SEVENTH FRAMEWORK PROGRAMME

Information & Communication Technologies
ICT

Cooperation Programme

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

Deliverable D3.5: Countermeasure Application - Results

Contractual Date of Delivery	November 30th 2015
Actual Date of Delivery	November 30th 2015
Deliverable Dissemination Level	Public
Editors	Youki Kadobayashi and Jouni Viinikka
Contributors	All *NECOMA* partners

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

[†] 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 and the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan.

Contents

List of Figures

1 Introduction

The main objective of workpackage 3 is to provide defense mechanisms against cyber attacks and malware. By defense mechanisms, we mean any element(s) of the protected network/system/application that can be *reconfigured* in response to an attack, coupled with the logic allowing its reconfiguration. These elements can be existing security mechanims such as firewalls, intrusion prevention systems, or anti-DDoS systems; or more generic elements, such as user directories, proxies, routers, switches, virtual machine hypervisors, and web browsers. We call such component a Policy Enforcement Point (PEP)

We previously proposed the design of several defense mechanims, both at infrastructure and endpoint layers, in the deliverable D3.4 [33]. This document reports the prototype mechanisms developed, the experience gained during the development and results on early experimentations during the validation of the prototypes.

The document begins with a reminder of the architecture we have proposed for combining the data collection, analysis, and countermeasure application in Chap. 2.

The infrastructure level mechanims are presented in Chap. 3, covering four DDoS mitigation mechanisms aimed largely at volumetric attacks currently plaguing the Internet because of amplification effects provided by design or implementation flaws in protocols and services such as DNS, NTP, SSDP and so on. The mechanisms share similarities in their basic filtering capability and granularity, but are based on different networking technologies and applicable in different locations in the infrastructure: at the target and its first upstream provider (Sect. 3.1) and (Sect. 3.2), at Internet exchanges (Sect. 3.3), or at the attack source (Sect. 3.4).

Another aspect covered by two mechanisms is threat detection and mitigation in cloud infrastructures: one for public, infrastructure as a service cloud environment (Sect. 3.5); and a second for intrusion detection inside the hypervisor (Sect. 3.6).

The endpoint level mechanisms are presented in Chap. 4, covering phishing mitigation through in-browser personalization (Sect. 4.1), protecting smartphone user against SMS fraudsters and premium dialers (Sect. 4.2), and offloading smartphone firewall functions for both URL and IP filtering to a wireless access point (Sect. 4.3).

2 Architecture

The current design of the whole system is an update and extension of the design proposed during the activities related to workpackage 1. An extensive explanation of the whole architecture was provided in the deliverable D2.1[32]. Figure 2.1 illustrates the current architecture design, which involves a rather complex system and process flow, which begins with data gathering and ends on effective exploitation of the various analysis results.

Effective exploitation happens when the results of data analyses, performed over data collected at the beginning of the process flow, are successfully interpreted and exploited to reconfigure deployed PEPs. This is the role of resilience mechanisms, as indicated in Figure 2.1. Analysis results are not directly used by resilience mechanisms but first integrated in the threat information sharing component (TISC), which retrieves information from either data stores (either internal or external) or analysis modules, and serves it to clients on request or in real time. Finally, resilience mechanisms will reconfigure deployed PEPs in order to maintain *acceptable* levels of availaibility of the protected targets in the face of disruptions.

Some examples of reconfiguration include dynamically modifying rules in a firewall filtering table, changing an SDN switch commutation table, or deactivating LDAP accounts.

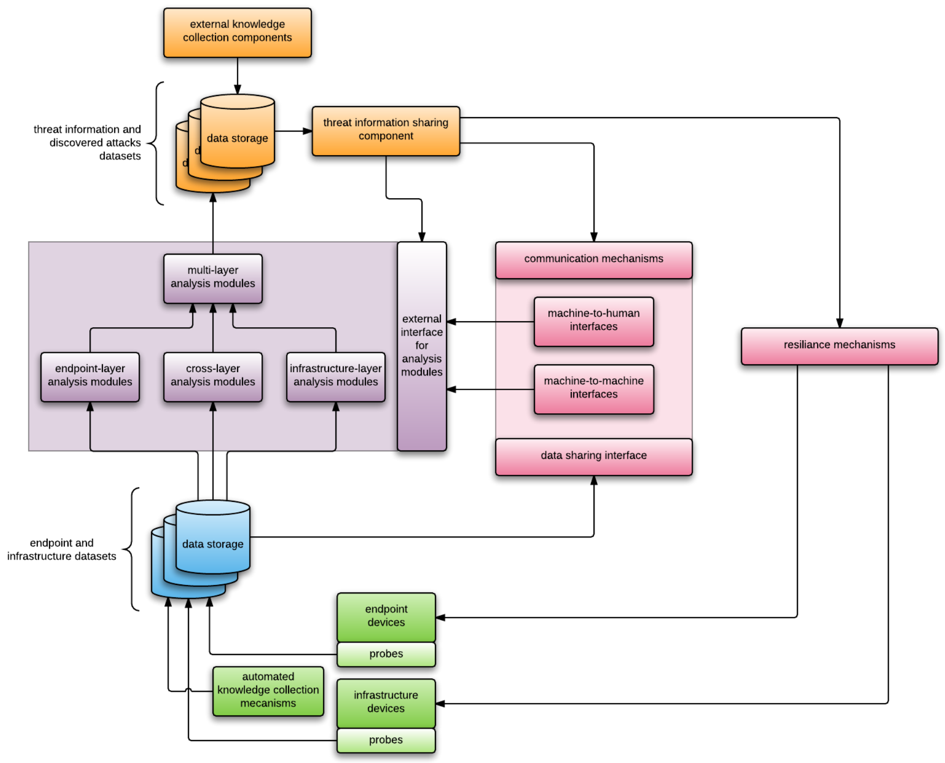

external knowledge collection components
threat information and discovered attacks datasets
data storage
threat information sharing component
multi-layer analysis modules
endpoint-layer analysis modules
cross-layer analysis modules
infrastructure-layer analysis modules
external interface for analysis modules
communication mechanisms
machine-to-human interfaces
machine-to-machine interfaces
data sharing interface
resiliance mechanisms
endpoint and infrastructure datasets
data storage
endpoint devices
probes
automated knowledge collection mecanisms
infrastructure devices
probes

Figure 2.1: Architecture Design

3 Infrastructure-level Cyberdefense Mechanisms

This chapter groups the description of prototypes for infrastructure-level defense mechanisms, improving resilience against DDoS attacks and the resilience of cloud environments.

All the DDoS mitigation mechanisms described focus on volumetric attacks. The common factor of such attacks is that they saturate the low level resources such as link bandwidth or router's packet processing capability. This saturation take place as the aggregate volume from distributed sources increases due to a funnelling effect, often close to the target. Close typically means right before the target, requiring countermeasures to be applied also upstream from the target.

- Section 3.1 describes an MPLS-based mechanism to push the defenses upstream from the choking point.
- Section 3.2 discusses the perspectives for autonomic cyberdefense provided by software-defined networking, and Sect. 3.2.3 desrcibes an SDN-based framework allowing the targeted organization to collaboratively mitigate the attack with its service provider.
- Section 3.3 describes an SDN-based Internet exchange allowing the mitigation of attack traffic on the peering links between the autonomous systems.
- Section 3.4 describes a LISP-based mitigation mechanism allowing the mitigation much further upstream, close to the attack source.

The mechanisms improving the resilience of cloud infrastructures are presented in the following sections.

- Section 3.5 describes detection and mitigation of attacks in a public, infrastructure as a service cloud environment.

- Section 3.6 describes an intrusion detection system capable of detecting attacks between virtual machines running on the same hypervisor host.

3.1 MPLS-based DDoS Mitigation

In this section, we describe a DDoS mitigation mechanism used for distributing part of the mitigation load upstream from the target.

3.1.1 Threat Description

We place ourselves in a DDoS mitigation situation, where the attack in one way or another overwhelms a defense mechanism and/or its underlying resources at a choke point, often close to the victim.

Figure 3.1 shows an example infrastructure, where a service is being protected by a *cleaning center* at the datacenter entrance and a legitimate user is able to access the service in a nominal situation. By cleaning center, we mean one or more nodes filtering out DDoS attack traffic by using access control lists, firewalling capabilities, intrusion prevention systems, DDoS mitigation systems, etc. The legitimate flow is depicted in green, and we suppose that the cleaning center is able to filter DDoS attacks to at least some extent.

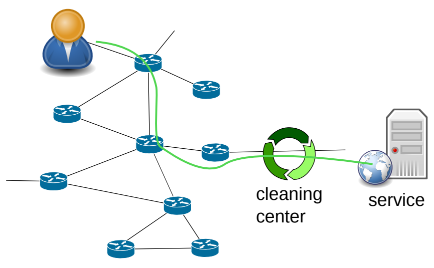


Figure 3.1: Infrastructure with routers under our control, a legitimate user accessing a protected service behind an on-premise cleaning center.

In the case of a small attack, the cleaning center and the surrounding network infrastructure is able to cope with the attack volume and to filter out the malicious packets while allowing the legitimate traffic to continue flowing. Figure 3.2 shows such a situation with the attack traffic depicted in red and the legitimate traffic in green.

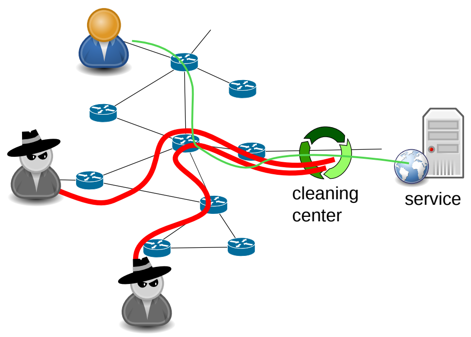


Figure 3.2: Infrastructure under a small attack, still handled by the cleaning center.

A large attack can either overwhelm the cleaning center's processing capacity, and/or the available resources upstream (e.g., router, load balancer or firewall processing capacity; or simply the available bandwidth on a network link).

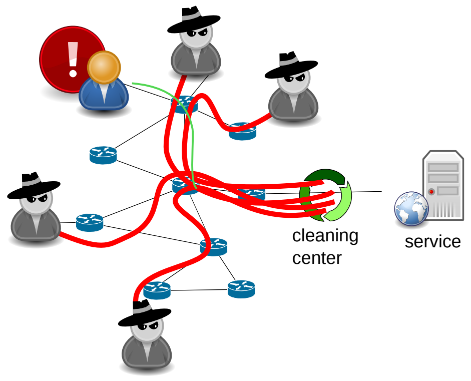


Figure 3.3: A large attack, overwhelming the network infrastructure and/or the cleaning center.

Figure 3.3 shows such an attack, where some low level processing capacity is being consumed by the attack before the attack traffic even reaches the datacenter and thus the cleaning center. One reason to such exhaustion is the funneling effect that concentrates the attack coming from distributed sources towards the target. In such a situation, the service is unavailable, no matter how efficient the cleaning center would be in removing the attack traffic, as the legitimate traffic is already being lost in transit due to congestion. Another situation we consider, but not shown in the figures, is that the network resources themselves are sufficient, but the processing capacity of the cleaning center is insufficient - the end result being more or less the same for the legitimate users and the service.

3.1.2 Proposal Overview

The objective is to find means to distribute the defense and to consume the attack at least partially before it reaches the choke point.

Figure 3.4 depicts the idea of distributing the mitigation upstream from the target, closer to the attack sources. We believe that, this way, it would be possible, for some types of attacks and in some environments, to divide the problem and prevent the attack aggregate from becoming too important to be processed.

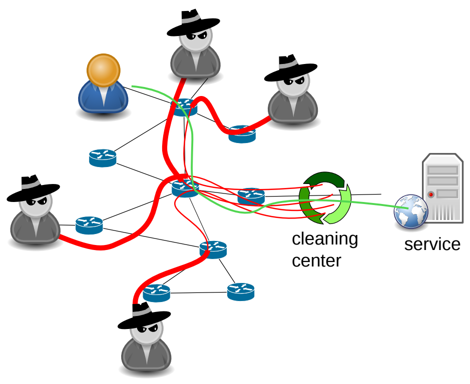


Figure 3.4: Distributed mitigation making use of upstream resources closer to attack sources.

There are different ways this type of distribution could be achieved. One approach would be the multiplication of target services in a distributed manner, e.g., by using a content delivery network to host them. Another option would be to use in some way resources available in the network core, e.g., by using several cleaning centers distributed in the core network and pulling in part of the attack traffic; or then using the network equipment in the core to fend off parts of the attack, preferably as close to the attack sources as possible and combine this with more precise mitigation by a cleaning center in front of the targeted services.

This proposal focuses on the last option. As the mitigation capacity of the core network equipment is limited to coarse-grained filtering based essentially on layer 3 and layer 4 criteria, we expect them to drop legitimate traffic as well. Thus, the objective is to use those equipment to drop only as much of suspicious traffic as necessary to avoid congestion and use the cleaning center for finer grained filtering for the remaining traffic.

We base our proposal on the MultiProtocol Label Switching (MPLS) protocol, in order for the mitigation to be executed on existing edge and MPLS routers. This routing paradigm defines paths - named Label Switching Paths (LSPs) - along the network. The ingress network router maps each packet to a LSP according to a policy, encapsulates the packet in an MPLS header and sets the MPLS label value to one corresponding to the LSP. Core routers then determine a packet's next-hop by looking at the label. A Forwarding Equivalence Class (FEC) refers to a group of packets sharing the same LSP.

We build on previous work from Hachem et al. [18] proposing an MPLS-based damage control approach. The network availability is ensured by degradation of traffic classes' Quality of Service (Qos) depending on the estimated impact. Traffic-Engineering (TE) and Differentiated Services (DiffServ) MPLS extensions are employed to generate and prioritize traffic classes. In our case, we use two traffic classes, one for suspicious i.e., supposed DDoS attack traffic, and another for legitimate flows. Instead of using an IDS to label the traffic, we seek to achieve the labeling by the routers themselves.

We set up a separate MPLS tunnel for each traffic class; use Traffic Engineering extensions to associate a bandwidth allocation for each tunnel to avoid congestion inside the MPLS network; and use the Differentiated Services extensions to define per-hop behaviors for each traffic class. In particular, these per-hop behaviors allow us to limit the resources available for suspicious traffic and thus to degrade its quality of service.

In the next sections, we present two ways of segregating suspicious traffic from the overall traffic, the first, in Sect. 3.1.3, is based on load-balancing features, and the second, in Sect. 3.1.4, on policy-based routing.

3.1.3 Approach Using Load Balancing Routing

The load-balancing feature in routers allows dispatching the packets between different tunnels according to some criteria. Our first idea was to influence the load-balancing in such a way that it differentiates between the suspicious and legitimate traffic, sending them to two different tunnels as described above. Then, with the help of MPLS Traffic Engineering and Differentiated Services extensions, we respectively degrade the QoS for suspicious traffic and ensure the legitimate traffic's QoS.

Figure 3.5 shows an example MPLS network implementing the concept. Ingress routers, called Load-Balancing Label Edge Routers (LB-LER) load-balance flows considered suspicious in tunnels represented with red arrows and flows considered legitimate in tunnels represented in green. The red tunnels used for suspicious traffic have a degraded quality of service defined with traffic engineering extensions. MPLS-wise, both types of traffic belong to the same Forwarding Equivalence Class.

When mitigation is not required, red tunnels regain some priority. Hence, Label Edge Routers are able to load-balance all FEC traffic through both red and green tunnels according to weights assigned to tunnels.

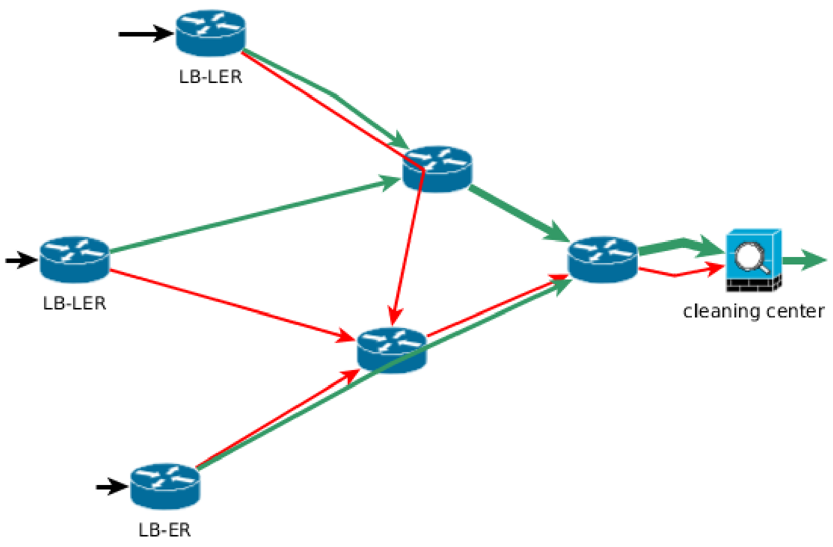


Figure 3.5: Use of Load-Balancing to mitigate attacks overwhelming the network infrastructure and/or the cleaning center.

One of our objectives is to use existing equipment for implementing the mechanism. The routers and firmware available to us (Cisco 7206 VXRs) provided load-balancing algorithms using the following parameters as input: source and destination IPs, router ID (an administrative value configured at the router), and administrative weights of available paths. The source and destination IPs being defined by the traffic itself, the remaining variables allowing us to influence the load-balancing behavior are the router ID and path weights.

Unfortunately, during the experimentations, it turned out that, in such a configuration, we are unable to control how flows are treated, i.e., to use the load-balancing algorithm to differentiate between flows we consider suspicious and legitimate.

3.1.4 Approach Using Policy-based Routing

The second idea was to use policy-based routing instead of load-balancing routing for DDoS mitigation. In policy-based routing, the routing decision is not based on routing tables, but on other criteria (i.e., the *policy*) defined by the administrator. This is described in Sect. 3.1.4.1.

The Cisco IOS versions in our test environment do not implement Differentiated Services-aware MPLS Traffic Engineering [12]. In other words, this means that bandwidth reservations for Traffic Engineering tunnels cannot be enforced with the help of built-in DiffServ-TE functions. Consequently, we need to set up a custom bandwidth policy enforcement, described in Sect. 3.1.4.2.

3.1.4.1 Defining Routing Policy with ACLs

Cisco proposes *route maps* for using other criteria than the routing table for deciding how a packet is routed. In our case, the route maps allow us to use an *Access Control List* (ACL) to match suspicious traffic.

ACLs allow the use of the following criteria to match a packet

- source IP address,
- destination IP address,
- layer 4 protocol,
- layer 4 port (for TCP and UDP), or
- any combination of the previous items,

and thus provides finer-grained control than routing tables.

For each attack requiring mitigation and for each router interface concerned (i.e., the ingress interfaces at the ingress routers), we define an ACL matching the suspicious traffic.

3.1.4.2 Enforcing Degraded QoS

To enforce a degraded quality of service for suspicious traffic, we use Cisco's *policy maps*. In a same way as route maps allow us to influence the routing decisions, policy maps allow us to map traffic policing to a Cisco *traffic class*. As with the route maps, we use the ACLs to map traffic to a traffic class, as depicted in Fig. 3.6.

We use a traffic policy that consists of two elements, first being used at the ingress router(s) and second on routers further down the path. The first element is a bandwidth-limiter for the traffic class, to ensure that the tunnel bandwidth remains within its allocated limits. The second element sets the EXP field of the MPLS header to a predefined value the MPLS domain uses to identify suspicious traffic. Such a traffic can be de-prioritized by the following routers, again by using policy maps tying the EXP field value to some mechanism, such as degrading the scheduling for packets carrying this EXP value. One limitation of this approach is that suspicious traffic is treated in the same manner for all attacks.

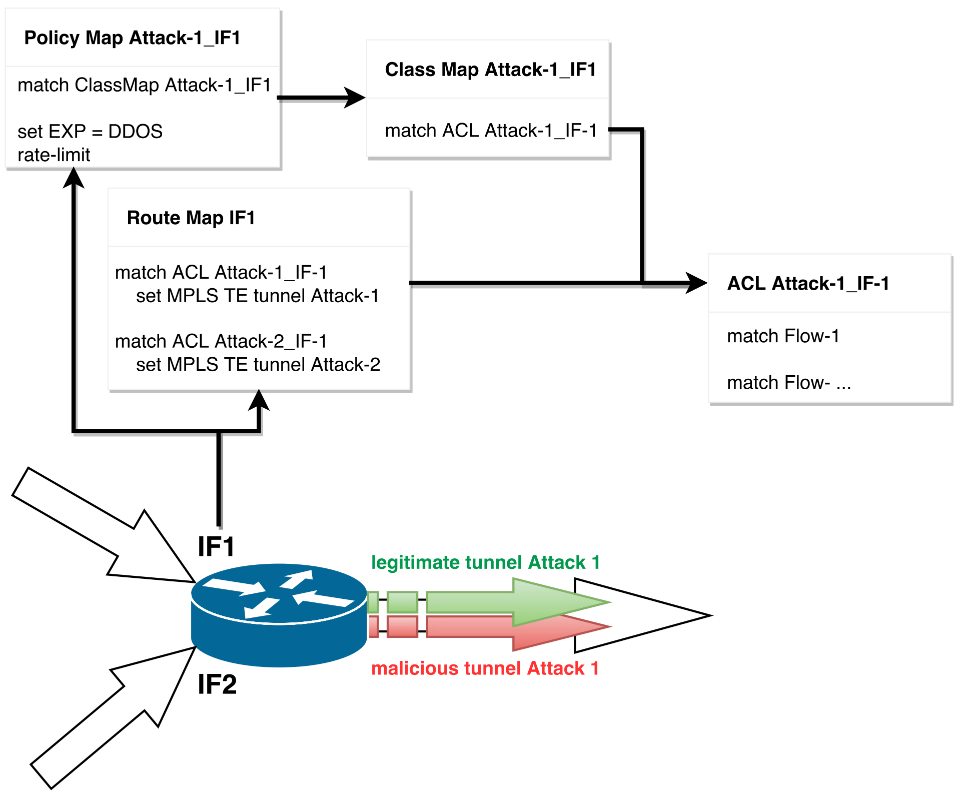


Figure 3.6: Policy Based Routing Implementation.

3.1.5 Reconfiguration Mechanism Implementation

In this section we describe a module allowing the reconfiguration of Cisco routers for mitigating DDoS attacks following the policy-based routing approach from Sect. 3.1.4.

The module has been developped in Python, requires a topology definition for the infrastructure used for mitigation, provides a set of commands allowing the deployment of the mitigation configuration on routers remotely.

3.1.5.1 Topology Model

We use a simple, JSON-based topology model for the network infrastructure used for mitigation. We currently feed the topology manually, but automated construction would be possible, e.g., starting from router configuration exports or from minimal connection information (IP, login, password) for collecting further details from the routers.

Below is an example of such a topology information in JSON format:

```
{
  "topology": {
    "routers": [
      {
        "name": "degR",
        "host": "192.0.0.1",
        "port": 23,
        "status": "up",
        "telnet_password": "cisco",
        "enable_password": "cisco",
        "interfaces": [
            { "name": "L0", "ip": "192.0.0.1/32",
              "type": "Loopback", "status": "up" },
            { "name": "G0/0", "ip": "172.16.5.2/24",
              "type": "Ethernet", "status": "up" },
            { "name": "G1/0", "ip": "172.16.3.2/24",
              "type": "Ethernet", "status": "up" },
            { "name": "G2/0", "ip": "172.16.4.2/24",
              "type": "Ethernet", "status": "up" },
            { "name": "G3/0", "ip": "172.16.7.1/24",
              "type": "Ethernet", "status": "up" }
          ]
      },
    ...
    ]
  }
```

3.1.5.2 Reconfiguration Functions

The module connects to network equipment using their remote command line interface and by interacting with the equipment by sending commands as a human operator would. It provides a wrapper for command-line access to the equipment over the network, accessible via its own command line interface or usable as a Python module. To set up a mitigation configuration using policy-based routing as described previously, the following parameters are required:

- ACL definition, e.g. a list of source IP addresses;
- the tunnel used to forward the suspicious traffic; and
- the bandwidth limit for the suspicious traffic.

The module automates the creation of the required route maps, class maps and policy maps. It can also dynamically set up MPLS tunnels based on an ordered list of inbound and outbound router interfaces through which the tunnel should go.

3.1.6 Experiments

This section describes the test environment set up for validating the implementation, in terms of testbed and traffic generation; and experimentations conducted with the module.

3.1.6.1 Testbed

The testbed used for validating the module at unit testing level is depicted in Fig. 3.7. We have chosen to use real equipment (vs. simulated environments such as GNS3[1]) to ensure that the module is really compatible with real equipment and in order to be able to have meaningful performance results.

Components Two Cisco 7206 VXR routers are placed at the ingress of the MPLS network, to handle incoming traffic. Both of them are used for the policy-based routing mitigation set up by the reconfiguration module. The ingress routers are connected to the egress router of the MPLS network, namely a Cisco 3660. A traffic generator outside the MPLS network is used to inject both legitimate and attack traffic and a traffic sink is the final destination of the traffic. There is a VLAN-capable switch providing connectivity between the routers.

[1] http://www.gns3.com/

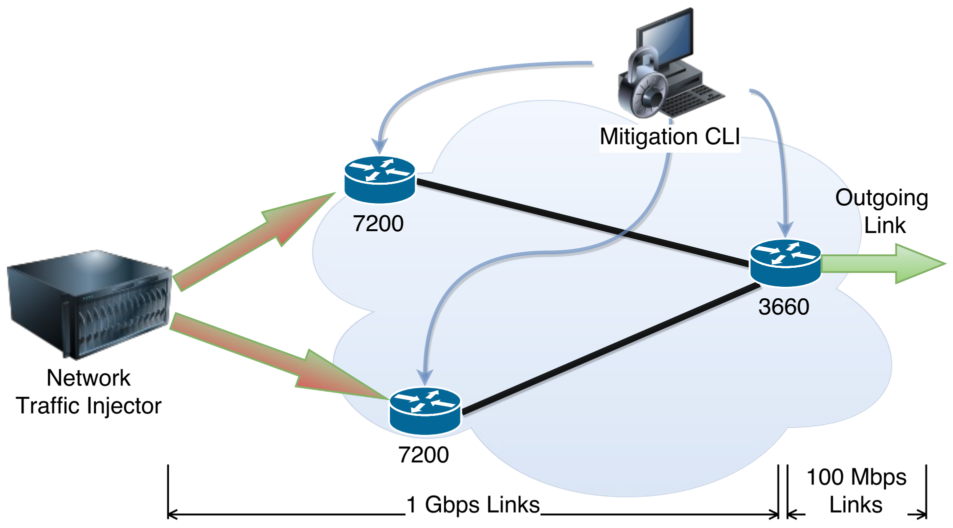


Figure 3.7: Testbed for MPLS-based DDoS Mitigation Mechanism.

L1 connectivity Each router interface as well as the traffic generator are directly connected to the ports of the switch with 1 Gbps links. The only exception is a 100 Mbps link going further from the 3660 towards the traffic sink. In such a configuration, the 100 Mbps link can largely be overwhelmed by the volume of traffic potentially coming through the two 1 Gbps upstream connections from the the 7206 VXRs.

L2 connectivity Each link presented in Fig. 3.7 corresponds to a VLAN with corresponding switch ports assigned to that VLAN.

MPLS overlay There are two MPLS tunnels starting from each of the two ingress routers, one for legitimate traffic and another for suspicious traffic. These tunnels terminate at the egress router.

Rate-limiting policy At each ingress router, the policy-map related to the attack class-map rate-limits the suspicious traffic to 10MBps.

ACL In this experimentation the ACLs are constructed using source IP and destination IP addresses for identified attack flows. The detection / identification is out of the scope of this experimentation, but could be provided e.g., by the cleaning center and combined with DDoS / botnet related information obtained from NASK datasets through n6.

3.1.6.2 Traffic Generation

We generate test traffic by combining legitimate traffic and attack traffic from different captures.

In order to have realistic legitimate traffic, we use captures from the MAWI repository [15]. As we place ourselves in a situation where the cleaning center is located at a datacenter entrance and not on a backbone link (where the MAWI traffic has been captured), we extract traffic corresponding to two /24 destination prefixes from the captures. The intention is to obtain realistic legitimate traffic that could correspond to a traffic incoming a datacenter towards the hosted IP ranges.

Attack traffic is generated by cloning and modifying a response to a DNS type ANY query for the domain `isc.org` (resulting in a large, almost 4000-byte response). The responses are modified so that the destination IP corresponds to the attack target and by using a pool of source IP addresses corresponding to open resolvers that could be used in a DNS amplification attack. More precisely, for each /24 prefix selected above, we

- use 4 target IP addresses in that prefix, two of them matching destination IPs present in the legitimate traffic;
- randomly generate a given number of source IPs.

The number of attacking sources can differ for each prefix: e.g., we have been using up to 4000 sources for the first prefix and up to 2500 sources for the second. Traffic for each prefix is reinjected towards a different ingress router.

3.1.7 Results

Overall, the testing was an iterative process, which allowed to validate and stabilize the design principles and the basic features including connection to routers, reconfiguration, etc.

In order to assess the efficiency, different tests were run, where we measured the following metrics:

- inbound legitimate traffic at the ingress routers,
- outbound legitimate traffic from the egress router, and
- outbound attack traffic from the egress router.

One sample of the observed results is shown in Fig. 3.8 covering four different phases:

- before the attack (seconds 0-5);
- during the attack, before the mitigation (seconds 5-20);
- reconfiguration for mitigation (second 20); and
- mitigation active (seconds 20-)

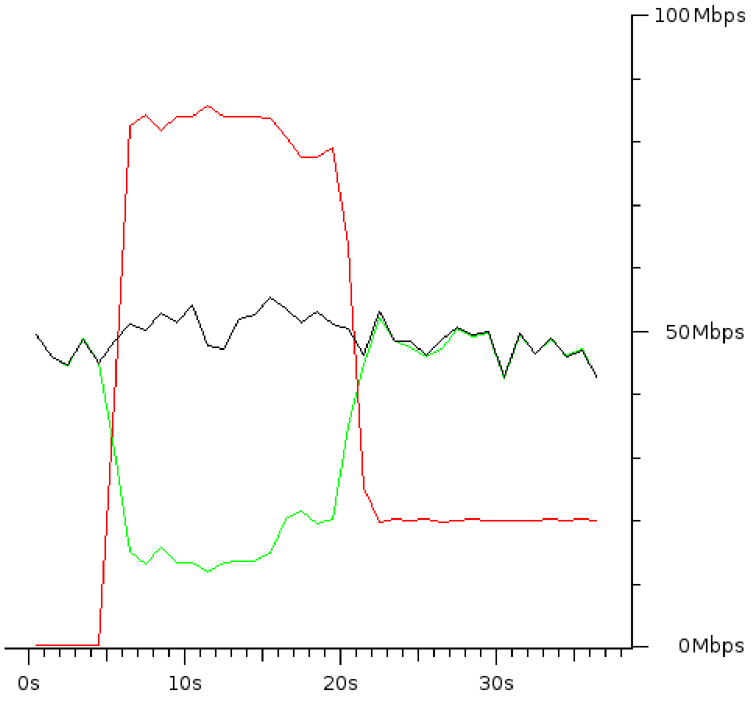


Figure 3.8: Legitimate and attack traffic volumes before the attack; during the attack; and while under attack, before and during mitigation.

and shows

- inbound legitimate traffic in black,
- outbound legitimate traffic in green, and
- outbound attack traffic in red.

The first phase lasts around 5 seconds during which only legitimate traffic is being injected. The outbound legitimate traffic is equal to the inbound legitimate traffic (around 50Mbps), without losses. During the second phase, from 5 s to 20 s, the attack traffic is being injected in addition to the legitimate traffic. For the test run shown here, we inject a total of 760 Mbps of attack traffic towards the ingress routers (around 380 Mbps per router).

At the egress router, the outgoing physical link (cf. Fig. 3.7) is congested and the malicious traffic fills over 80% of the bandwidth.

This results in link congestion at the egress router and to the loss of over 60% of the legitimate traffic. The reconfiguration for mitigation takes only few seconds during which 500 amplifiers are added to the ACLs (250 on each ingress router). The mitigation resolves the congestion. Once it has been set up, from an egress router's point of view, the traffic volume returns back to normal. As the suspicious traffic is bandwidth-limited on ingress routers up to 10 Mbps for each link towards the egress router, a total of 20 Mbps of suspicious traffic is reaching the egress router and being routed towards the destination.

Experimentations disclosed some performance limitations related to equipments on our testbed. Whereas the ingress routers are able to handle a large number of amplifiers (more than 40000 per router) flooding around 500 Mbps per router, the egress router is not able to forward more than 40000 packets per second (pps). Thus, the egress router can become a bottleneck by its packet processing capacity or by its link capacity, the shift happening around an average packet size of 312 bytes.

3.2 SDN-based Autonomic Cyberdefense

The rich history of arms race between attacker and defender has clearly indicated that defenders always trail behind attackers. One of the major root causes is that attackers always gain asymmetric advantages over defenders, e.g., in DDoS amplification attacks. Another important reason is that developing, deploying and operating in-depth cyber defense mechanisms at a large scale is always a challenging and cost prohibitive process. Indeed, we have learned lots of lessons from the DDoS research domain [30, 35, 50], in which many kinds of anti-DDoS mechanisms have been proposed. The majority failed to gain large-scale application and deployment in practice, chiefly due to labor-intensive cost and high operational complexity. Therefore, some research efforts on designing autonomic cyberdefense system have been seen in the past years. However, most of them are focused on the design of middleware algorithms and approaches, incurring additional operational complexity, as well as, scalability issues. The emergence of software-defined networking (SDN) and network functions virtualization (NFV), which are widely recognized to be promising solutions to reduce the complexity of network management [24], may bring technological advantages to implement those security middleware approaches in an efficient and systematic way.

3.2.1 Design Overview

As Figure 3.9 indicates, the designs are expected to systematically integrate the four functional modules, i.e., monitoring, analysis, countermeasure, and reaction together, and achieve *self-management* capabilities. While this framework is originally intended to lay a foundation to correlate the different *NECOMA* work packages through appropriate threat information exchange formats as well as enabling mechanisms or APIs, the work on different modules reported in the following sections can be treated as a set of

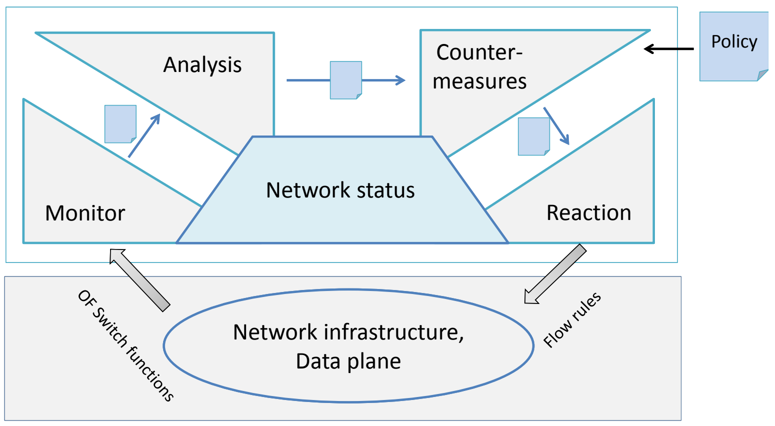


Figure 3.9: Design architecture of autonomic defense system.

stand-alone modules, covering the whole loop from detection to reaction, even if they have not yet been fully integrated together.

With the ultimate objective of developing autonomic cyberdefense mechanisms, our contributions are two-fold: (1) the implementation of basic security functions, e.g., firewalls and IDS/IPS, at the SDN controller, in order to study the feasibility of constructing multi-layered defense mechanisms; (2) the development of an autonomic defense framework, named ArOMA, for mitigating DDoS attacks.

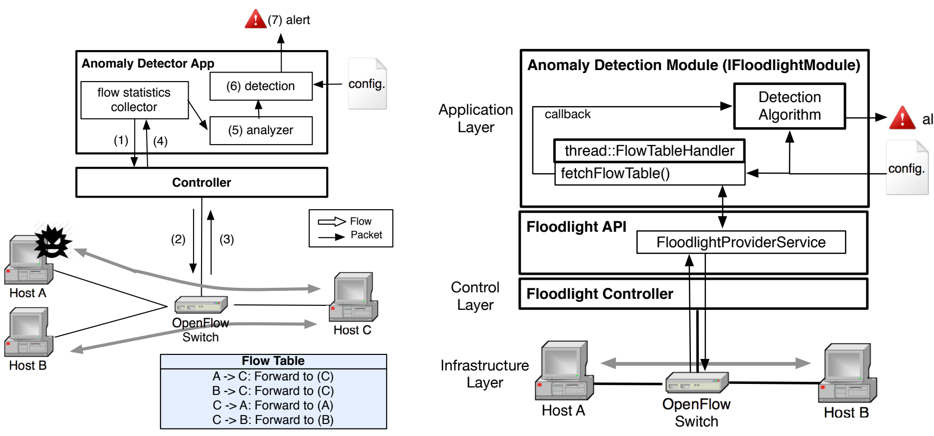
3.2.2 Enabling security functions by SDN

In [43], a security applications-oriented development framework termed FRESCO was proposed, which aims at providing a number of composable modules or modular libraries to implement basic security functions such as IDS/IPS, firewall, traffic monitor, or even customized security service. To further investigate the feasibility and effectiveness of SDN-based security mechanisms, four types of security applications have been implemented on the Floodlight controller[2], (i) in-line mode security functions, e.g., firewalls and IPS, (ii) passive mode security functions, e.g., traditional IDS, (iii) network based anomaly detection functions, e.g., scanner and DDoS detector, and (iv) advanced security functions, e.g., stateful firewall and reflector networks. Their performance has been tested on a small-scale yet practical testbed. In this Section, we selectively describe several representative functions, while the implementation details and discussions can be found in [49].

[2]An Open SDN Controller, available at `http://floodlight.openflowhub.org/`.

3.2.2.1 Anomaly-based NIDS

The advantages of implementing anomaly detection algorithms on SDN platforms have been identified in some previous work [8, 29]. Unlike traditional anomaly detection systems which rely on dedicated hardware devices or software modules to monitor network traffic and measure its statistics, SDN makes this process much easier by simply retrieving such information from the data plane. Here, we report the implementations of two simple variants of anomaly detectors, the network scan detector and the DDoS detector, both of which use traffic statistics to detect network anomalies.



Flow Table
A -> C: Forward to (C)
B -> C: Forward to (C)
C -> A: Forward to (A)
C -> B: Forward to (B)

Figure 3.10: Operational flow (left) and the Floodlight based implementation (right) of anomaly detectors as an SDN application.

Operational Flow of Anomaly Detectors as an SDN Application. Regardless of specific detection algorithms, the general operational flow of the Floodlight based implementation is illustrated in the left side of Figure 3.10, (1) the anomaly detector application initiates a request about traffic statistics, e.g., sFlow, to the controller, (2) the controller relays the request to the OpenFlow-enabled switch, (3) which then collects the data and returns it to the controller, (4) the controller then forwards the data to the application, (5) the information is extracted and represented in a certain format that can be processed by the detection algorithm, (6) anomaly detector runs, and (7) generates alerts in the presence of anomalies. Note that a database storing normal traffic patterns, which is omitted in the figure, is necessary to run the application.

Floodlight Based Prototype. Regardless of specific anomaly detection algorithms, a general prototype development framework is shown in the right side of Figure 3.10. Specifically, as the network anomaly detectors need to periodically collect traffic statistics, the OpenFlow-enabled switch is programmed to record flow level traffic statistics, such as byte or packet counts, in the flow table. Thanks to the intrinsic feature of Floodlight, this data can be easily fetched via the *FloodlightProviderService* API. To implement this, a *FlowTableHandler* thread is created to periodically request the flow table data from the controller, while the collection period can be specified via the configuration file. Then, the flow table data is provided as input to the anomaly detection module, which runs the specific anomaly detection algorithm and reports alerts in the presence of anomalies. The baseline files used by the detection algorithm can be provided via the configuration file.

As for the specific anomaly detectors, we chose to implement an *anomaly score based scan detection algorithm* [25] and a *DDoS detector*. Specifically, the scan detector computes an anomaly score for each destination port based on the collected flow table data, following the given algorithm below,

```
function scan_detection(stats){
    foreach(stat : stats){
        port_map[ stat.getDstPort() ] += 1
        counts += 1
    }
    foreach(port : port_map){
        prob = port_map[port] / counts
        learned_prob = learned_port_map[port]
        anomalyscore += -log_2[prob/learned_prob]
    }
    if(threshold < anomalyscore){
        alert();
    }
}
```

In addition, the DDoS detector simply maintains a counter of the number of bytes and packets in order to calculate the *byte rate* and *packet rate* from the collected data. The pseudocode is given below,

```
function ddos_detection(stats){
    foreach(stat : stats){
        bytes += stat.getByteCount()
        counts += stat.getPacketCount()
    }
    bytes -= previous_bytes;
    counts -= previous_counts;

    bps = bytes/time_interval
    pps = counts/time_interval

    previous_bytes = bytes;
    previous_counts = counts;

    if(threshold_bps < bps || threshold_pps < pps){
        alert();
    }
}
```

3.2.2.2 Performance Evaluation

One of the important performance metrics for evaluating the SDN applications is *throughput*, or the performance burden that could be incurred.

Experimental Setup. To evaluate this metric, three physical SDN testbeds are established, and each of them consists of an OpenFlow-enabled switch, a controller machine, and three physical hosts. In particular, three OpenFlow-enabled switches, namely, an HP 3500yl [19], an HP 3800 [20] and a Pica8 P3290 [36], are deployed respectively in the testbed for the purpose of performance comparison and to avoid any biased performance measurements due to switch-specific factors. Their specifications are listed in Table 3.1. Also, the specifications of the controller host and the other three host machines are described in Table 3.2.

Table 3.1: Specifications of three OF-enabled switches

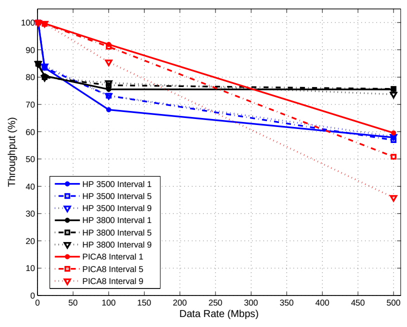
	HP 3500yl	HP 3800	Pica8 P-3290
Switch Fabric Capacity	101.8 Gbps	88 Gbps	176 Gbps
Forwarding Speed	75.7 Mpps	65.4 Mpps	132 Mpps
Latency	3.4 μs	2.8 μs	1 μs
Routing Table Size	10,000	10,000	12,000
MAC Table Size	64,000	65,500	32,000

Table 3.2: Specifications of the machines deployed in the testbeds

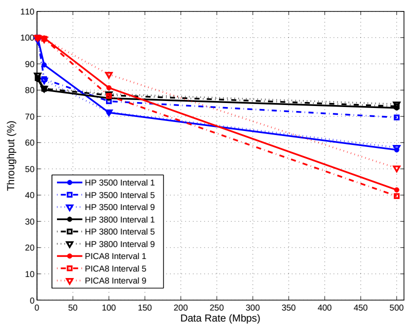
Type	NIC	CPU	RAM	OS
Controller	1Gbps x5	i5-4570	16GB	Ubuntu 12.04 64bit
Host A	1Gbps	i7-2640M	8GB	Ubuntu 12.04 64bit
Host B	1Gbps	i5-2450M	8GB	Windows 7 64bit
Host C	100Mbps	Atom N550	2GB	Ubuntu 13.10 64bit

Traffic Generation. For security applications that involve packet-matching processes, e.g., firewall, NIDS, and NIPS, we semi-randomly generated data traffic by intentionally excluding a specific port number. Meanwhile, we generated the firewall and Snort rules to include this port number in order to push the security applications to exhaustively match a packet with the whole rule-set (the worst case performance). For other applications, we randomly generated data traffic to measure the throughput. Specifically, the traffic was sent from Host A to B, and the measurement point was set at Host C.

Results and Analysis. To test the performance of two *anomaly-based NIDS, scan detector and DDoS detector*, we measured their performance in the presence of varying data rates and query intervals collecting traffic flow statistics. As indicated by the results shown in Figure 3.11a, the performance of the scan detection application was more affected by the switch



(a) Scan detection app.



(b) DDoS detection app.

Figure 3.11: Throughputs with varying data rates and query intervals.

than the flow table collection interval. This indicates that it is feasible to design practical anomaly-based IDS by monitoring the network states provided by the SDN functions. Also, as shown in Figure 3.11b, the DDoS detector performed in a way similar to what the scan detector did. Considering the two detectors used similar traffic features, such results are not surprising. It is worth noting that the computational overhead and latency incurred by the two anomaly detection algorithms were not taken into account, which essentially have no impact on the throughput. Nevertheless, it is still reasonable to conclude that depending on whether the traffic features used by the anomaly detectors are readily available or not through SDN functions (e.g., received packet counts of a flow), the performance of an anomaly-based IDS may vary a lot.

Discussions. *Is it possible to achieve autonomic (to some extent) defense mechanisms by using SDN platforms?* One of the salient features of autonomic defense mechanisms is that the complexity of management and deployment should be significantly reduced, and the intervention of security administrators should be minimized, achieving what we call *self-management*. Formally, the four essential properties of autonomic defense mechanisms are defined as follows,

- *Self-configuration*: high-level security policies can be enforced at the most appropriate points, while the rest of security components or systems can be adjusted in an automated and consistent manner.
- *Self-optimization*: the key defense elements continually improve their performance, efficiency, and effectiveness in terms of desirable performance metrics such as detection accuracy, computational overhead, and latency. More importantly, the consumed resources should bring negligible impact to the protected service infrastructure.
- *Self-healing*: the mechanism should be able to quickly identify, infer and determine the root causes of system failures so as to take appro-

priate response such as system reconfiguration, or attack signature updates.

- *Self-protection*: the mechanism is able to proactively avoid or mitigate the consequence of attacks, including the ones targeting the mechanism itself, based on the threat information exchange, correlation and analysis.

Comparing the developed prototypes shown in Figure 3.10 with the autonomic defense framework given in Figure 3.9, we can easily identify that SDN-based applications, more or less, preserve the four properties. In particular, the SDN controller provides a friendly interface which allows the security administrators to dynamically specify and update security policies, despite the fact that automated policy engines should be put in place to automatically translate security policies into policy functions that can be processed by the SDN controller. In addition to the *self-configuration* capability, the tight coupling between the data plane and flow rules via well-defined APIs, rather than monitoring sensors and manual configuration, can achieve dynamic resource allocation and significantly reduced communication overhead and latency, ideally leading to *self-optimization*. Moreover, *self-healing* and *self-protection* can be achieved thanks to the dynamic configurability of SDN enabled network devices, as well as flexible SDN applications authentication, on-demand traffic flow segregation, and trustworthy path computation [7].

3.2.3 ArOMA: Autonomic DDoS Mitigation Framework

As we have previously shown, the SDN controller provides global visibility of the network of deployed OpenFlow switches. Also, the SDN controller can dynamically configure the OpenFlow switches with rules translated from the high-level policies defined by the policy engine deployed as a controller application. This Section specifically presents our proposed autonomic defense framework ArOMA. After describing the threat model, we will outline the designs of our framework, describing the operational workflow, and finally cover some implementation details.

3.2.3.1 Threat Description

The key threat we are concerned with is the network resource exhaustion attack, in which attackers can send large volumes of traffic to exhaust the shared network resources between the ISP and the customer networks, such as link bandwidth, routers and servers. In particular, we focus on two types of DDoS flooding attacks, although other variants can be addressed as well, as long as the attacks are well characterized.

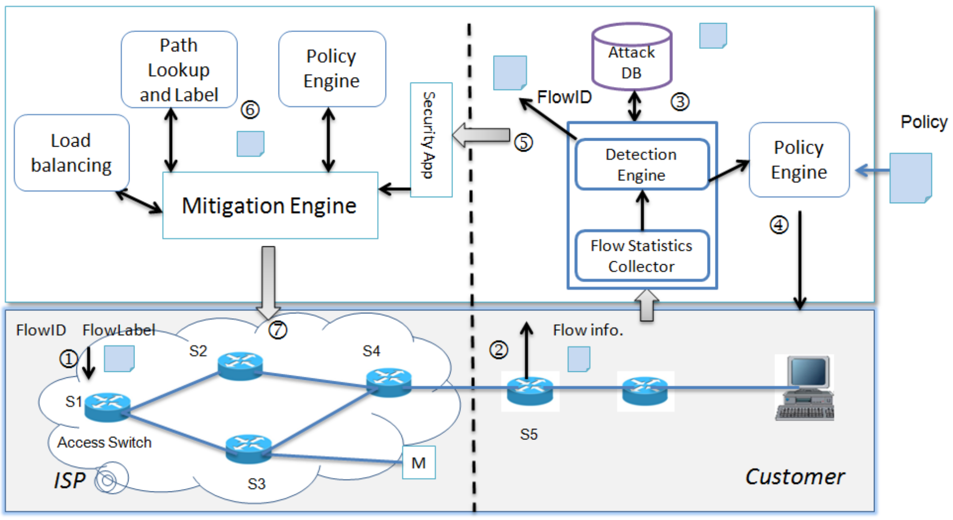


Figure 3.12: SDN enabled DDoS mitigation framework

- Destination flooding attack: these attacks target the customer network and its protected servers.
- Bandwidth flooding attack: these attacks aim to congest links in the ISP network as well as in the customer network so that legitimate traffic can not get its fair share of bandwidth. This type of attack traffic cause high collateral damage as it congests the ISP network, and disrupts the legitimate flows towards other customer networks.

3.2.3.2 Design Framework

The proposed framework ArOMA is shown in Fig. 3.12, which includes the following key features:

- DDoS mitigation and traffic engineering are provided as on-demand services to the customers, who need to subscribe to these services with the ISP beforehand. Then ISPs and customers collaboratively mitigate DDoS attacks.
- the ISP provides security functions at the application layer of its SDN controller, allowing the customers to send mitigation requests in real time.
- both customer and ISP networks have their own controller running in their networks.
- DDoS attacks are detected by a detection engine running at the customer's controller, which provides the flexibility to the customer to

develop anomaly detection (the feasibility has been studied in Section 3.2.2.1) according to their requirements. It helps the customer to deal with the traffic engineering done by the ISP with no prior knowledge, as the customer can later send the flow information causing the attack to the ISP.

3.2.3.3 Operational Workflow

As shown in Fig. 3.12, the framework is distributed across the ISP and customer networks, and the operational workflow (labeled with step numbers) can be described as follows:

1. as the traffic enters the ISP network, the controller associates a label to the flows, these labels are used for fast forwarding and rerouting the flows. The core routers in the ISP network do not require the states to be maintained as the forwarding is done based on the labels;

2. Flow statistics are periodically collected by the Flow statistics collector from the ingress switch in the customer network. We use an OpenFlow (OF) collector in the framework. To collect flow statistics, we rely on previous results [17] that showed how OpenFlow and sFlow can be used for the collection of flow statistics and the detection of anomalies in the traffic.

3. Collected flow statistics are forwarded to the detection engine for processing. The detection engine can run any anomaly detection algorithm that is most suited to its flows [17]. In the presence of malicious and suspicious flows, security alerts are generated by the detection engine, which trigger reaction from the policy engine;

4. The Policy engine in the customer network generates and install the rules at the ingress switch to process the traffic.

5. The FlowID (source IP Address, destination IP address) of suspicious (flows which are causing congestion but may be legitimate) and malicious (attack traffic) flows are encapsulated into security requests made to the security application at the ISP's controller;

6. The security application forwards the received FlowID (source IP Address, destination IP address) from the customer controller to the mitigation engine at the ISP controller.

7. The mitigation engine then communicates with the policy engine and path lookup module and redirects the flow to the appropriate middleboxes for processing.

3.2.4 Implementation and Evaluation of ArOMA

This Section describes the implementation and experiments of ArOMA on DDoS mitigation.

3.2.4.1 Implementation Details

Controller to controller communication. The communication between the controllers is done for the purpose of mitigating the attack. More specifically, the process to notify the ISP controller to start DDoS mitigation is indicated as Step 5 in Figure 3.12: the *Detection Engine* accesses the *Security Application* via a REST API and sends the `FlowID` of the malicious flows to the ISP controller, which then starts the mitigation.

Flow path reconfiguration at the ISP Controller.

The components at the ISP controller need to communicate in the following way,

1. when the `FlowID` is received by the *Security Application* at the ISP controller, it forwards the `FlowID` to the mitigation module;

2. on receiving the `FlowID`, the *Mitigation Engine* forwards it to the *Policy Engine* to update the policy associated to the detected flow (identified by the `FlowID`, and then it selects the functions most appropriate to process the flow (in the form of *middleboxes*). The *Policy Engine* maintains information about the security policies and the relevant middleboxes to process the corresponding flows;

3. when the *Mitigation Engine* receives the policy specifications from the Policy engine, it requests the *Path Lookup* module for selecting a path appropriate to enforce the policy and which contains the middlebox. The *Path Lookup* module is responsible for maintaining a list of paths spanning from ingress switch to egress switch and associates a label with each path. The matching label for the selected path (enforcing the policy) is returned;

4. the *Mitigation Engine* modifies the Flow Label to be inserted at switch S_1 in the 12 bit VLANID field of the packet, hence redirecting the flow.

3.2.4.2 Experiments

The purpose of our experiments is to validate the feasibility and effectiveness of our proposed autonomic defense framework ArOMA on DDoS mitigation in terms of performance metrics.

Platform. The topology of our experimental platform is similar to the data plane configuration shown in Figure 3.12, where the ISP and customer networks are managed by their respective SDN controllers. The components and specifications of the platform are given in Table 3.3. For simplicity, two routing paths are configured for the ISP network: one is QoS guaranteed and used for legitimate traffic (going through switches S_1, S_2, and S_4), while the other is for suspicious or malicious traffic (going through S_1, S_3 and S_4). In the customer network, the OpenFlow switch S_5 is attached to the customer controller. Also, three host machines are virtualized in the platform, respectively serving as legitimate host, attack host and customer machine, as shown in Figure 3.12.

Table 3.3: Platform specifications

Component	Qty	Specification or Configuration	Role
IBM RackSwitch G8052	5	48 Gigabit ports	OF enabled switch
DELL server R620	2	8-core 2.9 GHz CPU, 16GB RAM, 10 Gb/s Ethernet NICs	SDN controller host (ISP, customer)
SDN Controller	2	Ryu 3.20	Controlling switches in ISP (S_1 to S_4) and customer networks (S_5)
Hosts	3	Ubuntu server 12.04.2, 6-core 3 GHz CPU, 4GB RAM,20GB HDD	Playing as video streaming server L, attack host A and video streaming client C

Traffic generation. We have chosen to use a video streaming service as the source of legitimate traffic since, nowadays , video traffic accounts for more than 70 percent of all consumer Internet traffic [10]. In order to generate the legitimate video traffic between the customer machine C and the legitimate host machine L, we used TAPAS [9], a video streaming tool. In particular, one of the salient features of TAPAS is that it adopts adaptive video streaming strategy to adapt to time-varying networking conditions. Moreover, the hping3 tool is used to generate volumetric DDoS attacks with varying rates.

Evaluation metrics. We treat video streaming service as the target service to protect, and measure its quality in the face of DDoS attacks with respects to the quality-of-user-experience (QoE) metrics specified in Table 3.4. As the *time to rebuffer* increases, *average goodput* and *estimated bandwidth* decrease because of the attack traffic, hence reducing the QoE of watching the video. These metrics are evaluated from the logs generated at the client.

Table 3.4: Defined metrics to measure the impact of DDoS attacks

Metric	Definition	Unit
Time to rebuffer	the time taken to rebuffer the video when the streaming buffer gets empty due to network congestion	second
Average Goodput	useful data transferred by the network per unit time	KB/sec
Estimated bandwidth	rate of data transfer	KB/sec

3.2.4.3 Results and Analysis

We conducted several rounds of tests to comparatively study the given metrics in three cases,

- Video streaming services in normal condition (without attack traffic);
- Under attack without mitigation, and;
- Under attack with our autonomic mitigation module activated.

The metrics are measured at the video streaming client by collecting and analyzing the log files generated by TAPAS [9].

Time to rebuffer. The client will not be able to play the video as the streaming buffer gets depleted due to the overwhelming attack traffic. If this condition holds for a long time, the QoE of users is seriously degraded. Figure 3.13, where attacks are represented as packets per second(pks), illustrates the ability of our framework to maintain an acceptable QoE. Under no attack, the TAPAS [9] video client was able to complete the video transmission without any interruptions, and the buffer length could always be maintained above 15 seconds. In contrast, under DDoS attack, the video client was not able to buffer the video, leading to a depletion of the playout buffer down to zero. However, when the mitigation scheme was activated by the ISP upon the request of the customer controller via the REST API, the playout buffer returned to a normal level, allowing the video client to restore playing the video.

Average goodput and estimated bandwidth. We examined the *average goodput* and *estimated bandwidth*, and observed that they could maintain a rate above 550 KB/sec without attacks. However, when DDoS attacks were launched and the mitigation module was activated, the *average goodput* could be still maintained at 450 KB/sec and the average *estimated bandwidth* maintained at 400 KB/sec, as shown in Figure 3.14.

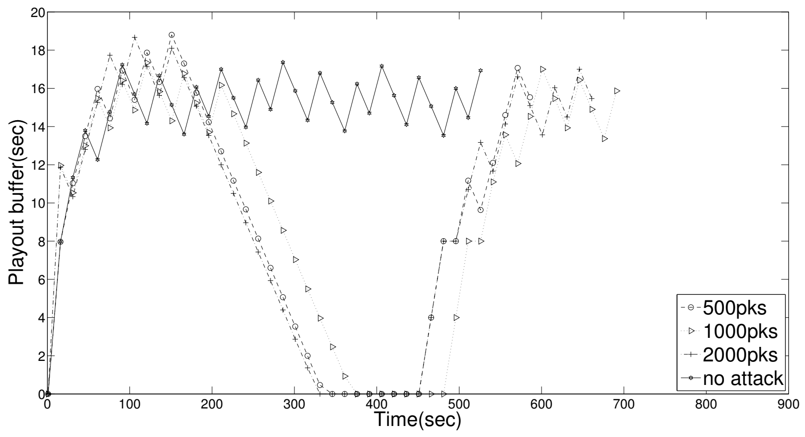


Figure 3.13: Playout Buffer

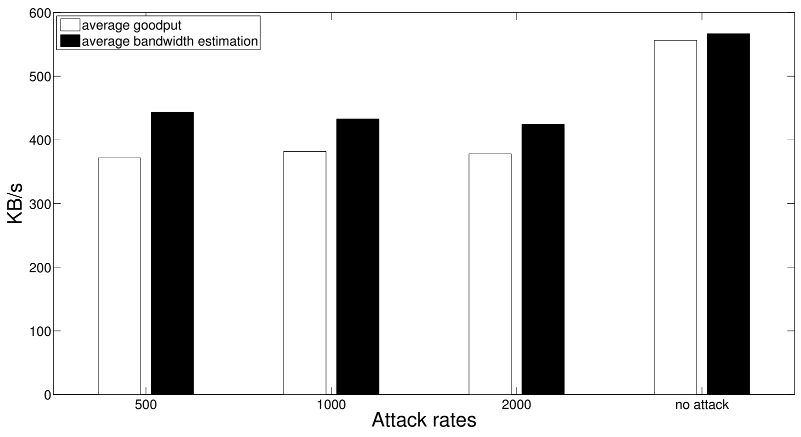


Figure 3.14: Average goodput and estimated bandwidth

3.3 DDoS Mitigation and Defense on Internet eXchanges (IX) with SDN Technologies

In this Section, we describe an SDN-based Internet eXchange point (IX), which provides flexible filtering and mitigation functions.

3.3.1 Threat Description

We assume UDP amplification-based attacks as an emerging threat on inter-domain networks. Figure 3.15 shows one of the considered DDoS attacks:

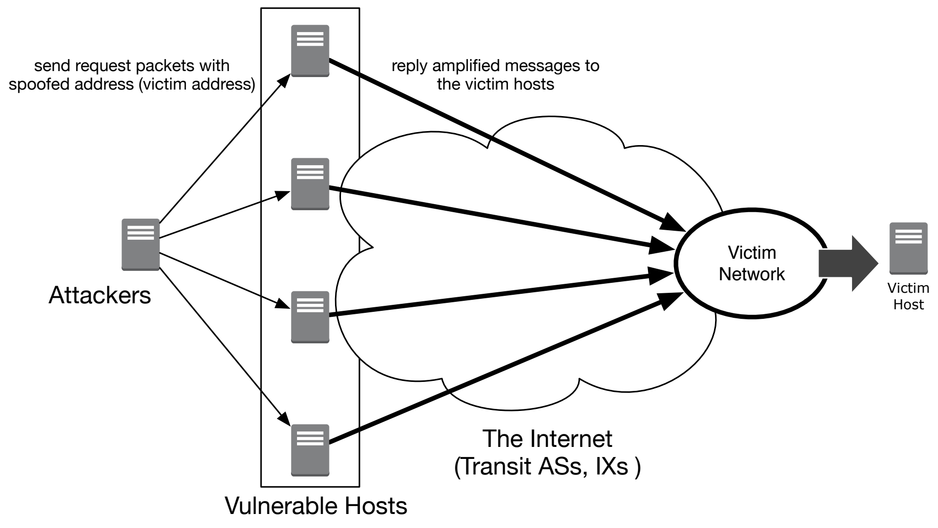

send request packets with spoofed address (victim address)
reply amplified messages to the victim hosts
Attackers
Victim Network
Victim Host
The Internet (Transit ASs, IXs)
Vulnerable Hosts

Figure 3.15: Amplification Attack

an amplification-based attack. The attack traffic is coming from multiple ingress points of the ISP to victim hosts. The traffic, which is amplified by vulnerable servers, is aggregated on multiple points through the servers to victim hosts. The aggregated traffic saturates the links between vulnerable hosts and victim hosts. Even if the attack could be defended against in front of the victim hosts by Access Control Lists (ACL), a firewall or an IPS, we cannot eliminate the traffic from links upstream on the Internet. It imposes constraints on link bandwidths of inter-domain networks across attackers and victims. In particular, saturated links lie between the victim network and the IX.

3.3.2 Internet eXchange Points

IXs are interconnection points between Autonomous Systems (ASs) on the Internet. The ASs directly communicate with each other on the IX using the Border Gateway Protocol (BGP). The networks exchange their routes and forward packets based on route information. ISPs can reduce and load balance their transit traffic through peering relationships on the IX. In general, the IX's infrastructure is comprised of commodity Layer-2 or Layer-3 switches. At present, native connection functions are provided on the infrastructure. However, as we mentioned in Sect. 3.3.1, current DDoS attacks are widely distributed on the Internet. Actually, the attack traffic may also come from ASs that are peered to victim networks on IXs, so that the attack traffic saturates links between the victim networks and the IXs.

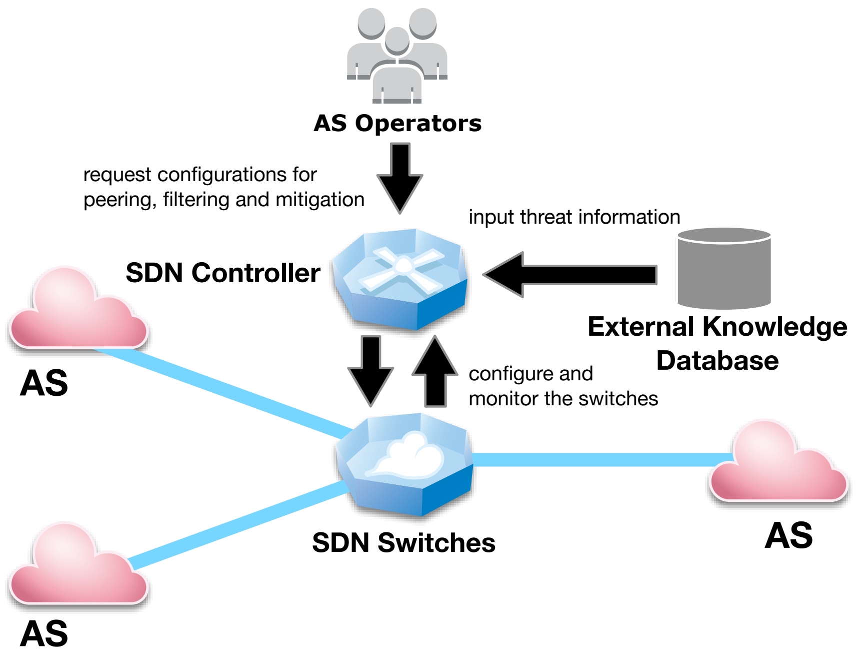

AS Operators
request configurations for peering, filtering and mitigation
input threat information
SDN Controller
External Knowledge Database
configure and monitor the switches
AS
SDN Switches
AS
AS

Figure 3.16: An architecture of SDN-IX

3.3.3 Design of SDN IX

In our approach, we enhance security functions on the IX by connecting ASs using SDN technologies. We designed a new IX which embeds filtering and mitigation mechanisms. Figure 3.16 shows an architecture of the SDN IX. The IX is comprised of the following three components: 1) SDN Switches: in the SDN IX, we adopt SDN capable switches to connect AS networks for dynamically and flexibly forwarding traffic on the IX; 2) IX Controller: the controller controls and coordinates the SDN switches based on user-defined rules. It provides peering, filtering and mitigation functions as applications; 3) Web User Interface: AS operators can manage their peering and filtering rules. Legacy command-based configurations are not suitable for operations on the SDN IX.

Indeed, legacy IXs are managed and configured by the IX's operators. The configuration mainly aims at establishing peering links between ASs. Additionally, individual AS operators cannot configure an IX's equipments and do not require such operations, because the configurations on the IX are static and are not used for any other purpose.

On the other hand, the operators are able to configure the equipments for peering with other networks and filtering attack traffic on the SDN IX. The operation model enables dynamical and flexible configurations by operators in comparison with current IXs. Because the SDN IX is not configured by only a single AS operator but by multiple AS operators, the IX switches should not be directly controlled by them to avoid misconfigurations or overlapping configurations. Additionally, the SDN IX configuration is more complex compared to current Layer-2 and Layer-3 networking, due to the specifications of dedicated flows and actions. These configurations should not affect any other ingress and egress traffic. Therefore, the operation model strives to maintain consistency and stability of the IX.

OpenFlow which is an instance of SDN technologies is a candidate technology for implementing the IX, but we do not restrict the implementation of the IX to OpenFlow. We explore and adopt suitable technologies for our implementation.

3.3.4 DDoS Defense on SDN IX

In this section, we describe how DDoS attacks are mitigated on the SDN IX. When an AS on the IX detects a DDoS attack from other ASs, the victim AS adds a filtering or mitigation rule on the SDN switches located in the SDN IX. The attack traffic is effectively filtered by the deployed rules on the switch's incoming ports. Therefore, it can reduce bandwidth usage by the malicious traffic on the links between the victim AS and the IX.

The rules are described in the following format {*Match [Source IP address, Source TCP/UDP port number, Destination IP address, Destination TCP/UDP port number], Action [drop]*}. The *Match* field specifies which packets have to be handled by this rule. In this case, *Source IP addres and TCP/UDP port number* represent the source of an attack while *Destination IP addres and TCP/UDP port number* represents the victim host. The *Action* field defines the switch's behavior when a packet is matched with the *Match* field. The action is set to *drop*. It means the packet is discarded on the SDN switch, not forwarded to the victim AS. AS operators can specify these fields to block incoming attack traffic.

Also, the IX plans to provide REpresentational State Transfer (REST) interfaces for operators of ASs interconnected on the IX. The operators can install configurations such as peering, filtering and mitigation rules on the IX via the REST interfaces. Additionally, the interface can be leveraged to enforce autonomic cyber cyberdefense. In this case, the IX can be dynamically configured to block attacks based on threat information through the API.

3.3.5 Architecture

In this Section, we describe the architecture of an SDN-based IX. SDN allows for providing multilayer path exchange and security functions on the IX. As a new network paradigm, SDN decouples the data plane and control plane, giving more flexibility to networks. At the control plane, we can define new network functions as applications. In our architecture, we adopt OpenFlow [28] to provide dynamic configurations in the architecture.

Figure 3.17 shows the design architecture of PIX-IE. PIX-IE is composed of the following two components:

Data Plane (OpenFlow switches): The AS border routers connect to other AS's routers over OpenFlow switches. The switches provide the functions introduced in the previous section. The switches are managed by a PIX-IE controller.

PIX-IE Controller: composed of the following submodules. The *User Interface* provides management and configuration interfaces to AS and IX operators on a web user interface. Each *Modules* provides a specific function. The *Negotiator* checks for flow conflicts with other pre-installed entries on the IX before writing them on the switches. If the candidate entry is validated by the Negotiator, the entry is installed on OpenFlow switches by the configurator. The *Configurator* handles configuration on OpenFlow switches through the OpenFlow API. Additionally, some databases are used in PIX-IE. The databases contain user, route and configuration information.

The PIX-IE architecture is not limited to the controller's implemented features. We can expand the number of supported functions by implementing a new module on PIX-IE. The incremental expandability is the reason why the IX is called *Programmable* Internet eXchange. The next few paragraphs describe the design principles and features of PIX-IE.

The key feature of PIX-IE is that ISP operators can configure IX facilities thorough the controller interface without any negotiation. Currently, IXes do not provide user interfaces to ISP operators. On PIX-IE, the ISP operators can apply IX configurations such as path exchanges or attack filters by themselves. The on-demand configuration dramatically changes the operation model of IXes.

Through this architecture, we adopt a proactive configuration model for OpenFlow control. The proactive model means that any configurations for the IX switches are predefined and managed by the controller. In OpenFlow, a reactive model is defined, which means a flow rule is not predefined, but the rule is defined by processing each packet on a controller. However, the reactive model is not a practical solution on IXes, due to the large number of flows. The flows will frequently cause control traffic from switches to the controller called *packet-in* to decide how the switches should handle the flows. The packet-in traffic will consume CPU and memory resources on the controller. In the worst case, the controller will be halted due to complete depletion of resources by the packet-in messages. Then, packet forwarding will also halt, or packets will be dropped on the IX. The proactive model can solve the problem because flow entries are not aware of packet-in on the model.

In the proactive model, the rules must be initiated by the controller. Therefore, the controller has to know the state of the network to avoid traffic

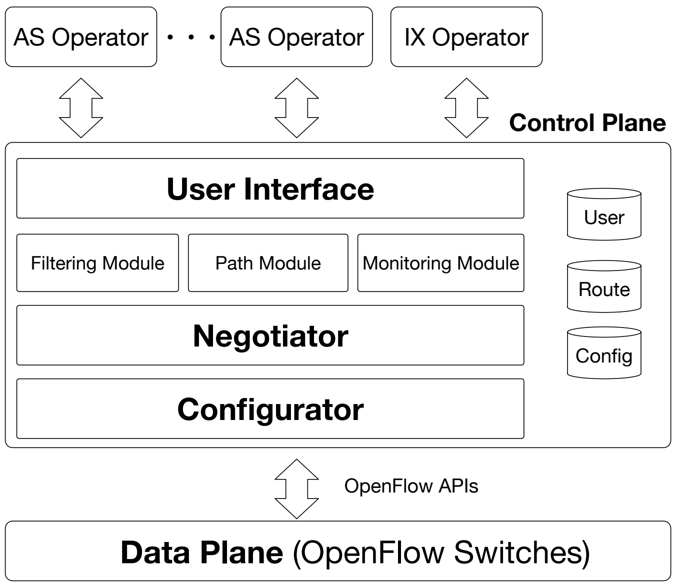


Figure 3.17: PIX-IE Architecture

misuses. It is necessary to avoid rule conflicts [23]. Flow conflicts are not only a problem in this particular IX use-case, but it is also a well-known issue on OpenFlow and any other SDN technologies. In particular, conflicts lead to incorrect packet forwarding or even packet loss on inter-domain networks. Accordingly, the controller has to consider the state of current flows and avoid conflicts between a new entry and the current entries. In addition, we need to isolate the flow rules of an AS from one another to avoid mis-forwarding or undesired filtering of other traffics.

An IX must be stable for packet forwarding as mentioned in the last section. On SDN-based IX, the traffic is forwarded to its destination by using predefined rules. Therefore, if the rules on the IX switches are flashed due to troubles, the traffic forwarding is suspended on the IX. Then, traffic will be dropped on the IXes because switches will have no entries for forwarding packets anymore. These failures are unacceptable to ISPs. Even in the case an IX loses the connection between the switches and the controller, the switches must ensure basic packet forwarding among ASes. In OpenFlow, the controllers may install flow entries with specific parameters (*idle_timer = 0 and hard_timer = 0*) to avoid flashing the entries by time-out.

3.3.6 Experiments

In 2015, we conducted an experiment by providing an on-demand DDoS mitigation on Interop Tokyo 2015. We implemented an SDN IX controller using the Ryu framework with OpenFlow 1.3.1 for demonstration purposes.

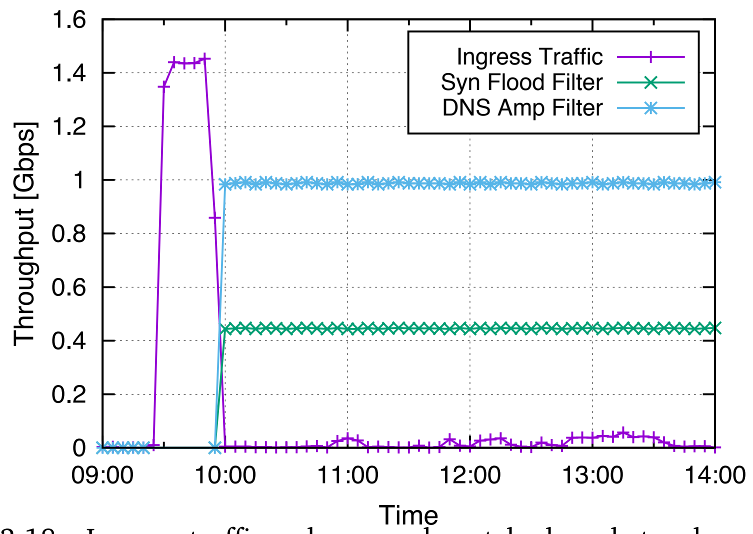


Figure 3.18: Ingress traffic volume and matched packet volume of SYN Flood and DNS Amp filters on PIX-IE

The filtering entries include 5-tuple values as match fields. The matched flows are dropped. The filters can be initiated via the controller's RESTful API. The API format is as follows: *http://{controller.fqdn}/filter/{ingress AS number}/{src IP}/{dst IP}/{TCP|UDP}/{src port#}/{dst port#}* Filtering rules are usually installed with a priority higher than the path exchange entries because these filters must be applied to incoming packets before forwarding them to ASes.

During the Interop exhibition, we generated two types of DDoS traffic for our experiment purposes by using a tester from a peering network of the ShowNet demonstration network towards internal victim IP addresses. The attacks were SYN flood attacks and DNS Amplification attacks. We detected the attack traffic by using Agurim, which is an sFlow analyzer[21]. After the attack detection, we initiated filtering IX switches via the controller API. The filtering entries were then installed on the ingress port facing the attack source.

We successfully mitigated the attack traffic using this mechanism. Fig. 3.18 shows the amount of traffic volume that was measured on a port of the PIX-IE switch. In the figure, the x-axis represents time, and the y-axis the throughput of total traffic.From the figure, we can see that the filters effectively dropped malicious traffic on the IX.

3.4 LISP-based DDoS Mitigation

We propose a new DDoS mitigation mechanism which can eliminate attack traffic at the attacker side using Locator/ID Separation Protocol (LISP).

3.4.1 Threat Description

In Section 3.3, we described a DDoS mitigation and defense mechanism deployed at IXs. The mechanism can filter attack traffic on the IX before it flows into the victim network. However, the IX cannot mitigate attack traffic coming from transit and private-peer networks of the victim network. Consequently, transit and private-peer links could be saturated by the attack traffic. Even if the attack traffic is filtered on a transit ISP of a victim network, the transit backbone is still suffering from the attack traffic. If the attackers learn that the attack is being blocked, they might adapt their attacks by changing their attack methods and/or sources. Therefore, it would be interesting to keep the attackers oblivious of mitigation actions. Additionally, the defense methods should be implemented as close to the attacker nodes as possible in order to reduce traffic loads.

3.4.2 State of The Art

Before describing our proposed method, we will discuss the difficulties of DDoS defense and existing defense methods. Existing DDoS defenses are categorized into the following phases: prevention, detection, identification, and routing-based mitigation. There are a variety of solutions used in each of these individual phases.

For DDoS attacks using spoofed source IP addresses, each packet can have a different source IP address, that is programmatically generated by the attack programs. Therefore, victim hosts or security devices have difficulty identifying the attack packets from the amount of traffic. Additionally, there exist DDoS-like short-lived events on existing services, known as *flash crowds* that produce traffic patterns that are similar to a DDoS attack when vast numbers of unique users attempt to access a service. However, such traffic is not offensive and must not be blocked.

Even if a security device unequivocally detects and identifies the sources of an attack, IP addresses may dynamically change over time. Additionally, according to a report from Arbor Networks [1], 77% of all DDoS attacks are less than one hour in duration, which means that the defense has a limited time to adapt to changes.

Usual prevention methods aim at blocking DDoS traffic in a victim network. Ingress/Egress Filtering [14, 6] involves blocking packets based on an ACL maintained at a router or a switch. Ingress filtering blocks attack packets that attempt to enter the network by filtering lists on an L2 switch or a router. In contrast, egress filtering blocks outgoing packets from the network. Such filtering operations have been accepted as best current practices (BCP) and are in widespread use in commercial networks. However, this type of blocking cannot reduce the attack traffic volume between the attacker's network and a victim's network.

In general, DDoS attacks are detected by IDS or IPS appliances in commercial networks [39]. However, IDS/IPS appliances are prone to state table problems when detecting such attacks. In fact, Arbor Networks [1] also states that DDoS attack detection often fails due to state table depletion. There are also a number of dedicated products aimed at DDoS detection that utilize flow analysis [27], and numerous researchers have attempted to resolve the problem of anomaly-based attack detection. However, all of these detection algorithms still produce a number of false positives.

Blackhole routing [26, 46] is a routing-based mitigation method that forwards malicious traffic to a null router device. However, this approach has the potential to disrupt legitimate traffic. Another potential solution is supplied by Arbor Networks, which provides Peakflow [3] to protect a network against DDoS attack. This product monitors network traffic and injects new IP routes that divert malicious traffic to a filtering device when an attack is detected. Such products work on enterprise and ISP networks, but they only function on the victim's network itself, and cannot eliminate malicious traffic between an attacker source and a victim network.

3.4.3 Proposal

To tackle this challenge, we developed a new DDoS mitigation method that can reduce load on victim servers and networks while protecting ongoing legitimate services. We propose a DDoS mitigation method that utilizes a two-stage map table extension of LISP [11, 16] in a way that keeps attackers oblivious to the defense efforts. While LISP was originally designed with a single map stage for IP based routing, we built an additional map table for the LISP architecture, which is called a mitigation table. This table is used to forward attack traffic to a decoy server that has the same IP address as the legitimate server. From the attacker's point of view, therefore, it is extremely difficult to distinguish whether the destination is legitimate or a decoy. Additionally, since simply adding a new mapping entry to the mitigation table can trigger the mitigation, this solution limits the scope of reconfiguration only to the mapping table i.e., does not require the reconfiguration of other network devices or legitimate servers. We implemented the two-stage extension on MapServer.

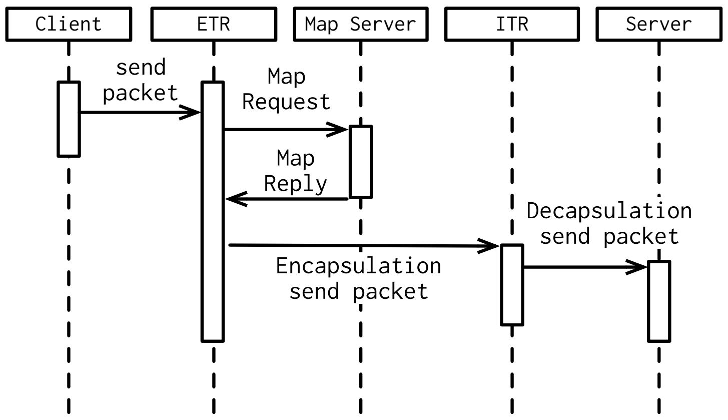


Figure 3.19: Packet forwarding sequence among LISP routers and the Map-Server.

3.4.4 Overview of LISP

LISP is a new Internet routing architecture with specifications standardized as RFC [11, 16] by the Internet Engineering Task Force (IETF). With the traditional IP architecture, IP addresses work as both the device and network identifier. Contrastingly, in LISP, the address role is separated into an end point identifier (EID) and a locator on the network. The EID is used to uniquely identify a device on the network. A routing locator (RLOC) is a routing point address for EIDs on the Internet.

EID and RLOC expressions can be used in the existing IPv4 and IPv6 address formats. In the separation architecture, network deployments can be made scalable because a device's EID addresses are aggregated by a few RLOCs. In addition, because the separation architecture enables device mobility on the network, we can move an EID network location by changing the RLOC without the need for any configuration modifications to the device.

A packet forwarding sequence on LISP is displayed in Figure 3.19. In this case, a client attempts to communicate with a server via the LISP infrastructure by sending a packet to the server (as can be seen in Figure 3.20). This packet is then forwarded to the client side LISP router, known as an egress tunnel router (ETR). When the ETR is receiving a packet from inside the network, it sends a MAP request to a MapServer in order to determine the next hop/gateway router for a destination EID. The MapServer retains the binding information between the RLOCs and EIDs. The MapServer then replies with the RLOC of a destination EID to the ETR. After receiving the RLOC, the ETR encapsulates the packet and forwards it to the router that has the RLOC. An RLOC router, which is called an ingress tunnel router (ITR), then decapsulates the packet and forwards it to the chosen server located in the network.

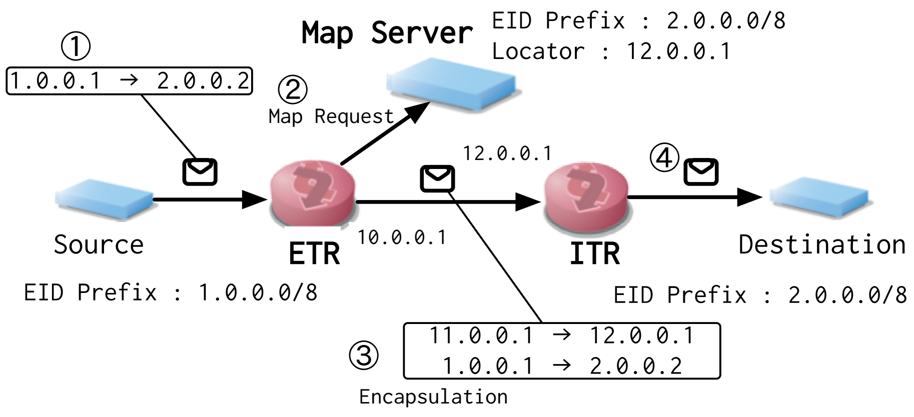


Figure 3.20: Packet forwarding on LISP networks.

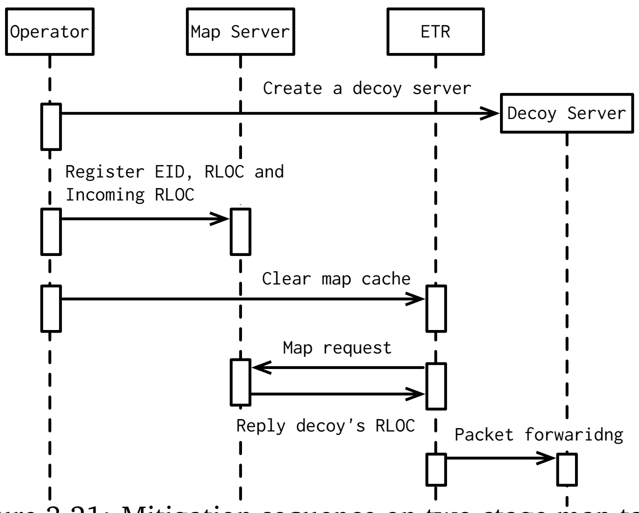


Figure 3.21: Mitigation sequence on two-stage map table.

3.4.5 Two-stage Map Table Extension

We extend the mechanism to enable DDoS mitigation by dynamically reconfiguring the mapping entries on the system. Using the entries, we lead malicious traffic to decoy servers of legitimate ones or the `null` interface. On the other hand, legitimate traffic is still forwarded to legitimate servers at the same time. The malicious traffic will be mitigated on border routers at the source of the attacks. Therefore, the mechanism can eliminate malicious traffic between the attacker's and victim's ASs.

In order to divert attack traffic to decoy servers located at network borders, it was first necessary to extend the LISP mapping system. To accomplish this, we began by assuming that the original and additional map tables are manually configured by network operators. As introduced previously,

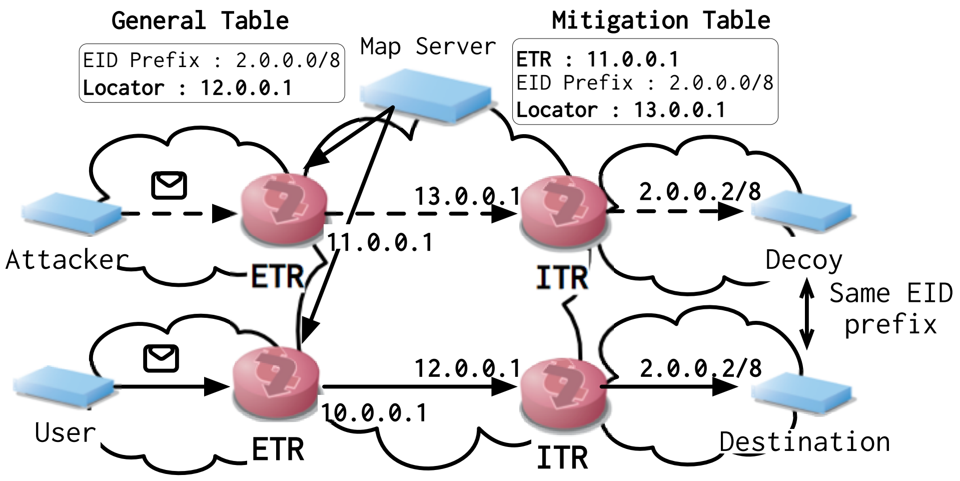


Figure 3.22: Packet forwarding on two-stage map table.

we refer to the additional map table as the "mitigation table", while the original map table is called the "general LISP map table". The sequence diagram of the mitigation approach based on the two-stage map table is shown in Fig. 3.21. The mitigation sequence proceeds as follows (see Fig. 3.22). When a network operator detects a DDoS attack on the network, he or she registers a new map entry to the mitigation table. The entry has three fields: a border router's address (incoming router), a destination address (EID prefix) and a router address that hosts a decoy server (locator). After the map entry registration, the operator frees up a map cache for the incoming router or the ETR. Next, the router needs to identify the address of attack packets via the MapServer. If a packet is coming from the attacker, the router obtains an RLOC that hosts a decoy server from the MapServer. Finally, the packet is forwarded to the decoy server. If a packet is not sent from the attacker, the MapServer sends a legitimate RLOC to the ETR. Accordingly, only attack packets are routed to the decoy server. We call this two-map-table-based system a "two-stage MapServer" The mitigation entry remains valid with a timer in much the same way as a domain name system (DNS) time to live (TTL) hop limit. When the timer expires, the ETR transmits requests for resolving the destination IP addresses of the DDoS packets to a MapServer. After that process, the DDoS packets will again be forwarded to the decoy network.

We implemented the proposed method by modifying a LISP MapServer implementation on Linux [47]. The software has a LISP router and a map table module written in C. We then added a mitigation table and its related functions in the map table module. The module code consists of a mere 350 lines of code. In total, the MapServer source has 2126 lines.

Additionally, we need an infrastructure for hosting decoy servers which behave as legitimate servers. The servers should be operated as legitimate ones to avoid detection of the mitigation mechanism by the attackers.

3.4.6 Advantages

To begin with, our method can reduce the traffic load on both the legitimate server and the network by forwarding attack traffic to decoy servers located on the periphery of a LISP-enabled network. Additionally, our two-map LISP extension does not require any configuration changes on a victim host because our extension can control attack traffic using only an EID and an RLOC binding on an extended MapServer. In contrast, other routing-based mitigation techniques, such as black hole routing, require network operators to modify configurations and install new routes at the routers. Therefore, the mitigation can be applied by network service providers. Moreover, since the decoy server has the same IP address as the legitimate server, attackers face extreme difficulty in recognizing whether or not their target is defended.

Existing DDoS defense methods have the following three disadvantages. The first is defense location. Almost all defense methods block attacks in the vicinity of the victim hosts. Methods of this type assume host loads such as network bandwidth or CPU usage can be reduced, but they eventually cannot eliminate the network traffic loads between the attackers and the victim. The second disadvantage is that several mitigation methods adversely impact legitimate services by filtering or dropping packets. The last disadvantage is the lack of obliviousness against attackers, who can easily recognize the mitigation effort from parameter changes measured by the attacker nodes. For example, firewalling or ACL blocking can be recognized by connections from multiple nodes located in other networks.

The two-stage map table extension overcomes such drawbacks of existing mitigation methods. In our proposal, we assume the attacker's location or the ingress routers of the attack traffic have been detected by IP traceback or some other method. Based on that assumption, our system leads the attack traffic to one or more decoy servers, which behave as the victim servers. Since the decoy server is located close to the attacker's network, the attacker's traffic does not leak to outside networks. Additionally, since the decoy servers have the same EID (IP address) as the targeted legitimate server, attackers are unable to distinguish between them.

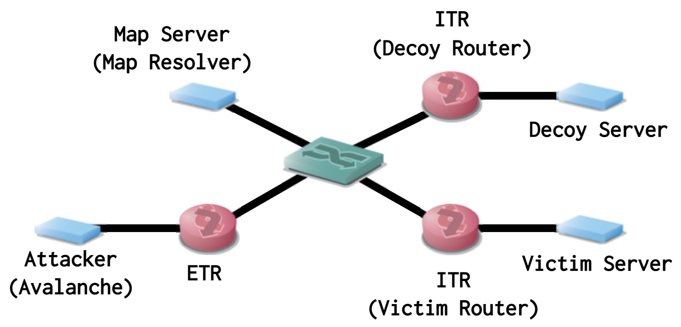


Figure 3.23: The experiment topology

3.4.7 Experiment

In this section, we evaluate the performance of our software implementation. The purpose of this experiment is to evaluate throughput and the time required to switch routes using our extension.

3.4.7.1 Methodology

Figure 3.23 shows the network topology used in our experiment, which consisted of the attacker's network, the victim's network and the decoy network. All links have a bandwidth of 1 Gbps. In this experiment, each server and router were equipped with a 2.4 GHz Intel Xeon E5620 CPU with 4 cores, 12 GB of memory and a 1000Base-T Ethernet port. The MapServer is running on the Linux server (Debian Squeeze). Each LISP router used [47]'s LISP implementation.

During our experiment, we measured the bandwidth of the victim and decoy server. To accomplish this, we collected received data volume from */proc/net/dev* per second in the servers. After starting the traffic generation on attacker node, we manually freed the map-cache on the ETR and reconfigured routing using our proposed method.

3.4.7.2 Basic Throughput

First, we tested basic performance using *iperf*, which is a common throughput testing method on Linux. A single *iperf* source can send user datagram protocol (UDP) packets to the victim server at a rate of 900 Mbps. In this experiment, we manually changed the map entry at approximately 100 seconds. Figure 3.24 shows a result of the experiment. After a few seconds, the traffic was moved directly to the decoy server. This result shows that our implementation can quickly change routes despite high traffic rates.

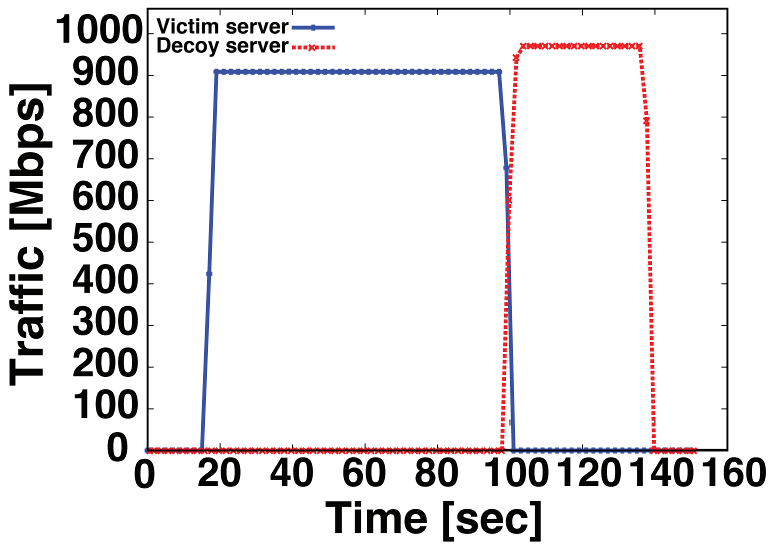


Figure 3.24: Traffic volume on victim and decoy server (iperf UDP).

3.4.7.3 UDP Traffic Mitigation

We then evaluated the performance of our method against a realistic UDP-based DDoS attack using a massive amount of spoofed source IP addresses. For the attacker role, we used an Avalanche 290, which is a commercial traffic generator. During the experiment, we placed a load on the victim host using the traffic generator. To simulate an UDP-based DDoS attack, we sent UDP datagrams at the rate of 0.18 Mpps to the victim server from the generator, spoofing 105 million unique source addresses and resulting in 700 to 900 Mbps of bandwidth usage.

Figure 3.25 shows the received traffic volume on the victim and decoy servers. At approximately 100 seconds on the x-axis, the traffic was smoothly forwarded to the decoy server. The total transition time was just a few seconds long, even though the traffic had a massive number of distinct source addresses.

3.4.7.4 TCP Traffic Mitigation

We measured how quickly our method could mitigate a TCP based-attack by conducting the following experiment. Here, we generated a massive number of *HTTP GET* requests to the victim server where virtual tester clients downloaded 1 GB of data on each request. We then measured the traffic bandwidth on the victim server and the decoy server using the same measurement methodology adopted during the UDP experiment.

Figure 3.26 shows the forwarding volume to the victim and the decoy on the server. In the figure, each line has a zigzag shape caused by TCP

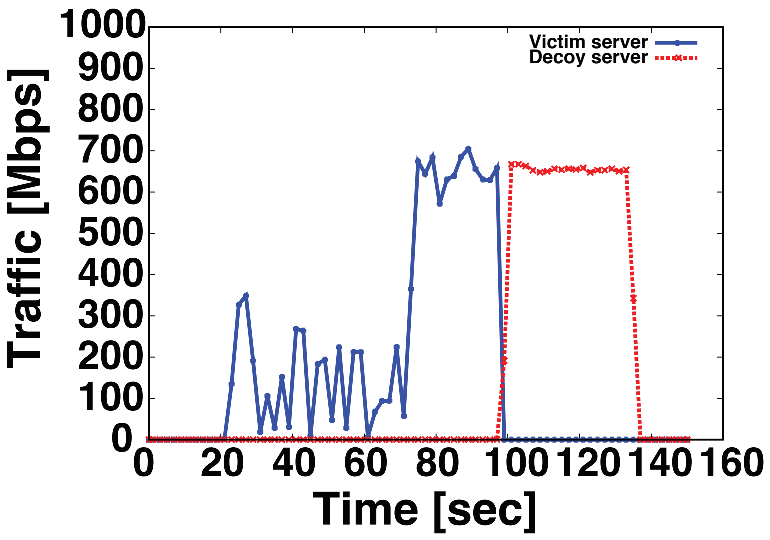


Figure 3.25: Traffic volume on victim and decoy server (DNS).

congestion between the clients and the server. After 100 seconds, the traffic was smoothly mitigated to the decoy server. Even though the destination changed, the attacker's TCP connections were smoothly established between the simulated clients on the traffic generator and the server. This indicates that an attacker would be unlikely to recognize the mitigation provided by the decoy server.

3.4.7.5 Experiment Summary

The above mentioned experimental results demonstrate that our LISP two-stage map extension has the potential to mitigate wire speed traffic. Since the mitigation only takes a few seconds, both TCP and UDP DDoS attacks would be quickly mitigated. Additionally, attackers would find it difficult to recognize the mitigation was in process because the server responses do not change when the mitigation takes affect. These are ideal features for keeping the mitigation undetected by the attackers.

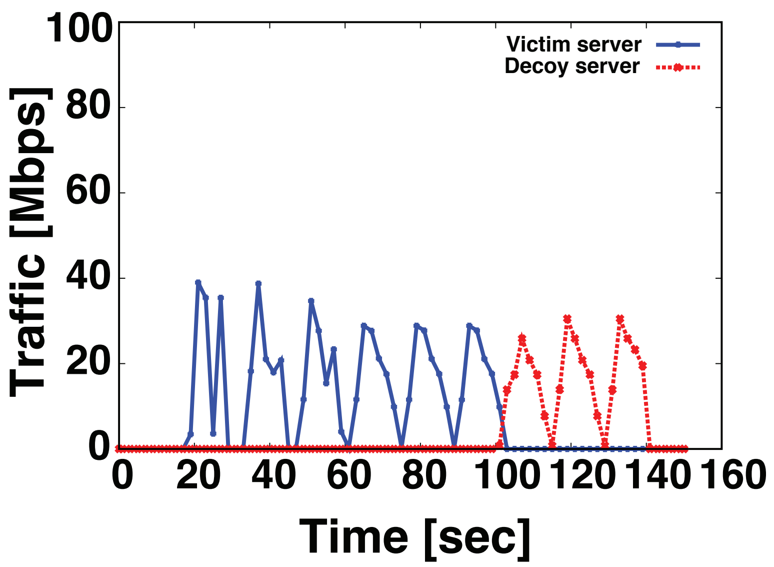


Figure 3.26: Traffic volume on victim and decoy server (HTTP GET).

3.5 Threat Detection and Mitigation for Public Cloud

In this Section, we describe threats against a public Infrastructure-as-a-Service (IaaS) cloud, the impacts of different attacks, and propose ways to mitigate such threats and attacks from the standpoint of the operator or the administrator of the IaaS public cloud.

Typically, a public IaaS cloud accommodates a large number of virtual machines for a large number of users. Different users share the overall resources of the public cloud with some separation mechanisms in place. In such environment, an incident concerning a user may affect other users, despite the separation mechanisms.

3.5.1 Threat Description

The public cloud has become a common infrastructure for services and computing. Especially, IaaS is very popular and widely deployed for migrating current services into cloud environments. However, there are some common threats to the IaaS cloud. These threats are roughly classified into the following three types

- information leak or cloud VM hijacking,
- denial of service inside the cloud, and
- denial of service from outside the cloud.

The first threat is a threat derived from the sotware vulnerabilities of an hypervisor. If there is a vulnerability that a cloud user can abuse to obtain permissions or intrude other VMs, confidential data on VMs may be stolen. In addition, the compromised VM can become a stepping stone for further attacks.

The second threat has its roots in resource allocation. If a VM on a cloud can exhaust network bandwidth, CPU resources, and memory resources, it may disturb other VMs' services. To avoid such a situation, most hypervisor implementations can limit the resources allocated to each VM. CPU resources and utilization for each VM are limited by abstraction of their virtual CPU (vCPU). Also, each VM has an amout of memory pre-allocated by the hypervisor. However, network and disk I/O are not limited by default in most hypervisor implementations. An ill-willed VM user can exhaust I/O resources and disturb the other VMs' services.

Another problem related to the second threat is masquerading. If the separation of resources is not sufficient, an ill-willed VM can forge another VM's ARP and NDP messages, allowing the VM to send fake Router Advertisement messages to other VMs in the same network group, eventually misleading the traffic of other VMs. As a result, the masquerading VM can interrupt the traffic directed to other VMs and hijack sessions from users.

The third threat relates to attacks from outside the cloud. Attackers can flood services running on a VM in the cloud with a large number of packets. The purpose of the attacks is to stop the service, whether it is for the challenge or for some benefit. Once such an attack begins, some resources of the cloud, such as the network bandwidth and the network I/O of the hypervisor may be exhausted. It follows that the services on other VMs running on the same hypervisor may be affected by the attacks. The service on a VM is more fragile than the service on a bare-metal server. Due to the hypervisor overhead, the I/O processing on a VM usually consumes twice the amount of resources compared to a bare-metal server, thus making it easier for an attacker to exhaust virtualized resources.

3.5.2 Proposal

To prevent the interruption of services, we propose a public cloud infrastructure that has the following capabilities:

- monitoring the traffic bandwidth and behaviors,
- monitoring the resources including I/O performance of hypervisors,
- multiple filtering and mitigating points, and
- filtering and mitigating mechanisms cooperated with a security information exchange system, such as NECOMAtter.

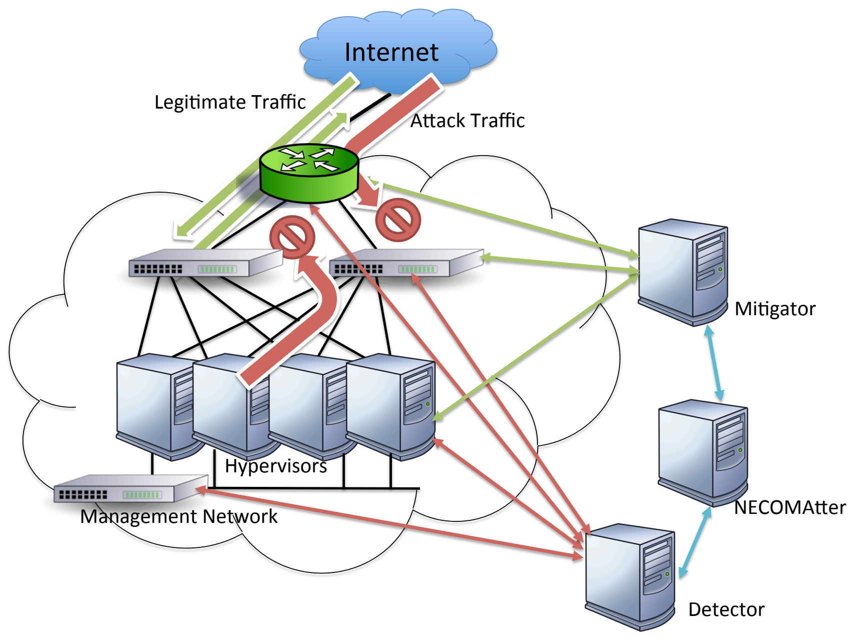


Figure 3.27: Countermeasure methods for cloud threats.

Monitoring the traffic bandwidth is efficient for detecting DoS attacks directed at services. If the traffic bandwidth toward a specific service increases distinctively compared to the total bandwidth, it might be a DoS attack.

Also, monitoring behaviors are important. By traffic behaviors, we designate the traffic pattern to/from a service. If the distribution of source addresses towards a specific service in a cloud is more widespread than before, it might be an indication of scans or DDoS attacks.

It is useful to poll and collect the status of hypervisors and VMs for finding abnormal behaviors inside a cloud. If a hostile user intrudes another VM and steals a certain amount of confidential data, the traffic between VMs using the *management* network will be increased. If there is an attack involving spoofing a VM inside a cloud, service traffics towards VMs will also reach the spoofed VM. We should monitor traffic behaviors at each VM from other VMs, or outside a cloud so that we can find abnormal changes in traffic behaviors.

In order to counter such threats, the following methods are useful.

- polling the status of hypervisors and VMs;
- monitoring the traffic flows inside and outside a cloud.

To find changes in traffic behaviors, as well as, suspicious traffics to some VM, traffic sampling methods, such as sFlow and NetFlow, are useful. Indeed, using the sampling technology, we can detect attacks by monitoring the traffic crossing not only physical network switches between hypervisors, but also the traffic crossing virtual network switches (such as Open vSwitch) between VMs.

In addition to monitoring network traffic, monitoring I/O state and performance from hypervisors may be an effective innovation. If the detector monitors the I/O state of network interfaces and storages from hypervisors, the cloud administrator can detect malicious behaviors of user's VMs without installing a special monitoring software inside the VMs. At present, most commercial public clouds deploy monitoring agent software inside user VMs. The cloud administrator can then monitor the information through the agent. However, if an attacker knows the monitoring mechanism used by the agent, the attacker may delete or replace the agent and send fake information to the administrators. Even if such situation happens, our proposed method, i.e., monitoring I/O state and performance on hypervisors, can detect malicious behaviors.

Once we detect such attacks, the suspicious traffic may be filtered or mitigated by SDN technology. Figure 3.27 shows an overview of the countermeasure methods against the threats. The method described in the Sect. 3.3 is applicable. In the case of a public cloud, filtering and mitigating at multiple points inside of the cloud is effective, because the public cloud being multi-tenant, the impacts of attacks and incidents should be reduced to the minimum. Reducing the impacts can be effectively performed through cooperation with a security information exchange system. Because such system can provide information external to the cloud, detection can be made at an early stage.

3.5.3 System Design

Our proposed system design against cloud threats is shown in Fig. 3.28. To monitor the status of VMs and hypervisors continuously, we adopt standardized protocols such as SNMP, SNMP trap, and OVSDB. If we were to define a dedicated protocol fo monitoring, it would be difficult for the existing cloud implementations to introduce this new protocol. So we decided to use existing and supported protocols in running implementations.

However, the standard hypervisor MIBs are not enough for monitoring and finding malicious behaviors of VMs. Thus, we have decided to enrich them. Furthermore, we want to avoid the need of installing some specific agents inside each VM for monitoring and thus we propose to base monitoring only on information available from hypervisors. We propose a method for monitoring only collective resources, exclusive resources, and I/O interruption status in a hypervisor in order to supervise VM behaviors. We think that it is a more suitable and deployable architecture for a public cloud than using specific agents in VMs.

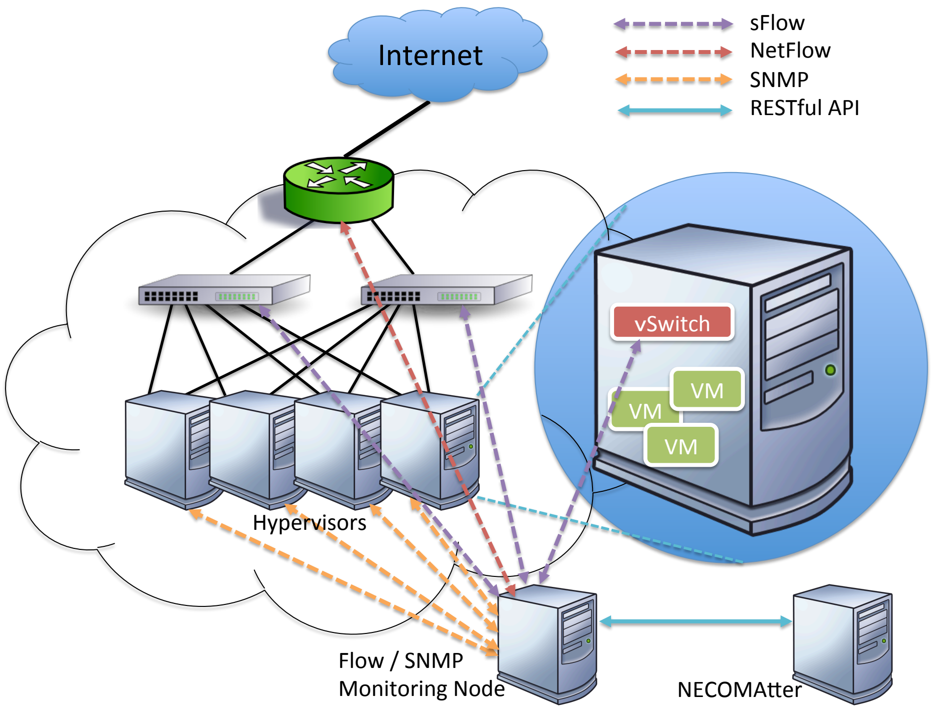


Figure 3.28: System design against cloud threats

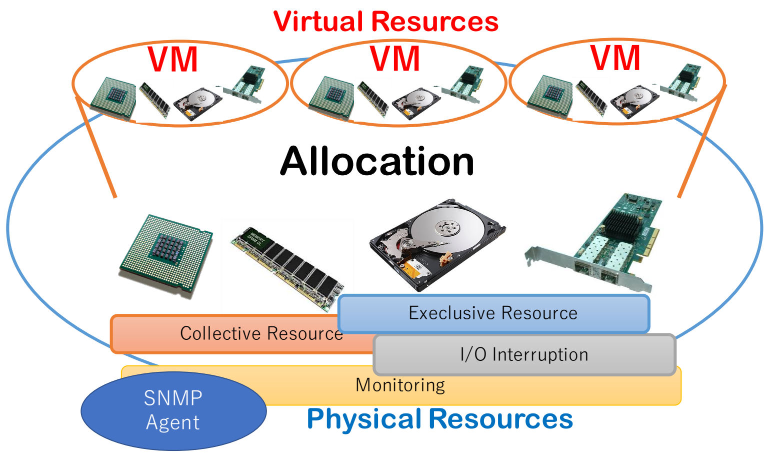


Figure 3.29: Resource Monitoring for Exclusive / Collective Resources in the Hypervisor.

The concept is shown in Fig. 3.29. A hypervisor provides resources for a VM, and a VM requests resources or I/O from hypervisors. If there is a malicious VM in a cloud and the VM sends attacks to outside hosts, the hypervisor I/O status used by the VM will show unusual values. If the VM sends attacks to inside hosts, the CPU usage of the virtual switch and virtual ethernet processes will show unusual values. We believe that the malicious behaviors of VMs are translated into unusual values of the hypervisor resources, thus we propose a polling model for the hypervisor.

Also, we propose to monitor traffics from inside and outside a cloud using sFlow and NetFlow. A threshold-based algorithm is used for the detection of malicious behaviors.

The system is composed of the following three types of modules, as well as, the NECOMAtter system which is described in Deliverable D3.2.

Monitoring modules They are implemented and installed at the hypervisor, Virtual Switches, Physical Switches, and Cloud Edge Routers. The monitoring module at the hypervisor monitors various physical resources of the hypervisor, as well as, the network traffic of physical network interfaces. Other monitoring modules monitor network traffic information using SNMP and flow technologies. The modules then summarize the information and tweet the results to the NECOMAtter system.

Detection module It collects the tweets from the monitoring modules and tries to detect unusual behaviors. If the module finds unusual or malicious behaviors, it tweets the mitigation information to the NECOMAtter system.

Mitigation module This module, which is deployed in the "mitigator" server in Figure 3.27, also monitors the tweets from the detection module. When it captures a tweet, the mitigation module extracts the point and method to mitigate from the tweet and tries to send requests for implementing access control rules on hypervisors, VMs, Physical Switches, Virtual Switches, and Cloud Edge Routers.

The three modules are connected via the NECOMAtter system and works as a single mitigation system.

3.5.4 Implementation

To fulfill the requirements of our proposed methods, we implemented and deployed four pieces of software, two monitoring modules, one detection module, and one mitigation module.

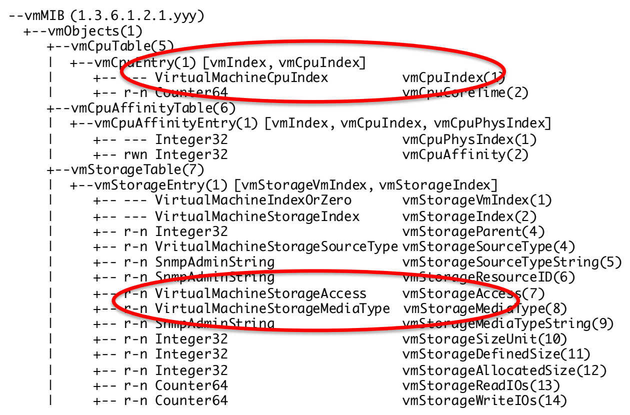
A monitoring module called ”virtsnmp” [4], implements the monitoring methods described in the previous subsection and proposeed in RFC7666 [5]. One of the *NECOMA* project members contributed to defining and standardizing the specification, that we deployed and tested in the actual environments. Another monitoring module called "Neutron-sFlow" allows retrieving sFlow information via the OpenStack's Neutron module by means of an OpenStack patch developed within the *NECOMA* project. A detection module called "agurim" provides real-time attack detection from pcap, NetFlow, and sFlow datasets. It was also developed by the *NECOMA* project and later improved towards its deployment. A mitigation module called "DNS DDoS Defense and Countermeasure (d4c)" [34] provides defense and mitigation architecture and functions in cloud and Internet Exchange (IX) environments. This software was developed by *NECOMA* project.

In this Subsection, we describe the implementation of the four modules, as well as, the overall architecture of the deployed system.

3.5.4.1 virtsnmp

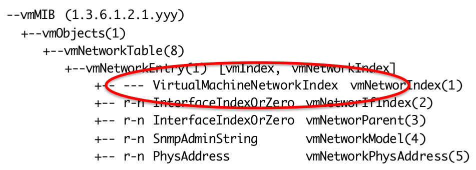
The virtsnmp is an SNMP agent software. To monitor the behaviors of hypervisors and VMs in a public cloud, the software collects the status of CPU, memory, and I/O of the HDD and network without using specific agents within the VMs, but from the hypervisors only. In previous standards, there is no definition to access network and storage I/O for each VM running on a hypervisor, so we supported the proposal of a standard and implemented the mechanism in this software. The proposed architecture of MIBs for a VM is shown in Figures 3.30 and 3.31.

The implementation architecture is shown in Fig. 3.33. virtsnmp works as a daemon process on a hypervisor. It integrates between snmpd and libvirtd and collects the required information from libvirtd and the host OS to



```
--vmMIB (1.3.6.1.2.1.yyy)
  +--vmObjects(1)
     +--vmCpuTable(5)
     |  +--vmCpuEntry(1) [vmIndex, vmCpuIndex]
     |     +-- --- VirtualMachineCpuIndex         vmCpuIndex(1)
     |     +-- r-n Counter64                      vmCpuCoreTime(2)
     +--vmCpuAffinityTable(6)
     |  +--vmCpuAffinityEntry(1) [vmIndex, vmCpuIndex, vmCpuPhysIndex]
     |     +-- --- Integer32                      vmCpuPhysIndex(1)
     |     +-- rwn Integer32                      vmCpuAffinity(2)
     +--vmStorageTable(7)
     |  +--vmStorageEntry(1) [vmStorageVmIndex, vmStorageIndex]
     |     +-- --- VirtualMachineIndexOrZero      vmStorageVmIndex(1)
     |     +-- --- VirtualMachineStorageIndex     vmStorageIndex(2)
     |     +-- r-n Integer32                      vmStorageParent(4)
     |     +-- r-n VritualMachineStorageSourceType vmStorageSourceType(4)
     |     +-- r-n SnmpAdminString                vmStorageSourceTypeString(5)
     |     +-- r-n SnmpAdminString                vmStorageResourceID(6)
     |     +-- r-n VirtualMachineStorageAccess    vmStorageAccess(7)
     |     +-- r-n VirtualMachineStorageMediaType  vmStorageMediaType(8)
     |     +-- r-n SnmpAdminString                vmStorageMediaTypeString(9)
     |     +-- r-n Integer32                      vmStorageSizeUnit(10)
     |     +-- r-n Integer32                      vmStorageDefinedSize(11)
     |     +-- r-n Integer32                      vmStorageAllocatedSize(12)
     |     +-- r-n Counter64                      vmStorageReadIOs(13)
     |     +-- r-n Counter64                      vmStorageWriteIOs(14)
```

Figure 3.30: Proposed MIB Architecture for vCPU and vStorage.



```
--vmMIB (1.3.6.1.2.1.yyy)
  +--vmObjects(1)
     +--vmNetworkTable(8)
        +--vmNetworkEntry(1) [vmIndex, vmNetworkIndex]
           +-- --- VirtualMachineNetworkIndex  vmNetworIndex(1)
           +-- r-n InterfaceIndexOrZero  vmNetworIfIndex(2)
           +-- r-n InterfaceIndexOrZero  vmNetworParent(3)
           +-- r-n SnmpAdminString       vmNetworkModel(4)
           +-- r-n PhysAddress           vmNetworkPhysAddress(5)
```

Figure 3.31: Proposed MIB Architecture for vNetwork.

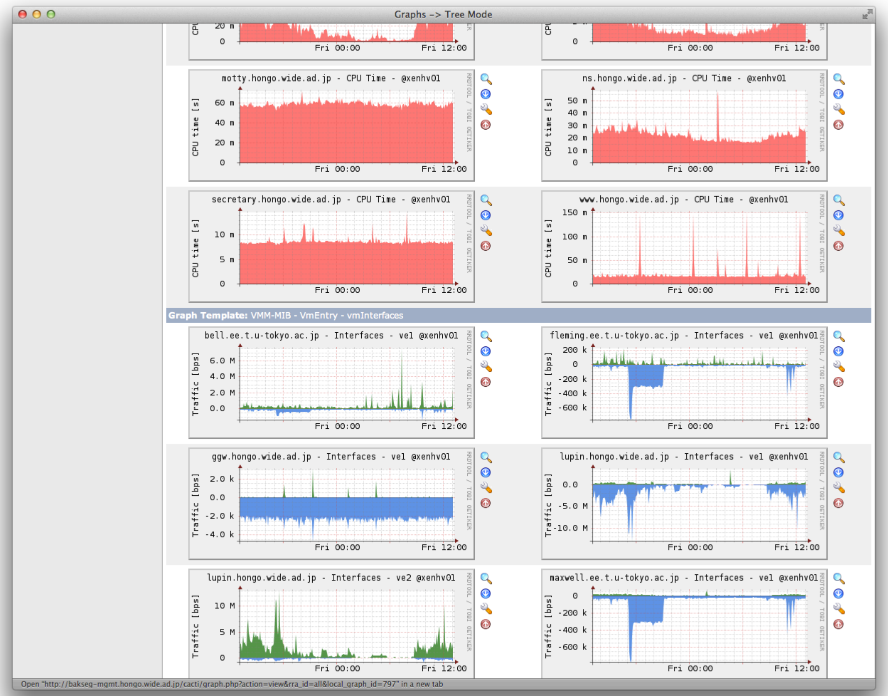

Graphs -> Tree Mode
motty.hongo.wide.ad.jp - CPU Time - @xenhv01
ns.hongo.wide.ad.jp - CPU Time - @xenhv01
secretary.hongo.wide.ad.jp - CPU Time - @xenhv01
www.hongo.wide.ad.jp - CPU Time - @xenhv01
Graph Template: VMM-MIB - VmEntry - vmInterfaces
bell.ee.t.u-tokyo.ac.jp - Interfaces - ve1 @xenhv01
fleming.ee.t.u-tokyo.ac.jp - Interfaces - ve1 @xenhv01
ggw.hongo.wide.ad.jp - Interfaces - ve1 @xenhv01
lupin.hongo.wide.ad.jp - Interfaces - ve1 @xenhv01
lupin.hongo.wide.ad.jp - Interfaces - ve2 @xenhv01
maxwell.ee.t.u-tokyo.ac.jp - Interfaces - ve1 @xenhv01
CPU time [s]
Traffic [bps]
Fri 00:00
Fri 12:00

Figure 3.32: Monitoring VM information from hypervisor.

monitor the hypervisor and the VMs. Then, it publishes the collected information to a monitoring server via the SNMP protocol. We adopted this architecture because the necessary modifications against existing implementations are minimal.

For monitoring VMs and visualizing the collected information, the cacti[3] tool is used. It collects information from hypervisor MIBs using SNMP and visualizes the results as shown in Fig. 3.32. In addition, agurim monitors network traffic from physical and virtual switches. The combination of results from cacti and agurim are useful for finding malicious behaviors inside and outside the cloud.

3.5.4.2 Neutron-sFlow

We made an extension of OpenStack[4] Neutron for retrieving sFlow information. OpenStack is a very popular and open source implementation of IaaS management. A VM is connected to a virtual port of the virtual switch,

[3] http://www.cacti.net/
[4] https://www.openstack.org/

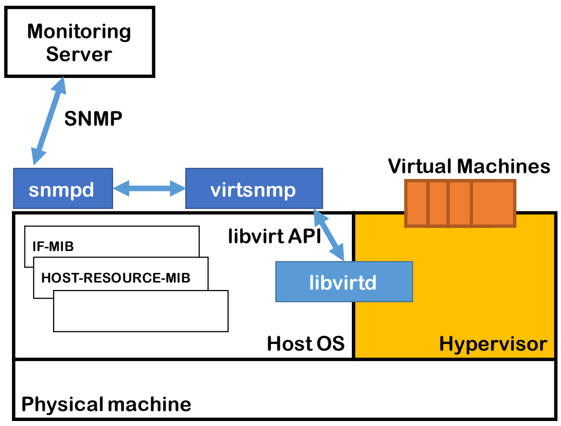


Figure 3.33: Proposed Architecture to collect information for Hypervisor and VMs.

so if the operator can monitor the network flows of each virtual port separately, the flows between VMs are easily monitored. When OpenStack is deployed as an IaaS, Open vSwitch is also installed as a virtual switch inside the hypervisor. It has the capability to export sFlow information, but Neutron, OpenStack's network controller, does not provide the method to retrieve this information. Hence, a patch was made by a member of the *NECOMA* project and is available online[5].

3.5.4.3 agurim

agurim is a network traffic monitor based on flexible multi-dimensional flow aggregation to identify significant aggregate flows in traffic. A network administrator can dynamically switch views based on traffic volume or packet counts, address or protocol attributes, with different temporal and spatial granularities. The supported data sources are pcap, sFlow, and NetFlow. Also, we developed DDoS detection modules which can work with NECOMAtter, the security information exchange component described in Deliverable D3.3.

The main view provides dual plots, a volume-based plot on the left and a packet-based plot on the right. Each plot presents the seven most significant aggregate flows, by default. The legend label shows each aggregate flow with the main attribute and its share of the total traffic, along with the sub-attributes and their shares within the aggregate flow. In the address view, the main attributes are the source and destination addresses, and the sub-attributes are the protocols. In the protocol view, the main attribute is

[5]https://gist.github.com/upa/fd91ff40908b070b7173

the protocol, and the sub-attributes are the addresses. Addresses are presented with their prefix length when aggregated. The detection algorithm is threshold-base. If the top seven aggregated flows include a /32 IPv4 destination flow or a /128 IPv6 destination flow, it may indicate the presence of a malicious flow. The detection module posts the flow information to NECOMAtter.

Using agurim for cloud monitoring, the operator can find discriminative traffic behaviors. It mostly depends on the flow traffic volumes, but agurim can render separate views based on the collected information. So the operator can define the granularity of a set of collected flow information. This means the operator can make views per hypervisor and per VM, for instance.

The software was originally implemented by a member of the *NECOMA* project and enhanced to detect network anomalies for this project. The details of agurim are described in [22]. The software is available online[6],[7].

3.5.4.4 d4c

d4c was developed for mitigating attacks both from outside and inside the cloud. It was developed mainly for mitigating DNS DDoS attacks. However, the methodology is applicable not only to DNS-based attacks, but to other types of DDoS attacks as well. We are currently developing other modules to extend d4c.

Figure 3.34 shows the overview of a d4c deployment. When a malicious traffic is detected by agurim, the OpenFlow controller of the public cloud sends messages to both d4c and the OpenFlow switches, which then leads malicious flows to d4c. There, d4c mitigate DNS packets bearing specific patterns. After d4c has filtered the malicious DNS queries, it forwards only proper DNS queries to VMs inside or outside the public cloud. Currently, d4c can mitigate DNS queries using DNS QNAME patterns, and the patterns are specified by the options. It uses netmap [37] technology for performing the high-performance mitigation. The software is publicly available online[8].

d4c is an example implementation of our proposed mitigation concept. The key points of our proposed concepts are (1) finding malicious behaviors through various monitoring points, (2) separate the malicious network flows from normal flows, and (3) analyze the malicious flows with deep insights using mitigation software. To instantiate the concept, various kinds of network monitoring and mitigation points are needed in a cloud system. We call this mitigation architecture as ”demand-and-opportunity based mitigation.”

[6]http://mawi.wide.ad.jp/~agurim/about.html
[7]https://github.com/necoma/agurim/
[8]https://github.com/upa/d4c

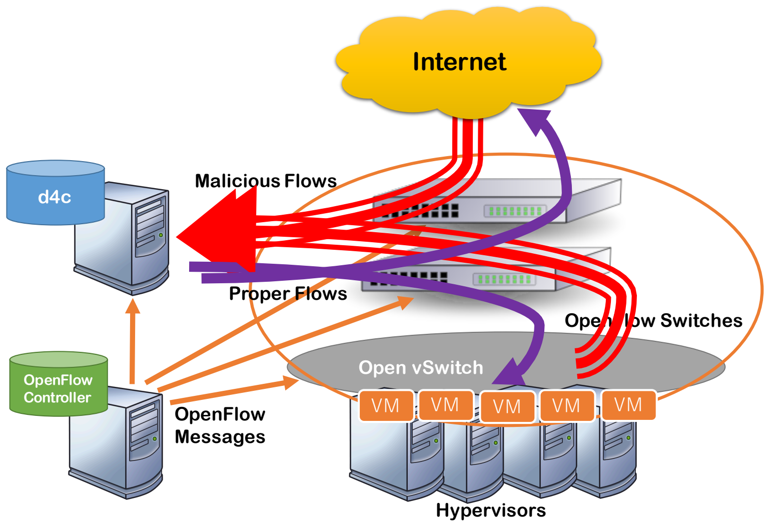


Figure 3.34: Deployment Overview of d4c.

3.5.5 Results

Using a combination of four pieces of software, we implemented our proposed threat detection and mitigation system for a public cloud. As a testbed for the system, we used a public cloud called "WIDE Cloud" [42]. The Cloud is a public IaaS cloud for researchers and students, available to be freely used by anybody who joins the WIDE Project[9]. Over 100 users were registered, and 400 VMs were running as of November 2015.

Figure 3.35 shows the operational flows of the implemented system. The detector collects various information such as SNMP, sFlow, NetFlow, and syslog. Then, the NECOMAtter BoT working in the detector analyzes the collected data and try to find malicious behaviors. The collected information is also sent to MATATABI and recorded. If it finds malicious behavior, it posts information to NECOMAtter, the mitigator and human operators. The mitigator monitors the timeline of NECOMAtter. If it finds a post that needs a response, it takes the appropriate pre-defined action to trigger the mitigation mechanisms, physical switches, and virtual switches. In the case of a simple incident such as an NTP amplification attack, the mitigator pushes filtering ACLs to physical switches automatically. However, if the incident is not well-known and difficult to defend against, the human operator should ultimately decide to enable the mitigation or not. The system was implemented into the WIDE cloud and routinely worked for daily operation.

[9]http://www.wide.ad.jp

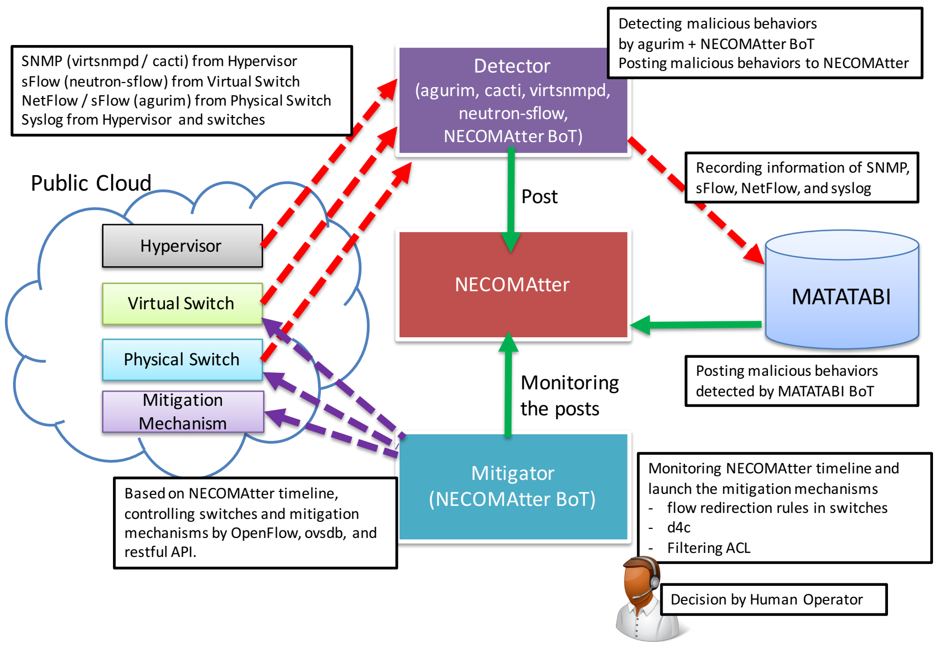

SNMP (virtsnmpd / cacti) from Hypervisor
sFlow (neutron-sflow) from Virtual Switch
NetFlow / sFlow (agurim) from Physical Switch
Syslog from Hypervisor and switches
Detector (agurim, cacti, virtsnmpd, neutron-sflow, NECOMAtter BoT)
Detecting malicious behaviors by agurim + NECOMAtter BoT
Posting malicious behaviors to NECOMAtter
Public Cloud
Hypervisor
Virtual Switch
Physical Switch
Mitigation Mechanism
Post
Recording information of SNMP, sFlow, NetFlow, and syslog
NECOMAtter
MATATABI
Posting malicious behaviors detected by MATATABI BoT
Monitoring the posts
Mitigator (NECOMAtter BoT)
Based on NECOMAtter timeline, controlling switches and mitigation mechanisms by OpenFlow, ovsdb, and restful API.
Monitoring NECOMAtter timeline and launch the mitigation mechanisms
- flow redirection rules in switches
- d4c
- Filtering ACL
Decision by Human Operator

Figure 3.35: System flow diagram of the implemented system.

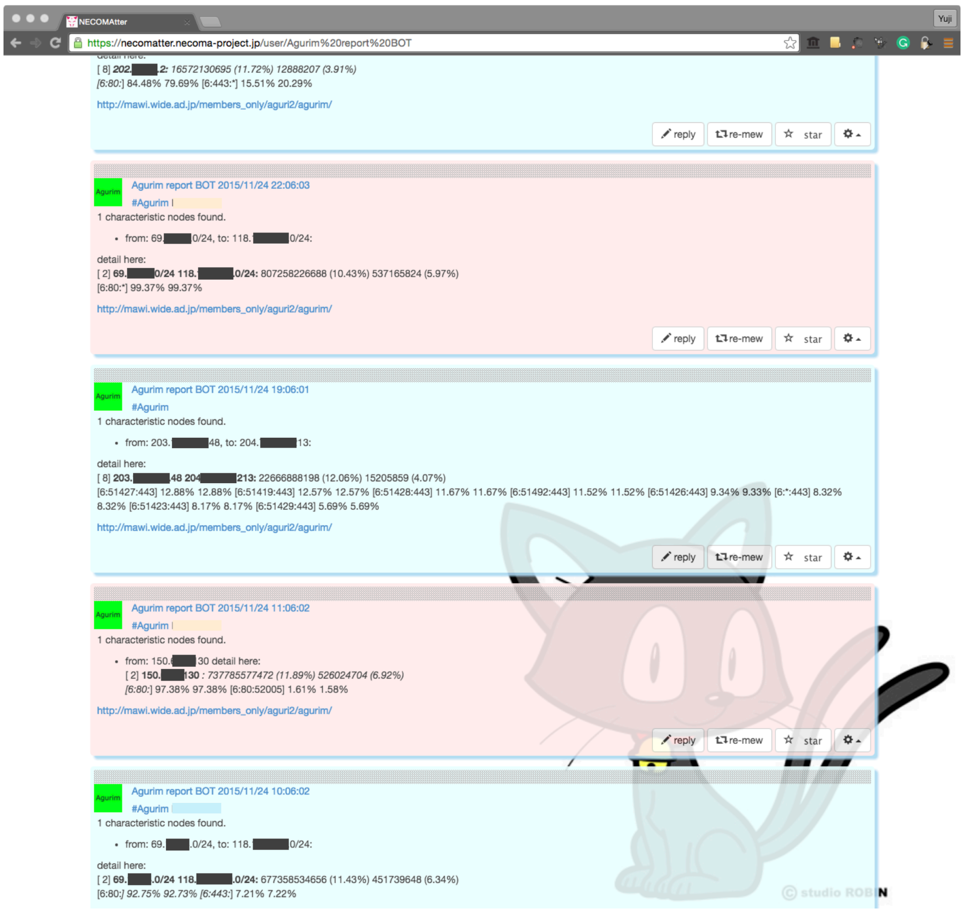
3.5.5.1 Malicious behaviors detected by agurim

agurim found malicious traffic behaviors in the WIDE Cloud. Figure 3.36 shows the screen capture of NECOMAtter. agurim posted malicious network behaviors to NECOMAtter. Each of the posts includes IP addresses, protocol numbers, port numbers, and its share of the whole traffic bandwidth. agurim found the distinctive flows and aggregated the information, before rendering it. According to the aggregated information, the NECOMAtter BoT for agurim picked up the information about the first occurrence of the event, as well as, the information about the event exceeding the monitoring threshold, and posted to the NECOMAtter feeds. Table 3.5 shows the malicious behaviors detected by agurim, from July to November in 2015.

Based on the posted information, the mitigation mechanisms such as flow redirection rules, d4c, or ACL filtering are launched by the mitigator.

3.5.5.2 Evaluation of the mitigation performance

To demonstrate the performance of our concept, “demand-and-opportunity based mitigation”, we evaluated the mitigation performance of the d4c implementation. As described before, d4c can filter malicious DNS queries by specifying the contents and condition. For simple evaluation, we evaluated the throughput performance with the following conditions.


NECOMAtter
Yuji
https://necomatter.necoma-project.jp/user/Agurim%20report%20BOT
reply
re-mew
star
Agurim
© studio ROBIN

[8] **202.[redacted].2:** *16572130695 (11.72%) 12888207 (3.91%)*
[6:80:] 84.48% 79.69% [6:443:*] 15.51% 20.29%

http://mawi.wide.ad.jp/members_only/aguri2/agurim/

Agurim report BOT 2015/11/24 22:06:03
#Agurim
1 characteristic nodes found.

- from: 69.[redacted]0/24, to: 118.[redacted]0/24:

detail here:
[2] **69.[redacted]0/24 118.[redacted].0/24:** 807258226688 (10.43%) 537165824 (5.97%)
[6:80:*] 99.37% 99.37%

http://mawi.wide.ad.jp/members_only/aguri2/agurim/

Agurim report BOT 2015/11/24 19:06:01
#Agurim
1 characteristic nodes found.

- from: 203.[redacted]48, to: 204.[redacted]13:

detail here:
[8] **203.[redacted].48 204[redacted]213:** 22666888198 (12.06%) 15205859 (4.07%)
[6:51427:443] 12.88% 12.88% [6:51419:443] 12.57% 12.57% [6:51428:443] 11.67% 11.67% [6:51492:443] 11.52% 11.52% [6:51426:443] 9.34% 9.33% [6:*:443] 8.32% 8.32% [6:51423:443] 8.17% 8.17% [6:51429:443] 5.69% 5.69%

http://mawi.wide.ad.jp/members_only/aguri2/agurim/

Agurim report BOT 2015/11/24 11:06:02
#Agurim
1 characteristic nodes found.

- from: 150.[redacted]30 detail here:
  [2] **150.[redacted]130** *: 737785577472 (11.89%) 526024704 (6.92%)*
  [6:80:] 97.38% 97.38% [6:80:52005] 1.61% 1.58%

http://mawi.wide.ad.jp/members_only/aguri2/agurim/

Agurim report BOT 2015/11/24 10:06:02
#Agurim
1 characteristic nodes found.

- from: 69.[redacted].0/24, to: 118.[redacted]0/24:

detail here:
[2] **69.[redacted].0/24 118.[redacted].0/24:** 677358534656 (11.43%) 451739648 (6.34%)
[6:80:*] *92.75% 92.73% [6:443:]* 7.21% 7.22%

Figure 3.36: Screenshot of the posts by agurim on NECOMAtter (Partially masked).

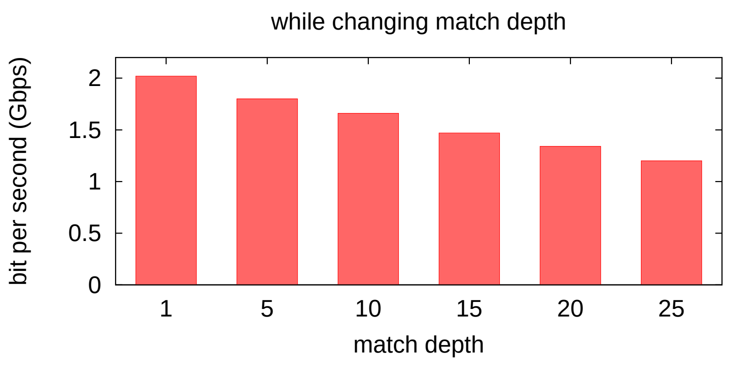


Figure 3.37: Throughput of DNS packet forwarding with text matching on one QNAME.

1) Throughput of DNS packet forwarding with no condition

2) Throughput of DNS packet forwarding with text matching on one QNAME

3) Throughput of DNS packet forwarding with text matching on several QNAMEs

The evaluation environment is shown in Table 3.6. The dummy DNS packets are sent using an IXIA hardware network tester, with a 10Gbps Ethernet interface, and evaluate the throughput of received packets.

Table 3.7 shows the result of the evaluation on scenario 1). When the packet size is larger than 512 kbytes, the performance is almost wire-rate. However, normal DNS inquiries with QNAME do not exceed 512 kbytes, so the result for 64-byte packets is also important. The implementation of d4c uses netmap[37] for its packet I/O. So one physical CPU core is allocated to one 10Gbps Ethernet. The results can be interpreted as the maximum performance of one CPU core with one 10Gbps interface. If the server has enough CPU cores and 10Gbps interfaces, the maximum performance will increase. However, we think that 2.02Gbps is enough for mitigating DNS inquiries.

Figure 3.37 shows the result of the evaluation of scenario 2). In this evaluation, the length of the QNAME parameter is changed. Usually, a DNS QNAME is a Fully Qualified Domain Name (FQDN), and the name is separated into several levels by dots. The dummy packets generated by the tester include a QNAME. The "match depth" in the graph shows the number of domain name levels (separated by dots) in the QNAME. For example, the depth of "xxx.zzz" is two, and the depth of "xxx.yyy.zzz.aaa.bbb" is five. The performance decreases as the depth increases.

The result of the evaluation of scenario 3) is shown in Figure 3.38. In this evaluation, the number of QNAME varies. In the previous evalu-

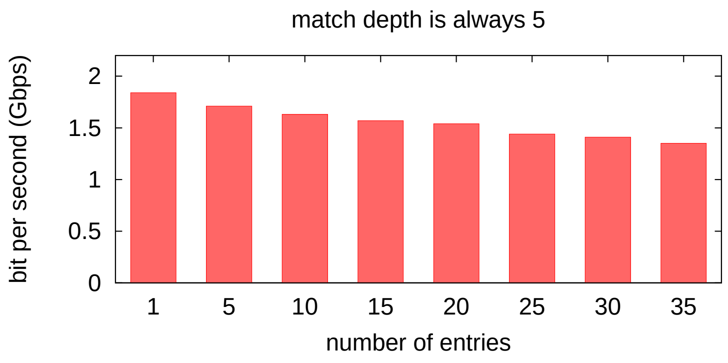


Figure 3.38: Throughput of DNS packet forwarding with text matching on several QNAMEs.

ation, the number of QNAMEs was constant (one), but the length of the QNAME changed. The "number of entries" represents the number of specified QNAMEs for mitigation. The depth of each QNAME is five.

From the above evaluations, the concept of "demand-and-opportunity based mitigation" has proven to be feasible. The number of servers for mitigation could be increased when a greater throughput is needed for mitigation.

Table 3.5: Detected malicious behaviors by agurim

Date and Time (JST)	IP address	Description
2015/11/24 19:06:01	203.178.XXX.48	https
2015/11/24 09:06:05	203.178.XXX.48	https
2015/11/13 07:06:01	203.178.XXX.142	https
2015/11/08 08:06:02	203.178.XXX.19	NTP amp
2015/11/08 08:06:02	203.178.XXX.52	NTP amp
2015/11/04 02:06:01	203.178.XXX.19	UDP flood
2015/11/02 04:06:02	203.178.XXX.19	UDP scan
2015/10/28 07:06:01	203.178.XXX.200	http
2015/10/26 05:06:01	203.178.XXX.19	UDP scan
2015/10/17 23:06:02	203.178.XXX.177	NTP amp
2015/10/01 08:06:02	203.178.XXX.42	http
2015/10/01 02:06:02	203.178.XXX.177	NTP amp
2015/09/19 01:06:02	203.178.XXX.100	DNS amp
2015/09/18 05:06:01	203.178.XXX.210	NTP amp
2015/09/17 02:06:02	203.178.XXX.80	unknown
2015/09/10 06:06:02	203.178.XXX.42	NTP amp
2015/09/06 05:06:01	203.178.XXX.175	unknown
2015/09/02 08:06:02	203.178.XXX.175	unknown
2015/09/01 07:06:03	203.178.XXX.177	NTP amp
2015/08/29 09:06:01	203.178.XXX.210	NTP amp
2015/08/15 08:06:01	203.178.XXX.80	http
2015/08/05 02:06:02	203.178.XXX.80	rsync
2015/07/13 05:06:02	203.178.XXX.142	TCP scan
2015/07/06 08:06:01	203.178.XXX.208	unknown
2015/06/30 11:06:01	203.178.XXX.208	TCP scan
2015/06/19 14:06:01	203.178.XXX.80	unknown
2015/06/06 22:41:07	203.178.XXX.206	http
2015/06/02 13:06:02	203.178.XXX.8	GRE

Table 3.6: Evaluation environment

Server	DELL PowerEdge R430
CPU	Intel Xeon E5-2603v3 x 2 (total 12 cores)
Memory	64GBytes
Network Interface Card	Intel X520-DA2

Table 3.7: Throughput of DNS packet forwarding with no condition

Packet Size	pps	bps
64byte	4.21M	2.02G
512byte	2.32M	9.53G
1500bytes	821K	9.84G

3.6 Improving Cloud-based Intrusion Detection through High-Performance Virtualization

In this Section, we describe an intrusion detection system for cloud environments taking advantage of performance-improving technologies provided by hypervisors.

3.6.1 Proposal

Managing many virtual machine instances is a complex procedure achieved by the virtualization software. This vast and complex piece of software is prone to attacks due to vulnerabilities that may exist in its code. A malicious VM can take advantage of these vulnerabilities to attack the other VMs hosted on the same physical machine. In this section, we describe the implementation and the results of a cloud-based intrusion detection system (IDS) based on Single Root I/O Virtualization technology (SR-IOV). In the following paragraphs and sections we introduce an overview of our system and the results obtained. The implementation is based on the Snort IDS which was integrated with Xen hypervisor[10]. We also used other free and stable components such as BASE, Apache MySQL and Barnyard2 in order to monitor all inter- and intra-hypervisor traffic and display real-time results via a Web interface.

3.6.2 Tools and Technologies

This Section lists different tools and technologies on which our proposal builds.

3.6.2.1 The Snort IDS

Snort[11] is one of the most widely used open source Network Intrusion Detection Systems (NIDS). It has a rule set that contains about 10000 rules. Searching every packet for all of these strings requires significant resources, both in terms of the computation capacity needed to process a packet, as well as, the amount of memory needed to store the rules. Snort from version 2.6 and onwards uses only flavors of the Aho-Corasick algorithm for exact-match pattern detection. Specifically, it contains a variety of implementations that are differentiated by the type of finite automaton they use (NFA or DFA), and the storage format they use to keep it in memory (full, sparse, banded, trie, etc.). It should be mentioned, however, that the best performance is achieved with the full version that uses a deterministic finite

[10]http://www.xenproject.org/
[11]https://www.snort.org/

automaton (DFA) at the cost of high memory utilization. According to the extensive knowledge we compiled on Snort IDS[12], we thus chose it for the cloud-based intrusion detection of the current implementation.

3.6.2.2 SR-IOV Technology

SR-IOV feature was used because it provides better I/O performance than the traditional method of triggering a DMA operation whenever a VM is willing to use the network interface. This means that an interruption of the CPU was avoided by using SR-IOV. Otherwise the hypervisor should translate the memory address of this DMA operation to the one in the corresponding hosts address space.

3.6.2.3 Logging and Results Presentation

The Snort IDS produces a large amount of output in text and binary format. We have used the Barnyard2[13] library in order to parse those logs and insert the data into a MySQL database. The Barnyard2 interpreter allows Snort to write to the disk in an efficient manner, oblivious of network traffic loss, thus allowing Snort to log all outputs to a MySQL database via Barnyard2. The results stored in the database are then displayed in a Web interface through BASE[14] (Basic Analysis and Security Engine). The BASE tool is able to search and process the database containing the security event logs stored by Snort. The tool is also capable to display both layer-3 and layer-4 packet information as presented in the respective figures. The results are then presented to a Web interface via an Apache HTTP server.

3.6.3 Architecture and Results

The system's architecture is presented in Figure 3.39. The intra-hypervisor communication is achieved via the Linux bridge. Snort instance is running on Dom-0 and is able to monitor all available network interfaces (ethX, bridge and all vifX.Y) used by the VMs of the system.

[12] http://dcs.ics.forth.gr/Activities/Projects/snort.html
[13] https://github.com/firnsy/barnyard2
[14] http://sourceforge.net/projects/secureideas/

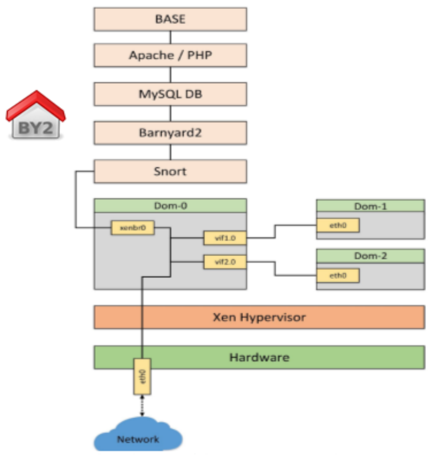
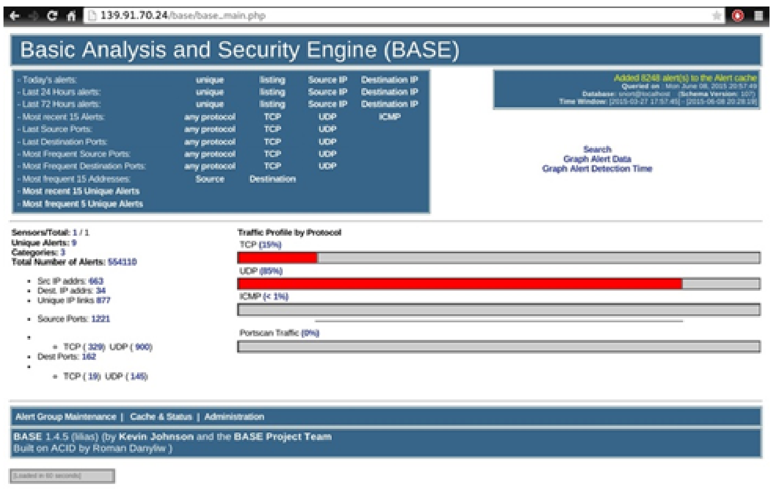


Figure 3.39: The architecture of the cloud-based IDS implementation.

The network traffic that is produced by either Dom-0 (Host OS) or Dom-X (the VMs) is captured by Snort which produces logs. Figure 3.40 and Figure 3.41 present the results of the data captured by Snort while operating in such a system.


139.91.70.24/base/base_main.php

Basic Analysis and Security Engine (BASE)

- Today's alerts:	unique	listing	Source IP	Destination IP
- Last 24 Hours alerts:	unique	listing	Source IP	Destination IP
- Last 72 Hours alerts:	unique	listing	Source IP	Destination IP
- Most recent 15 Alerts:	any protocol	TCP	UDP	ICMP
- Last Source Ports:	any protocol	TCP	UDP	
- Last Destination Ports:	any protocol	TCP	UDP	
- Most Frequent Source Ports:	any protocol	TCP	UDP	
- Most Frequent Destination Ports:	any protocol	TCP	UDP	
- Most frequent 15 Addresses:	Source	Destination		
- Most recent 15 Unique Alerts				
- Most frequent 5 Unique Alerts				

Added 8248 alert(s) to the Alert cache
Queried on : Mon June 08, 2015 20:57:49
Database: snort@localhost (Schema Version: 107)
Time Window: [2015-03-27 17:57:45] - [2015-06-08 20:28:19]

Search
Graph Alert Data
Graph Alert Detection Time

Sensors/Total: 1 / 1
Unique Alerts: 9
Categories: 3
Total Number of Alerts: 554110

- Src IP addrs: 663
- Dest. IP addrs: 34
- Unique IP links 877
- Source Ports: 1221
-
  - TCP (329) UDP (900)
- Dest Ports: 162
-
  - TCP (19) UDP (145)

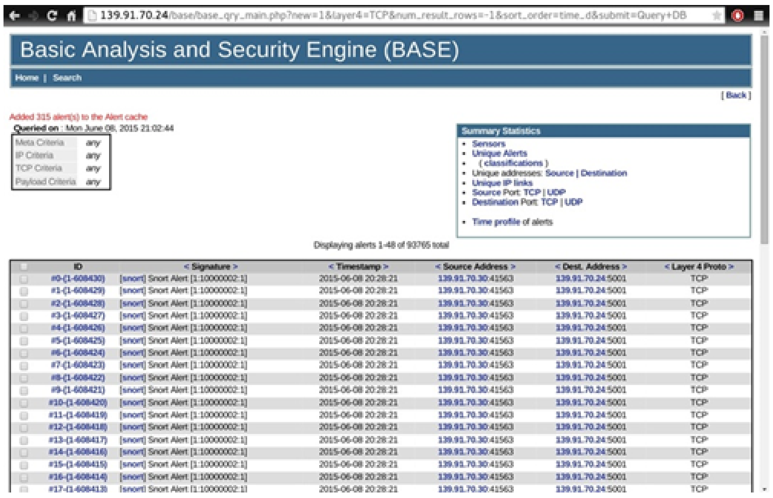
Traffic Profile by Protocol
TCP (15%)
UDP (85%)
ICMP (< 1%)
Portscan Traffic (0%)

Alert Group Maintenance | Cache & Status | Administration

BASE 1.4.5 (lilias) (by Kevin Johnson and the BASE Project Team
Built on ACID by Roman Danyliw)

[Loaded in 60 seconds]

Figure 3.40: An overview of the data captured by Snort with the help of BASE engine.


139.91.70.24/base/base_qry_main.php?new=1&layer4=TCP&num_result_rows=-1&sort_order=time_d&submit=Query+DB
Home | Search
[Back]

Basic Analysis and Security Engine (BASE)

Added 315 alert(s) to the Alert cache

Queried on : Mon June 08, 2015 21:02:44

Meta Criteria	any
IP Criteria	any
TCP Criteria	any
Payload Criteria	any

Summary Statistics

- Sensors
- Unique Alerts
- (classifications)
- Unique addresses: Source | Destination
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying alerts 1-48 of 93765 total

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
#0-(1-608430)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#1-(1-608429)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#2-(1-608428)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#3-(1-608427)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#4-(1-608426)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#5-(1-608425)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#6-(1-608424)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#7-(1-608423)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#8-(1-608422)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#9-(1-608421)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#10-(1-608420)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#11-(1-608419)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#12-(1-608418)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#13-(1-608417)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#14-(1-608416)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#15-(1-608415)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#16-(1-608414)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP
#17-(1-608413)	[snort] Snort Alert [1:10000002:1]	2015-06-08 20:28:21	139.91.70.30:41563	139.91.70.24:5001	TCP

Figure 3.41: Alerts captured for the respective VMs of the system.

The system administrator can navigate through the Web interface and focus on specific traffics of the respective VMs monitored. Future work on the described implementation could include the usage of custom filters to separate the traffic produced per VM in order to provide clearer results to the user of the system.

4 Endpoint-level Cyberdefense Mechanisms

This Chapter groups the description of endpoint-level defense mechanism prototypes, allowing improved resilience for browsers and smartphones. These mechanisms are protecting directly the end-user against phishing attacks and malware.

- Section 4.1 describes how users with different level of expertise can be protected against phishing websites. The mechanism uses eye movement tracking to ensure that the user sees the information that is most relevant to determine whether a website is a phishing site or not.
- Section 4.2 describes a defense mechanism deployed on Android smartphones protecting the user against malware making calls or sending messages to premium numbers.
- Section 4.3 describes offloading the firewall function from smartphones to wireless access points.

4.1 Improving Resilience through Personalization

In this Section, we describe means to counter phishing websites with eye movement tracking, via a browser extension, and by adapting the defense to the user's level of experience.

4.1.1 Threat description

Phishing is a fraudulent activity defined as the acquisition of personal information by tricking an individual into believing the attacker is a trustworthy entity. The number of phishing sites is still growing. According to trend reports published by the Anti-Phishing Working Group [2], the number of reported phishing sites was 44,212 in March 2014, far surpassing the 25,630

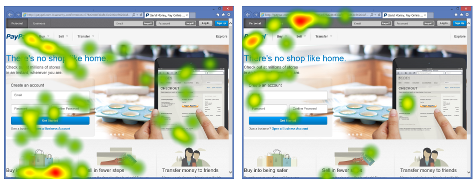

(a) Novice users (b) Experts

Figure 4.1: Eye-tracking in a phishing site

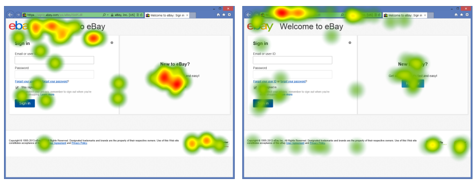

(a) Novice users (b) Experts

Figure 4.2: Eye-tracking in a legitimate site

in March 2008. The annual worldwide impact of phishing in 2013 could be as high as $5 billion [38].

Phishing attackers lure people through the use of a phishing email, as if it was sent by a legitimate corporation. Email recipients are then attracted to a phishing site, which is the replica of an existing web page, and are fooled into submitting personal, financial, and/or password data.

4.1.2 Overview of Resilient Defense for Phishing

In order to improve resilience for phishing prevention, we would like to introduce two key terms - Personalization and Information Pipelining. Personalization here means adjusting cyberdefense to each end-user. The resilient defense mechanism involves the recognition of users' awareness when visiting trustworthy or phishing websites, and the differentiation of defense measures according to the type of user. Information Pipelining for resilient phishing protection needs to interconnect defense modules developed in our Work Package 2 for providing suitable defense for the users.

The recognition of awareness is a fundamental concept for determining the skills of a user. We consider such case in which a user assesses a website by its contents, and ignore meaningful signals displayed in the browser's address bar. Figures 4.1 and 4.2 respectively show the heat maps of the eye fixation locations and durations on both phish and legitimate website for novice users (a) and expert users (b). The red color denotes the areas that attracted the user's gaze the most and green denotes moderate gaze activity. In the phishing case, the novice looked at the web content but ignored the browser's address bar while assessing credibility, as shown in Figure 4.1a. Since the text and visuals in phishing sites are quite similar to the ones in legitimate sites, the novice failed to label the phishing site correctly. In the legitimate case, the novice also only paid attention to the content of a web page as shown in Figure 4.2a. By 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 Figures 4.1b and 4.2b.

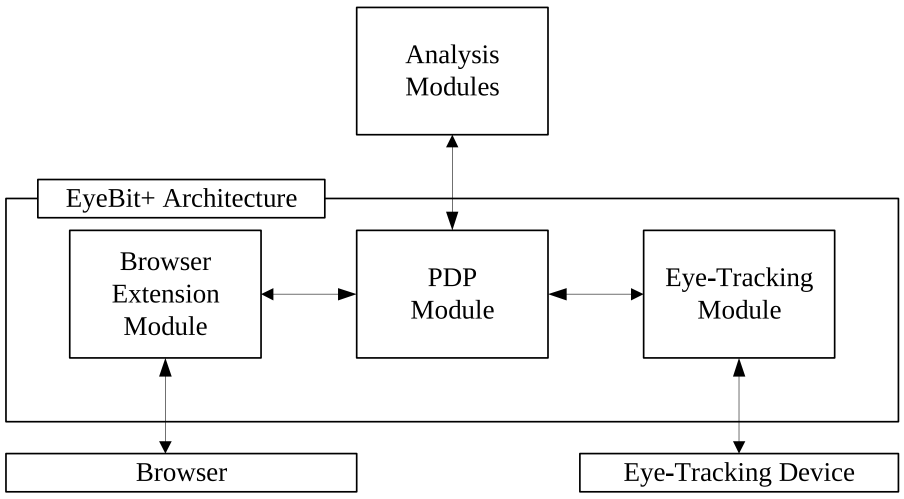


Figure 4.3: The architecture of EyeBit+

We therefore hypothesized that the analysis of eye movement on the particular areas of interest (AoIs) would allow to extract what are the criteria that helped the user in making a trust decision. To assess our hypothesis, we conducted two types of participant-based experiments. In the first experiment, we analyzed the correlation between eye movements and decision criteria to confirm whether eye fixations can be used as decision criteria indicators. The second experiment investigated whether the eye movement allows to estimate the likeliness of a user to fall victim to phishing.

Our results showed that the average error was 32.4% if users assessed the credibility of the website by paying attention to web content, 21.3% if users looked at the URL of the site, and 13.5% if users checked the security information of the browser. We also verified our ability to predict the likelihood that users may fall victim to phishing attacks on the basis of the analysis of their eye movement patterns, and found that it can be estimated with a probability of 79.3%. Further, we observed that experts would try to find some trustworthiness information as soon as they have begun browsing the websites.

4.1.3 Implementation

This section demonstrates the implementation of our personalized phishing protection system, named *EyeBit+*, which upgrades the proof of concept of EyeBit [31], for interacting with analysis modules developed in WP2.

4.1.3.1 Personalization

The requirements of components for personalization are as follows.

- **Web inputs control via reconfiguration.**
  The module 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.**
  The module 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.**
  The module should be able to locate the address bar within the screen.

The architecture is shown in Fig. 4.3. It consists of (i) an eye-tracking module, (ii) a browser extension module, and (iii) a PDP 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[1] as the eye-tracking device. Its software development kit (SDK) allows to program APIs to communicate with eye tracker web server which capture and return user's eye positions in JavaScript Object Notation (JSON) format. Eyebit+ can then retrieve values from this server via HTTP transactions.

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 stored 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 algorithms are shown in Algorithm 1. At the begining of the browsing experience, the system deactivates all input forms in the websites. From our observation, we recognize experts as users that will check the address bar as soon as they begin browsing the websites. Our system provides

[1]The Eye Tribe Tracker: https://theeyetribe.com

Algorithm 1 Pseudo Code of Personalization Component

```
for all websites do
    Deactivate web input forms
    if a user see the address bar at time $T$ then
        if $T < 5$ seconds then
            A user is expert
            Check with ATOS's algorithm
            if the website is labeled as phishing then
                Deactivate input form
            else
                Activate input forms
            end if
        else if $5 \geq T < 10$ seconds then
            A user is the average type
            Check with ATOS algorithm
            if the website is labeled as phishing then
                Deactivate input form
            else
                Check with UT algorithm
                if the website is labeled as phishing and its confidence is high then
                    Deactivate input form
                else
                    Activate input forms
                end if
            end if
        else
            A user is novice
            Check with ATOS algorithm
            if the website is labeled as phishing then
                Deactivate input form
            else
                Check with UT algorithm
                if the website is labeled as phishing and its confidence is medium or high then
                    Deactivate input forms
                else
                    Activate input forms
                end if
            end if
        end if
    end if
end for
```

ATOS's high precision phishing detection for experts. If the algorithm labels the site as legitimate, the system reactivates all input forms. It should be noted that the algorithm works faster and achieves high precision in compensation for recall; it does not label legitimate sites as phishing, but sometimes labels phishing sites as legitimate if the phishing site is sophisticated enough. If the number of dots in the URL is reasonably lower, this algorithm often labels a site as legitimate.

For the average users, we would like to provide another defense, UT's machine learning-based phishing detection. It checks information of registered domains, popularity of contents, suspicious links, symbols and forms as well as the number of dots in the URL. This algorithm calculates the likelihood of the site to be a phishing one, and also outputs the confidence for its detection results. We can categorize its confidence into low, medium, and high. The site seems to be definetely phishing if the confidence is high. The medium confidence means it seems to be legitimate but has a possibility of being a phishing site. For the average users, our system provides ATOS's and UT's algorithms. Even if the ATOS's algorithm labels the site as legitimate, it is still deactivated if the site was deemed to be phishing one with a high confidence rate.

In the case of novice users, who often fail to make the correct decision, the system should emphasize the elimination of all potential phishing websites. The algorithm works similiarly as for the average user, but the system activates input forms only when the ATOS's algorithm judges the site legitimate and UT's algorithm has deemed that there is a low level of confidence to consider the site as a phishing one.

4.1.3.2 Information Pipelining

The overview of our information pipelining is shown in Fig. 4.4. Our resilient cyberdefense system, the policy decision point (PDP), communicates with the web browser extension which is the policy enforcement point (PEP) of the proposed design. The PEP also takes inputs from an eye-tracking camera which acts as a sensor. Additionally, it includes analysis modules developed in Task T2.3 (Threat Analysis Platform). The initial workflow is as follows.

Phase 1 After an end-user starts browsing, a browser extension retrieves his/her eye movement records with an eye-tracking camera, as well as the website's URL.

Phase 2 The browser extension module sends the URL and the user type to the PDP module (novice, expert, etc.[2]).

Phase 3 If the end-user is an expert, the PDP sends the URL to the phishing analysis module provided by ATOS. It only checks the domain name, so response time might be very short. If the end-user is neither an expert nor a novice, the PDP sends the URL to both phishing analysis modules. UT's module analyzes contents and reputation, so it needs more time compared to ATOS' module. If the user is a novice, the PDP sends a request to the IP address, DNS, and AS number analyses modules.

Phase 4 Each module returns its analysis results to the PDP. Analysis results would differ for each type of users.

Phase 5 The PDP sends the decision results to the PEP.

[2]A novice is a user not used to check the address bar while an expert quickly checks the bar while browsing. Intermediate levels/patterns may apply as well

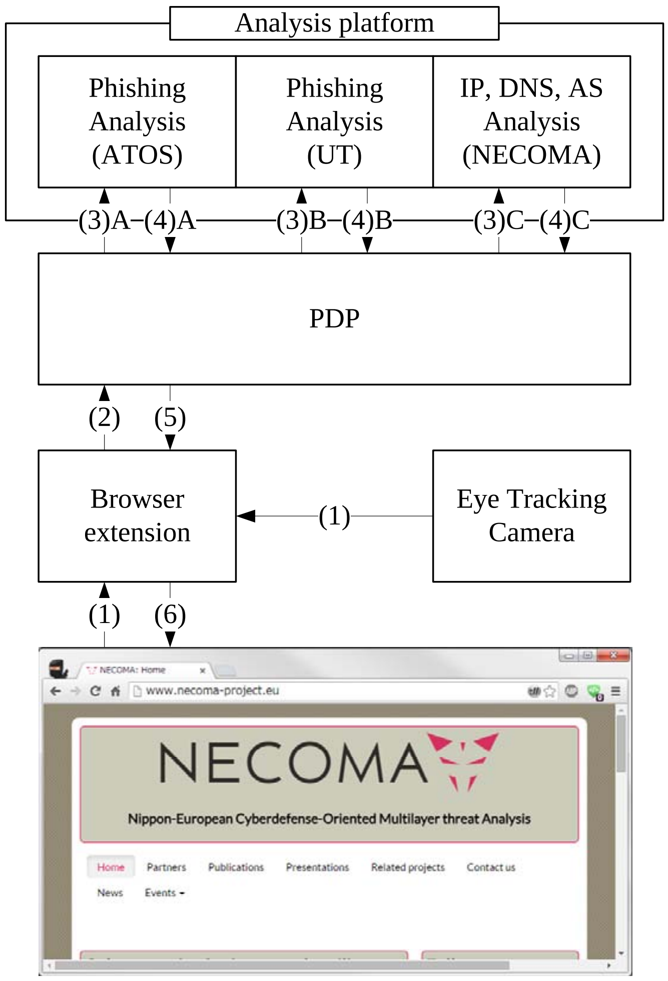


Figure 4.4: Design overview of personalized defense.

Phase 6 The PEP commits changes to defend the user, e.g., blocking, alerting, or deactivating web input forms.

Next, for each phase, we will describe the data flow between components.

4.1.3.3 Phase 1: Interaction with the browser extension

As a component of the browser, the browser extension can retrieve a URL which was just visited by the end-user. It also monitors users' behavior, e.g., eye movements. The mechanism was already developed in [31] and is currently available for download. The EyeTracking Module[3] bridges eye-

[3]`https://github.com/necoma/eyebit_server`

```
{
    "address": [
    {
        "ip":    xxx.xxx.xxx.xxx
        "asn":    xxxx
    ],
    "category": "phish",
    "source": "name",
    "url": "http://...."
}
```

Figure 4.5: Request message from the PDP to the analysis module in Phase 3

tracking devices with the browser extension. Eye-Tribe provides an SDK to output JSON-formatted data containing eye position in the screen and its status, e.g., eye fixation. The module therefore can identify whether the users look at the browser's address bar or not. We assume that modern web browsers do show the website's URL and security information in the address bar.

4.1.3.4 Phase 2: Message from the browser extension to the PDP

Based on the eye tracking results, the PDP module will categorize users into three types: experts, novices, and others. The tentative categorization is explained as follows: experts are users that look at the address bar within 5 seconds after they start browsing a web site; novices are users who never check the bar; the rest of users are categorized as others.

It should be noted that we initially thought that NECOMAtter was available for exchanging messages between the browser extension and PDP modules. However, due to privacy concerns, we re-designed our architecture in order to incorporate the PDP module into EyeBit+.

4.1.3.5 Phase 3: Message from the PDP to analysis modules

The conversation in phase 3 can be described using a JSON object as shown in Figure 4.5. The PDP selects the suitable analysis module according to the type of the user. It then requests the module with a URL. If necessary, it also requests the module using the IP address, and the AS number.

```
{
    "address": [
    {
        "ip":    xxx.xxx.xxx.xxx
        "asn":    xxxx
    ],
    "category" : "phish",
    "source": "name",
    "url": "http://....",
    "confidence": "high",
    ]
}
```

Figure 4.6: Response message from the analysis module to the PDP in Phase 4

4.1.3.6 Phase 4: Message from analysis modules to the PDP

The conversation in phase 4 can be described using a JSON object as shown in Figure 4.6. Each analysis module is expected to respond using the n6 format (more details are provided in Deliverable D3.2). The response might include the analysis results with a confidence level.

4.1.3.7 Phase 5: Message from the PDP to the browser extension

Based on the response received from the analysis modules, the PDP will decide if it provides resilient defense or not. For experts, it will adjust to reduce false positive errors when they can identify phishing by themselves. In contrast to that, novices cannot identify whether the site is phishing or not, the PDP will therefore provide a defense focusing on reducing false negative errors. The decision algorithm was described in Algorithm 1.

4.1.3.8 Phase 6: Interaction with the browser extension

According to the message from the PDP, the browser extension, as the PEP, will enforce a defense, succesfully protecting users from phishing sites. Possible defense methodologies include compulsory blocking, deactivation of forms, and displaying alert messages.

4.1.4 Preliminary Evaluation

This Section provides the results of our performance experiments. We prepared an emulated phishing site, made from Web pages of legitimate enter-

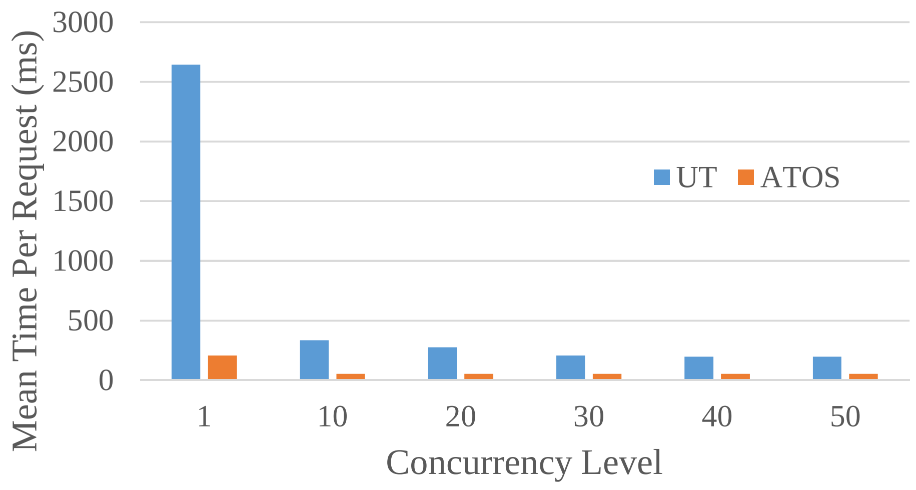


Figure 4.7: Stress Test Analysis of the Analysis Modules using ApacheBench

prises, in a test environment. We then performed performance evaluation with ApacheBench, a stress test tool.

Figure 4.7 shows the benchmark results of using UT's analysis module and ATOS's analysis module on several concurrency levels, i.e., the number of clients which concurrently send requests to the modules. We queried these analysis modules via HTTPS 100 times, and observed the mean time per request. Since UT's analysis module needs to analyze the content with natural language processing, we found that the overhead of the UT's module was greater than for the ATOS's module. It can be naturally assumed that there might be a trade-off between the security and the convenience of users. However, the performance overhead for UT's module was just a few seconds, therefore, we speculated that the loss of convenience might be acceptable for Web users who are likely to be victims of phishing sites.

4.2 Smartphone User Protection

In this section, we first describe the weaknesses in the Android ecosystem that make it possible for SMS fraudster and premium dialer malware to exist. Finally, we present our system which introduces significant countermeasures against these families of malware.

4.2.1 Threat Description

Android made its first appearance on September 2008 and since then it has managed to become the leading OS in the market, largely because of its open source nature. IDC reports that Android leads the smartphone market with almost 85% of the market share in the second quarter of 2014. Thanks to its popularity, Android has become a common target for malware authors, aiming to exploit it for their own personal gain. Cisco mentions in its 2014 Annual Security Report that of all mobile malware in 2013, 99% targeted Android devices. Similarly, F-Secure states in its Mobile Threat Report for the first quarter of 2014, that 99% of their findings during that period of time were designed to run on the Android platform. One class of malware that can cause financial loss to Android users is SMS fraudsters and premium dialers.

4.2.2 Tools and Technologies

We will describe the tools and technologies on which our proposal builds.

4.2.2.1 Applications on Android

There are two primary sources for applications:

- **Pre-Installed Applications**: Android includes a set of pre-installed applications including phone, email, calendar, web browser, and contacts. These function both as user applications and to provide key device capabilities that can be accessed by other applications. Pre-installed applications may be part of the open source Android platform, or they may be developed by a manufacturer for a specific device.

- **User-Installed Applications**: Android provides an open development environment supporting third-party applications. Third-party applications can be installed from various app stores, including Google Play Store, or even obtained from the internet and other unknown sources as a package file.

4.2.2.2 Application Security Model

The Android Application Sandbox isolates application data and code execution from other applications. Direct communication and data exchange between applications is only possible through Binder, a kernel-level interprocess communication (IPC) driver and core component of Android. Binder also enables the communication between applications and the Android framework (middleware). The Android framework allows applications to access system resources (network, storage, cameras, etc.) only if the requesting application has the permissions needed. As a result, permissions are declared for each application by its developers. These permissions, as of Android 5.1, are granted once and forever during installation time. In addition, users cannot selectively reject any of them during installation or run-time.

4.2.2.3 Exploiting the Android Security Model and Ecosystem

As most users are unaware of the importance of permissions, they blindly accept any permission listed during installation time. This combined with the ability to install applications from unknown sources and authors, creates a significant security threat. Well known applications can be repackaged with malicious code and distributed through third-party stores. Apps that request the permissions required to send messages (SMS, MMS) and perform phone calls can therefore exploit the functionality provided by the framework, which allows them to communicate with premium numbers, even in the background without the user knowing. As a result, this type of malware has become pretty common, costing users money.

4.2.3 Architecture and results

Our system requires modification of the Android core, as well as the Messaging application, which is developed by Google and comes pre-installed on every device. We also include a new system-level pre-installed application that is required for our system to operate.

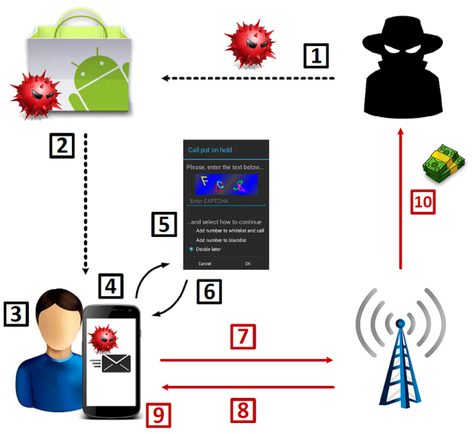

1
2
3
4
5
6
7
8
9
10
Call put on hold
Please, enter the text below...
Enter CAPTCHA
...and select how to continue
Add number to whitelist and call
Add number to blacklist
Decide later
Cancel
OK

Figure 4.8: Execution sequence of the system

As shown in Figure 4.8, when an application attempts to initiate a phone call or send a message, the application we introduced takes over and shows a pop-up dialog, displaying the destination number to the user and allowing him to either blacklist it or allow the action to be performed. In order for the selected action to be performed, the user has to correctly solve a CAPTCHA code or a simple mathematical operation. This implementation is secure as third-party applications by design cannot access or alter the pop-up dialog of any other application. The subsequent blacklists and whitelists are stored in a private database and can be modified at any time by the user as he is being prompted on each call and message sent. The process is displayed in Figure 4.9.

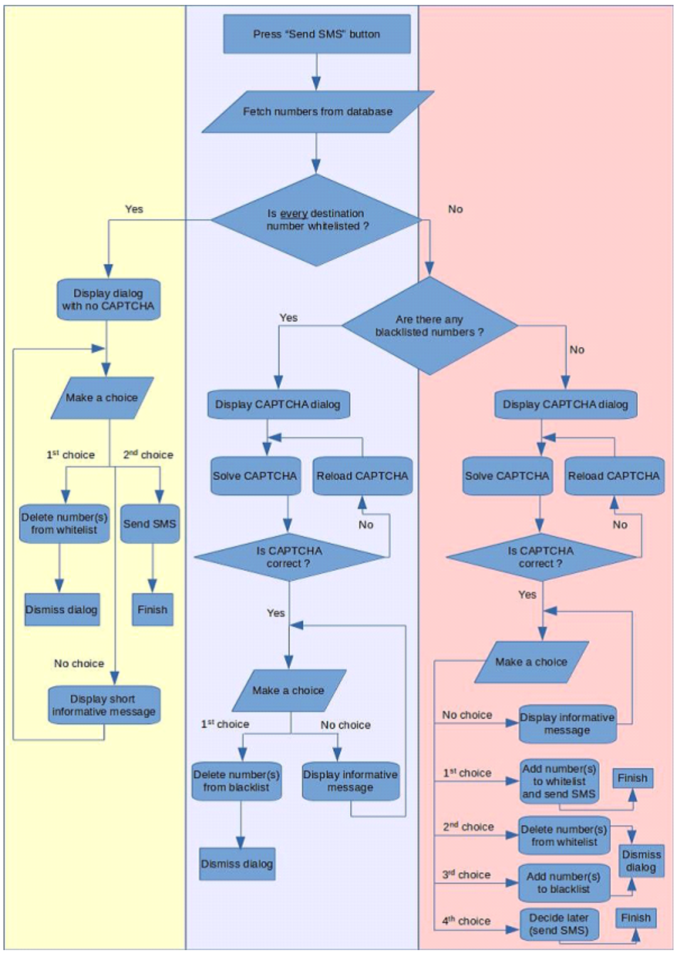

Press "Send SMS" button
Fetch numbers from database
Is every destination number whitelisted ?
Yes
No
Display dialog with no CAPTCHA
Are there any blacklisted numbers ?
Make a choice
1st choice
2nd choice
Delete number(s) from whitelist
Send SMS
Dismiss dialog
Finish
No choice
Display short informative message
Display CAPTCHA dialog
Solve CAPTCHA
Reload CAPTCHA
Is CAPTCHA correct ?
Delete number(s) from blacklist
Display informative message
Add number(s) to whitelist and send SMS
Add number(s) to blacklist
3rd choice
4th choice
Decide later (send SMS)

Figure 4.9: User actions for solving a CAPTCHA puzzle when sending an SMS from the built-in Messaging app

Future work includes integrating web-sources into the system. Blacklists could be updated periodically (e.g. once per week) with data obtained from web servers that distribute known malicious or premium numbers and informing users about the application that triggered the system. Our system can also be improved and strengthened in scenarios where the operating system has been modified (e.g. rooted) and thus is running in a less secure environment.

4.3 Offloading Smartphone Firewalling Using OpenFlow-capable APs

The attack surface of today's cyberspace has been significantly enlarged by the rapid increase in smartphone usage and the proliferation of diverse mobile applications, which introduce a large variety of zero-day vulnerabilities. According to Schmidt et al. [41], smartphones started being targets for malware in June 2004, and as of January 2014, there are roughly 700,000 of cumulated Android malware samples observed according to a report published by Sophos [44].

There are many motivations for attackers to target smartphones. One is the number of users. According to BI Intelligence, at the end of 2013, 6% of the global population own a tablet and 22% own a smartphone, while 20% own a PC[4]. Another motivation is the fact that smartphones often hold much more personal information compared to PCs, keeping detailed records of users' contacts and SMS history as well as sensitive account information regarding banking, emails, social networks, etc.

4.3.1 Threat Description

Mobile malware is one of the most significant cyber threats on smartphones. One outsanding example is iBanking, a criminal software targeting Android terminals. According to the report from RSA [38], iBanking Mobile Bot is controlled over HTTP or via SMS, and it allows bot herders to steal personal information by reading incoming SMS messages, intercept calls to the phone, and obtain files as well as contact lists from the phone. It also has a function of phone fraud which allows a bot master to gain money by calling premium rate telephone service and charging the victim an expensive toll fee. Another function of iBanking is to record audio using the device's microphone. Note that there are many services that require users to input credentials such as credit card and/or PIN number over tone signaling, and this function can recognize the context of typing by the key tones [40].

Smartphones are sometimes used in DDoS attacks. Android DDoS Origin[5] is a malware controlled via SMS messages containing the target IP address and port number. The malware is used to generate traffic from the smartphone towards the target.

As well as malware, smartphone users are faced with phishing, since the user interfaces for smartphones are constrained by their small screens. In legacy personal computers, symbols indicating trust have been developed for a long time. For example, web browsers display a padlock icon to notify about a valid server certificate, the colour green is used in the address bar

[4]http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10
[5]http://news.drweb.com/show/?i=3191&lng=en

when a server is equipped with an extended validation (EV) certificate. The users can therefore be aware of trust information. Thus, improving awareness about security indicators is important in the security of user interfaces. In the case of smartphones, there are too few principles for security awareness.

4.3.2 State of the Art

Generally, smartphone platforms have a method for allowing users to grant permission to an application that needs to access certain types of data. In the case of Android, whenever an application wants to read the contact list, it declares the request for this permission in its manifest file, effectively asking users to grant the permission. If a user thinks something is wrong, he/she may prevent the application from accessing the contact list. In the case of iOS, a permission request dialog box appears at runtime whenever an application first requests access to any of the following six resources: the user's geophysical location, the address book contacts, photos and videos, calendars, reminders, and for Bluetooth pairing[6].

The permission request process is intended to inform users about the risks of installing applications, and hence, users can only make correct security decisions based on permissions if they understand what the permission warnings mean.

However, in the case of Android, users had limited understanding of the permission warnings according to Felt et al. [13]. It should be noted that the authors also observed that Android permissions fail to be informative to most users, while not being completely ineffective. In the case of iOS, Tan et al. reported that permission requests that include explanations are significantly more likely to be approved [45].

There are also risks that users gain administrator's privileges by "rooting" Android or "jailbreaking" iOS. There are various motivations for their act, however it breaks the security model and allows all applications, including malicious ones, to access the data owned by other applications.

Another concern is the battery consumption of smartphones. Recently, almost all client OSes are equipped with the personal firewall function. However, there is a serious factor making the protection of the smartphone difficult, and that is battery consumption. Firewall often acts as a service and it stays active on the smartphone throughout the smartphone's operation. It is therefore important to ensure it has no severe impact on the battery [48].

[6]https://developer.apple.com/library/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS6.html

4.3.3 Proposal Design

In order to provide a resilient protection for smartphone devices, we decided to utilize wireless access points (APs). As we mentioned in Sect. 4.3.1, the defense at the smartphone itself is difficult. Instead, we propose to offload smartphone firewalling function to network switching devices.

One type of defense mechanism is URL filtering, intended to block specific web sites for all smartphones. The objective of filtering is to prevent malware infection, botnet C&C, and phishing. In such attacks, the HTTP protocol is respectively used for malicious applications download, communications with the bot master, and fraudulent websites access.

Packet filtering is another possible defense to prevent smartphone devices from joining DDoS campaigns. Once a smartphone device is infected with a trojan horse, it starts to send data packets to a specified host (the victim) whenever it receives a DDoS attack command. It should be noted that the scope of DDoS mitigation is usually at the infrastructure layer rather than the endpoints, even when protecting smartphones. However, a major principle of DDoS protection stipulates that the mechanism should filter DDoS traffic as close to the attacker as possible. We therefore address DDoS mitigation at the AP, which may be the closest to the source as possible when we consider the case of a smartphone joining a DDoS campaign.

In order to provide a resilient firewall for smartphones, we decided to employ OpenFlow-capable wireless APs. Since OpenFlow provides powerful traffic control schemes, it facilitates the implementation of URL filtering based on the packet payload, as well as packet filtering based on header information of the network and transport protocols such as IP address, and TCP/UDP port numbers.

Alternatively, another challenge when using OpenFlow is to reuse the DDoS mitigation rules generated in Sect. 3.3. PIX-IE is designed to utilize OpenFlow-based solutions, and we therefore consider that the packet filtering rules developed for PIX-IE are applicable to the OpenFlow-capable APs.

In Fig. 4.10, there is a high level overview of the architecture we propose. As we can see, APs are placed close to the smartphone devices, and provide *URL filtering* to thwart smartphone malware, C&C activity, and phishing; as well as *packet filtering* to prevent smartphones from participating in DDoS campaigns.

From these considerations, the requirements of APs are summarized as follows.

Filtering ability The APs must have the ability to protect smartphones from malicious entities. It must therefore have the capacity to filter malicious URLs and to block suspicious IP addresses.

Operability The APs should be able to scale with the load of configurations, in a short time span.

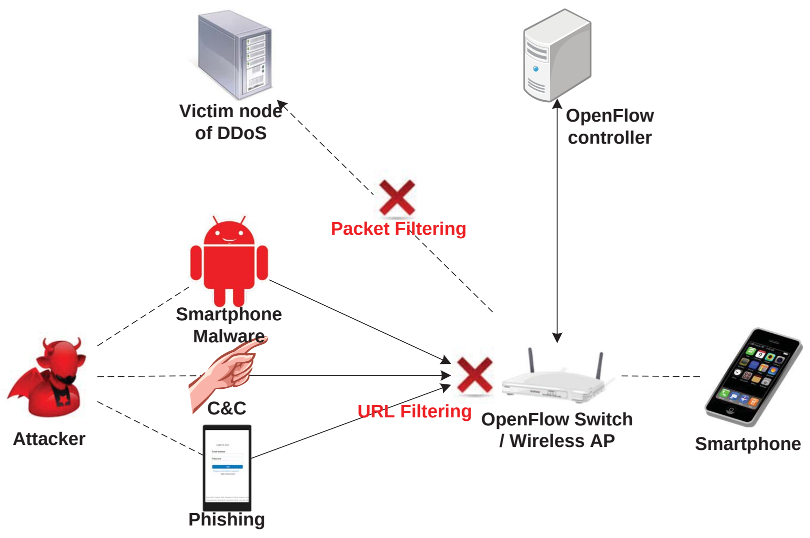


Figure 4.10: An overview of smartphone protection using OpenFlow-capable APs

Table 4.1: Specification of Wireless Access Point

Product	WZR-HP-1G300H
Vendor	Buffalo
CPU	Atheros AR 7161 (680Mhz)
Ram	128 MB
Disk	32 MB
Ethernet	AR 8136
Wireless	Atheros AR9223 (2.4GHz) and AR9220 (5.0GHz) 802.11abgn

4.3.4 Implementation

We explored suitable ways for URL and packet filtering. In the case of URL filtering, proxy servers are widely used to block the access to malicious websites and we consider them well suited for this task. In the case of packet filtering, we found that Software-Defined Network technologies can provide powerful schemes for adding and/or deleting IP addresses in order to prevent smartphone from accessing malicious hosts.

We chose Ryu[7], a Python framework for OpenFlow Controller (OFC), to develop our implementation as an OpenFlow application. Since there are not many OpenFlow-capable wireless access points available, we installed OpenWRT[8], a Linux-based firmware for extensible configuration, into wire-

[7]Ryu: https://github.com/osrg/ryu
[8]OpenWRT: http://wiki.openwrt.org/start

Algorithm 2 Pseudo Code of Resilient Firewall

```
for all packets do
  Read the list of suspicious IP addresses
  if the source or destination IP address is listed then
    Discard the packet
  else
    Forward the packet
  end if
end for
```

less access points and then used Open vSwitch[9] as OpenFlow Switch (OFS). Our configuration for OFS are as follows.

Network Configuration.
: We configured three types of network interfaces: namely, wireless, internet, and management interfaces. The wireless interface is used for within the network where users' smartphone devices are connected. The internet interface is connected to the Internet, and the management interface is for interacting with the OFC. We also setup a pseudo device that we call a bridge interface, in order to forward packets between wireless and internet interfaces.

Access Point Configuration.
: To accept connections from smartphone devices, each AP is configured to support 802.11 n/g wireless networking protocols. The configuration also includes the AP's Service Set Identifier (SSID), encryption, authentication, and signal strength.

OpenFlow Configuration.
: For interactions between OFC and OFS, we assign the private IP address to AP's management interface, and connect to OFS with Fast Ethernet.

Aside from OFS, our algorithm for packet forwarding runnning on OFC is summarized as Algorithm 2. Each packet incoming to OFS will ask OFC to check the IP address with the list of suspicious IP addresses. If it is listed, the OFC controls OFS to discard the issued packet.

Our application developed for OFC runs an HTTP server and accepts requests using the following APIs.

`/add_blocking_ip/{IPADDR}`
: This API is for adding specified IP addresses to the list of suspicious IP addresses.

[9]Open vSwitch: http://openvswitch.org/

`/del_blocking_ip/{IPADDR}`
: This API is for deleting specified IP addresses from the list of suspicious IP addresses.

`/delassoc_client/{IPADDR}`
: This API is used for deassociating a smartphone device which is assigned to the specified IP address.

It should be noted that the disassociation of a smartphone device is not currently supported in OpenFlow protocols. When this API is called, it runs similar OS commands that extract the MAC address from the IP address, and disconnect the issued MAC address powered by IW[10], a command line configuration utility for wireless devices.

[10]IW: https://kernel.org/pub/software/network/iw/

5 Conclusion

We have described in this deliverable the prototype implementations of different defense mechanisms, a total of six for infrastructure and three mechanisms for endpoints.

The proposed infrastructure level mechanisms have focused on the mitigation of DDoS attacks in the routing equipment; and on improving the resilience cloud infrastructures both in terms of detection and mitigation of attacks targeting a cloud environment, as well as attacks remaining inside the cloud. At the endpoint level we have covered phishing at the browser level; and smartphone firewalling and protection against malware.

Even if our experimentations are to be extendend, the initial results seem promising and as importantly, the mechanisms have been designed in a way that allows 1) them to adapt dynamically to the current threat situation using internal detection mechanisms and external threat information such as produced by workpackage 2, 2) the automation of countermeasure application, which in turn allows us to move toward more timely reactions and limiting the risk of errors resulting from manual application, in line with our initial objectives.

Some of the mechanisms build on tried and mature, largely deployed technologies such as MPLS; some on more recent technologies, still in active development, such as SDN; or even requiring further advances in technology in order to be real world deployable like EyeBit awaiting the eye tracking to become feasible for example in the cameras embedded in smartphones or in computer screens. This opens up different avenues for going forward between immediate deployements for real world experiments and preparing for future.

There have also been changes and deviations from the designs presented in deliverable D3.4: the actual development and experimentations have revealed issues that were not seen at the design phase and requiring adaptations, but this seems quite normal for any research work.

Currently these prototypes make use of threat information and analysis provided by workpackage 1 and workpackage 2 to different degrees, the challenge of building a complete pipeline from data collection, through data analysis to putting that analysis in action in countermeasure application remains to be addressed in the workpackage 4.

Bibliography

[1] D. Anstee, D. Bussiere, and G. Sockrider. Worldwide Infrastructure Security Report 2012 Volume VIII. Technical report, Jan. 2013.

[2] Anti-Phishing Working Group. Phishing Activity Trends Report - Q1, 2014. Available at: http://docs.apwg.org/reports/apwg_trends_report_q1_2014.pdf, Aug. 2014.

[3] Arbor Networks. Peakflow. http://www.arbornetworks.com/products/peakflow.

[4] H. Asai. Virtal monitoring of hypervisor by snmp. Available at: https://github.com/drpnd/virtsnmp, 2013.

[5] H. Asai, M. MacFaden, J. Schoenwaelder, K. Shima, and T. Tsou. Management Information Base for Virtual Machines Controlled by a Hypervisor. RFC 7666 (Proposed Standard), Oct. 2015.

[6] F. Baker and P. Savola. Ingress Filtering for Multihomed Networks. RFC 3704 (Best Current Practice), Mar. 2004.

[7] R. Bifulco and G. Karame. Towards a richer set of services in software-defined networks. In *Proceedings of the NDSS Workshop on Security of Emerging Technologies (SENT)*, 2014.

[8] R. Braga, Braga, E. Mota, Mota, and A. Passito, Passito. Lightweight ddos flooding attack detection using nox/openflow. In *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*, LCN '10, pages 408–415, Washington, DC, USA, 2010. IEEE Computer Society.

[9] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. Tapas: A tool for rapid prototyping of adaptive streaming algorithms. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, VideoNext '14, pages 1–6, New York, NY, USA, 2014. ACM.

[10] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. *SIGCOMM Comput. Commun. Rev.*, 41(4):362–373, Aug. 2011.

[11] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), Jan. 2013.

[12] F. L. Faucheur. Protocol Extensions for Support of Diffserv-aware MPLS Traffic Engineering. RFC 4124 (Proposed Standard), June 2005.

[13] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android Permissions: User Attention, Comprehension, and Behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, pages 3:1–3:14, July 2012.

[14] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May. 2000. Updated by RFC 3704.

[15] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda. MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking. In *ACM CoNEXT '10*, Philadelphia, PA, December 2010.

[16] V. Fuller and D. Farinacci. Locator/ID Separation Protocol (LISP) Map-Server Interface. RFC 6833 (Experimental), Jan. 2013.

[17] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. *Computer Networks*, 62(0):122 – 136, 2014.

[18] N. Hachem, H. Debar, and J. Garcia-Alfaro. HADEGA: A novel MPLS-based mitigation solution to handle network attacks. In *31st IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 171–180, Dec 2012.

[19] HP. Hp 3500 and 3500yl switch. http://h17007.www1.hp.com/us/en/networking/products/switches/HP_3500_and_3500_yl_Switch_Series/index.aspx.

[20] HP. Hp 3800 switch. http://h17007.www1.hp.com/us/en/networking/products/switches/HP_3800_Switch_Series/index.aspx.

[21] M. Kato, K. Cho, M. Honda, and H. Tokuda. Monitoring the dynamics of network traffic by recursive multi-dimensional aggregation. In *Presented as part of the 2012 Workshop on Managing Systems Automatically and Dynamically*. USENIX, 2012.

[22] M. Kato, K. Cho, M. Honda, and H. Tokuda. Monitoring the Dynamics of Network Traffic by Recursive Multi-dimensional Aggregation. In *OSDI2012 MAD Workshop*, Oct. 2012.

[23] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13. USENIX, 2013.

[24] H. Kim and N. Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[25] C. Krügel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, SAC '02, pages 201–208, New York, NY, USA, 2002. ACM.

[26] W. Kumari and D. McPherson. Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF). RFC 5635 (Informational), Aug. 2009.

[27] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc*, SIGCOMM '05, pages 217–228. ACM, 2005.

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[29] S. Mehdi, J. Khalid, and S. Khayam. Revisiting traffic anomaly detection using software defined networking. In R. Sommer, D. Balzarotti, and G. Maier, editors, *Recent Advances in Intrusion Detection*, volume 6961, pages 161–180. Springer Berlin Heidelberg, 2011.

[30] J. Mirkovic and P. Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, Apr. 2004.

[31] D. Miyamoto, T. Iimura, G. Blanc, H. Tazaki, and Y. Kadobayashi. EyeBit: Eye-Tracking Approach for Enforcing Phishing Prevention Habits. In *Proceedings of the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, Sep. 2014.

[32] NECOMA Consortium. Deliverable D2.1: Threat Analysis. Technical report, November 2014. (confidential).

[33] NECOMA Consortium. Deliverable D3.4: Countermeasure Application - Design. Technical report, November 2014. (confidential).

[34] NECOMA Project. Dns ddos defence and countermeasure. Available at: https://github.com/upa/d4c, 2015.

[35] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Comput. Surv.*, 39(1), Apr. 2007.

[36] Pica8. Data Sheet: Pica8 P-3290. http://www.pica8.com/documents/pica8-datasheet-48x1gbe-p3290-p3295.pdf.

[37] L. Rizzo. netmap: A novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, Bellevue, WA, Aug. 2012. USENIX Association.

[38] RSA Online Fraud Resource Center. The Current State of Cybercrime 2014. Available at: http://www.emc.com/collateral/white-paper/rsa-cyber-crime-report-0414.pdf, 2014.

[39] K. Scarfone and P. Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). *NIST SP800-94*, 2007.

[40] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. In *Proceedings of the Network and Distributed System Security Symposium*, Feb. 2011.

[41] A.-D. Schmidt, H.-G. Schmidt, L. Batyuk, J. H. Clausen, S. A. Camtepe, S. Albayrak, and C. Yildizli. Smartphone malware evolution revisited: Android next target? In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pages 1–7, 2009.

[42] Y. Sekiya. Software technologies of composing iaas clouds –wide cloud as a case study–. *Computer Software*, 29(2):2–15, 2012.

[43] 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 *Proc. of NDSS*, 2013.

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

[45] J. Tan, K. Nguyen, M. Theodorides, H. Negrón-Arroyo, C. Thompson, S. Egelman, and D. Wagner. The Effect of Developer-specified Explanations for Permission Requests on Smartphone User Behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 91–100, Apr. 2014.

[46] D. Turk. Configuring BGP to Block Denial-of-Service Attacks. RFC 3882 (Informational), Sep. 2004.

[47] Y. Ueno, K. Horiba, and K. Kataoka. Design and Implementation of Software LISP Router. *Internet Conference 2011 (IC2011) - Work in Progress*, Oct 2011.

[48] J. Vincent, C. Porquet, M. Borsali, and H. Leboulanger. Privacy Protection for Smartphones: An Ontology-Based Firewall. In *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings*, pages 371–380, 2011.

[49] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang. Enabling security functions with SDN: A feasibility study. *Computer Networks*, 85:19–35, 2015.

[50] S. T. Zargar, J. Joshi, and D. Tipper. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069, 2013.