MAD: A Middleware Framework for Multi-Step Attack Detection

Panagiotis Papadopoulos, Thanasis Petsas, Giorgos Christou and Giorgos Vasiliadis

Institute of Computer Science,
Foundation for Research and Technology-Hellas
{panpap, petsas, gchri, gvasil}@ics.forth.gr

***Abstract*—Signature-based network intrusion detection systems (NIDS) are one of the most popular tools used to detect and stop malicious attacks or unwanted actions. However, as network attacks become more sophisticated and diversified, the accuracy of signature-based NIDS that rely only on live network traffic decreases significantly. Recent research efforts have proposed to archive the raw contents of the network traffic stream to disk, in order to enable later inspection of activity that becomes interesting only in retrospect. Unfortunately, the ever increasing network traffic and capacity make the collection and archiving of multi-gigabit network streams very challenging.**

In this paper, we review different mechanisms and techniques to efficiently store the captured network traffic to disk. We also propose an architecture that will integrate all these mechanisms into a single middleware platform that will be used by network monitoring applications in order to enhance their functionalities. Our approach will offer the ability to analyze and correlate multiple security activities, as well as, in terms of forensic analysis, to perform post-mortem incident analysis in order to asses the given damage.

I. Introduction

Signature-based Network Intrusion Detection System (NIDS) is one of the most popular tools for detecting and preventing malicious network attacks and the presence of such a tool is a cornerstone in any modern security architecture. Typically, a NIDS captures the network traffic at ingress and egress points in the network, and performs the required analysis and processing. In order to detect any malicious activity, a set of pre-defined signatures is matched against the live captured traffic.

To cope with high traffic volumes, several works have been proposed for improving the performance of Network Intrusion Detection Systems (NIDSs), either by accelerating the packet processing throughput [1]–[5], or by balancing detection accuracy and resource requirements [6], [7]. However, as network attacks become more sophisticated and diversified, the accuracy of signature-based NIDS that rely only on live network traffic decreases significantly [8], [9]. Recent research efforts have proposed to archive the raw contents of the network traffic stream to disk, in order to enable later inspection of activity that becomes interesting only in retrospect [9]. As a result, current live network stream can be combined with previously archived packets, in order to enrich the accuracy of signature-based NIDS.

Unfortunately, the challenge of collecting, analyzing and finally archiving a multi-gigabit stream is significant. Especially for 10 GbE networks, where packet arrivals can be as short as 1.25 microseconds (for a 1.5KB MTU), storing full packet traces even for a couple of hours can result to thousands of gigabytes of data. In order to reduce the space requirements, previous works use a *cutoff* limit, typically ranged between 10-20 KB per connection [9]. However, an attacker who has knowledge of the cutoff value can easily evade detection by transmitting data (or forcing the host server to send data) until the cutoff value has passed. To make matters worse, there may be cases in which a single network connection may exchange large amounts of data passing the cutoff value. For instance, web proxies that maintain persistent connections with the back-end servers will always pass the cutoff value, after serving a few clients.

In this paper, we examine different mechanisms and techniques that can be used to store the captured network traffic to disk. In particular, we propose utilizing several approaches that have been traditionally used to minimize the stored size of large volumes of data, such as compression and deduplication. We also propose utilizing more domain-specific techniques, such as aggregation and sampling of network packet traces. Our aim is to evaluate the different mechanisms that can be used to reduce the size of the required data, characterize the sustained performance, discuss the disadvantages, and show the corresponding trade-offs. All these mechanisms can be combined into a single framework that can be used as a middleware by network monitoring applications in order to enhance their functionalities.

The rest of the paper is organized as follows. In Section II we describe the notion of the multi-step attacks and the advanced persistent threats. In Section III we present the design of our system in detail, when in Section IV we discuss how we control the ever-increasing size of the stored historical traffic. In Section VII we compare MAD to previous works in the field and finally, in Section VIII we present some future directions and conclusions.

II. Multi-Step Attacks

Automated attacks, such as worms and viruses, are easy to detect using signature-based NIDS and virus scanning solutions. Besides automated attacks, there is a constant interest for detecting more sophisticated malicious actions that follow long-term steps of actions. These targeted attacks consist of multiple correlated steps in order to reach a specific target and combine several attack methodologies (e.g. drive-by downloads, SQL injections, malware, spyware, phishing, spam emails etc.) and tools (e.g. zero-day vulnerability exploits, viruses, worms, and rootkits). Unlike the traditional network attacks, which consist of an automated malicious script, in these *Multi-Step Attacks* usually exists a level of coordinated human involvement. These types of attacks are usually designed for political or economic espionage or sabotage, and are fired against governments, organizations, highly competitive companies, political activists etc.. Below, we describe the main characteristics of the most popular multi-step attack, the *Advanced Persistent Threat* (APT), we discuss its phases, and show a typical attack example.

A. Main Characteristics of APTs

Persistence. The main characteristic of APTs is their multi-phase nature. Critical information, as the exact location of the data, deployed security controls and the existence of vulnerabilities are not known a priori to the attacker. As a consequence, in order to steal valuable information, the attacker must first identify several vulnerabilities and overcome various security measures to finally gain access to privileged hosts and extract data from the network. Thus, detection of such threats requires a different approach, not constrained by the observation of a single event.

Evasiveness. APTs are usually designed to evade the common security mechanisms deployed by most organizations. The attackers are able to bypass the deployed firewalls and gain access to hosts by delivering threats through commonly used protocols (like HTTP, POP, SMTP, etc.). In order to be stealthy, the intruders usually need to create and install custom malwares on privileged hosts to avoid being identified by anti-virus products. In addition, in order for the exfiltrated data to be sent out of the target network while avoiding firewalls, encryption techniques may be used.

Complexity. An APT is based on a complex combination of attack vectors targeting as many vulnerabilities as possible in the targeted organization, including social engineering, exploits sent through email, remote administration software, Remote Access Trojans (RATs), or other custom malicious software. As a result, it is very difficult for a single security measure to provide defense against all these different vectors. It is easily anticipated thus, that a multi-layer approach must be used as a countermeasure to this complex mix of attacks.

B. The APT Phases

An APT contains a number of phases that usually span over a large period of time. What follows is a brief description of such steps.

- **Step 1: Host Reconnaissance.** First, the attacker collects useful information by scanning and studying the targeted victim. This way, she makes herself familiar with the target system and conducts a full host reconnaissance to assess its security vulnerabilities.
- **Step 2: Persistent Incursion.** In the second phase the intruder launches a "low-and-slow" attack to avoid detection aiming to breach the target. More specifically, in this phase the intruder takes advantage of possible host's vulnerabilities detected in the above preliminary step. To achieve this, she uses several methods like execution of SQL injection, exploitation of zero-day vulnerabilities, usage of targeted malware, or even a mix of the above methods. If the target shows resistance, the attacker usually changes strategy and deploys a new different type of attack against the host until she finally cracks its defenses.
- **Step 3: Control, Discover, Update, Spread.** Once the attacker has successfully compromised a host, then she tries to map the network topology and the organization defenses from the inside. In essence, as a second objective, she aims to spread the infection and take control of a larger part of the organization's network. Hence, she scans for unprotected data and networks as well as vulnerabilities, exposed credentials, and paths to additional nearby resources. Since the scanning process results may contain hosts with several different hardware vendors and software versions, the attacker probably needs to update her tool-chest. To achieve this, she usually downloads additional tools to the compromised host before starting to spread the infection.
- **Step 4: Capture and Exfiltration.** The final step of an APT contains the extraction of valuable data off the target network and the total control of a number of hosts. In this phase the intruder may secretly install rootkits on key systems and access points in order to be able to eavesdrop and capture data as they flow through the organization network. Finally, the attacker sends the harvested data to a node outside the organization network. The data travel to the node either in the clear or wrapped in encrypted packets or zipped files with password protection. In that node the collected data are studied and analyzed to extract secrets and assess their value.

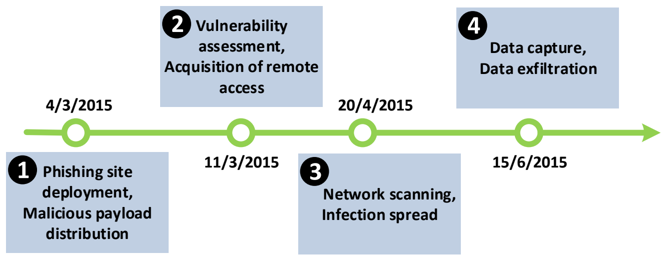
2 Vulnerability assessment, Acquisition of remote access
4 Data capture, Data exfiltration
4/3/2015
20/4/2015
11/3/2015
15/6/2015
1 Phishing site deployment, Malicious payload distribution
3 Network scanning, Infection spread

Figure 1. A simple multi-step attack example. To remain stealthy, the adversary has the different attack steps spread in the timeline.

At this point it is worth mentioning that the intruder aims to remain inside the organization and harvest information over a long-term. As a consequence, all of her actions are designed to avoid detection at all cost and make the analysis of the malware as difficult as it gets[1].

C. *Example of an APT*

Figure 1 shows a simple APT attack example. First, the attacker sets up a phishing site that includes a zero-day payload (for example: CVE-2011-0609 - Adobe Flash Player v10 vulnerability[2]) and distributes by spoofed emails his malicious link to a handful of employees at the targeted organization (step 1). After a week a user is finally deceived, she downloads the vulnerable plug-in and installs it to her workstation. Then the attacker studies the remote host that installed the plug-in in order to collect information about it and make himself familiar with its existing defenses. After that the attacker is able to exploit the vulnerabilities he found and gain remote access to the victim's host (step 2).

As soon as it gains control of the remote host, the intruder starts searching it for unprotected data and credentials. Besides local data harvesting, he also scans the nearby hosts to map the organization's internal network topology and identify strategic assets or employees with higher level privileges. This way, he locates the next target and once again conducts a full host reconnaissance to assess the victim's vulnerabilities aiming to spread the infection (step 3). In this step, the intruder may need to update his tool-chest by downloading malware and tools that will help him with the exploitation of the new host's vulnerabilities.

As soon as the intruder has successfully infected a number of workstations, he finally owns the administration keys for the organization's main data storage server. At this point, he constructs a stealth channel via encrypted files over FTP to transfer the data from the server to a host he controls out of the target's network (step 4). Finally, in the external host the valuable, exfiltrated data gets analyzed, and the intruder exploits them in every possible way (by blackmailing, selling secrets business competitors etc.).

III. The MAD System Design

In this section, we describe the proposed design of our system, called (MAD). MAD is a middleware platform for archiving the retrospective traffic, as well as for performing queries on the archived traffic. Similar to other typical network traffic analysis systems, the architecture can be broken into the following main subsystems: packet capturing, indexing, and storage management. These subsystems are implemented in MAD through four components shown in Figure 2 and described below in more detail.

Packet Capturer: The packet capturing component is responsible for tapping the network link, monitor the traffic and filter the received packets. It primarily consists of a thread that uses a network packet capturing library, such as the *libpcap*, in order to access the full packets of the monitored traffic. All captured packets are passed to the upper processing layers first

[1]There were incidents in the past like Hydraq Trojan [10] that the attackers used obfuscation techniques to keep themselves hidden from the victim.

[2]http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0609

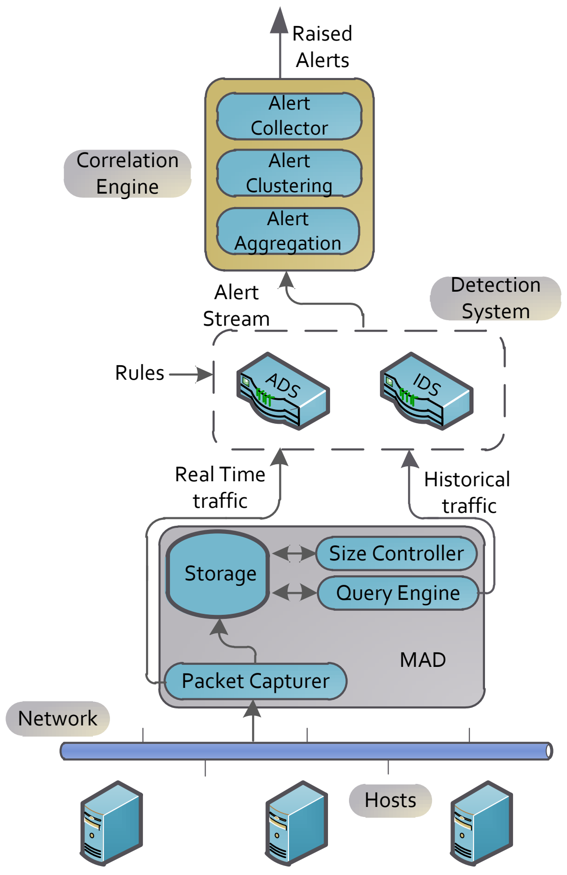

Figure 2. The MADś system overview. It consists of four main components: Packet capturer, Query engine, Size Controller and Storage. It aims to help traditional detection systems correlate the historical traffic with the real-time traffic in order to successfully detect sophisticated multi-step attacks.

— where the detection system lays (that can either an anomaly detection system or an intrusion detection system) — and then to the storage component where they are archived.

Query engine: As our second component, we design a query engine, which is responsible for the communication between the storage component and the coupled IDS. In particular, the query engine contains a listener that responds to every GET-request originating from the upper level detection system. Thus, it is achieved in a three-steps procedure, first by translating the detection system's GET-request into the corresponding SQL query, then by applying that SQL query to the storage's packet indexer and finally by sending the results back to the detection system. As a result, the detection system, which is configured with the appropriate rules by the network administrator, is able to combine the historical traffic with the direct real-time traffic that originates from the query engine and the packet capturer respectively.

Correlation Engine: The detection of attacks, similar to these discussed in Section II, require correlation of both real and historical traffic. As such, our approach will be used alongside with a correlation engine as presented in [11] or [12]. The Correlation Engine is an optional component able to correlate the intruder's attack steps by linking NIDS alerts. The linked alerts are two alerts that the postcondition of the first is a necessary precondition of the second and this precondition was not satisfied before the first one. In other words, the first attack step that corresponds to the first alert prepares the execution of the next attack step that corresponds to the second alert. Hence, if the NIDS raises an alert for illegal access of an attacker in a machine that runs a vulnerable service, the Correlation Engine by processing MAD's output could identify that this alert is linked with a past alert that corresponds with a remote buffer overflow that made the way for that following illegal access.

Size Controller: The historical traces reflect the knowledge of the detection system regarding the network actions that happened in the past. The more knowledge MAD provides to the detection system, the more accurate the multi-step attack detection procedure will be. Thus, the necessity of storing a large volume of historical traffic is apparent. Although, since the available storage capacity is not inexhaustible, it's very crucial for the MAD to apply an efficient storage size management plan. The size controller is the component that makes the appropriate actions to achieve the highest possible storage size reduction for the stored traces. In Section IV, we discuss this controller's functionality more thoroughly.

Storage: The Storage component is the MAD's core component and consists of three blocks as shown in Figure 3. The first block named *Headers Database* is responsible of the indexing procedure. In this procedure we index the packets by storing the fields of their header in a relational database. Besides packet's headers in the database we keep a pointer to its full payload stored in the second block as well. The second block consists of a RAID-0 volume where packets payloads are stored in a serialized form, grouped by flow. The last block named *Dispatcher* receives the monitored raw packets from Packet Capturer and after separating the headers from the payload, it stores them in the appropriate block. In addition Dispatcher is responsible of responding to the GET requests of the query engine. To conclude, the Storage component is responsible to store and more important maintain a large volume of historical traces in order to respond fast to search queries regarding this volume.

IV. Storage Size Management

As stated earlier, the more "knowledge" MAD maintains, the more powerful the Correlation Engine will be and as a consequence the more accurate the multi-step attack detection will become. By "knowledge" here we define the historical traffic retrieved from the network. This traffic consists of several network traces that include several events. Considering that some of these events may constitute specific steps to a sequence, able to end up to a multi-step targeted attack, it is easy to anticipate that maintaining such network history knowledge is a point of paramount significance in our system, which results in storing large amounts of network traces.

The fact that we have to keep a large volume of data on disk for a long time, creates the necessity of controlling the size of our Storage component. As a consequence, in Size Controller component we use several mechanisms to reduce the size of data to the minimum possible. These mechanisms include

compression for storage space reduction, deduplication for eliminating duplicate or redundant information, cutoff to the streams that are candidate for storage, classification and finally aggregation and sampling as discussed below.

A. Compression

Compression is the most efficient and fast method to reduce the required size of a large volume of data. The most naive way to succeed this would be the use of a generic data compression approach e.g., Gzip [3]. Nevertheless, such an approach lacks of efficiency since it does not take into account the structure of data streams during compression. Both [13] and [14] present flow-based algorithms for trace compression that result in 25% of the size of the original trace. Their algorithms are based on storing compressed packet headers along with their timestamps as flow records. Although, an IDS needs to perform frequent database queries, the decompression of packet headers would add an additional overhead on the response latency. In [15], there is a study attempting to quantify the amount of information included in various types of packet traces and the limits of trace compression that can be achieved taking advantage of traces' joint information. The conclusions of this study present the guidelines for the development of practical network trace compression algorithms, on which we rely, in order to reduce MAD's storage size requirements.

B. Deduplication

Deduplication is a technique used for reducing duplicates and redundant data, leading to better storage utilization. There are many studies that apply deduplication techniques in order to reduce the amount of data in large data centers with a view to lowering energy consumption and achieving storage space reductions [16]–[18]. These studies follow a set of different ways to accomplish that, such as *block-level* or *file-level* approaches. In our case, where the candidate data for deduplication are packet traces, the level of deduplication has to be based on the characteristics of the network packets or flows. Packet-level elimination techniques on network links can reduce resource utilization in ISP networks by 10-50% as measured in [19]. By applying such techniques in MAD's stored traces, we are able to achieve similar storage space savings and improve the maintenance of our historical traces.

C. The Cutoff heuristic

Another way to effectively decrease the required size of the needed storage is to use selective packet discarding techniques, i.e., by discarding the less important packets of a trace or a flow. Many systems that deal with high traffic volumes at real time are used to apply a *per-flow cutoff* in order to discard the least significant packets when they are under heavy load [9],

[3]http://www.gzip.org/

[20]. In [9] the authors show that due to the “heavy-tailed” nature of Internet traffic, one can record most connections in their entirety, yet skip the bulk of the total volume, by only storing up to a (customizable) cutoff limit of bytes for each connection. As a result, by employing a cutoff of 1020 KB per connection, it is possible to store the raw high-volume traffic of several days, without losing the complete record for the vast majority of the connections. Although the attacks that our proposed system tries to detect are differentiated from the traditional ones, as they consist of multiple steps (flows) of separate time periods that needed to be correlated, a cutoff would make sense in our system too. A per-flow cutoff would also be applied in our collected historical traces, as the interesting parts of the different flows that may constitute a multi-step attack intuitively seem to be hidden in the first packets of each flow. We are planning to investigate further this assumption by performing experiments similar to this study [21] on traces that contain multi-step threats.

D. Aggregation and Sampling

To decrease our storage requirements we also use data aggregation and sampling techniques. Both techniques have their limitations. For example, aggregation is effective but it requires the traffic features of interest to be known in advance. Similarly, sampling techniques are effective but they are based on the selection of a representative set of packets uniformly over the collecting data period. Many aggregation techniques, especially on historical traces, have already been proposed and were useful in our approach [22]–[24]. Moreover, sampling techniques have been proposed [25]–[28], which are common in high-end routers due to the limitations of their available storage and processing resources. Techniques like these of *NetFlow* [29], which is based on traffic rate prediction to adjust the sampling rate properly, can be applied to our historical traces. In addition *RRDTrace* [30] seems to be another ideal candidate that could supply our system with sampling. RRDTrace stores raw network packets in fixed-size disk space for arbitrary long periods, while preserving the most recent packets with many more details. Thus, summaries could be constructed for our old historical traces that contain the first parts of a multi-step attack, while the most recent information that will trigger the alerts can be saved in more detail.

E. Classification

We also classify the traffic according to the content of the packets payload it contains, in order to filter out portions of traffic that does not need to be stored. For example, if we are interested in detecting multi-step attacks that target specific applications, we can discard the rest non-suspicious traffic. This can be achieved through applying traffic classification techniques like the ones presented in [31]–[33]. Moreover, a recent study showed that discarding specific traffic like P2P traffic could also improve the accuracy of various anomaly detectors [34].

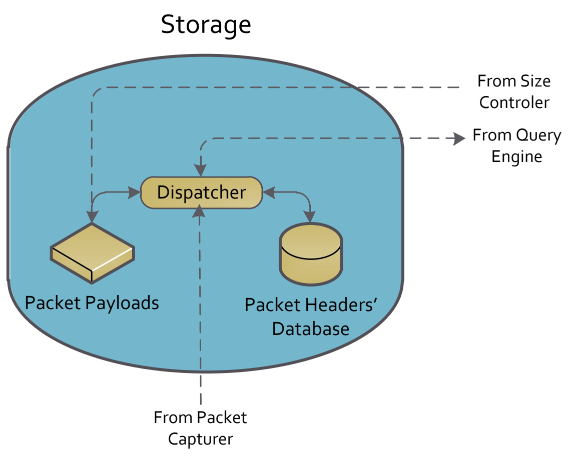

Figure 3. The Storage component. The dispatcher, in direct communication, with the Packet capturer gets raw packets and separates their payload and headers before storing them in two different blocks. It is also responsible of marshaling the GET requests of the query engine

F. Mixing Different Mechanisms

Finally, we offer the opportunity to combine any of the above different mechanisms together. As a result, users can have the benefits of many approaches, based on their needs. For instance, it is possible to first use classification to filter-out non-interesting traffic, and then use deduplication to store the remaining traffic to disk. Finally, there are also several other existing mechanisms able to further-improve the performance of MAD, which are already presented in other existing works [35].

V. Implementation

The implementation of MAD middleware will be based on an SQL relational database for storing the incoming network traffic. More specifically, the database is responsible of separately storing the headers and the payloads of the captured packets, in two different tables. As soon as a packet arrives a new row for SQL database will be created containing two columns: (i) a file ID pointing to a locally stored random access file and (ii) the offset in the corresponding file where the payload is located. To achieve high throughput and avoid packet losses, both SQL queries and payload writes will be batched. As a result, SQL insertion queries will performed per thousands of incoming packets.

The input of the database is passed through the Packet Capturer component of MAD, which sniffs packets from the network and forwards them to both Storage and Detection system. The latter is able to combine both real time and historical traffic performing this way an efficient retrospective analysis. In our prototype, as a detection system we utilize the popular rule/policy driven and signature-based, network intrusion detection system (NIDS) of Snort [36].

VI. Discussion

Besides the main task of maintaining the required historical traffic needed for the NIDS retrospective analysis, MADcould also be able to further-improve the accuracy and effectiveness of the detection system. For instance, MADwould be able to counteract against situations of packet drops that occur when NIDS/NIPS operate under heavy load [37], by storing the incoming network traffic streams into the disk. In addition, MAD would enable NIDS to perform a better packet process scheduling by implementing prioritization. In essence, by utilizing our middleware, NIDS would be able to assign priorities to different network flows (e.g. flows originating from port 80 only). As a consequence, the CPU power would be used to process packets from first priority flows, letting the rest be requested from the Query Engine and processed during idle times.

VII. Related Work

The work most related to ours is "Toolkit for Intrusion Alert Analysis" (TIAA) [12]. TIAA is a system that provides support for interactive intrusion alerts. It utilizes a knowledge base and a database and is comprised of three components: the alert collection, the alert correlation, and the interactive analysis subsystem. TIAA is equipped with a correlation engine which produces alerts, called hyper-alerts, based on previous intrusion alerts collected by the alert collection subsystem. TIAA also provides a graphical user interface along with a set of interactive analysis utilities that can be applied in the aforementioned hyper-alerts. Finally, the analysis results are visualized and presented to the user graphically. Although TIAA appears to be very similar to what our system provides, our system is more generic and flexible. MAD is a middleware that provides a generic API allowing the easy implementation of intrusion detection systems that can take advantage of any historical data it maintains. It also provides the user with the ability to implement any IDS application on top of it easily; for instance, TIAA would be an example of an application that can be easily implemented using the MAD API.

Maier et al. present Time-Machine, a system capable of accessing past network traffic quickly in order to perform network analysis and security forensics [9]. Time Machine is built on the proof-of-principle prototype that was previously proposed in [8], increasing its performance and extending its functionality. Time-Machine stores network traffic in the most detailed form, that is packets, similar to our proposed architecture. Moreover, Time-Machine provides a remote interface that allows a user to perform queries on the stored packets. In order to reduce the amount of the stored traffic, Time-Machine applies a cut-off to each flow to be stored (storing only the its first *N* bytes). The same functionality is also supported by our system, through the MAD API, even though we apply many other mechanisms too, such as compression, deduplication, etc (see Section IV).

Apart from the aforementioned approaches that appear to be the most similar to our system, there have been several other studies that deal with recording high-volume network traffic. Antonelli et al. [38] provide a long-term traffic archive using tapes. The main drawback of this system when comparing with MAD is the lack of a real-time query engine or interface. Nonetheless, there exist many approaches in the literature that provide real-time queries engined to stored traffic archives. For example, Reiss et al. [39] introduce a system that performs declarative queries on high-bandwidth streams, as well as real-time monitoring and comparison against past history. Gigascope [40] is a stream database for network applications like traffic analysis, intrusion detection, etc., that supports SQL-like queries on stored streams in the spirit of our approach, but does not provide historical archiving. Desnoyers et al. introduced Hyperion [41] which is a novel stream archival system used to store large amounts of data streams. The indexing is based on Bloom filters and provides good query performance. Finally, Cooke et al. [42] presented a framework for archiving security data which can be transformed into flows later, along with algorithms that can be used for their gathering and storing needs.

VIII. Conclusions and Future Work

As network attacks become more sophisticated and diversified, the need for an accurate attack detection grows. The accuracy of current state-of-the-art Network-based Intrusion Detection Systems rely only on live network traffic, being this way an adequate countermeasure against attacks that follow long term steps of actions. In such attacks, called multi-step attacks, the intruder launches several actions that, although are not performed in the same time period, they are correlated and are aiming to harm a specific target. In contrast to traditional automated attacks, these attacks include also a coordinated human involvement combining several attack methodologies.

In this work we presented MAD, a middleware framework for Multi-step Attack Detection, which aims to improve the accuracy of existing signature based NIDS. MAD runs between the NIDS and the underlying network interface and supports real-time packet capturing, as well as historical network traffic archiving, indexing and querying. Our system is designed to offer both real-time along with historical traffic analysis, in order to pass the highest possible knowledge as input to the coupled NIDS. This input along with any rules, given by the network administrator, will allow NIDS to produce a more accurate alert stream, able to detect multi-step attacks that consist of a large number of steps or actions. In addition, we presented the required mechanisms that will allow the NIDS to efficiently perform queries to the archived traffic, and reviewed popular mechanisms that can be used to reduce the size of stored data.

Our future work includes the extensive evaluation of our system both in terms of performance and effectiveness, by measuring the produced false positive alert ratio. Finally we plan to make a case study by running MAD against a pool of several known multi-step attacks and measure the detection percentage it can achieve.

IX. Acknowledgments

The project is being co-financed by the European Union Seventh Framework Programme (FP7/2007- 2013) under grant agreement No. 608533 (NECOMA) and the Prevention of and Fight against Crime Programme of the European Commission - Directorate-General Home Affairs (project GCC).

References

[1] V. Paxson, R. Sommer, and N. Weaver, "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," in *Sarnoff Symposium, 2007 IEEE*, April 2007, pp. 1–7.

[2] L. Foschini, A. Thapliyal, L. Cavallaro, C. Kruegel, and G. Vigna, "A Parallel Architecture for Stateful, High-Speed Intrusion Detection," in *Proceedings of the International Conference on Information Systems Security (ICISS)*. Hyderabad, India: Springer, December 2008, pp. 203–220.

[3] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An FPGA-Based Network Intrusion Detection Architecture," *Information Forensics and Security, IEEE Transactions on*, vol. 3, no. 1, pp. 118–132, March 2008.

[4] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," in *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, ser. RAID '08, 2008, pp. 116–134.

[5] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "MIDeA: A Multi-parallel Intrusion Detection Architecture," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11, 2011, pp. 297–308.

[6] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational experiences with high-volume network intrusion detection," in *ACM Conference on Computer and Communications Security*, 2004.

[7] W. Lee, J. a. B. D. Cabrera, A. Thomas, N. Balwalli, S. Saluja, and Y. Zhang, "Performance Adaptation in Real-time Intrusion Detection Systems," in *Proceedings of the 5th International Conference on Recent Advances in Intrusion Detection*, ser. RAID'02, 2002.

[8] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer, "Building a time machine for efficient recording and retrieval of high-volume network traffic," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '05, 2005, pp. 23–23.

[9] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, "Enriching network security analysis with time travel," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08, 2008, pp. 183–194.

[10] S. O. Blog, "The trojan.hydraq incident," http://www.symantec.com/connect/blogs/trojanhydraq-incident, 2010, [Online; last accessed Nov. 2015].

[11] F. Manganiello, M. Marchetti, and M. Colajanni, "Multistep Attack Detection and Alert Correlation in Intrusion Detection Systems," in *Information Security and Assurance*, ser. Communications in Computer and Information Science, T.-h. Kim, H. Adeli, R. Robles, and M. Balitanas, Eds., 2011, vol. 200, pp. 101–110.

[12] P. Ning, Y. Cui, D. S. Reeves, and D. Xu, "Techniques and tools for analyzing intrusion alerts," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 2, pp. 274–318, May 2004.

[13] M. Peuhkuri, "A method to compress and anonymize packet traces," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, ser. IMW '01. ACM, 2001.

[14] G. Iannaccone, C. Diot, I. Graham, and N. McKeown, "Monitoring very high speed links," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, ser. IMW '01. New York, NY, USA: ACM, 2001.

[15] Y. Liu, D. Towsley, T. Ye, and J. C. Bolot, "An information-theoretic approach to network monitoring and measurement," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '05. USENIX Association, 2005.

[16] D. Geer, "Reducing the storage burden via data deduplication," *IEEE Computer*, vol. 41, no. 12, pp. 15–17, 2008.

[17] J. Wei, H. Jiang, K. Zhou, and D. Feng, "Mad2: A scalable high-throughput exact deduplication approach for network backup services," *Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on*, vol. 0, pp. 1–14, 2010.

[18] Q. He and Z. X. Li, Zhanhuai, in *International Conference on Future Information Technology and Management Engineering*, ser. FITME '10. IEEE, 2010.

[19] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: The implications of universal redundant traffic elimination," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008.

[20] T. Limmer and F. Dressler, "Improving the Performance of Intrusion Detection using Dialog-based Payload Aggregation," in *30th IEEE Conference on Computer Communications (INFOCOM 2011), 14th IEEE Global Internet Symposium (GI 2011)*. Shanghai, China: IEEE, 2011.

[21] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, "Improving the accuracy of network intrusion detection systems under load using selective packet discarding," in *Proceedings of the Third European Workshop on System Security*, ser. EUROSEC '10. New York, NY, USA: ACM, 2010.

[22] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: Existing techniques and new directions," in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '02. New York, NY, USA: ACM, 2002.

[23] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein, "Enabling real-time querying of live and historical stream data." in *SSDBM*. IEEE Computer Society, 2007.

[24] S. Chandrasekaran and M. Franklin, "Remembrance of streams past: Overload-sensitive management of archived streams," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04, 2004.

[25] B.-Y. Choi, J. Park, and Z.-L. Zhang, "Adaptive random sampling for load change detection," in *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '02. New York, NY, USA: ACM, 2002.

[26] J. Drobisz and K. J. Christensen, "Adaptive sampling methods to determine network traffic statistics including the hurst parameter," in *Proceedings of the 23rd Annual IEEE Conference on Local Computer Networks*, ser. LCN '98. Washington, DC, USA: IEEE Computer Society, 1998.

[27] E. A. Hernandez, M. C. Chidester, and A. D. George, "Adaptive sampling for network management," *J. Netw. Syst. Manage.*, vol. 9, no. 4, pp. 409–434, Dec. 2001.

[28] N. Duffield, C. Lund, and M. Thorup, "Flow sampling under hard resource constraints," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '04/Performance '04. New York, NY, USA: ACM, 2004, pp. 85–96.

[29] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '04. New York, NY, USA: ACM, 2004, pp. 245–256.

[30] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, "RRDtrace: long-term raw network traffic recording using fixed-size storage," in *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, ser. MASCOTS '10, 2010, pp. 101–110.

[31] S. Zander, T. T. T. Nguyen, and G. J. Armitage, "Automated traffic classification and application identification using machine learning." in *LCN*. IEEE Computer Society, 2005.

[32] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, 2006.

[33] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: Multilevel traffic classification in the dark," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '05. ACM, 2005.

[34] 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. Springer, 2010.

[35] A. Papadogiannakis, G. Vasiliadis, D. Antoniades, M. Polychronakis, and E. P. Markatos, "Improving the Performance of Passive Network Monitoring Applications with Memory Locality Enhancements," *Computuer Communications*, vol. 35, no. 1, pp. 129–140, Jan. 2012.

[36] M. Roesch, "Snort," www.snort.org.

[37] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, “Operational experiences with high-volume network intrusion detection,” in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 2–11.

[38] C. J. Antonelli, J. Coffman Kevin, Fields, and P. Honeyman, “Cryptographic wiretapping at 100 megabits,” in *SPIE 16th Int. Symp. on Aerospace Defense Sensing, Simulation, and Controls*, 2002.

[39] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein, “Enabling real-time querying of live and historical stream data,” in *Proceedings of the 19th International Conference on Scientific and Statistical Database Management*, ser. SSDBM ’07. IEEE Computer Society, 2007.

[40] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, “Gigascope: A stream database for network applications,” in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’03. ACM, 2003.

[41] P. J. Desnoyers and P. Shenoy, “Hyperion: High volume stream archival for retrospective querying,” in *Proceedings of the 2007 USENIX Annual Technical Conference*, ser. ATC’07. USENIX Association, 2007.

[42] E. Cooke, A. Myrick, D. Rusek, and F. Jahanian, “Resource-aware multi-format network security data storage,” in *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense*, ser. LSAD ’06. ACM, 2006.