OrchSec: An Orchestrator-Based Architecture For Enhancing Network-Security Using Network Monitoring And SDN Control Functions

Conference Paper · May 2014

DOI: 10.1109/NOMS.2014.6838409

CITATIONS
106

READS
2,688

4 authors, including:

Adel Zaalouk
Red Hat
10 PUBLICATIONS **235** CITATIONS

SEE PROFILE

Rahamatullah Khondoker
Fraunhofer Institute for Secure Information Technology SIT
63 PUBLICATIONS **497** CITATIONS

SEE PROFILE

OrchSec: An Orchestrator-Based Architecture For Enhancing Network-Security Using Network Monitoring And SDN Control Functions

Adel Zaalouk
RWTH Aachen University
Aachen, Germany
Email: adel.zaalouk@rwth-aachen.de

Rahamatullah Khondoker, Ronald Marx, Kpatcha Bayarou
Fraunhofer Institute for Secure Information Technology (SIT)
Rheinstr. 75, Darmstadt, Germany
Email: firstname.lastname@sit.fraunhofer.de

***Abstract*—The original design of the Internet did not take network security aspects into consideration, instead it aimed to facilitate the process of information exchange between end-hosts. Consequently, many protocols that are part of the Internet infrastructure expose a set of vulnerabilities that can be exploited by attackers. To reduce these vulnerabilities, several security approaches were introduced as a form of add-ons to the existing Internet architecture. However, these approaches have their drawbacks (e.g., lack of centralized control, and automation). In this paper, to address these drawbacks, the features provided by *Software Defined Networking* (SDN) such as network-visibility, centralized management and control are considered for developing security applications. Although the SDN architecture provides features that can aid in the process of network security, it has some deficiencies when it comes to using SDN for security. To address these deficiencies, several architectural requirements are derived to adapt the SDN architecture for security use cases. For this purpose, OrchSec, an Orchestrator-based architecture that utilizes *Network Monitoring* and *SDN Control* functions to develop security applications is proposed. The functionality of the proposed architecture is demonstrated, tested, and validated using a security application.**

I. Introduction

The Internet was not designed with security in mind, it was rather designed to provide connectivity between end-hosts, and to facilitate the process of information sharing. For providing services to end-hosts, the Internet follows a *client-server* model. In this model, the *server* is responsible for answering service-requests sent by one or more *clients*. This design model was later utilized for executing threats such as *Denial of Service* (DoS) and *Distributed Denial of Service* (DDoS) which aim to disrupt the functionality of the serving-entity.

DoS and DDoS are attacks intended to prevent legitimate users from accessing network resources. The main difference between a DoS and DDoS is that in the former, the attack originates from one source, alternatively, the latter requires multiple attack sources (One-to-One Vs. Many-to-One). DoS/DDoS attacks can be executed in different forms. For example, *Domain Name System* (DNS) Amplification is a form of a DDoS attack through which the DNS components themselves are exploited in an attempt to magnify the flooding attack consequence. This can be done by making use of the fact that small DNS queries can generate large responses [1].

978-1-4799-0913-1/14/$31.00

For mitigating against Internet attacks such as DNS Amplification, several security approaches were introduced as a form of add-ons to the existing Internet architecture.However, these approaches have their drawbacks. To understand these drawbacks, security approaches against DNS Amplification are considered as a showcase. Traditional security approaches against DNS amplification rely on attack prevention mechanisms (e.g., policies for not leaving DNS resolvers open and publicly accessible) rather than reactive detection and mitigation. The reason for this limitation is that there are no mechanisms to obtain the state of the network (i.e., lack of network-visibility). Additionally, there are no automated mechanisms for steering malicious traffic away from the network once an attack is detected [2].

SDN provides features that can be used to address these problems. SDN decouples the control-plane from the data-plane of network devices, and migrates the network intelligence (e.g., network applications such as routing, QoS, etc) as well as the state of the underlying network to a centralized controller. Moreover, SDN provides an abstraction layer that allows the controller to communicate with the underlying network (e.g., using OpenFlow [3]). Through this abstraction layer, the controller commands the network to steer / drop traffic in an automated manner. Additionally, applications can be developed on-top of the controller, which reduces the time needed for testing and deployment (i.e., extensible architecture).

Few research was done regarding the use of SDN to improve or enable security in traditional networks. For example, in [4][5][6] the authors develop security applications within SDN controllers for mitigating against attacks such as Scanning and DoS. Furthermore, in [7][8] the authors develop frameworks for writing security applications. However, the following problems can be noticed in these works:

- Relying only on SDN capabilities without making use of other security-assisting technologies (e.g., Network Monitoring) which could reduce the overhead on SDN controllers.
- The security applications are tightly coupled to the SDN controller (i.e., needs to be re-implemented if another controller was used). Therefore, they lack

flexibility.

- Using only a single SDN controller which the increases chance of network downtime because the controller will act as a single point-of-failure.

Therefore, before starting to develop security applications on-top of SDN, the SDN architecture needs to be extended to address architectural limitations. In Section II, to address the drawbacks of SDN, several architectural requirements are derived. To fulfill these requirements, four design iterations towards extending the existing SDN architecture are attempted. Each design iteration has some pros and cons. Ultimately, an architecture was proposed to fulfill the requirements and to address the drawbacks described in the SDN architecture.

Furthermore, in Section III, to address the problems of DNS Amplification security approaches, a security application was developed on-top of the proposed architecture. Moreover, in Section IV, an experiment is conducted to test and validate the behavior of the prototype security application.

Finally, in Section V, the paper is concluded and further future modifications, extensions and enhancements to the proposed architecture are explained

II. Architectural Design

In this section, an architecture for developing security applications using SDN and network monitoring is proposed.

A. Architectural Requirements

To develop an architecture capable of addressing the SDN architecture problems mentioned in Section I, several architectural requirements are derived in this section.

1) Requirements For A Reliable SDN Architecture: In [9], the authors reveal vulnerabilities in the SDN architecture. Based on these vulnerabilities, the authors derived requirements for a reliable and dependable SDN architecture. Some of these requirements are:

- **Replication of Controllers:** Having multiple instances of the controller can increase reliability and fault tolerance. This can be achieved by failing-over (i.e., migrating functionalities) to different controller instances when a problem occurs.
- **Using Diverse Controllers:** To improve the robustness of the architecture, replication with diverse controllers is encouraged. The main advantage here is to avoid common-mode faults (i.e., faults caused by the same component). For example, if a controller instance has a bug or a vulnerability, then running an instance of another SDN controller can help preventing system failures due to the bugs introduced by the first controller instance.

2) Requirements For Decoupling Control And Monitoring Functions: SDN decouples the control-plane from the data-plane. However, it does not decouple the control functionalities from the monitoring functionalities. For example, an OpenFlow would be responsible for communicating control messages to the switches and monitoring traffic on the network at the same time. This would increase the overhead on the controller, and thus reduce the performance. To solve this problem, it is required to decouple the control and the monitoring functionalities, to reduce the overhead on the SDN controller. For example, by making use of sampling-based network monitoring technologies to signal a first step of an attack, it becomes un-necessary to forward traffic all the time to the controllers. Traffic should be forwarded only when there is an indication of an attack signaled by the network monitor.

3) Requirements For a Flexible SDN Architecture: A flexible SDN architecture should have loosely-coupled components that do not rely on each other's functionality. For example, when developing an application, the application should be able to function regardless of the type of the underlying SDN controller being used. In the context of SDN, these type of applications are called *Northbound* applications. Consequently, another requirement for the proposed architecture is to develop applications on a Northbound API to ensure flexibility.

4) Requirements For High Resolution Attack Detection: The capability of a security application is dependent on the amount of information given to it as an input. For example, OpenFlow provides flow-level information (e.g., Packets per flow), however, it does not provide packet level information (e.g., Packet Length). A high-resolution attack detection approach provides as much information about network traffic as possible. Therefore, to increase the detection capability of a security application, the SDN architecture should provide means for accessing packet-level information to increase the resolution of attack detection.

B. Proposed Architecture

Based on the requirements stated in the previous section, several design iterations towards addressing these requirements have been attempted. These iterations are described in the next sections.

1) First Iteration: Sampling Based Security: The architecture for the first design iteration is shown in Figure 1. As shown in the left side of the figure, the architecture consists of the following main components:

- **Virtual / Physical Switches**: Devices that communicate using a southbound protocol (e.g., OpenFlow [3]) with the SDN controller.
- **Network Monitor**: A component that can inspect network traffic, apply traffic filters and trigger events in case a certain pattern of traffic occurred.
- **Controller(s)**: Components that can communicate with the switches using a southbound protocol such as OpenFlow. In the proposed architecture, it is recommended to use more than one controller for diversity and replication.

Mechanism: On the right side of Figure 1, the interaction between the components is shown by arrows. Basically, detection applications are developed on top of the network monitor. Using a REST API [10] , applications can periodically query for traffic seen by the network monitor (1). Once applications detect a suspicious activity, an action is immediately taken (e.g., using OpenFlow to install dropping rules on the switch) to separate the source of this activity from the rest of the

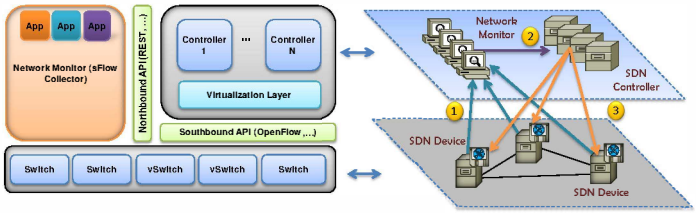


Fig. 1: Sampling Based Security Architecture

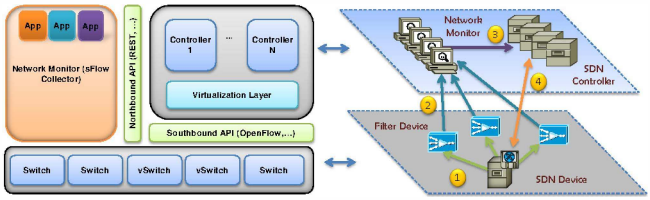


Fig. 2: High Resolution Sampling Architecture

network (2) (3). In this architecture, applications are developed on-top of the network monitor.

Pros: Applications in this approach are developed on the northbound (i.e., loosely-coupled from the controllers) and multiple controllers are used to ensure diversity and replication.

Cons: The main problem in this approach is that the technology used in the network monitor for attack detection relies on *packet sampling* [11]. Sampling can be a reasonable alternative for detecting events such as elephant flows (i.e., a flow with a large number of packets) or for detecting attack patterns without having access to all the packets sent. However, when it comes to detecting attacks that require inspecting all the packets of a certain type, sampling will not be sufficient due to the following limitations [12]:

- **Flow-Shortening**: Only a small fraction of the flow packets are observed.
- **Flow-Reduction**: Not all the flows are observed.

According to [12], the flow-shortening reduction problem can be minimized by increasing the sampling rate. However, there will be a sampling bias which will result in false-positives and mis-leading alarms.

2) Second Iteration: High Resolution Sampling: According to [13], the sampling resolution can be improved by grouping traffic with similar characteristics (e.g., same protocol type) together and then sampling the traffic of individual groups. Therefore, in an attempt to improve the sampling resolution, and to reduce the number of false-positives resulted from the flow-shortening problem described in the previous iteration, the second iteration architecture was designed. As can be seen in Figure 2, the architecture is not very different from the previous iteration except for an additional component, which is the filter device.

- **Filter Device:** A component that collects traffic of some type (e.g., UDP traffic) and sample the traffic back to the network monitor.

Mechanism: In the first design iteration, traffic was sampled from a pool of packets having different types, which led to decreased sampling resolution and diminished the capability of detecting attacks. In this iteration, OpenFlow is used to filter the desired traffic from network switches (i.e., filtering traffic into similar groups) and forwards it to the filter devices (1). The filter devices sample the filtered traffic (e.g., ARP traffic) and sends it to the network monitor (2). Finally, the network

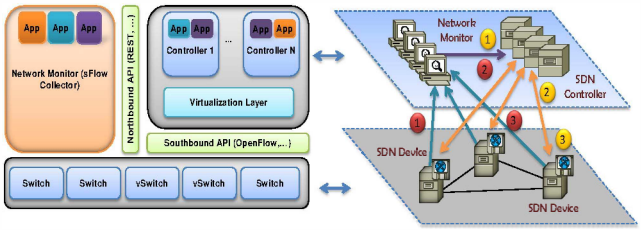


Fig. 3: Delegating Attack Detection Architecture

monitor runs the corresponding security application, and uses the Northbound API of the SDN controller(s) to separate the source of the malicious traffic from the rest of the network when suspicious behavior is detected (3) (4).

Pros: The budget of sampling (i.e., the amount of traffic available for sampling) for the desired traffic is much higher, this is because undesired traffic is filtered before the sampling process, unlike the first iteration where sampling was done without filtering of desired traffic.

Cons: Although this design increased the sampling efficiency compared to the previous design, the problem of Flow-shortening was not completely solved. This is a trade off from using sampling (to decrease the overhead, traffic has to be reduced).

This architecture can complement other architectures whenever sampling is required.

3) Third Iteration: Delegating Attack Detection: The second design iteration can help minimizing the problem of flow-shortening, yet it does not prevent it because of the limitation posed by sampling in general. Depending on the type of attack to be detected, it should be decided whether sampling is the right alternative or not. In some cases, attack detection requires having a copy of all the packet on the network, in this case, sampling will not be the right alternative. To solve this problem, whenever a high-resolution attack (i.e., an attack that requires access to all packets) needs to be detected, the logic for detecting this attack is delegated to the OpenFlow controller instead of the network monitor.

Mechanism: As shown in Figure 3, a typical scenario would be as follows; The first step is to configure which attacks should be detected. Consequently, the type of attacks configured are checked for feasibility (i.e., deduce whether the attack is a high or a low resolution attack) by the network

monitor. Whenever an attack is marked as infeasible, that steps marked with the red markers on the right side of the figure are followed. Alternatively, the steps marked with yellow markers are executed. In case an attack is marked as infeasible, the network monitor contacts the controller to for delegating the attack detection tasks (1). The controller turns on the correct detection module accordingly, and starts to inspect high-resolution traffic (2). If an attack is detected by the controller detection module, an action is immediately taken by separating the source of the attack traffic from the rest of the network (3). Moreover, if the type of the attack can be handled by the network monitor, the same procedure described in the first iteration is followed.

Pros: High-resolution attack detection is delegated to the SDN controllers, therefore, sampling is not used in this particular case, and attack detection becomes more efficient than the previous two iteration.

Cons: High resolution detection applications are tightly coupled to the controllers. This reduces flexibility in application development, which is against the requirements listed in Section II-A.

4) Final Iteration: OrchSec Architecture: Although the third iteration prevented the limitation of sampling (i.e., flow-shortening) by delegating detection logic to SDN controllers, application development flexibility was reduced. Therefore, to keep the system flexible and increase the capability of the architecture to detect different types of attacks, an additional component is needed to orchestrate the use of network monitoring and SDN technologies according to the attack type being handled, this extra component is added to the proposed *OrchSec* architecture.

The design of the proposed architecture is shown in Figure 4. On the left side of the figure, the main components of the architecture are shown. On the right side, the figure shows the interaction between these components. Most of the components are similar to the components of the first three iterations. However, two components have been added:

- **Orchestrator**: The Orchestrator constitutes the most important component in the architecture where all of the security applications are developed. In the Orchestrator, applications for detecting attacks of different types can be switched on or off on-demand. Furthermore, the Orchestrator communicates with the network monitor for updates about important events and network status, from the knowledge obtained from the network monitor, the Orchestrator commands the controller to observe certain traffic patterns and redirect traffic belonging to these patterns back to the Orchestrator for further inspection (i.e., co-ordinating the functionalities of the network monitor and the SDN Controllers). Therefore, the Orchestrator is considered to be the brain of the whole architecture.
- **The Orchestrator Agent**: The Orchestrator Agent is an application installed on each SDN controller that enables the controller to communicate with the Orchestrator.

Mechanism: There are two types of attacks that can be handled by this architecture:

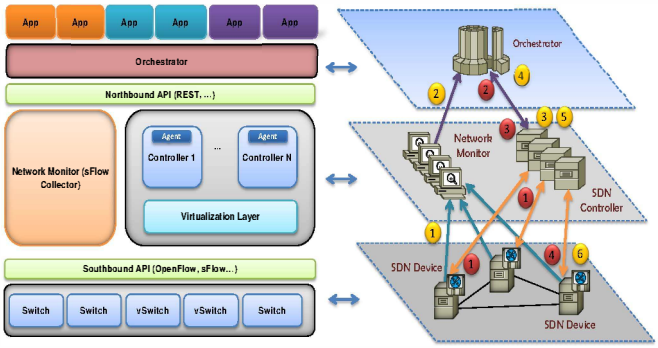


Fig. 4: *OrchSec* Architecture

- **High-Resolution Attacks**: Attacks that can not be detected without having access to all the packets in the network. Detecting these attacks using sampling will not be sufficient, therefore, packets belonging to this traffic flow should be forwarded for a pre-defined time window to the Orchestrator for further inspection (e.g., ARP Cache Poisoning, ARP Spoofing).
- **Low-Resolution Attacks**: Attacks that does not necessarily require access to all the packets. To reduce the overhead, sampling can be used to detect large-flow events, and thereupon informing the Orchestrator to take further actions (e.g., DoS, DDoS, DNS Amplification attacks).

Figure 4 shows the architecture components interaction steps that are followed for detecting high-resolution attacks (red marked numbers) and low-resolution attacks (yellow marked numbers).

For detecting low-resolution attacks the following steps are performed; The network monitor first checks for events happening in the network (1), once a suspicious event is detected, the Orchestrator is immediately informed (2). Next, the Orchestrator asks the controller to match traffic of a specific pattern for a pre-defined time interval (to avoid overloading the controller with attack traffic and to keep the overhead minimal) (3), and then redirects traffic header-fields back to the Orchestrator (4). Accordingly, the detection logic (i.e., the security application) running on-top of the Orchestrator will detect if there is an attack. Finally, if an attack is detected, an action (e.g., dropping traffic) is immediately taken using the API provided by one of the controllers (5) (6).

For detecting high-resolution attacks, the following steps are followed; The Orchestrator commands the controller to redirect packet headers of traffic belonging to a certain type (e.g., ARP traffic) back to it (1). The Orchestrator then forwards the packet headers to the corresponding detection logic module for inspection (2). Once an attack is detected, the Orchestrator asks one of the controllers to block attacker traffic (3). Finally, the controller installs flow-rules that blocks attacker's traffic on the switches using OpenFlow (4).

The Orchestrator is a multi-threaded server that receives update messages from the network monitor and the SDN controller(s). After processing these messages (i.e., looking

at the headers assigned to each packet), the Orchestrator decides whether more packets are needed and thus decides to establish a communication channel with one of the controllers to issue commands to the Orchestrator-agents. Additionally, the Orchestrator can run more than one security application at the same time. Consequently, when a packet is received, a number of copies equal to the number of enabled security applications is forwarded to each application for orthogonal attack inspection (i.e., each application looks for a different attack pattern).

Pros:

- Replication and diversity of controllers to ensure reliability and resilience.
- Using a network monitor for detecting the initial phases of low-resolution attacks using sampling, can reduce the overhead of forwarding large-flows to the SDN controllers even when it is not needed.
- Flow-shortening is not a problem for high-resolution attacks any more because traffic is forwarded to the Orchestrator
- To allow for increased flexibility in application development, all the applications developed on-top of the Orchestrator are loosely-coupled northbound applications

Cons: To keep the overhead minimal when handling high-resolution attacks, it is required to forward traffic types that constitutes a small portion of the overall traffic (e.g., ARP traffic). Forwarding other types of traffic without a time-limit to the Orchestrator can increase the overhead of attack detection. Another alternative is to fall-back to the second design iteration where traffic is sampled with high resolution, but again this would decrease the efficiency and introduce false positives.

C. Fulfilling Architectural Requirements

The proposed architecture fulfills the requirements derived in Section II-A as follows:

- The architecture makes use of multiple diverse (i.e., different controller types) controller instances, this would ensure reliability in case of controller failure.
- By using a network monitor, access to packet-level information would be enabled, and more information would be provided as an input to the detection logic.
- In the Orchestrator-based architecture, applications are not implemented inside the controllers. Instead, the applications are developed as Northbound applications which are not tightly coupled to the controller. This would ensure flexibility when developing applications.
- By using a network monitor, the control and the monitoring functionalities are decoupled. The controller(s) will only be responsible for issuing control messages (e.g., installing flow-rules on the switches) and the network monitor would be responsible for doing the monitoring functions (e.g., monitoring the performance of the network, etc.).

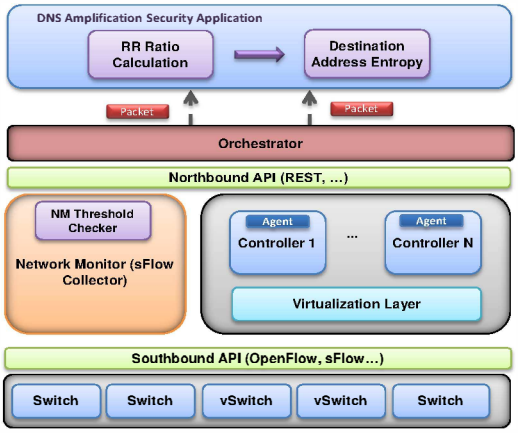


Fig. 5: Proposed DNS Amplification Application

III. OrchSec Security

To show how security applications can be developed on-top of the proposed Orchestrator-based architecture, several security requirements are derived for mitigating against DNS Amplification attacks. Furthermore, an application is proposed and described to fulfill these requirements using *OrchSec*. More security applications (i.e., ARP security and DoS) that use the *OrchSec* can be found in [14].

A. Security Requirements

DNS amplification attacks are difficult to address. This is because the attacker usually hides behind other network devices such as open (publicly accessible) DNS resolvers or compromised hosts. Although the impact of DNS amplification attacks might go un-noticed, the combined effect can cause serious damage to the target network [15].

Traditional security approaches against DNS amplification rely on attack prevention mechanisms (e.g., policies for not leaving DNS resolvers open and publicly accessible) [2]. However, hosting service providers have limited or no control on their physical and virtual servers. As a result, customers with low awareness of reflection attacks (e.g., DNS Amplification) leave their DNS resolvers open and publicly accessible. This facilitates the process of generating a DNS amplification attack [15].

To address the problem of defending against DNS Amplification, the following requirements are derived:

- Monitoring of multiple network devices at the same time for high-visibility.
- Access to DNS packet-level information (e.g., DNS Opcode, packet-length, etc.) to be able to differentiate malicious traffic from valid traffic.
- The ability to selectively drop or steer malicious traffic in an automated behavior.

B. DNS Amplification: Prototype Design

SDN Related Work: In [15], Phaal describes how a network monitoring technology such as sFlow [16] can be

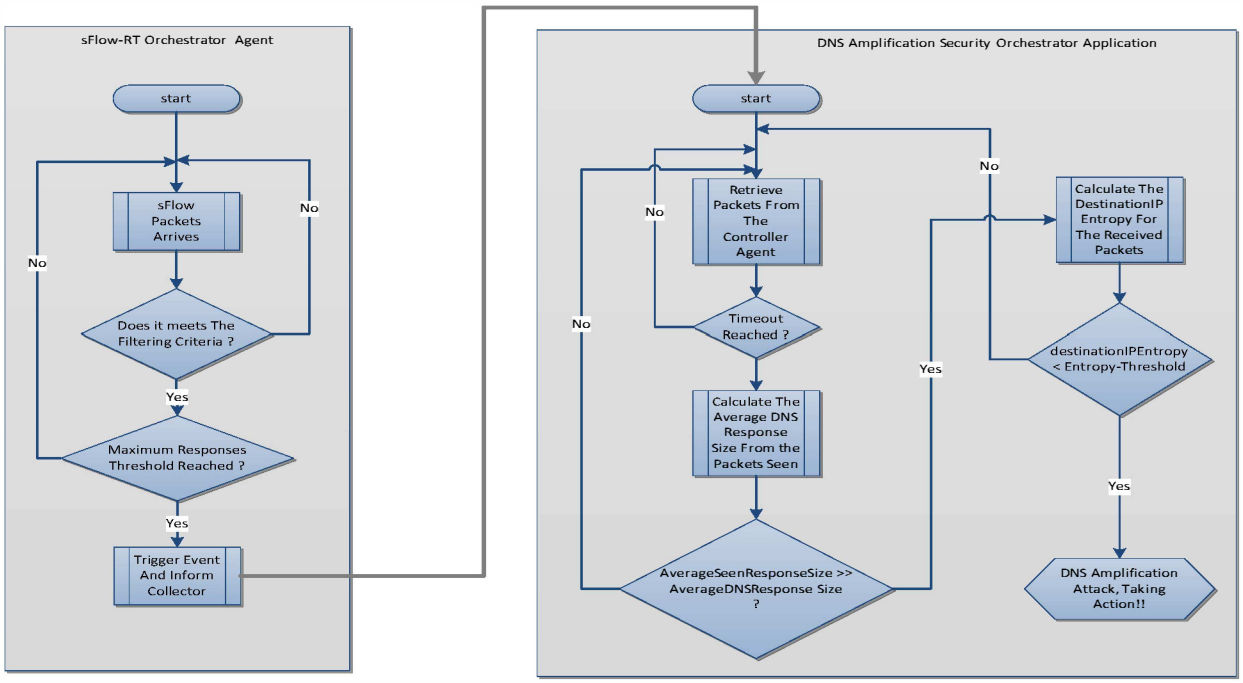


Fig. 6: DNS Amplification Prototype Flowchart

used to monitor DNS traffic and how DNS attributes can be extracted from sFlow packets. Phaal also stated that combining the capabilities of sFlow and SDN can increase the efficacy of defending against DNS amplification attacks. However, using sFlow DNS attributes alone does not suffice for accurately detecting DNS Amplification because of the sampling limitation mentioned in Section II-B1. Alternatively, sFlow can be used to signal suspicious DNS traffic behavior (e.g., high number of DNS request), then, using OpenFlow, traffic can be steered for further inspection.

Non-SDN Related: Only a few research have been done towards handling DNS Amplification attacks. In [17], the authors built a low-cost hardware solution to defend against amplification attacks, the approach works well except that it is hardware-based, which makes it hard to update and extend. In [2], Kambourakis et al. propose a solution for the DNS amplification attack, which works by storing all the incoming DNS requests and replies. Whenever an un-matched reply is received, a danger counter is incremented until it reaches a danger-threshold. When the threshold is reached, an attack alert is generated and a DNS amplification attack is assumed to be happening. The problem with this approach is that it does not scale for large networks because it needs to store all the DNS traffic queries and responses.

The prototype application design encompasses the following main building blocks as shown in Figure 5:

- **Network Monitor Threshold Checker**: This component resides at the Network Monitor and it is responsible for monitoring DNS responses. When the number of DNS responses exceeds a pre-defined response-count threshold, the Orchestrator is immediately notified.
- **Reply-Ratio (RR) Calculation And Threshold Comparison**: The Orchestrator commands one of the SDN controllers to forward suspicious traffic to it. When the Orchestrator starts receiving DNS response packets, it forwards these packets to this module. In this module, the size of the DNS response packets is observed. Furthermore, the average DNS packet response size is calculated. If the average response size for all the received packets exceeds the value of the response-size threshold (i.e., the ratio (RR) between the average response size and response-size threshold), then the application transients to the first danger state.
- **Destination Address Entropy Calculation And Threshold Comparison**: If the RR Ratio was high, then this component calculates the destination IP address entropy. If the destination IP address entropy value is low, then it is assumed that there is a DNS amplification attack, and an action is taken accordingly.

C. DNS Amplification: Prototype Implementation

In the previous section, the incorporation of the prototype application in the proposed architecture was described. In this section, the DNS Amplification detection algorithm is explained.

The detection algorithm makes use of the *Entropy* concept. *Entropy* is a statistical method to quantify information (i.e.,

a method to measure observational variety contained in the data) [18]. Using *Entropy*, it can be identified whether the information received varies or not. The *Entropy* is used in the detection algorithm to calculate the following:

- **Destination IP Entropy**: Determines the variation of destination IP addresses in the network. For example, if only one IP is used (i.e. targeted IP), then the entropy value will be low, and high otherwise.
- **Source And Destination Transport layer Port Entropy**: Determines the variation of destination or source transport layer ports used. For example, if only HTTP (e.g., Port 80) is used then the entropy value will be low and high otherwise.

The *DNS Amplification* security application works by configuring the sFlow Orchestrator-Agent to monitor DNS responses. Once the amount of responses monitored by sFlow exceeds a maximum-responses count, the Orchestrator is informed. Respectively, the Orchestrator asks the Orchestrator-agent inside the controller to redirect traffic for a specific time-window back to it for further inspection. Once the Orchestrator starts to receive traffic from the controller's Orchestrator-agent, it executes the following steps as shown in Figure 6:

- Packets are collected at the Orchestrator for a pre-defined time window, or until a timeout value expires.
- When the timeout value is reached, the average DNS response size is calculated for all the collected packets.
- If the average size calculated was greater than the average standard size of a DNS response packet, then the algorithm moves to the next step. Otherwise, it is assumed that there is no attack.
- If the destination IP entropy value is less than the entropy-threshold, then a DNS amplification attack is detected and further action is taken. Otherwise, no action is taken, as the threat level is not considered high enough.

Once a *DNS Amplification* attack is detected, attacker traffic is rate-limited using OpenFlow.

IV. Experimental Examination

To show the behavior of the DNS Amplification security application which is developed on-top of the proposed architecture, an experiment is conducted in this Section.

A. Testing Enviroment

The machine used for this experiment runs *Ubuntu 12.04 LTS*, with an Intel Core i7-3630QM, a 2.40GHz x 8 CPU, and 8GiB of RAM.

For generating network topologies, Mininet [19], a network emulator that uses process-based virtualization was installed on a *VirtualBox* [20] VM, the VM is connected to the Internet through *Network Address Translation* (NAT) (for software installation and updates), and a host-only adapter is configured on the VM to enable it to communicate with the host-system. Additionally, SSH was used to allow access to the VM for running different software at the same time. On the other hand, the Network Monitor (i.e., sFlow-RT [16]) and the SDN controllers (i.e., Floodlight [21], and POX [22]) were installed on the host-system.

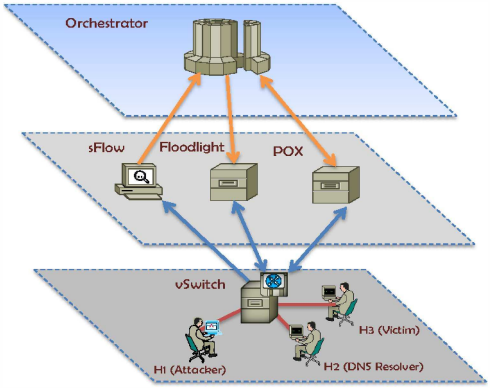


Fig. 7: DNS Amplification Experiment Setup

B. Experiment Setup

As shown in Figure 7, the DNS Amplification experiment setup consists of two OpenFlow controllers (i.e., Floodlight, and POX), an sFlow network monitor, an Orchestrator, a switch, and three hosts. The sFlow network monitor is used for detecting large DNS flows (flows with large amounts of traffic), the Orchestrator will communicate with both sFlow and the OpenFlow controllers for detecting and mitigating DNS amplification. The hosts are divided into attackers and victims as will be described in the *Attack Scenario*.

C. Attack Scenario

To generate the DNS amplification attack, two main steps should be considered: Attackers should spoof the source IP address of the victim, and Open DNS resolvers should be simulated, and configured to send large DNS responses to the victim.

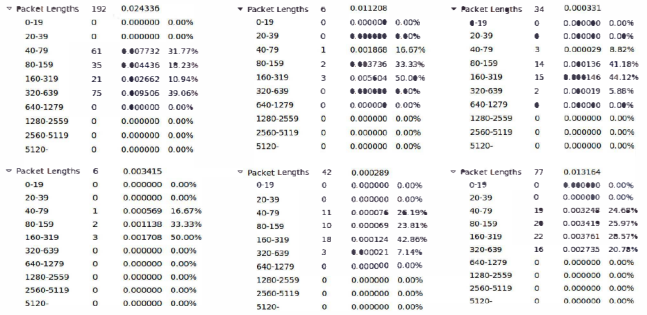
The setup for this experiment consists of three hosts as shown in Figure 7. H1 is the main attacker (there can be more than one attacker, however, for simplicity, only one attacker is considered), H2 will act as the open DNS resolver, and H3 will be the victim.

Two python scripts were written for this attack, the *DNSAmplificationAttack* and the *DNSServer*. For the attack script, Scapy's Python library [23] was used to craft DNS queries in large amounts, and for the DNS server script, the DNSlib [24] was used and the server was configured to send large *TXT* DNS records as query responses to the victim (H3).

D. Attack Mitigation

To mitigate against the *DNS Amplification* attack, the Orchestrator is configured to run the *DNS Amplification* security application. Additionally, the *AverageDNSResponseSize* must be defined in the security application.

One of the main characteristics of a DNS Amplification attack is generating large-sized DNS responses, therefore, to estimate the maximum DNS response size, DNS queries to six of the top-most visited web sites (as stated in [25]) were



Packet Lengths	192	0.024336	
0-19	0	0.000000	0.00%
20-39	0	0.000000	0.00%
40-79	61	0.007732	31.77%
80-159	35	0.004436	18.23%
160-319	21	0.002662	10.94%
320-639	75	0.009506	39.06%
640-1279	0	0.000000	0.00%
1280-2559	0	0.000000	0.00%
2560-5119	0	0.000000	0.00%
5120-	0	0.000000	0.00%

Packet Lengths	6	0.011208	
0-19	0	0.000000	0.00%
20-39	0	0.000000	0.00%
40-79	1	0.001868	16.67%
80-159	2	0.003736	33.33%
160-319	3	0.005604	50.00%
320-639	0	0.000000	0.00%
640-1279	0	0.000000	0.00%
1280-2559	0	0.000000	0.00%
2560-5119	0	0.000000	0.00%
5120-	0	0.000000	0.00%

Packet Lengths	34	0.000331	
0-19	0	0.000000	0.00%
20-39	0	0.000000	0.00%
40-79	3	0.000029	8.82%
80-159	14	0.000136	41.18%
160-319	15	0.000146	44.12%
320-639	2	0.000019	5.88%
640-1279	0	0.000000	0.00%
1280-2559	0	0.000000	0.00%
2560-5119	0	0.000000	0.00%
5120-	0	0.000000	0.00%

Packet Lengths	6	0.003415	
0-19	0	0.000000	0.00%
20-39	0	0.000000	0.00%
40-79	1	0.000569	16.67%
80-159	2	0.001138	33.33%
160-319	3	0.001708	50.00%
320-639	0	0.000000	0.00%
640-1279	0	0.000000	0.00%
1280-2559	0	0.000000	0.00%
2560-5119	0	0.000000	0.00%
5120-	0	0.000000	0.00%

Packet Lengths	42	0.000289	
0-19	0	0.000000	0.00%
20-39	0	0.000000	0.00%
40-79	11	0.000076	26.19%
80-159	10	0.000069	23.81%
160-319	18	0.000124	42.86%
320-639	3	0.000021	7.14%
640-1279	0	0.000000	0.00%
1280-2559	0	0.000000	0.00%
2560-5119	0	0.000000	0.00%
5120-	0	0.000000	0.00%

Packet Lengths	77	0.013164	
0-19	0	0.000000	0.00%
20-39	0	0.000000	0.00%
40-79	19	0.003249	24.68%
80-159	20	0.003419	25.97%
160-319	22	0.003761	28.57%
320-639	16	0.002735	20.78%
640-1279	0	0.000000	0.00%
1280-2559	0	0.000000	0.00%
2560-5119	0	0.000000	0.00%
5120-	0	0.000000	0.00%

Fig. 8: DNS Response Sizes For Top-most Visited Web-sites

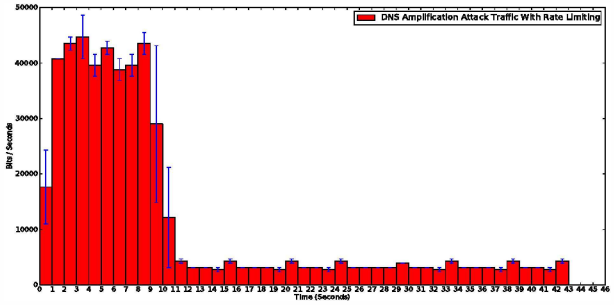


Fig. 10: DNS Amplification Experiment Result

generated and response sizes were recorded using wireshark as shown in Figure 8. From the figure, it can be noticed that the maximum DNS packet size is between 320-639 Bytes. Accordingly, it is assumed that responses with length greater than 450 Bytes are considered an indication of a DNS amplification attack. Furthermore, once a *DNS Amplification* attack is detected, the attack traffic has to be rate-limited. In this paper, OpenFlow 1.0 was used because it is widely supported by several SDN controllers [26]. However, it does not provide mechanisms for rate-limiting. Therefore, as a work-around for this limitation, the *Ping-Pong* application was developed.

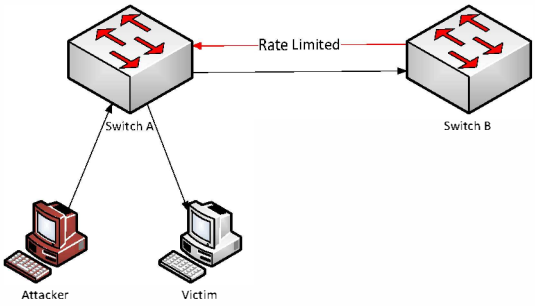


Fig. 9: Ping-Pong Rate Limiting Application

The Ping-Pong Application: Mininet allows modification of link performance parameters such as capacity and delay. This feature can be used by the *Ping-Pong* application to rate-limit traffic as shown in Figure 9. In the figure, an additional switch is added (i.e., *Switch A*) to the original topology. One or both of the links between *Switch A* and *Switch B* can be configured using Mininet to have a lower capacity than it usually has (i.e., rate-limiting the link). When an attack is detected, the rate-limiting application instructs *Switch A* to forward the traffic to *Switch B*. When *Switch B* starts receiving the attack traffic, the rate-limiting application instructs it to re-send the traffic back to *Switch A* through the rate-limited link. When *Switch A* receives the rate-limited traffic back, it will forward normally to its destination.

OpenFlow 1.0 provides *DROP* as an action for dropping traffic, however only the rate-limiting option is feasible for handling DNS Amplification attacks. The reason for this limitation is that attackers do not generate the attack themselves, instead they hide behind open (i.e., publicly-accessible) DNS resolvers, and let the DNS resolvers carry out the attack for them. Therefore, if the public DNS resolvers were blocked, the whole network can be denied of service (because the public DNS resolvers reply to local DNS requests). On the other hand, if the traffic of the open DNS resolvers is rate-limited, the attack will not affect the network and the network will continue to function normally.

E. Results

Figure 10 shows the behavior of the DNS Amplification security application. It can be seen from the figure that once the DNS attack is detected, the *Ping-Pong* application was used to rate limit the traffic at around 10 seconds. The attack response-time can be reduced by minimizing the sFlow packet-count threshold and reducing the time needed for collecting attack-traffic in the DNS security application.

V. Conclusion & Future Work

In this paper, *OrchSec*, an architecture that utilizes *Network Monitoring* and *SDN Control* functions for developing security applications was presented. *OrchSec* aims to enhance network security by reducing the overhead on *SDN Controllers* through the decoupling of control and monitoring functions. Furthermore, the architecture motivates application development flexibility by decoupling application development from the *SDN Controller*, making the process of application development controller-agnostic. Moreover, using *OrchSec*, applications for mitigating against network attacks such as *ARP Spoofing / Cache Poising*, *DoS / DDoS* were developed. To test, validate, and demonstrate the functionality of *OrchSec*, a *DNS Amplification* security application was developed. Using the application, *OrchSec* was able to detect attacker traffic by incorporating the functionalities of sFlow-RT (the network monitor) to signal initial signs of an attack to the Orchestrator which in-turns commands the SDN controller(s), based on the detection logic, to take further actions against the attacker such as rate-limiting and dropping attack traffic.

In the future, *OrchSec* can utilize the functionalities provided by other OpenFlow versions (i.e., OpenFlow 1.x) to provide a rich set of attack mitigation actions. Furthermore, other applications to cover a wide range of security services can be developed on-top of *OrchSec*.

REFERENCES

[1] S. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks." IEEE, 2013.

[2] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "A fair solution to DNS amplification attacks," in *Digital Forensics and Incident Analysis, 2007. WDFIA 2007. Second International Workshop on*. IEEE, 2007, pp. 38–47.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE, 2010, pp. 408–415.

[5] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 127–132.

[6] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Recent Advances in Intrusion Detection*. Springer, 2011, pp. 161–180.

[7] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular composable security services for software-defined networks," in *Proceedings of Network and Distributed Security Symposium*, 2013.

[8] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. IEEE, 2012, pp. 1–6.

[9] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 55–60.

[10] R. Fielding, "Representational state transfer," *Architectural Styles and the Design of Netowork-based Software Architecture*, pp. 76–85, 2000.

[11] R. E. Jurga and M. M. Hulbj, "Technical report packet sampling for network monitoring," 2007.

[12] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006, pp. 165–176.

[13] R. Kawahara, T. Mori, N. Kamiyama, S. Harada, and S. Asano, "A study on detecting network anomalies using sampled flow statistics," in *Applications and the Internet Workshops, 2007. SAINT Workshops 2007. International Symposium on*. IEEE, 2007, pp. 81–81.

[14] A. Zaalouk, "Taking In-Network Security To The Next-Level With Software Defined Networking (SDN)," 2014.

[15] P. Peter, "DNS Amplification Attacks," http://blog.sflow.com/2013/10/dns-amplification-attacks.html, 2013, [Online; accessed 01-January-2014].

[16] M. Wang, B. Li, and Z. Li, "sflow: Towards resource-efficient and agile service federation in service overlay networks," in *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*. IEEE, 2004, pp. 628–635.

[17] C. Sun, B. Liu, and L. Shi, "Efficient and low-cost hardware defense against DNS amplification attacks," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*. IEEE, 2008, pp. 1–5.

[18] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[19] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[20] Oracle, "Virtual Box," http://www.virtualbox.org/, [Online; accessed 01-January-2014].

[21] BigSwitch, "Floodlight Controller," http://www.projectfloodlight.org/floodlight/, 2013, [Online; accessed 01-January-2014].

[22] NoxRepo.org, "About POX," http://www.noxrepo.org/pox/about-pox/, 2013, [Online; accessed 01-January-2014].

[23] P. Biondi, "Scapy, a powerful interactive packet manipulation program," 2010.

[24] "Dnslib," https://pypi.python.org/pypi/dnslib, [Online; accessed 01-January-2014].

[25] "Most visited websites," "https://ebizmba.com/articles/most-popular-websites", [Online; accessed 01-January-2014].

[26] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," 2013.