the first stage of the pilot study.

Due to the small number of the participants, it is difficult to accurately determine that EyeBit could exert the educational materials in long time period. However, based on the observations of the pilot and follow-up study, we assumed that EyeBit is helpful for getting the habit of seeing address bar while making trust decisions.

IV. Discussion

A. Potential inconvenience caused by EyeBit

The primary motivation for our experiment was to assess the effectiveness of EyeBit in influencing users' behavior to check the address bar. Essentially, it must be investigated the address bar to correctly judge, therefore, time increase for seeing the address bar must be acceptable.

However, the significant time increase might penalize users' convenience. In general, there is a tradeoff between usability and security, and hence EyeBit would penalize the user's experience to the benefit of security. There are various methodologies for estimating the convenience, and here we tentatively employed the overhead of time spent on the site as a measure.

Fig. 5 shows the average time for making decision, where x axis denotes the number of seconds, y axis denotes each participant, Average(A) is the average time of $A_1 \cdots A_5$ and Average(B) is the average time of $B_1 \cdots B_5$. The blue bar denotes the average time for the first stage (websites 1 – 3), the orange bar denotes the second stage (websites 5 – 7), and the gray bar denotes the last stage (websites 9 – 11). Note that we could not obtain the time on the last websites in each stage due to the limitation of our experiment system, the time spent in the website 4, 8, and 12 were not measured.

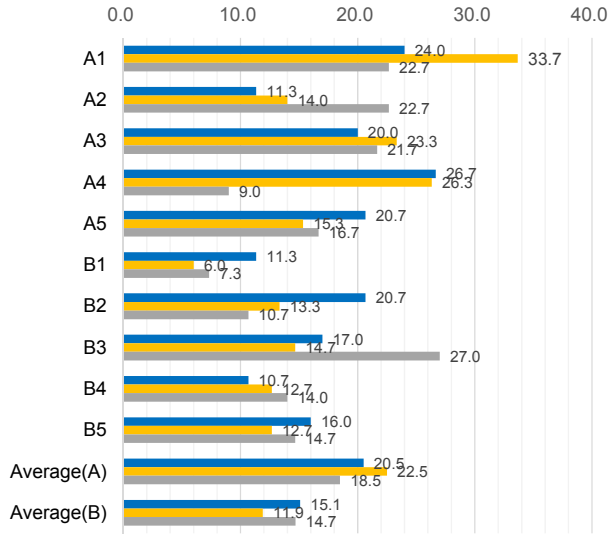


Fig. 5: The average time required for making decision for websites 1-3 (blue), websites 5-7 (orange), and websites 9-11 (gray), in respectively

In comparison to the results, we could not observe significant increase of time raised from EyeBit. We confirmed that it took 22.5 seconds at the second stage for the participants with EyeBit, whereas 11.9 seconds for the participants without EyeBit. However, once the user gets in habits of seeing the address bar, the average time was deceased to 18.5 seconds, whereas it took 14.7 seconds. In regard to the differences among individuals from the first stage, we assumed that the inconvenience caused of EyeBit would be negligible.

B. Educational approach

Education is one of the straightforward ways to counter phishing since phishing problems are caused of human errors. There were much number of educational materials. For example, Merve et al. [37] proposed educational materials and a strategy on preparing to avoid phishing attacks.

Despite claims by security and usability experts that user education about security does not work [38], there are some evidence that well designed user security education can be effective. Kumaraguru et al. proposed to employ a comic as an educational material [39]. They tested the educational effectiveness of 30 subjects with three types of educational materials. Their results suggested that typical security notices were ineffective. Their results also indicated that that their comic strip format was more effective than the text and graphics. Sheng et al. [40] found that the game is a novel educational material. The main character of the game was Phil, a young fish living in the Interweb Bay. Phil wanted to eat worms so he can grow up to be a big fish, but has to be careful of phishers that try to trick him with fake worms (representing phishing attacks). They conducted the total correctness of subjects' classification before and after the education. By using this game, the correctness increased from 69%, before the education, to 87%. In the case of using existing training materials, the correctness increased from 66% to 74%.

In contrast with past studies [37], [39], [40], our approach focused on getting habits, rather than development of educational materials. Since educational materials are often ignored by users, our EyeBit was designed for getting end users to pay attention to the address bar.

An alternative approach employed learning science principles in which phishing education is made part of a primary task for users [41]. This intended to extend their past research [39], and analyzed the individual user characteristics for improving their educational materials. Our EyeBit gave further evidence to their observations [41] on personalization of phishing prevention.

C. Evaluation of effectiveness

All at first, getting habitual actions usually takes time. EyeBit selected a methodology which enforces end users to see the address bar before using input forms, and there are many alternative methodologies for getting habits. Comparative study among methodologies should be considered in future works.

In order to confirm these effectiveness gained by EyeBit, we will evaluate the effectiveness for long-term retention. Nevertheless there are few research focused on observing the effectiveness in long time periods, Kumaraguru et al. [42] conducted the evaluation of various educational materials. In the case of EyeBit, we should evaluate the effectiveness to end users in different mental modes; the results might be different if the users feel stressed. It may be difficult due to the potential violation of the ethics whenever we intentionally make stressful events to participants.

Furthermore, we will conduct our evaluation of modularity for EyeBit in regard to cognitive aspects. Aside from anti-phishing, understanding users' mental state with eye-tracking may be feasible solution to personalize cyber defense systems. Since recent social engineering employs psychological manipulation techniques, the anomalies in mental state might be recognized by observable behavior. To assess this hypothesis, we will analyze end users' behavior, find its characteristics, and develop personalized defense mechanisms in consideration of the attributes of each end user. As shown in section II-C, eye movement will give much insights while estimating the users' behavior and its foundation.

For evaluating the effectiveness, elimination of bias might be discussed. In our experiments, there were some bias due to the number of samples and/or biased samples. In order to thwart the bias, we will present our prototype of EyeBit at shared code repository [43]. It is possible by distributing the work as browser-extension with some feedback and getting a large population of users to agree to use it.

D. Limitation of implementation

In this study, we used two eye-tracking devices that are designed for non-mounting monitoring. They are usually affected by sudden movements of head, neck and/or face. In order to suppress it, the head-mount eye-tracking device might be available. Our experiments were intended to thwart participants' inconvenience caused by equipment of the head-mounted device. However, it might be worthwhile to evaluate EyeBit in the case of using this type of devices.

As we mentioned in section III-C, the limitation of our prototype was recognition of the address bar. EyeBit should identify a browser window on the screen at first, and then recognize the position of its address bar. One possible way is pattern matching in a digitized image. Alternative is estimating from the position of the browser's top-left corner. In both cases, adjusting for each participant will be necessary.

A potential implementation issue is lack of support for smartphone devices. Recently, people use smartphone devices as well as personal computer. However, smartphone users are also faced with cyber crimes, since the user interfaces for smartphones are constrained by their small screens, browsers in the smartphones often lack a function for showing trustworthy indicators. Due to the small size of the smartphone browser's address bar, it is necessary not only checking the users' gaze to the address bar, but also monitoring their additional activities. This might entail the best practice for browsing with smartphones, but it is beyond the scope of this paper.

V. Conclusion

Basically, habits of checking the address bar will exert security education and knowledge, improve a chance to be aware of security information from browsers, and work like a conditioned reflex action regardless of the users' primal concern. This paper therefore focused on enforcing end users to get the habit of checking the address bar. Our key contribution is development of EyeBit, which aims end users acquiring the habit of seeing browser's address bar before entering any data into websites. EyeBit was able to control web input forms, and deactivate all of them until the end users saw the address bar. By interacting with eye-tracking devices, it finally activated the forms when the users saw there.

We confirmed that the effectiveness of seeing the bar, at first. Our participant-based test showed that the decision accuracy increased by checking the address bar. Based on the observation, we designed and implemented EyeBit as a browser extension with eye-tracking camera. In the pilot study, we performed new participant-based test. The effectiveness of the education with EyeBit succeeded to form the behavior of novices. We found the inconvenience caused of EyeBit was negligible. One month later, we performed a follow-up study to observe behavior of novices. The pilot study could not show significant difference in the case of the education without EyeBit, however the follow-up study indicated that EyeBit could decrease false negative errors in which a participant deems a phishing website as legitimate. The eye movements of novices who employed EyeBit in the pilot study often checked the address bar. We therefore considered that EyeBit was helpful for getting the habit of seeing address bar while making trust decisions.

Acknowledgment

This research has been supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA). The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the Ministry of Internal Affairs and Communications, Japan, or of the European Commission.

References

[1] C. Abad, "The economy of phishing: A survey of the operations of the phishing market," *First Monday*, vol. 10, no. 9, 2005.

[2] R. Dhamija, J. D. Tygar, and M. A. Hearst, "Why Phishing Works." in *Proceedings of Conference On Human Factors In Computing Systems*, Apr. 2006.

[3] D. Miyamoto, H. Hazeyama, Y. Kadobayashi, and T. Takahashi, "Behind HumanBoost: Analysis of Users' Trust Decision Patterns for Identifying Fraudulent Websites," *Journal of Intelligent Learning Systems and Applications*, vol. 4, no. 4, pp. 319–329, 2012.

[4] B. J. Fogg, L. Marable, J. Stanford, and E. R. Tauber, "How Do People Evaluate a Web Site's Credibility? Results from a Large Study," Stanford, Tech. Rep., Nov. 2002.

[5] Z. E. Ye, Y. Yuan, and S. Smith, "Web Spoofing Revisited: SSL and Beyond," Department of Computer Science, Dartmouth College, Tech. Rep. TR2002-417, Feb. 2002.

[6] P. Kumaraguru, A. Acquisti, and L. F. Cranor, "Trust modeling for online transactions: A phishing scenario," in *Proceedings of the 3rd Annual Conference on Privacy, Security, and Trust*, Oct. 2005.

[7] M. Wu, R. C. Miller, and S. L. Garfinkel, "Do Security Toolbars Actually Prevent Phishing Attacks?" in *Proceedings of Conference On Human Factors In Computing Systems*, Apr. 2006.

[8] Netcraft, "Netcraft Anti-Phishing Toolbar," Available at: http://toolbar.netcraft.com/.

[9] A. Herzberg and A. Gbara, "TrustBar: Protecting (even Naïve) Web Users from Spoofing and Phishing Attacks," Cryptology ePrint Archive, Report 2004/155, Tech. Rep., Jul. 2004.

[10] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. C. Mitchell, "Client-side defense against web-based identity theft," in *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, Feb. 2004.

[11] S. M. Zahran and G. H. Galal-Edeen, "A Categorization Technique for Resolving Information System Failures Reasons," *International Journal of Electrical and Computer Science*, vol. 12, no. 5, pp. 67–77, 2012.

[12] F. Raja, K. Hawkey, and K. Beznosov, "Revealing hidden context: improving mental models of personal firewall users," in *Proceedings of the 5th Symposium On Usable Privacy and Security*, July 2009, pp. 1–12.

[13] K. Hawkey, D. Botta, R. Werlinger, K. Muldner, A. Gagne, and K. Beznosov, "Human, Organizational, and Technological factors of IT security," in *Extended Abstracts on Human Factors in Computing Systems*, April 2008, pp. 3639–3644.

[14] R. Werlinger, K. Hawkey, D. Botta, and K. Beznosov, "Security Practitioners in Context: Their Activities and Interactions with Other Stakeholders within Organizations," *International Journal of Human-Computer Studies*, vol. 67, pp. 584–606, 2009.

[15] S. E. Parkin, A. van Moorsel, and R. Coles, "An information security ontology incorporating human-behavioural implications," in *Proceedings of the 2nd international conference on Security of information and networks*, October 2009, pp. 46–55.

[16] British Standards Institution, "ISO/IEC 27002:2005 - Information Technology - Security Techniques - Code of Practice and Information Security Management," 2005.

[17] S. Alfawaz, K. Nelson, and K. Mohannak, "Information security culture: a behaviour compliance conceptual framework," in *Proceedings of the 8th Australasian Conference on Information Security*, July 2010, pp. 47–55.

[18] R. West, "The Psychology of Security," *Communications of the ACM*, vol. 51, pp. 34–41, 2008.

[19] M. Grootjen, M. A. Neerincx, and J. C. van Weert, "Task Based Interpretation of Operator State Information for Adaptive Support," ACI/HFES-2006, Tech. Rep., 2006.

[20] D. Watson and R. Friend, "Measurement of social-evaluative anxiety," *Consulting and Clinical Psychology*, vol. 33, pp. 448–457, 1969.

[21] G. F. Wilson, "An analysis of Mental Workload in Pilots during flight using multiple psychophysiological measures," *International Journal of Aviation Psychology*, vol. 12, pp. 3–8, 2002.

[22] S. Miyake, "Multivariate workload evaluation combining physiological and subjective measures," *International Journal of Psychophysiology*, vol. 40, pp. 233–238, 2001.

[23] H. van Kuilenburg, M. Wiering, and M. den Uyl, "A Model Based Method for Automatic Facial Expression Recognition," in *Proceedings of the 16th European Conference on Machine Learning*, Oct. 2005.

[24] A. Haag, S. Goronzy, P. Schaich, and J. Williams, "Emotion Recognition Using Bio-Sensors: First Steps Towards an Automatic System," in *Proceedings of Affective Dialogue Systems, Tutorial and Research Workshop*, June 2004, pp. 36–48.

[25] T. Crawford, S. Higham, T. Renvoize, J. Patel, M. Dale, A. Suriya, and S. Tetley, "Inhibitory control of saccadic eye movements and cognitive impairment in alzheimer's disease," *Biological Psychiatry*, vol. 9, no. 57, pp. 1052–1060, 2005.

[26] B. Noris, K. Benmachiche, J. Meynet, J.-P. Thiran, and A. Billard, "Analysis of Head-Mounted Wireless Camera Videos for Early Diagnosis of Autism," *Advances in Soft Computing*, vol. 45, pp. 663–670, 2007.

[27] R. J. Leigh and D. S. Zee, *The Neurology of Eye Movements*, 4th ed. Oxford University Press, 1991.

[28] D. E. Irwin and J. R. Brockmole, "Mental rotation is suppressed during saccadic eye movements," *Psychonomic Bulletin and Review*, vol. 7, no. 4, pp. 654–661, 2000.

[29] S. Tokuda, G. Obinata, E. Palmer, and A. Chaparro, "Estimation of mental workload using saccadic eye movements in a free-viewing task," *Proceedings of the 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 4523–4529, August 2011.

[30] C. K. Ora and V. G. Duffyb, "Development of a facial skin temperature-based methodology for non-intrusive mental workload measurement," *Occupational Ergonomics*, vol. 7, pp. 83–94, 2007.

[31] L.-M. Wang, V. G. Duffy, and Y. Du, "A composite measure for the evaluation of mental workload," in *Proceedings of the 1st International Conference on Digital Human Modeling*, 2007, pp. 460–466.

[32] J. Voskamp and B. Urban, "Measuring Cognitive Workload in Non-military Scenarios Criteria for Sensor Technologies," in *Proceedings of the 5th International Conference on Foundations of Augmented Cognition*, June 2009, pp. 304–310.

[33] H. Genno, K. Ishikawa, O. Kanbara, M. Kikumoto, Y. Fujiwara, R. Suzuki, and M. Osumi, "Using facial skin temperature to objectively evaluate sensations," *International Journal of Industrial Ergonomics*, vol. 19, pp. 161–171, 1997.

[34] Tobii Technology, "Tobii TX300," Available at: http://www.tobii.com.

[35] The EyeTribe, "The Eye Tribe Tracker," Availabel at: https://theeyetribe.com.

[36] Anti Phishing Working Group and CMY-CyLab, "Education Landing Page Program," Available at: http://phish-education.apwg.org/r/about.html.

[37] A. Van der Merwe, M. Loock, and M. Dabrowski, "Characteristics and Responsibilities Involved in a Phishing Attack," in *Proceedings of the 4th International Symposium on Information and Communication Technologies*, Jan. 2005.

[38] J. Evers, "Security Expert: User education is pointless," Available at: http://news.com.com/2100-7350_3-6125213.html, Oct. 2007.

[39] P. Kumaraguru, Y. Rhee, A. Acquisti, L. F. Cranor, J. I. Hong, and E. Nunge, "Protecting people from phishing: the design and evaluation of an embedded training email system," in *Proceedings of Conference On Human Factors In Computing Systems*, Apr. 2007, pp. 905–914.

[40] S. Sheng, B. Magnien, P. Kumaraguru, A. Acquisti, L. F. Cranor, J. I. Hong, and E. Nunge, "Anti-Phishing Phil: the design and evaluation of a game that teaches people not to fall for phish," in *Proceedings of the 1st Symposium On Usable Privacy and Security*, Jul. 2007.

[41] P. Kumaraguru, Y. Rhee, S. Sheng, S. Hasan, A. Acquisti, L. F. Cranor, and J. I. Hong, "Getting users to pay attention to anti-phishing education: evaluation of retention and transfer," in *Proceedings of the Anti-Phishing Working Groups 2nd Annual eCrime Researchers Summit*, Oct. 2007, pp. 70–81.

[42] P. Kumaraguru, J. Cranshaw, A. Acquisti, L. F. Cranor, J. I. Hong, M. Ann, and T. Pham, "School of Phish: A real-World Evaluation of Anti-Phishing Training," in *Proceedings of the 5th Symposium On Usable Privacy and Security*, Jul. 2009.

[43] NECOMA Project, "necoma - github," Available at: https://github.com/necoma/eyebit, (to appear).

3

Short Papers Papers

This chapter contains copies of the accepted short research papers, as they will appear in the actual proceedings.

The Vulnerability Dataset of a Large Software Ecosystem

Dimitris Mitropoulos*, Georgios Gousios†, Panagiotis Papadopoulos‡,
Vassilios Karakoidas*, Panos Louridas*, and Diomidis Spinellis*

*Department of Management Science and Technology
Athens University of Economics and Business
{dimitro, bkarak, louridas, dds}@aueb.gr

†Software Engineering Research Group
Delft University of Technology
G.Gousios@tudelft.nl

‡Institute of Computer Science
Foundation for Research and Technology, Hellas
panpap@ics.forth.gr

***Abstract*—Security bugs are critical programming errors that can lead to serious vulnerabilities in software. Examining their behaviour and characteristics within a software ecosystem can provide the research community with data regarding their evolution, persistence and others. We present a dataset that we produced by applying static analysis to the Maven Central Repository (approximately 265GB of data) in order to detect potential security bugs. For our analysis we used *FindBugs*, a tool that examines Java bytecode to detect numerous types of bugs. The dataset contains the metrics' results that FindBugs reports for every project version (a JAR) included in the ecosystem. For every version in our data repository, we also store specific metadata, such as the JAR's size, its dependencies and others. Our dataset can be used to produce interesting research results involving security bugs, as we show in specific examples.**

***Keywords*—Security Bugs, Software Security, Static Analysis, FindBugs, Software Ecosystem, Maven Repository, Software Evolution.**

I. Introduction

A security bug is a programming error that introduces a potentially exploitable weakness into a computer system [1]. Compared to other bug categories, failures due to security bugs have two distinct features: they can severely affect an organization's infrastructure [2], and they can cause significant financial damage to an organization [3], [4]. Specifically, whereas a software bug can cause a software artifact to fail, a security bug can allow a malicious user to alter the execution of the entire application for his or her own gain. Such bugs could give rise to a wide range of security and privacy issues, like the access of sensitive information, the destruction or modification of data, and denial of service. Moreover, security bug disclosures lead to a negative and significant change in market value for a software vendor [5]. One of the most common approaches to identify security bugs is *static analysis* [6]. This kind of analysis involves the inspection of the program's source or object code without executing it.

A software ecosystem can be seen as a collection of software projects, which are developed and co-evolved in the same environment [7]. Components can be interdependent and have multiple versions. Examples of such ecosystems include Python's PyPI[1] (Python Package Index), Perl's CPAN[2] (Comprehensive Perl Archive Network), Ruby's RubyGems[3] and the Maven Central Repository.[4] Maven is a build automation tool used primarily for Java projects hosted by the Apache Software Foundation. It uses XML to describe the software project being built, its dependencies on other external modules, the build order, and required plug-ins. To build a software component, it dynamically downloads Java libraries and Maven plug-ins from the Maven Central Repository, and stores them in a local cache. The repository can be updated with new projects and also with new versions of existing projects that can depend on other versions.[5]

TABLE I. Bug categorization according to FindBugs.

Category	Description
Bad Practice	Violations of recommended and essential coding practice.
Correctness	Involves coding misting a way that is particularly different from the other bug sakes resulting in code that was probably not what the developer intended.
Experimental	Includes unsatisfied obligations. For instance, forgetting to close a file.
Internationalization (i18n)	Indicates the use of non-localized methods.
Multi-Threaded (MT) Correctness	Thread synchronization issues.
Performance	Involves inefficient memory usage allocation, usage of non-static classes.
Style	Code that is confusing, or written in a way that leads to errors.
Malicious Code	Involves variables or fields exposed to classes that should not be using them.
Security	Involves input validation issues, unauthorized database connections and others.

To analyze the Maven repository we used FindBugs,[6] a static analysis tool that was also used for research purposes in [8] and [9]. FindBugs' role is to examine Java bytecode to detect software bugs and separate them into nine categories. Two of them involve security issues (see Table I). In this paper we present: a) the construction process to obtain the collection

[1]https://pypi.python.org/pypi
[2]http://www.cpan.org/
[3]http://rubygems.org/
[4]http://mvnrepository.com/
[5]Note that in the Maven repository, versions are *actual* releases.
[6]http://findbugs.sourceforge.net/

TABLE II. DESCRIPTIVE STATISTICS MEASUREMENTS FOR THE MAVEN REPOSITORY.

Measurement	Value
Projects	17,505
Versions (total)	115,214
Min (versions per project)	1
Max (versions per project)	338
Mean (versions per project)	6.58
Median (versions per project)	3
Range (over versions)	337
1^{st} Quartile (over versions)	1
3^{rd} Quartile (over versions)	8

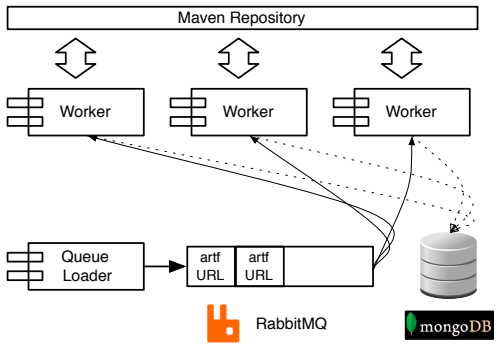


Fig. 1. The data processing architecture.

of the metrics results that the FindBugs tool produces for every project version of the repository (115,214 JARs), b) our dataset and c) how researchers can use the dataset and produce meaningful results concerning security bugs.

II. CONSTRUCTION PROCESS

Initially, we obtained a snapshot of the Maven repository and handled it locally to retrieve a list of all the names of the project versions that existed in it. Then, we filtered out projects written in programming languages other than Java because FindBugs analyzes only Java bytecode. The statistic measurements concerning the repository can be seen in Table II.

Due to the large volume of our dataset, we designed our data processing step in a distributed way. A schematic representation of our data processing architecture can be seen in Figure 1. In particular, we created a series of processing tasks based on the JAR list we have obtained and added them to a task queue mechanism (a RabbitMQ[7] message broker). Then, we executed twenty five workers (custom Python scripts) that checked out tasks from the queue, processed each project version and stored the results to the data repository (a MongoDB[8] database system).

A typical processing cycle of a worker included the following steps: as soon as the worker was spawned, it requested a task from the queue. This task contained the JAR name, which was typically a project version that was downloaded locally. First, specific JAR metadata were calculated and stored (see Section III). Then, FindBugs was invoked by the worker and its

[7]http://www.rabbitmq.com/
[8]http://www.mongodb.org/

TABLE III. BUG DESCRIPTION.

type	EI_EXPOSE_REP2
category	MALICIOUS_CODE
source File	ANTLRHashString.java
class	antlr.ANTLRHashString
method	setBuffer
sourceline start	97
sourceline end	98

results were also stored in the data repository. Note that before invoking FindBugs, the worker checked if the JAR is valid in terms of implementation. For instance, for every JAR the worker checked if there were any *.class* files before invoking FindBugs.

When the data collection was completed, we ran some tests to check the validity of the results. A common issue that we discovered was the out-of-memory crashes of FindBugs, which demanded the repetition of the process for the corresponding JARs, with the appropriate settings in the Java Runtime Environment (JRE).

III. DATASET ENTRIES

FindBugs reports *bug collections* that include all the bugs discovered in a JAR file. For every registered bug, there are numerous accompanying features like the class, the method and the line that the bug was found (see Table III). FindBugs' results also include additional information like the number of classes included in the examined JAR and others.

As we mentioned earlier, our data were stored in a MongoDB database that stores its records in JSON-like documents. However, FindBugs outputs its results in XML format. Hence, all the data were converted to the JSON format by mapping all XML elements to JSON objects.

As we discussed in Section II, our workers calculated and stored specific metadata together with the FindBugs' results. Such metadata included the JAR's size (in terms of bytecode), its dependencies, and the ordinal version number of the version. This number was derived from an XML file that accompanies every project in the Maven repository called *maven-metadata.xml*. The following listing shows the format of the metadata each worker collected. Note that the results of FindBugs are too large to fit, thus in order to see a complete instance please visit our GitHub repository (see Section VII):

```
{"JarMetadata": {
    "version": "1.0.0",
    "version_order": "1",
    "jar_size": "34768",
    "dependencies": [
        {
            "version" : "2.0",
            "groupId" : "org.apache.maven",
            "artifactId" : "maven-project"
        },
        { /* other dependencies */ }
    ],
    "group_id": "org.apache.myfaces.buildtools",
    "jar_filename": "myfaces-jdev-plugin-1.0.0.jar",
    "artifact_id": "myfaces-jdev-plugin",
    },
 "BugCollection": { /* FindBugs data */ }
}
```

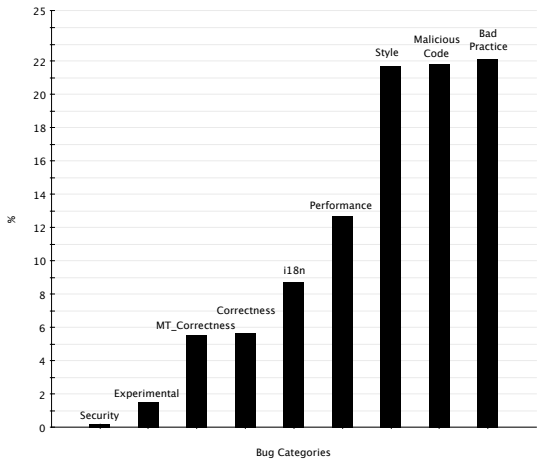


Fig. 2. Bug percentage in Maven repository.

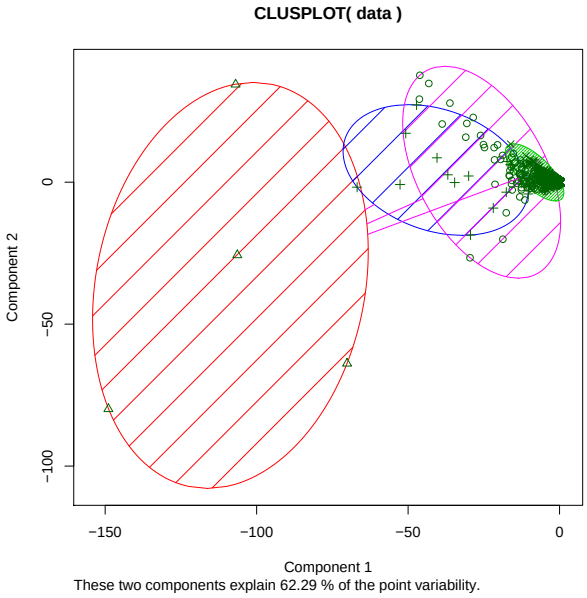


Fig. 3. Basic *k-means* clusters of all the versions that exist in the ecosystem.

IV. Gathering Experience Returns for Security Based on our Dataset

Since MongoDB provides a rich query interface, it was easy to find out how software bugs are distributed among the repository (see Figure 2) or identify the main clusters that are formed based on the number of the bugs of every version (see Figure 3). An interesting observation is that the *Malicious Code* bugs, together with the *Bad Practice* bugs are the most popular in the repository. Also, a simple query like the following, will reveal that from the total number of versions, 45,559 of them contained at least one bug coming from the *Malicious Code* category:

```
db.findbugs.find({
     'BugCollection.BugInstance.category' :
     'MALICIOUS_CODE'}).count()
```

TABLE IV. Correlations between version and software bugs count.

Category	Spearman Correlation	p-value
Security	0.04	< 0.05
Malicious Code	0.03	$\ll 0.05$
Style	0.03	$\ll 0.05$
Correctness	0.04	$\ll 0.05$
Bad Practice	0.03	$\ll 0.05$
MT Correctness	0.09	$\ll 0.05$
i18n	0.06	$\ll 0.05$
Performance	*(0.01)*	0.07
Experimental	0.09	$\ll 0.05$

Another observation involves specific bugs thath we could consider as critical and they are a subset of the *Security* category. Such bugs are related to vulnerabilities that appear due to the lack of user-input validation and can lead to damaging attacks like SQL injection and Cross-Site Scripting [10]. Also, as FindBugs' bug descriptions indicate,[9] if an application has bugs coming from this category, it might have more vulnerabilities that FindBugs doesn't report. Table V presents the number of versions where at least one of these bugs exists. In essence, 5,341 project versions, contained at least one bug related to user-input validation issues. Given the fact that other projects include these versions as their dependencies, they are automatically rendered vulnerable if they use the code fragments that include the security bugs.

Furthermore, we have created a series of scripts to exhibit how the dataset can be used to capture correlations regarding the evolution of security bugs. First, based on the dataset we produced some metadata that contained the number of bugs per category in each project version. Based on these metadata we estimated the relation between bugs and time (see Table IV). Specifically, we calculated the Spearman correlations between the defects count and the ordinal version number across all projects. The zero tendency that can be seen on Table IV applies to all versions of all projects together.

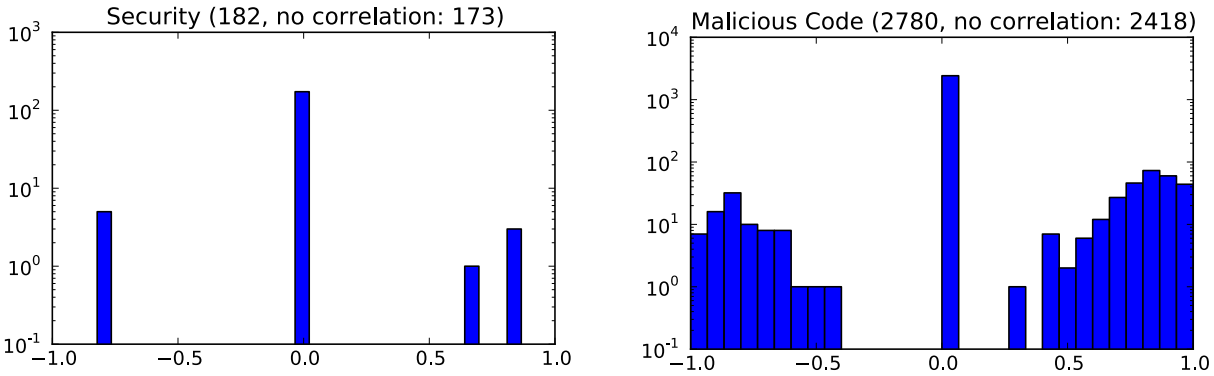
The situation was different in individual projects where we performed Spearman correlations between security bug counts and version ordinals in all projects we examined. These paint a different picture from the above table, shown in Figure 4. The spike in point zero is explained by the large number of projects for which no correlation could be established—note that the scale is logarithmic. Still, we can see that there were projects where a correlation could be established, either positive or negative. Such results indicate that we cannot say if vulnerabilities decrease or increase as projects mature.

In addition, we explored the relation between security bugs with the size of a project version, measured by the size of its JAR file by carrying out correlation tests between the size and the security bug counts for each project and version. The results can be seen in Table VI. An interesting observation is that the *Security* category stands out by having a remarkably lower effect than the other categories. As we mentioned earlier, many bugs that belong to this category are related to user-input validation issues. Hence, it seems that even if a programmer adds code to a new version, if this code does not require user input, the possibility of such bug is minimal.

[9] http://findbugs.sourceforge.net/bugDescriptions.html

TABLE V. NUMBER OF PROJECT VERSIONS THAT CONTAIN AT LEAST ONE SECURITY BUG RELATED TO USER-INPUT VALIDATION ISSUES.

Bug Description	Number of Project Versions
HRS: HTTP cookie formed from untrusted input	151
HRS: HTTP response splitting vulnerability	1,579
SQL: non-constant string passed to execute method on an SQL statement	1,875
SQL: a prepared statement is generated from a non-constant String	1,486
XSS: JSP reflected cross site scripting vulnerability	18
XSS: Servlet reflected cross site scripting vulnerability in error page	90
XSS: Servlet reflected cross site scripting vulnerability	142



(a) (b)

Fig. 4. Correlations between version and software bugs count.

TABLE VI. CORRELATIONS BETWEEN JAR SIZE AND SOFTWARE BUGS COUNT.

Category	Spearman Correlation	p-value
Security	0.19	$\ll 0.05$
Malicious Code	0.65	$\ll 0.05$
Style	0.68	$\ll 0.05$
Correctness	0.51	$\ll 0.05$
Bad Practice	0.67	$\ll 0.05$
MT Correctness	0.51	$\ll 0.05$
i18n	0.53	$\ll 0.05$
Performance	0.63	$\ll 0.05$
Experimental	0.36	$\ll 0.05$

Figure 5 presents the pairwise correlations between all bug categories. To establish these correlations, we calculated the correlations between the number of distinct bugs that appeared in a project throughout its lifetime. Our results show that bugs coming from the *Security* category are not correlated with the bugs coming from other categories. This indicates that security bugs of this kind do not appear together with the other bugs.[10]

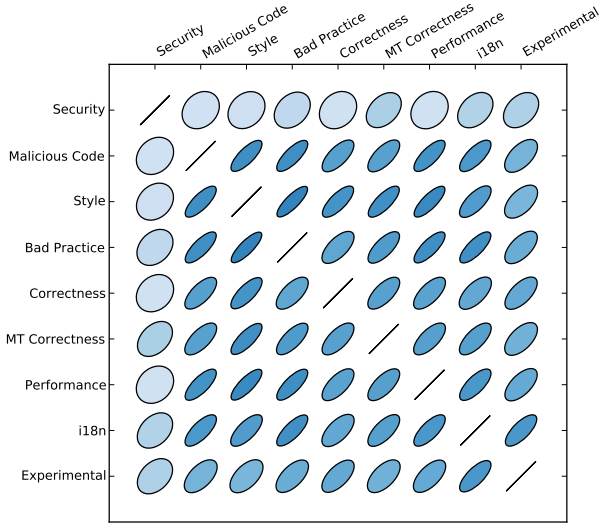


Fig. 5. Correlation matrix plot for bug categories. Blue color indicates positive correlation. The darker the color (and the more acute the ellipsis slant), the stronger the correlation.

V. THREATS TO VALIDITY

During our dataset analysis we faced some limitations that concerned the non-availability of some JARs. Specifically, there were some JARs included in the initial JAR list, that were not available online, when the FindBugs result collection was performed.

A threat to the internal validity of our dataset construction process could be the false alarms of the FindBugs tool [8], [12], [13]. Specifically, reported security bugs may not be applicable to an application's typical use context. For instance, FindBugs could report an SQL injection vulnerability in an application that receives no external input. In this particular context, this would be a false positive alarm. False alarms of static analysis tools and how they can be reduced are issues that have already been discussed in literature [9], [14] and they are beyond the scope of this paper.

[10]Further research concerning the examination of security bugs based on this dataset can be found in our previous paper [11].

VI. RELATED WORK

Our work is related to the creation of datasets to facilitate research and the examination of software vulnerabilities.

The Maven ecosystem has been previously analyzed by Raemaekers et al. [15] to produce the *Maven dependency*

dataset. Apart from basic information like individual methods, classes, packages and lines of code for every JAR, this dataset also includes a database with all the connections between the aforementioned elements. Our work differs from this research because it reports all bugs coming from the output of a static analysis tool, for each JAR contained in the Maven repository.

Identyfying software bugs in multiple projects is not a new idea [16]. On the security front, Ozment and Schechter [17] examined the code base of the OpenBSD operating system to determine whether its security is increasing over time. Massacci et al. [18] observed the evolution of software defects by examining six major versions of Firefox. In addition, Shahzad et al. [1] analysed large sets of vulnerability datasets to observe various features of the vulnerabilities that they considered critical, while Edwards et al. [19] have examined the evolution of security bugs by examining different versions of four projects.

VII. Conclusions

In this paper, we have presented a dataset that includes all the software bugs that each JAR of the Maven central repository contains along with some other metadata mentioned in Section IV. We have also shown how our data can be used to extract results concerning the evolution and the behaviour of security bugs.

Initially, we made some observations concerning the security bugs of the Maven repository as a whole. Then, based on our dataset, we found that we cannot say with confidence if security bugs increase or decrease as projects mature. We also showed that there were many projects where security bug counts do not change as projects evolve. Concerning the relation between severe security bugs and a project's size we showed that they are not proportionally related. Given that, we could say that it is productive to search for and fix security bugs even if a project grows bigger. In addition, the pairwise correlations between all categories indicates that even though all the other categories are related, severe bugs do not appear together with the other bugs. Such findings indicate that projects have their own idiosyncrasies regarding security bugs and could help us answer questions like: what are the common characteristics of the projects where security bugs increase over time? Finally, the analysis of a vulnerability management dataset like the NVD[11] (National Vulnerability Database), to identify disclosed vulnerabilities and check if there is a correlation between them and our dataset, could provide interesting results.

By selecting a large ecosystem that includes applications written only in Java, we excluded by default measurements that involve vulnerabilities like the infamous buffer overflow vulnerabilities [20]. Still, by examining software artifacts with similar characteristics facilitates the formation of an experiment. Thus, future work on our approach could also involve the observation of other ecosystems like the ones mentioned in Section I and projects in different languages like Ruby, Python *etc.* Concluding, the complete set of our data and source code will become available upon publication.

[11] http://nvd.nist.gov/

VIII. Acknowledgments

The present research is under the Action 2 of AUEB's[12] Research Funding Program for Excellence and Extrovesion of the academic year 2014/2015. It is financed by the University's Research Center.

The project is being co-financed by the European Regional Development Fund (ERDF) and national funds and is a part of the Operational Programme "Competitiveness & Entrepreneurship" (OPCE II), Measure "COOPERATION" (Action I).

[12] Athens University of Economics and Business.

References

[1] M. Shahzad, M. Z. Shafiq, and A. X. Liu, "A large scale exploratory analysis of software vulnerability life cycles," in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 771–781.

[2] H. Shahriar and M. Zulkernine, "Mitigating program security vulnerabilities: Approaches and challenges," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 11:1–11:46, Jun. 2012.

[3] J.-E. J. Tevis and J. A. Hamilton, "Methods for the prevention, detection and removal of software security vulnerabilities," in *Proceedings of the 42nd annual Southeast regional conference*, ser. ACM-SE 42. New York, NY, USA: ACM, 2004, pp. 197–202. [Online]. Available: http://doi.acm.org/10.1145/986537.986583

[4] D. Baca, B. Carlsson, and L. Lundberg, "Evaluating the cost reduction of static code analysis for software security," in *Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security*, ser. PLAS '08. New York, NY, USA: ACM, 2008, pp. 79–88.

[5] R. Telang and S. Wattal, "Impact of software vulnerability announcements on the market value of software vendors - an empirical investigation," in *Workshop on the Economics of Information Security*, 2007, p. 677427.

[6] B. Chess and J. West, *Secure programming with static analysis*, 1st ed. Addison-Wesley Professional, 2007.

[7] M. Lungu and M. Lanza, "The small project observatory: A tool for reverse engineering software ecosystems," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 289–292. [Online]. Available: http://doi.acm.org/10.1145/1810295.1810356

[8] N. Ayewah and W. Pugh, "The google FindBugs fixit," in *Proceedings of the 19th international symposium on Software testing and analysis*, ser. ISSTA '10. New York, NY, USA: ACM, 2010, pp. 241–252.

[9] J. Spacco, D. Hovemeyer, and W. Pugh, "Tracking defect warnings across versions," in *Proceedings of the 2006 international workshop on Mining software repositories*, ser. MSR '06. New York, NY, USA: ACM, 2006, pp. 133–136.

[10] D. Ray and J. Ligatti, "Defining code-injection attacks," in *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ser. POPL '12. New York, NY, USA: ACM, 2012, pp. 179–190. [Online]. Available: http://doi.acm.org/10.1145/2103656.2103678

[11] D. Mitropoulos, V. Karakoidas, P. Louridas, G. Gousios, and D. Spinellis, "Dismal code: Studying the evolution of security bugs," in *Proceedings of the LASER Workshop 2013, Learning from Authoritative Security Experiment Results*. Usenix Association, Oct. 2013, pp. 37–48. [Online]. Available: http://www.dmst.aueb.gr/dds/pubs/conf/2013-LASER-BugEvol/docs/html/evol.html

[12] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *SIGPLAN Not.*, vol. 39, no. 12, pp. 92–106, Dec. 2004.

[13] "An evaluation of findbugs," http://www.cs.cmu.edu/~aldrich/courses/654-sp07/tools/Sandcastle-FindBugs-2009.pdf, accessed: 2014-08-08.

[14] S. Heckman and L. Williams, "A model building process for identifying actionable static analysis alerts," in *Proceedings of the 2009 International Conference on Software Testing Verification and Validation*, ser. ICST '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 161–170.

[15] S. Raemaekers, A. v. Deursen, and J. Visser, "The maven repository dataset of metrics, changes, and dependencies," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 221–224. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487129

[16] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A few billion lines of code later: using static analysis to find bugs in the real world," *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[17] A. Ozment and S. E. Schechter, "Milk or wine: does software security improve with age?" in *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06. Berkeley, CA, USA: USENIX Association, 2006.

[18] F. Massacci, S. Neuhaus, and V. H. Nguyen, "After-life vulnerabilities: a study on firefox evolution, its vulnerabilities, and fixes," in *Proceedings of the Third international conference on Engineering secure software and systems*, ser. ESSoS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 195–208.

[19] N. Edwards and L. Chen, "An historical examination of open source releases and their vulnerabilities," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 183–194. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382218

[20] A. D. Keromytis, "Buffer overflow attacks," in *Encyclopedia of Cryptography and Security (2nd Ed.)*, 2011, pp. 174–177.

MATATABI: Multi-layer Threat Analysis Platform with Hadoop

Hajime Tazaki
The University of Tokyo, Japan
tazaki@nc.u-tokyo.ac.jp

Kazuya Okada
Nara Institute of Science and Technology, Japan
kazuya-o@is.naist.jp

Yuji Sekiya
The University of Tokyo, Japan
sekiya@wide.ad.jp

Youki Kadobayashi
Nara Institute of Science and Technology, Japan
youki-k@is.aist-nara.ac.jp

***Abstract*—Threat detection and analysis are indispensable processes in today's cyberspace, but current state of the art threat detection is still limited to specific aspects of modern malicious activities due to the lack of information to analyze. By measuring and collecting various types of data, from traffic information to human behavior, at different vantage points for a long duration, the viewpoint seems to be helpful to deeply inspect threats, but faces scalability issues as the amount of collected data grows, since more computational resources are required for the analysis. In this paper, we report our experience from operating the Hadoop platform, called MATATABI, for threat detections, and present the micro-benchmarks with four different backends of data processing in typical use cases such as log data and packet trace analysis. The benchmarks demonstrate the advantages of distributed computation in terms of performance. Our extensive use cases of analysis modules showcase the potential benefit of deploying our threat analysis platform.**

***Keywords*—Cybersecurity, Multi-layer threat analysis, Hadoop**

I. INTRODUCTION

In parallel to the growth of the Internets functionality as a distributed system, the number of critical threats is also increasing, rendering pro-active defensive approaches troublesome. Various categories of malicious activities have been seen in the wild, and multiple countermeasures have been proposed and deployed. Unfortunately, no defense mechanism has been able to completely hinder attacks and bring an end to this perpetual arms race.

A significant obstacle against deploying a countermeasure for such threats, is the lack of knowledge of what is happening in the system: a single point of observation at the firewall has no knowledge about the other egress nodes of an enterprise network, or scanning applications deployed at various endpoints lack global information which would provide information whether their probing is assisting a Distributed Denial of Service (DDoS) attack. The limited number of observation points and type of information collected needs improvement.

The recently established NECOMA project[1] aims at improving the situation of current cyber-security, by introducing new insight regarding countermeasure. It assumes the missing pieces for creating robust countermeasure are the lack of 1) information or types of datasets for threat analysis and 2) locations that observers should look at. In the other words, *the more data is available about an attack, the more data can be analyzed and thus, the higher the probability that the most effective countermeasures are taken.*

However, such a huge collected dataset easily faces a scalability problem in terms of not only the storage of the collected data, but also computation resource of the analysis itself because it also requires a fair amount of computational resource to investigate the datasets across the multiple sensors at different layers and various locations.

This paper reports our experience on the development of a big-data platform, called MATATABI, in order to fulfill the requirements for cyber-threat analysis (detailed in § II). We combined possible techniques and ideas available to satisfy the requirements, by incorporating and tweaking existing open source software. Our development follows several basic studies ([1][2][3]) in the past, but aimed at creating more complete system that allows us to detect complex security threats involving multiple data sources and locations of attackers.

Our contributions of this paper include:

- We designed and implemented the data collection and analysis platform, MATATABI, to handle multi-terabytes measurement data for the security threat analysis.
- We studied performance of our system with data querying benchmarks and gave the best practice for the implementations of threat analysis modules.

II. REQUIREMENTS

Considering the amount of collected data for the threat analysis, the design of data collection and analysis platform must satisfy the following properties;

R-1) the input and output performance of data must be scalable in terms of size of analyzed data (**scalability**).
R-2) the computation resource must be scaled out with the number of nodes: hopefully real-time performance for the detection of threats on the fly (**real-time analysis**).
R-3) the platform must be adaptable with the multiple kinds of data types (e.g., formatted text, binary data, etc) and layers (e.g., network traffic, user behaviors, etc) without introducing new ways to analyze them (**uniform programmability**).

R-1 and R-2 are obvious requirements, but R-3 is also important since one cannot predict threats, but is required to detect them once they happen.

[1] http://www.necoma-project.eu/

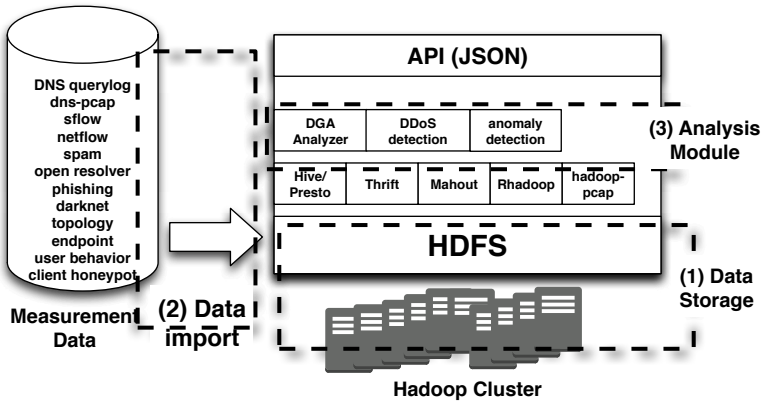


Fig. 1. Overview of MATATABI. Based on the Hadoop platform, we integrated the data storage with import modules, analysis scripts, and an application programing interface in a single platform.

We will describe the system and how it fulfills the aforementioned requirements in the following section.

III. DESIGN OF MATATABI PLATFORM

This section presents the design of our platform, so called MATATABI that serves data collection and analysis for the threat detection in order to fulfill the requirements described in § II. Figure I depicts the overview of our implemented platform, which consists of three key components; 1) data storage in a distributed environment, 2) data import modules, 3) analysis modules, and 4) Application Programming Interface (API) with the help of Apache Hadoop [4] software[2].

A. Data Storage and Base Software

The data storage component relies on the Hadoop Distributed File System (HDFS) to locate and access data in a distributed environment so that applications are agnostic to access the data where they are running on. We employed totally nine cluster nodes in total (Table I) distributed across several Japanese universities.

On top of served distributed file system, various data access utilities such as Hive [5] (SQL liked interface), Presto-db [6] (distributed query engine), and language bindings (Thrift [7], Rhadoop [8]) are employed in order to create analysis modules (§ III-C). These varieties of utilities are helpful not only for its friendliness to Hadoop environment, but also we can reuse existing analysis implementations used at stand-alone environments without reimplementing a lot. Furthermore, SQL like interface provided by Hive or Presto-db is useful to analyze multiple layers of data sources with simple query statements.

B. Data Import Module

The data import module basically works for copying various kinds of collected data into HDFS so that the analysis module implemented by the Map-Reduce framework can access directly. It will benefit *locality* during data reading process, as it is arranged by Hadoop automatically.

[2]At the moment, we used Apache Hadoop 2.2.0 version.

TABLE I. EQUIPMENTS OF HADOOP CLUSTER.

	CPU	RAM	Storage
master	2.5GHz (8 cores)	24GB	1.9TB
hadoop1	2.2GHz (16 cores)	38GB	52GB
hadoop2	0.8GHz (8 cores)	68GB	77GB
hadoop4	0.8GHz (8 cores)	68GB	77GB
hadoop6	0.8GHz (8 cores)	32GB	253GB
hadoop7	2.2GHz (16 cores)	50GB	1.9TB
slave02	2.0GHz (24 cores)	64GB	6.6TB
slave03	2.0GHz (24 cores)	64GB	6.6TB
slave04	2.0GHz (24 cores)	64GB	6.6TB

Table II describes the list of data import modules which we have used at the present moment. Some of them are converted to Hive-oriented table, others are stored as-is (binary data).

For the data access via Hive, Hive Serializer/Deserializer (SerDe) is used to read and write HDFS data with a custom format. It allows us to reduce the cost of implementing a data import module. We slightly modified RIPE pcap SerDe [9] for the data stored in `pcap` format.

Figure 2 is an example of a Hive database schema, which represents a custom format definition of `pcap` file containing DNS packets. With `PcapDeserializer` of the RIPE module, `pcap` files can be queried with an SQL-like language.

C. Analysis Module

The analysis module works on top of data store which provides high computation resource with flexible data access interface. Unlike ordinary applications for threat analysis running on a standalone machine, the module will benefit from the distributed computations by Map-Reduce or distributed query engine of Presto-db.

An example of the process of our implemented analysis module, and the corresponding queries to the Hive/Presto-db can be seen in figure 3: 1) to look for events in collected datasets which contain suspicious indications of threat, 2) find the behavior of the indications into another dataset to identify correlations among multiple data sources (where 192.168.10.1 is the IP address that first query detects it as suspicious host), and 3) extract some attributes from the indications and store into a blacklist.

TABLE II. DATA CONVERSION INTO HDFS.

	format	parser	data size (per day)	remark
DNS pcap	as-is	PcapDeserializer (hadoop-pcap [9])	5GB	date/node partitioned
Netflow	csv	CSV	1.2GB	`nfdump`, lzo compress, date/node partitioned
sFlow	csv	CSV	4.1GB	`sflowtool`, lzo compress, date/node partitioned
DNS querylog	ssv (bind9)	SSV	1.5G	date/node partitioned
SPAM email	SSV		4.5MB	date/MUA partitioned

```
CREATE EXTERNAL TABLE IF NOT EXISTS dns_pcaps (ts bigint,
                            protocol string,
                            src string,
                            src_port int,
                            dst string,
                            dst_port int,
                            len int,
                            ttl int,
                            dns_queryid int,
                            dns_flags string,
                            dns_opcode string,
                            dns_rcode string,
                            dns_question string,
                            dns_answer array<string>,
                            dns_authority array<string>,
                            dns_additional array<string>)
PARTITIONED BY (dt string, server string)
ROW FORMAT SERDE
   'net.ripe.hadoop.pcap.serde.PcapDeserializer'
STORED AS INPUTFORMAT
   'net.ripe.hadoop.pcap.io.PcapInputFormat'
OUTPUTFORMAT
           'org.apache.hadoop.hive.ql.io.\
                        HiveIgnoreKeyTextOutputFormat'
LOCATION 'hdfs:///dns-pcaps/';
```

Fig. 2. Example Hive table scheme for `pcap` data.

```
1) select * from dns_pcaps where regexp_like \
   (dns_question, '[a-z0-9]{32,48}');
2) select * from netflow where srcip='192.168.10.1';
3) select time,client_fqdn from suspicious_flow ;
```

Fig. 3. Steps for an analysis module to seek suspicious flows through multiple data sources.

D. MATATAPI: API for MATATABI

The analytical results obtained from our platform are valuable not only for our own purpose, but also the others who try to detect threats from their analysis. Multi-dimensional analysis using different datasets at the different physical or logical space will help early threat detection: if indication of threats were detected in advance and propagated these information to others, one can countermeasure against such threats.

For that purpose, we designed MATATAPI, an application programing interface (API) for MATATABI, in order to provide an interface for accessing analytical results. Our design is a simple wrapper for the existing Presto REST API, which is available by Presto-db and generates JavaScript Object Notation (JSON) objects for a certain request through the API. All we need to provide an API is 1) to create a Hive (or Presto-db) table, and 2) to write a bridge program from client requests to Presto REST API.

```
def Query (time_min, time_max):
   headers = {
   'context-type': 'text/plain',
   'x-presto-user': 'presto',
   'x-presto-catalog': 'hive',
   'x-presto-schema': 'default'
   }
   url='http://master:8080/v1/statement'

   data = "select dga_domain from zeus_dga_result
           where dt >= '%s' AND dt <= '%s'"
           % (time_min, time_max)
   request = urllib2.Request (url, data, headers)
   return json.loads (urllib2.urlopen(request).read())
```

Fig. 4. A sample python program of MATATAPI bridge program.

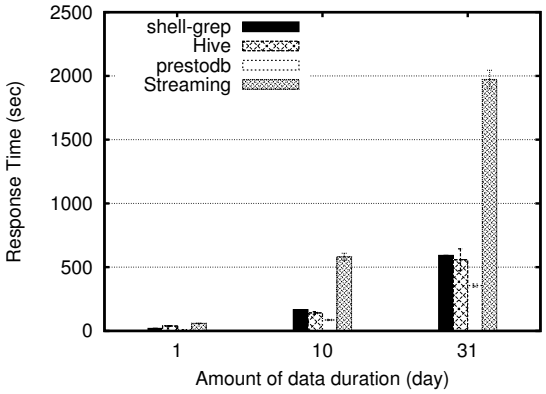


Fig. 5. Performance parsing on plain text data (DNS query log).

Figure 4 represents an example of the bridge program written in `python`, where it bridges requests received via http transport and runs as a CGI program on a web server.

IV. MICRO-BENCHMARKS

In order to assess the performance of the data processing of use cases that are typical to threat analysis, we conducted micro-benchmarks in this section. The objectives of this benchmark are:

- to observe the scalability for the amount of data (data size), and
- to present a best practice for implementing the analysis module.

A. Regular Expression Match on Plain Text Files

The first benchmark shows the response time for data query when the amount of data scanned for the queries increases.

```
'[a-z0-9]{32,48}.(ru|com|biz|info|org|net)'
```

Fig. 6. Regular expression of a DGAed domain name.

Since the data collected for the analysis increases day by day and is easy to fill up the storage, it is important to understand how much data we can process in order not to degrade the performance of analysis.

To measure the performance, we setup scripts that conduct a simple query to pick events from the data sources. The scripts we used are 1) `grep` command-based shell script without Hadoop (`shell-grep`), 2) Hive query language, 3) Presto-db SQL, and 4) python script of Hadoop streaming. All the scripts parse and look for strings using a regular expression shown in figure 6, which represents the Domain Generation Algorithm (DGA) for ZeuS Bot [10]), and then print the results to the console screen.

As a target data for this benchmark, we used a) formatted text based log files which contain `bind9`[3] querylog, and b) `pcap` files which contain DNS traffic.

We ran the scripts on our Hadoop cluster described in Table I (except `1) shell-grep` script, which uses a single node, master, with local storage) and measured the execution time of each script.

Figure 5 represents the result of response time of each script as a function of data size (i.e., we changed the number of date to be parsed) with the 95% confidence interval computed for 3 replications of script execution.

The performance of single node data queries present a slower response time, while distributed computation by Hive and Presto-db gives faster response (40% faster in the best case), when the size of data parsed increased. Note that although our hadoop streaming script dispatched query jobs into distributed node, we did not see much performance gain. This may be due to an implementation matter of the streaming script that we used, but it is possible to achieve such a performance with a simple mapper/reducer implementation for the hadoop streaming.

B. Regular Expression Match on pcap Files

Figure 7 represents another benchmark using different datasets, `pcap` files contain DNS traffic, in the same environment. In this benchmark we have completed only one iteration of the benchmark due to time constraints.

Unlike the performance on queries to plain text files shown in figure 5, the `shell-grep` performance is worse: the response time was 21 times slower in the worst case than the one of Presto-db. This is possible considering heavy tasks in each filtering process: extracting gzip compressed data of original `pcap` files, decoding packet data from `pcap` format by `tshark` command, and regular expression matching by `grep` command. On the other hand, other scripts using hadoop infrastructure employs RIPE `hadoop-pcap` library which effectively dispatches reading files and string matching

[3]https://www.isc.org/downloads/bind/

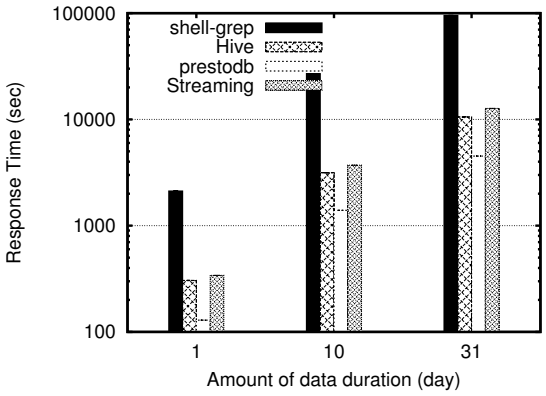


Fig. 7. Performance parsing on binary data (`pcap` file).

operation into distributed nodes, resulted in a high performance gain compared to the `shell-grep` implementation. Note that the RIPE `hadoop-pcap` library also stores the raw `pcap` data into HDFS, and then decompresses and decodes the file when data query is issued so, the both response times of `shell-grep` and others involve same procedure.

Presto-db achieves almost the best result among the four different scripts, with a low implementation cost for the data parsing script. If an analysis is based on simply looking up events on a Hive table, Presto-db is the best possible choice for a data store.

C. Processing Multiple Datasets

The last benchmark is a performance measurement on querying multiple datasets at the same time to analyze common interests between different datasets. This is an interesting feature of multi-layer threat analysis: if a dataset contains interests of threat, the other dataset may give the behavior of malicious activities more deeply.

In this benchmark, we used a query across the two Hive tables, Netflow records (i.e. netflow) and suspicious ZeuS DGA domain name list (i.e. zeus_dga_result), as depicted in figure 8. The query tries to find traffic flows that communicates with the Command and Control (C&C) server detected by a DNS traffic scan using JOIN operation of Hive and Presto-db. The netflow table for one-month traffic has about 757,144,720 records while the zeus_dga_result table has 2,171 records. Since the size of the netflow table is relatively big and a query takes a long time for the JOIN operations, we carefully looked at three different file formats available for the Hive table, which are `TextFile`, `SequenceFile`, and `RCFile` (Record Columnar File) [11], and observed the variance of response time.

```
SELECT netflow.* FROM netflow
   JOIN zeus_dga_result ON (zeus_dga_result.c2c_sv =
   netflow.sa AND zeus_dga_result.dt=netflow.dt);
```

Fig. 8. Steps for an analysis module to seek suspicious flows through multiple data sources.

Figure 9 represents the execution time of the query by Hive and Presto-db as a function of the amount of data processed

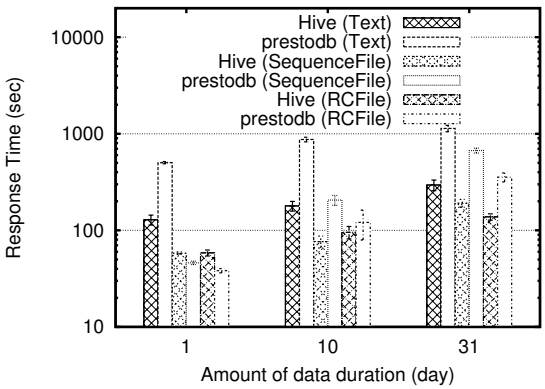


Fig. 9. Performance of SQL JOIN queries on various internal formats of Hive.

(i.e., days), along with the standard deviation computed for five replications. We measured three different types of file structure (i.e., `TextFile`, `SequenceFile`, `RCFile`) stored in HDFS.

When we used `TextFile` for the data structure, we did not have a benefit in performance since the structure is not able to correctly split the job processing across multiple nodes, resulting in a small number nodes executing the query. On the other hand, `SequenceFile` and `RCFile` are well designed for splitting jobs under MapReduce environment or Presto-db distributed SQL engine, and the performance improves.

Note that while Presto-db outperforms Hive with regular expression matching as shown in previous sections, the result is almost opposite with JOIN operation: Presto-db only outperforms one-day data with `SequenceFile` and `RCFile` structure.

V. Use Cases

In this section, we present several use cases of MATATABI as a threat analysis platform with huge amount of data.

A. Implemented Analysis Modules

ZeuS DGA detector
The first case is the detection of compromised hosts by the ZeuS botnet in an enterprise network by scanning DNS queries with a particular pattern of domain names as used in § IV-A. This module detects compromised hosts of ZeuS bot in a managed network, where a host queries suspicious domain names, based on the Domain Generation Algorithm (DGA), is considered a potential compromised host. In the case of proxied query via a DNS forwarder, we looked at traffic information filtered by the IP address of DNS answer records to identify the client IP address.

NTP amplifier detector
This module searches for Network Time Protocol (NTP) servers sending traffic with a particular packet size corresponding to a well-known NTP-amplification attack [15]. It reports the IP addresses of NTP amplifiers, and the IP address and Autonomous System (AS) number of targeted victims.

An additional module for the detector extracts NTP flows at the backbone sampling traffic (i.e., sFlow records) and lists the top ten NTP flows within a given time period.

Anomalous heavy-hitter detector
By using simple statistical tests, this modules detects IP addresses sending or receiving an abnormally high number of packets or bytes, for example, caused by DoS attacks.

Phishing likelihood calculator
This module is an implementation of a previously proposed system [12], which provides a binary detection whether a given URL points to a phishing site or not. The module consists of dataset preparation by crawling contents on pre-known phishing sites provided by PhishTank[4], analyzed by machine learning method with the help of Mahout. The dataset is updated every day since phishing sites changes frequently.

DNS amplification detector
The module tries to detect anomalous DNS traffic, causing amplification attack which fills the link capacity and makes denial of services. It looks at two different datasets, backbone sFlow traffic records and a list of open DNS resolver servers [13], and ranks top 10 speakers of DNS flow which communicates with open resolvers in sFlow datasets.

UDP fragmentation
As realizing an additional way for cache poisoning attack on DNS server based on IP fragmented packets [16], we started to observe how much traffic employed fragmented packets in the backbone traffic. The script simply extracts a record from sFlow dataset and implements a counter-based detection approach.

DNS anomaly detection
This module tries to detect anomalies of DNS response packets by adapting a machine learning method. Various statistical features such as IP addresses, the country code of DNS server, Malware Domain List[5], legitimate domain list, and the AS number of the DNS server are used for the analysis.

SSL scan detector
This module extracts SSL/TLS scans sFlow traffic data, which frequently happened right after the discovery of Heartbleed bug in OpenSSL library. The module simply counts packets destined to a specific port number, and containing the TCP SYN flag.

DNS failure graph analysis
This module tries to find suspicious non-existing domain names and IP addresses that might belong to botnets. The analysis is an implementation of an existing method, DNS failure graphs [14], based on a clustering technique.

Visualization
Figure 10 shows an example of visualization, representing a ranking of frequent asked domain names that matched with the regular expression of the ZeuS DGA, based on the result which the module generates. This visualization is implemented by using d3js[6] with the data available via MATATAPI.

[4]http://www.phishtank.com/
[5]http://www.malwaredomainlist.com/
[6]http://d3js.org/

TABLE III. ANALYSIS MODULES ON MATATABI.

Name	datasets	frequency	LoC (#lines)	remark
ZeuS DGA detector	DNS pcap, netflow	daily	25	hadoop-pcap
UDP fragmentation detector	sflow	daily	48	
Phishing likelihood calculator [12]	Phishing URLs, Phishing content	1-shot	–	Mahout (RandomForest)
NTP amplifier detector	netflow, sflow	daily	143	pyhive, Maxmind GeoIP
	sflow	daily	24	
DNS amplifier detector	sflow, open resolver [13]	daily	37	
Anomalous heavy-hitter detector	netflow, sflow	daily	106	pyhive
DNS anomaly detection	DNS pcap, whois, malicious/legitimate domain list	daily	57	hadoop-pcap, Mahout (RandomForest)
SSL scan detector	sflow	1-shot	36	
DNS failure graph analysis [14]	DNS pcap	daily	159	pyhive

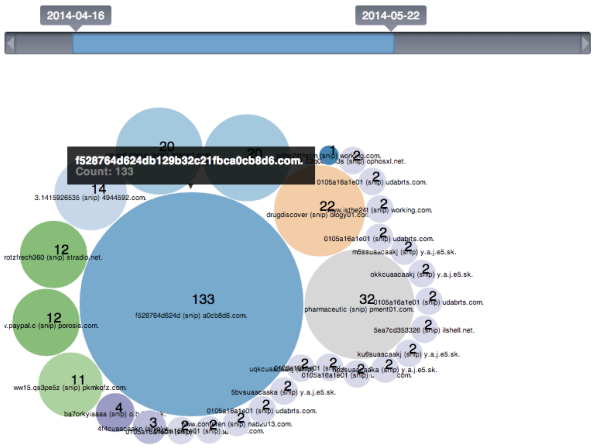


Fig. 10. Visualization of the number of DGAed queries asked.

B. Summary

Table III summarizes all the implemented analysis modules that we have come up with so far (almost for one year). Thanks to the pre-processed data by import module of each dataset and uniform programmability of MATATABI, multiple experiments have been conducted. Furthermore, the script are small and easy to implement, with most of the ranging from from 20 to 160 Lines of Code (LoC).

VI. DISCUSSIONS

Early warning on a threat: As the number of cyber-attacks grows, detection mechanisms and countermeasures against threats targeting at enterprise are becoming indispensable. The convention on cybercrime [17], which the Japanese government has signed, states that the maximum duration for preserving computer data shall be 90 days. Once a company encounters a cyber-attack, it is required to analyze in detail the information collected during the attack, contained in the data logs, deploy countermeasures for both the source and targets of the attack, and identify the range of influence. As indicated with our benchmark in figure 5, MATATABI with DNS querylog is able to process data from one month (31 days) within 500 seconds, and might be possible to process within 20 minutes if the stored data spans a duration of 90 days. The hadoop-based infrastructure allows us to increase the number of recorded information processed and gives a potential to analyze multiple data sources for a long duration, which include the target incident to be detected.

Best practice for implementing analysis module: currently Presto-db presents the best performance on simple data querying as shown in § IV, but the software is still young and has limitations on creating table onto the original database, among others. Therefore we still use Hive for such operations in the data import and analysis modules. Furthermore, the performance result of SQL JOIN operation between Hive and Presto-db suggests that Hive with `RCFile` achieves good response time in our benchmarks, even for large amount of parsed data. Calculations after the queries need different processing on data and can use available utilities such as Rhadoop, which is provided by the streaming feature of Hadoop.

Users can benefit from each method available for Hadoop depending on what the analysis modules require.

The adaptability of open source tools: The open source tools that we incorporated into our system, introduced extended functionality during initial deployment, without requiring much effort to be invested during the software development. Indeed, it requires to modify such software not only fixing bugs in the original one, but also optimizations for our purposes to handle the huge size of data[7].

Using a combination of Apache Hadoop, Hive, and Facebook Presto-db makes a faster deployment of our MATATABI system, but introduces difficulties in terms of operating the system. The issues we have faced so far are: 1) appropriate resource dispatch between concurrent threat detections, 2) inability to estimate each job's duration, which may keep occupying processors' resource, 3) precise access control on each job, among others. These are not originally addressed by a particular software, and require careful system operation in the end.

VII. RELATED WORK

P^3 [1] studied Hadoop platform to analyze a large amount of traffic data, with improving RIPE `hadoop-pcap` by reducing computational overhead. DDoS-Hadoop [2] extends P^3 to introduce a counter-based DDoS anomaly detection method on Hadoop. Both very early studies are our basis and gave the potential benefits of multi-terabytes traffic analysis with `pcap` and netflow data.

Li et al. [3] proposed a system to classify the host roles (i.e., clients or servers, etc) by using sFlow traffic with machine learning supported Hadoop environment. Their objective is to provide an analysis platform to detect hosts role from

[7]All of our modified and implemented modules are available at https://github.com/necoma.

measurement data in timely manner with an online system, while P^3 was focused on an offline analysis. Our proposed MATATABI is also based on an offline analysis, however it is also possible to be an online system providing faster performance to inspect packets and flows on the fly if further optimization to the data access performance would be archived.

Hashdoop [18] introduced a way to speed-up the detection of network anomalies by distributing heavy computations among Hadoop cluster nodes. It shows 15 times speedup, at maximum, compared to standalone version of detectors. Also, accuracy evaluation with MAWILab [19] report as ground truth data highlights better detection with Hashdoop. The key benefit is to use hash function during job splitting to preserve spatial and temporal structure of traffic dataset for the anomaly detection. We plan to integrate their effort into our system in near future.

VIII. CONCLUSIONS AND FUTURE WORK

We have reported MATATABI, a data collection and threat analysis platform that uses the Hadoop environment. The system has been designed to meet a set of requirements for security threat analysis, and achieves scalability with uniform programmable analysis module in a timely manner. Then we have presented benchmarks on querying stored data and shown speedups of up to 21 times (compared to a single machine), when the backend of MATATABI uses Presto-db as a data store. We have also showcased the use cases with our designed analysis modules to detect cyber threats with small amount of effort required for implementation.

Our system is running daily to analyze and detect security incidents from collected data, but there is still room for improvement on the system. At first, most of analysis with MATATABI is running every 24 hours as batch processes, but shorter period and hopefully real time analysis would be desired for certain threats. That would pose another challenge for the system's design such as faster data import rather than importing files collected by measurement sensors. Another direction for a broader system design is to integrate the threat information sharing system with the provisioning mechanism for defense, making information pipeline for more resilient mechanism on cyber-threats.

ACKNOWLEDGMENTS

This research has been supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA). The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the Ministry of Internal Affairs and Communications, Japan, or of the European Commission. This work was also supported by JSPS KAKENHI Grant Number 26330101. Thanks to all the volunteers, Romain Fontugne, Daisuke Miyamoto, Wataru Tsuda, for the development of analysis modules on MATATABI.

REFERENCES

[1] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 5–13, Jan. 2012. [Online]. Available: http://doi.acm.org/10.1145/2427036.2427038

[2] ——, "Detecting ddos attacks with hadoop," in *Proceedings of The ACM CoNEXT Student Workshop*, ser. CoNEXT '11 Student. New York, NY, USA: ACM, 2011, pp. 7:1–7:2. [Online]. Available: http://doi.acm.org/10.1145/2079327.2079334

[3] B. Li, M. H. Gunes, G. Bebis, and J. Springer, "A supervised machine learning approach to classify host roles on line using sflow," in *Proceedings of the First Edition Workshop on High Performance and Programmable Networking*, ser. HPPN '13. New York, NY, USA: ACM, 2013, pp. 53–60. [Online]. Available: http://doi.acm.org/10.1145/2465839.2465847

[4] "Apache Hadoop," http://hadoop.apache.org/, (Accessed 30th January 2014).

[5] "Apache Hive data warehouse," http://hive.apache.org/, (Accessed 30th January 2014).

[6] "Presto: Distributed SQL Query Engine for Big Data," http://prestodb.io/, (Accessed 30th January 2014).

[7] "Apache Thrift," http://thrift.apache.org/, (Accessed 30th January 2014).

[8] "RHadoop," https://github.com/RevolutionAnalytics/RHadoop/wiki, (Accessed 30th January 2014).

[9] "Large-scale PCAP Data Analysis Using Apache Hadoop," https://labs.ripe.net/Members/wnagele/large-scale-pcap-data-analysis-using-apache-hadoop, (Accessed 30th January 2014).

[10] C. Polska, "ZeuS P2P+DGA variant mapping out and understanding the threat," http://www.cert.pl/news/4711/langswitch_lang/en, (Accessed 30th January 2014).

[11] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu, "Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, April 2011, pp. 1199–1208.

[12] D. Miyamoto, H. Hazeyama, and Y. Kadobayashi, "An evaluation of machine learning-based methods for detection of phishing sites," in *Advances in Neuro-Information Processing*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5506, pp. 539–546. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02490-0_66

[13] Y. Takano, R. Ando, T. Takahashi, T. Inoue, and S. Uda, "A Measurement Study of Open Resolvers and DNS Server Version," in *Internet Conference 2013*. IEICE, 2013.

[14] N. Jiang, J. Cao, Y. Jin, L. Li, and Z.-L. Zhang, "Identifying suspicious activities through dns failure graph analysis," in *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, Oct 2010, pp. 144–153.

[15] C. Rossow, "Amplification hell: Revisiting network protocols for ddos abuse," in *NDSS Symposium 2014*, Feb. 2014, pp. 224–232.

[16] A. Herzberg and H. Shulman, "Fragmentation considered poisonous, or: One-domain-to-rule-them-all.org," in *Communications and Network Security (CNS), 2013 IEEE Conference on*, Oct 2013, pp. 224–232.

[17] C. of Europe, "Convention on Cybercrime," Nov. 2001.

[18] R. Fontugne, J. Mazel, and K. Fukuda, "Hashdoop : A mapreduce framework for network anomaly detection," Toronto, Canada, 2014.

[19] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab : Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *CoNEXT '10*, Philadelphia, USA,

Collaborative repository for cybersecurity data and threat information

Jean Lorchat
Internet Initiative Japan
Tokyo, Japan
jean@iijlab.net

Cristel Pelsser
Internet Initiative Japan
Tokyo, Japan
cristel@iijlab.net

Romain Fontugne
National Institute of Informatics
Tokyo, Japan
romain@nii.ac.jp

***Abstract*—In this paper, we attempt to show how to build a collaborative repository for cybersecurity data and threat information by building on top of a privacy-aware storage system: Tamias. We set the following goals: allow data sharing with a very high level of control over the sharing scope, enhance collaboration of entities that may not know each other but deal with similar threats, and manage different levels of trust between each parties. These levels of trust will define how much information is shared with each entity.**

I. Introduction

The Internet has radically changed the way that people communicate. By providing a worldwide, almost unregulated, information exchange system, it has made possible a large number of innovations, with impact comparable to that of the first newspaper, radio or television broadcasts.

However, the largely unregulated nature of the Internet and the fact that users and corporations alike interconnect their own information systems to this network, has also made it a playing ground of unprecedented size for criminals. Certainly, there are existing countermeasures. But similarly to the arms race existing within the military industry, better defense tools are soon followed by more sophisticated threats.

Of course, many entities are devoted to the study of these threats and the gathering of cybersecurity-related information. However, because of the trust issues raised by the impersonal nature of communications on the Internet, these entities hardly exchange any sensitive data. A Cybersecurity Emergency Response Team (CERT) might provide information about an attack against its network, after redacting the details of the exact targets. By doing so, it makes sure that malicious parties don't try to exploit those victims further. But this might also prevent other operators from getting crucial information about potential *victims-turned-sources* of subsequent attacks.

Also, in many cases, datasets collected by entities monitoring security information are very large. Within this data, one must look for both known and unknown threats. Each entity might have different algorithms to spot these threats, but given the large amount of data, it is likely that some attacks are not detected. So even sharing aggregated information about detected threats might be insufficient in certain cases.

Finally, defense against cyber-threats requires collaboration among defense entities because it is hard to get a global view from within a single entity. But parties might not want to share all their data with all their peers, for the trust reasons we have mentioned previously, lowering the value of the collaboration.

In this paper, we will have a look at the existing threat data exchange systems. We will then show how the Tamias [1] distributed storage system can be leveraged to build a collaborative repository for cybersecurity data and threat information. We will first introduce the Tamias privacy-aware distributed storage system, then detail each of the principles of the system and how they can be adapted to build our repository: identities, stored objects and federation.

II. Related Works

The need for sharing information in the context of cybersecurity has always been evident. In this sense, there are numerous recommendations and a few designs that try to tackle this problem.

In [2], several suggestions are described for efficient data sharing among CERTs, however the approach favored by the authors is to use secure messaging systems such as PGP. While this is an obvious choice for privacy and trust, it is cumbersome to exchange large amounts of data. Also, it does not allow to revoke access when the threat is gone.

Another work [3] known as *fordrop* (standing for Forensics Dropbox) proposes an architecture for a social network about threats that is based on the XMPP [4] and ActivityStreams [5] standards and allows participants to publish information about malware detected in their networks. However, this is especially restricted to non-sensitive information, so that it does not have to deal with trust and privacy issues.

On the other hand, standardization organizations have also started working on sharing threat data, in order to define exchange formats that can be used by all participants. In the IETF, there has been work on the IODEF [6] format for incident description. This standard only defines the exchange format, and thus does not specify how various entities can actually exchange information. At the ITU-T, work is still ongoing on the CYBEX [7] X.1500 standard, that will include the definition of both the exchange format and overall architecture.

III. The Tamias Distributed Storage

Tamias is a distributed storage system built on top of Tahoe-LAFS [8]. It adds identity and access capabilities management, fine-grained sharing, access capability scoping, and identity-related services. Tahoe-LAFS consists of a peer-to-peer network of storage nodes. Each storage node hosts indexed buckets of encrypted data. Prior to storage, objects are encrypted on the local client with the key being based on the hashed contents. In this system, knowing the storage

index allows to retrieve encrypted contents, and knowing the key allows to decrypt the object. In this Tahoe-LAFS system, access is granted by sharing a capability consisting of the storage index and encryption key. Thus the name of Least Authority File System, because knowing the access capability grants the right to share it further automatically.

In order to scope those access capabilities, Tamias introduces an identity for each storage user. This identity is made of a private/public keypair. The public key is shared with other parties and allows to authenticate messages received, as long as a proper introduction was made beforehand. Then, this public-key is associated with each bucket that the storage client uses. Furthermore, the storage server will not serve a block to anyone else than the genuine client, or a client that can show an access authorization signed by the actual owner of the bucket.

This access authorization is made of several pieces of information, such as the storage index to which the authorization relates, an expiration date, and a target identity. The whole is protected by a signature, precluding forgery. By using these access authorizations, it is possible to share different levels of information to different groups, at various trust levels. For example, partners bound by a NDA might receive access authorizations for raw data, whereas occasional partners only get anonymized data, and another lower level of trust would only gain access to aggregate results.

Finally, in order to ease the identity bootstrapping process, Tamias provides a globally writable object known as the phonebook. While everyone can publish to the phonebook, each entry is signed by the user public-key. It is thus possible for the entity to publish information about itself, such as identity details, website, and so on. Actually, any kind of information could be posted to the phonebook, since the phonebook leverages RDF [9] to provide semantics.

The phonebook is an example of service that leverages identities. There are other that are directly integrated with the Tamias client. The inbox is another such service. For example, if user Arthur and user Brutus trust each other, Arthur can create an inbox dedicated to Brutus, and provide him the access authorization to this object. Brutus can then write to Arthur about new access authorizations, or any other RDF-based information. The last integrated mechanism leveraging identities is the public inbox. This is an inbox that is published by Arthur in the phonebook and is writable by anyone. It allows users that Arthur does not trust to write to him, for example in order to self-introduce themselves to Arthur or solicit access to specific data.

Relying on Tamias' scoped sharing, identity properties and identity services, we propose to build a collaborative repository of cybersecurity data. Using access authorizations, entities can have a very fine-grained control of what is being shared, and to whom. It is also possible to use the phonebook mechanism to publish general and public information about ongoing threats and trigger collaboration with interested parties.

A. Identity Principles

We have explained in the previous subsection, that in the Tamias storage system, each participant is defined by his identity. For the purpose of building trust specifically for sharing threat data information, offline exchange seems to be the most trustworthy way. For example, if two entities already have a trusted means of communication, they can use it for this exchange of public keys.

On the other hand, in a sort of *friend of a friend* fashion, participants can introduce their trusted peers to each other. This is an integrated feature of Tamias (see Section III). Finally, we also propose to extend the phonebook for reputation building. With additional RDF grammar elements, it becomes possible for an entity to publish a recommendation about another one. By summing all those recommendations, an interested entity can evaluate the reputation of a new partner. This provides another metric for entities when choosing which partners to trust.

B. Storage principles

In the Tamias storage system, it is possible to store objects of arbitrary size. Once inside the storage, these objects can not be easily leaked to the outside. Indeed, stealing the access authorization of another entity is not sufficient to gain access. Knowing the storage index, or even the encryption key, does not grant access to the buckets.

In order to be able to fetch a block from a storage server, a Tamias client must show an access authorization that was signed by the actual owner of the file. The owner of the file is the one whose public key has been recorded with the block when it was first stored. Also, the access authorization itself has a time limit, so that any authorization eventually expires. Finally, block owners can revoke access authorizations directly at the storage server level, if they want to stop an access authorization before it expires.

In addition, since the storage network is distributed, access to the data can be much faster, provided that the distribution is consistent with respect to locality. Indeed, an increased number of storage servers brings an increased number of resources, thus making the system faster as it grows.

C. Federation principles

The global Tamias storage system in itself has been designed to work as a federation of small Tamias storage networks for scalability purposes. Using a federation of Tamias networks allows each entity to host the datasets that it collected and wants to be able to share. This way, it prevents the entity from suffering from quota limitations in the storage network.

In addition to this, entities can now trust whole networks. By doing this, each entity builds its own view of a global repository where the information sources are limited to the partners that they trust. Then, as for identities of separate entities, it is possible to recommend networks to each other, thus controlling the scope of information and the coverage of the datasets.

IV. Preparing Tamias for threat data

In this section, we introduce our proposal to build a collaborative repository based on Tamias to share threat data.

A. Data Semantics

One advantage of the Tamias storage system, is the fine-grained sharing mechanism that allows to specify which user can access which data in a very detailed way. As explained in section III, access authorizations are sent to intended recipients through their shared inbox using RDF tuples.

In addition to sharing messages, we propose to define the following messages that describe datasets. They can be sent to the intended user at the same time as the access authorization itself.

Properties
: This allows to describe the properties of the dataset itself. For example, what kind of token can be found inside: IP addresses, malware signatures, packets, URLs, time range, and more. But it could also be a reference to file types, e.g. pcap, sflow, netflow, rfc822.

Analysis results
: By attaching analysis results to the dataset, an entity can tag its datasets for easier collaboration. This way, a partner can try to look for datasets that have specific results associated, such as traces of NTP attacks, Zeus Botnet activity, etc...

Standards
: An entity will use this kind of message to specify that an object is described by an information exchange standard. It could be IODEF [6], CYBEX [7], or the n6 [10] JSON format among others.

Alternate view
: This type of message allows to offer another view of the dataset. Depending on the level of trust (see subsection IV-B) between the sharing parties, an alternate view might be the only view available. For example, it might refer to an anonymized version of the dataset.

Besides writing to intended recipients, users can also choose to publish dataset-related information directly to the public repository (the phonebook), or to a message box shared by a task-force group.

B. Trust levels

For the purpose of easy and safe sharing of threat data within our repository, we propose to define several trust levels. The granularity of those levels is defined by the user, because it depends on the kind of data and the relationships he maintains with the other users of the system. For example, let's consider a dataset of packets coming from the sampling of a link traffic. We can define the following levels of trust, from the highest to lowest:

High
: Sharing at this level grants access to the raw dataset to the recipient. However, it is not exactly similar to providing a copy of the dataset because the access authorization will eventually expire.

Moderate
: Sharing at this level grants access to an anonymized version of the dataset. All details that can identify a specific victim or person of interest in the dataset have been altered.

Low
: Sharing at this level only provides information about a summary of the dataset. For example, it could be the output of the various anomaly detectors that have been run on the dataset.

Least
: Sharing at this level will provide only information about specific threat information. For example, the IP source of an NTP attack, or a list of malware signatures that have been spotted in the dataset. This might allow to advertise the dataset to unknown parties before starting the identity exchange process.

C. Collaboration

In order to foster collaboration, it is important to go beyond the usual partners of an entity and allow entities to discover each other in the context of a specific threat.

For that purpose, we propose the creation of a distributed journal of recent information. It is a message box similar to the phonebook, but hosting information with a short lifetime. Each entity can then publish, to this journal, any information that is related to its current situation. It could be details of an ongoing threat, or request for specific information. Other entities can then look at this journal and find out if they can relate to this information.

Another important feature that would enhance collaboration is the creation of short-lived groups. These groups can be created quickly and announced to the journal described above. It allows to share directly with all the members at a given trust level. Some examples of groups might be "DNS reflection attack victims club", but also "Botnet XYZ take-down coordination center". These groups eventually disappear when the related activity is concluded.

Then, pursuing the publish/subscribe concept, we propose a journal subscription application. This application runs locally within the Tamias client and parses the journal updates to look for any kind of information about which the user has specified interest. The user can describe his subscriptions with full-text search in new messages, or using keywords in conjunction with the semantic attributes that underlie the journal itself.

D. Sharing lifetime

Finally, an important property of the Tamias storage system is that access authorizations always expire. This helps to bring a sense of safety to all the players because they can feel confident that the data can not be misused at a later date. In addition to this, we propose to extend the client revocation mechanism with triggers. Whenever an access authorization is created, the user has the opportunity to attach it to a trigger. If this trigger is activated, the client will automatically revoke access to all the authorizations involved. For example, Arthur might choose to share its raw traffic sampling traces in the context of a task force investigating NTP-amplification attacks. Upon sharing, he decides to create an associated trigger that he calls *end of the NTP threat*. At a later date, when the NTP problem is dealt with, he just has to enable this trigger,

at which point the client revokes access to the raw data everywhere it was shared with this trigger.

E. Examples

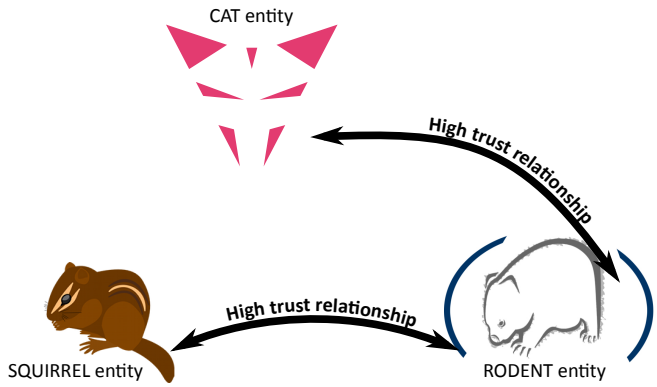


Fig. 1. Trust relationships as they stand in the first example scenario.

1) **Various trust levels and notifications:** All the principles we have detailed here are illustrated in Figures 1–3. In this example, we have three entities participating in the repository. As seen in Figure 1, the entity on top is the CAT entity, while we have SQUIRREL on the bottom left and RODENT on the bottom right. RODENT trusts both SQUIRREL and CAT, but SQUIRREL and CAT do not know each other.

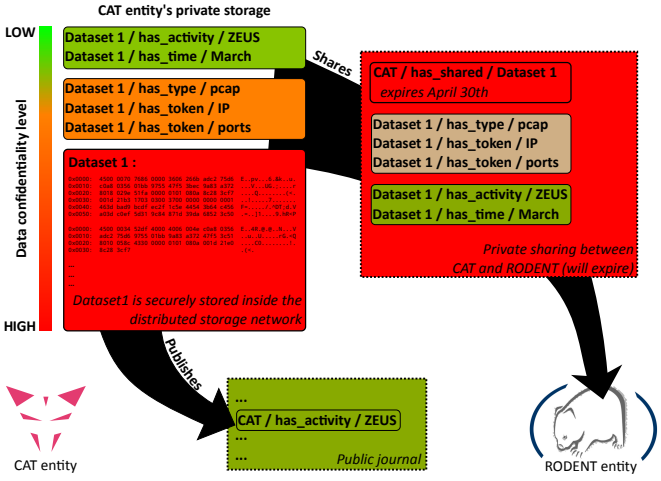


Fig. 2. The CAT entity stores Zeus-related activity in its private storage space, publishes summarized related information to the phonebook and shares it entirely with RODENT.

Now, we can see in Figure 2 that CAT owns a dataset that it shared with RODENT. Since they trust each other, CAT provided a large description about the dataset. It included the type, time coverage, a list of tokens and the results of an analysis for Zeus activity. Also, CAT decided that in order to fight the Zeus botnet, it would publish summary information informing all participants that it has traces of Zeus activity, without any details about the dataset though.

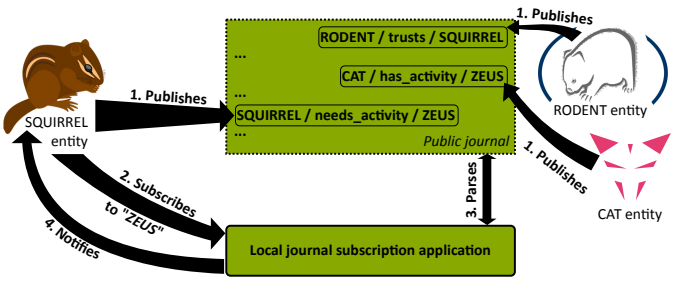


Fig. 3. The SQUIRREL entity publishes its needs, it also subscribes to Zeus-related updates and gets a notification.

Meanwhile, in Figure 3, SQUIRREL has published information about its needs. Namely, people from the SQUIRREL entity are looking at Zeus activity and are thus eager to find more information about it. They publish this requirement to the public journal. From then on, there are several possibilities. By subscribing to Zeus information, SQUIRREL gets a notification about the message that CAT has published. Otherwise, SQUIRREL might also look at the journal and realize that CAT has interesting data. Although they do not know each other, RODENT actually trusts CAT and could serve as third-party for introduction. Conversely, if CAT proactively looks for more Zeus information, it might notice SQUIRREL request and propose to establish a trust relationship, by seeing that RODENT trusts SQUIRREL.

From this example, it appears that there are many ways to take advantage of this system, but also, that the system is not fully automated. This is an important point if entities want to be sure that nothing happens outside of their control. Even though CAT and SQUIRREL have a common friend, no sharing happens before CAT decide to do so. If CAT chooses not to trust SQUIRREL, notwithstanding that RODENT trusts it, it can opt to not trust SQUIRREL nor share data with it.

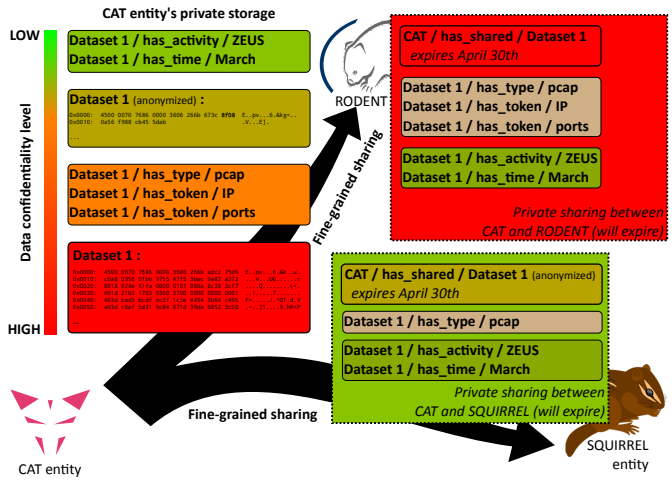


Fig. 4. The CAT entity decides it can trust SQUIRREL thanks to BADGERS endorsement, however it only grants partial access to the data by providing the anonymized version of the dataset.

2) **Anonymization and fine-grained sharing:** In Figure 4 we show how our system can be used to provide different views of the dataset to entities that have different trust relationships. In this example, we build upon the previous example and

assume that the CAT entity has decided to mildly trust SQUIRREL, after it made sure that RODENT was actually trusting SQUIRREL as well. In this situation, we see that the same dataset is shared to both entities, however SQUIRREL receives much less information compared to what RODENT already had. This is because the anonymized dataset has jumbled IP addresses and ports. For this reason, the present tokens specified by CAT are not present in the sharing message with SQUIRREL.

In this situation, we assumed that CAT already had an anonymized version of the dataset, where IP addresses and ports have been transformed to protect the actual values. In the future, it might be possible to provide application plugins that would be able to anonymize data prior to sharing with targets such as SQUIRREL that do not satisfy the required trust level for full access to the data. This kind of plugin would have to be prepared once for each kind of threat data standard though.

V. Conclusion

We have proposed a system based on the Tamias distributed storage to efficiently share large amounts of threat data. Our solution enables fine-grained sharing of threat data with a per-destination control of the amount of information provided. Access policy is based on the identity of each destination and its trust level as perceived by the owner of the threat data.

While this is a work in progress, we can already anticipate that it will address some very important problems such as provisioning trust into the threat data exchange system, limiting the scope of sharing in both time and space, helping the discovery of related datasets, and providing basic collaboration tools for information sharing.

Our future work will of course include a prototype open-source implementation based on the existing Tamias code, which will lead to more detailed work on various aspects of this specification. Nevertheless, we expect to reach interesting results thanks to the unique and powerful nature of the underlying Tamias storage system.

Acknowledgments

This research has been supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA).

References

[1] J. Lorchat, C. Pelsser, R. Bush, K. Shima, H. S. (IIJ), and L. J. (SUNET), "TAMIAS: a distributed storage built on privacy and identity," in *The 28th Trans European Research and Education Networking Conference, 21 - 24 May, 2012, Reykjavik, Iceland*, May 2012.

[2] ENISA - European Union Agency for Network and Information Security, "Detect, SHARE, Protect - Solutions for Improving Threat Data Exchange among CERTs," Nov 2013, https://www.enisa.europa.eu/activities/cert/support/data-sharing.

[3] J. Berggren, "Social CERT," The 28th Trans European Research and Education Networking Conference, 21 - 24 May, 2012, Reykjavik, Iceland, May 2012.

[4] XMPP Standards Foundation, "The Extensible Messaging and Presence Protocol (XMPP)," http://xmpp.org.

[5] Activity Streams Community, "A format for syndicating social activities around the web," http://activitystrea.ms/.

[6] R. Danyliw and J. Meijer and Y. Demchenko, "RFC5070 - The Incident Object Description Exchange Format," http://www.ietf.org/rfc/rfc5070.txt.

[7] A. Rutkowski, Y. Kadobayashi, I. Furey, D. Rajnovic, R. Martin, T. Takahashi, C. Schultz, G. Reid, G. Schudel, M. Hird, and S. Adegbite, "Cybex: The cybersecurity information exchange framework (x.1500)," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 5, pp. 59–64, Oct. 2010. [Online]. Available: http://doi.acm.org/10.1145/1880153.1880163

[8] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the Least-Authority FileSystem," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*. ACM, 2008, pp. 21–26.

[9] W3C/RDF Working Group, "The Resource Description Framework (RDF)," http://www.w3.org/standards/techs/rdf.

[10] CERT Polska, "n6 - network security incident exchange," http://www.cert.pl/projekty/langswitch_lang/en.