The Interdisciplinary Center, Herzliya

Efi Arazi School of Computer Science

M.Sc. Program

Mitigating Layer 2 Attacks: Re-Thinking the Division of Labor

by

Nir Solomon

Final project submitted in partial fulfillment of the requirements

for the M.Sc. degree in Computer Science,

The Interdisciplinary Center, Herzliya

November 2015

This work was carried out under the supervision of Prof. Anat Bremler-Barr from the Efi Arazi School of Computer Science, The Interdisciplinary Center, Herzliya.

This research was (partially) funded by the Office of the Chief Scientist of the Israel Ministry of Economy under the Neptune consortium for network programming.

Abstract

We provide an overview of Layer 2 attacks in OpenFlow: ARP Poisoning and a new DDos attack on the Controller, both implemented by us. We will describe our approach to mitigate these attacks, called *Switch Reactive ARP-query*. The key idea is to shift responsibilities back from the control-plane to the data-plane in order to reduce the load on the Controller.

ARP Poisoning is the kind of attack in which an attacker is able to alter or change the victim's ARP cache in order to leverage it to Man in the Middle (MitM) attack or a Denial of Service (DoS) attack. A Distributed Denial of Service (DDoS) is a form of attack in which the victim's resources are being depleted by multiple adversaries.

Both of these attacks are relevant in an OpenFlow-managed SDN network, where the contradicting relationship between the whole view of the network and the centralized Controller may clash.

In this paper, we have successfully mitigated ARP Poisoning attacks and have decreased dramatically and bounded the number of packet-ins, the main cause for the DDoS on the Controller.

Table of Contents

1. Introduction 6

2. Background: Layer-2 Attacks 9

2.1 ARP Poisoning 9

2.2 Controller DDoS 10

3. Related Work 12

4. Executing Layer-2 Attacks in OpenFlow 14

4.1 ARP Poisoning 14

4.2 Controller DDoS 15

5. Solution Requirements - Desired Properties 19

6. Suggested Solution 21

6.1 Shifting Control-Plane Responsibilities 21

6.2 Algorithm 22

6.3 Rate Limiting 24

6.4 The Division of Labor 26

7. Evaluation 28

7.1 Implementation 28

7.2 ARP Poisoning 28

7.3 DDoS - Naive Attacker 29

7.4 DDoS - Sophisticated Attacker 31

7.5 Network Overload 32

8. Conclusions 34

9. References 35

List of Figures

Figure 1: Normal ARP traffic and ARP Poisoning traffic 11
Figure 2: Spoofed ARP packet 15
Figure 3: Attack traffic is simulated by creating a large set of packet-in events 18
Figure 4: The number of packet-ins received at the Controller for a naïve attacker 30
Figure 5: The number of flows installed for a naïve attacker 30
Figure 6: The number of packet-ins received at the Controller for a sophisticated attacker 31
Figure 7: The number of flows installed for a sophisticated attacker 32

List of Algorithms

Algorithm 1: Reactive ARP Query Algorithm, running on the OpenFlow Switch 26

1. Introduction

In recent years, Software Defined Networks (SDN) rose to fame by its ability to enable developers and administrators to control and program their network dynamically, by turning the switching infrastructure programmable. One of the many expectations from the SDN paradigm is to solve some of the network security threats using its entire view of the network. One such security threat is the ability to perform an ARP Poisoning attack: a technique that creates a Man-in-the-Middle by sending spoofed ARP packets into the network. This attack enables the attacker to sniff and drop confidential data destined to the victim.

Many expect that SDN's ability to gather information from the entire network, and the ability to create a whole view of it, will help mitigate and prevent network attacks from happening. We are now able to aggregate network statistics and match rules on packets - all within a centralized machine called the Controller. Contrary to the above, even with the Controller's whole view, we will show that an ARP Poisoning attack is still easy to implement and deploy in an SDN network.

All the more, we will show that the usage of a centralized Controller in SDN opens the door to new threats; a DDoS attack on the Controller, using only the network's endpoints. By consuming the resources of the Controller in a short time, triggering a large amount of packet-in events, we have managed to overload it. These packet-in events occur when the switch receives a packet with no matching flow installed. In such case, the switch will request more information from the Controller regarding the desired action. This kind of attack, of course, is a new threat that

inherently exists in OpenFlow networks. Both ARP Poisoning and DDoS attacks are successful implementations of Layer 2 attacks. Some partial solutions to mitigate DDoS have been proposed [1], such as using proactive Flows. In a proactive approach all the flows are installed beforehand, which helps reduce the number of packet-ins sent to the Controller. This approach may help against a DDoS, but requires a large management overhead. It is also not as dynamic as a reactive approach, where flows are installed dynamically, on the fly.

Our approach to this problem is to shift some of the responsibilities of the Controller back to the switches, enabling a large view of the network, while mitigating an overload of the Controller. Our solution, called Switch Reactive ARP Query works by keeping a partial view of the network in the data-plane switches, along with the existing whole-view in the control plane. By keeping this view, each switch is able to retain the mapping between the MAC address of each host connected to it, and the physical port and IP address of that host. For new hosts that connect to the network, we use a challenge-response mechanism called Reactive ARP Query in order to validate that the host is not an attacker. Retaining this mapping makes ARP Poisoning attacks much harder to execute and also helps mitigate the DDoS attack on the Controller, while reducing the load that is being imposed on the Controller. We will show that our solution enables the prevention of ARP Poisoning and to scale against Controller DDoS attacks. As we will demonstrate, our simulations determine that our solution helps mitigate both attacks, with minor overhead.

In summary, our paper makes the following contributions:

- Overview of an ARP Poisoning attack and its execution in an OpenFlow network.
- Presentation of a new attack vector on OpenFlow - ARP level Distributed Denial of Service (DDoS) attack on the OpenFlow Controller.
- Extension of the current implementation of OpenFlow to include new mechanisms to mitigate Layer 2 attacks, and an evaluation.

2. Background: Layer-2 Attacks

2.1 ARP Poisoning

The Address Resolution Protocol (ARP) is the Protocol used to create a mapping between Layer 3 addresses (i.e. IP address) and Layer 2 addresses (i.e. MAC address). As the switch uses MAC addresses to forward the network traffic, ARP is used whenever a endpoint host tries to connect to another on the switched network and the mapping of the IP and the MAC is not known. The latest mapping between the IP and MAC addresses is stored in a local ARP cache of each host. In order to create the mapping, the host's system will first try to find the mapping in the local cache. If the mapping is cached, ARP is not used. If the mapping is not cached, the system will broadcast a packet over the local network, asking "Who-has the destination IP?". The ARP request packet will travel from the first host to its switch, who will in turn broadcast the packet to all other interfaces. The packet will then reach the destination host, who will in turn send an ARP reply.

An attacker can abuse this protocol [2], and create an ARP Poisoning attack which is a Man-in-the-Middle (MitM) attack between the victim host and the counterpart system it communicates with. This is done by associating the attacker's MAC address with the IP address of the counterpart system. This is a known issue in legacy networks. Figure 1 illustrates normal ARP traffic and ARP Poisoning attack in legacy networks.

The Address Resolution Protocol has no authentication mechanism and is stateless. As such, the attack is performed by creating a spoofed ARP packet that poisons the victim's ARP cache. By poisoning the cache, the attacker associates his MAC address with the IP address of another host. By using an ARP Poisoning attack, the attacker is then able to inspect the packets, modify them or drop them, breaking the CIA (Confidentiality, Integrity and Availability) model, while keeping it completely transparent for the victim.

We will show that executing such an attack in an SDN network is relatively easy. In this paper, we will show that spoofed ARP packets may also be used to create a Denial of Service (DoS) attack in the OpenFlow network.

2.2 Controller DDoS

A Distributed Denial of Service (DDoS) attack is the attempt to make a machine or service unavailable to its legitimate users, by consuming its resources by a large group of attackers. The attacking machine may participate in the attack without the consent of its owner, such as in a Botnet, where a malicious program takes control of the machine. A DDoS scenario may quickly worsen, such as when sending a packet to the victim will cause him to send a reflected packet to another host [3]. In many cases, the best solution to contain the damage is to use Response Rate Limiting - RRL [3]; by rate limiting the victim's responses, we can bound the number of reflected packets that the victim will send to another host. We will show that it is possible to create a DDoS on the OpenFlow Controller, while using only ARP packets.

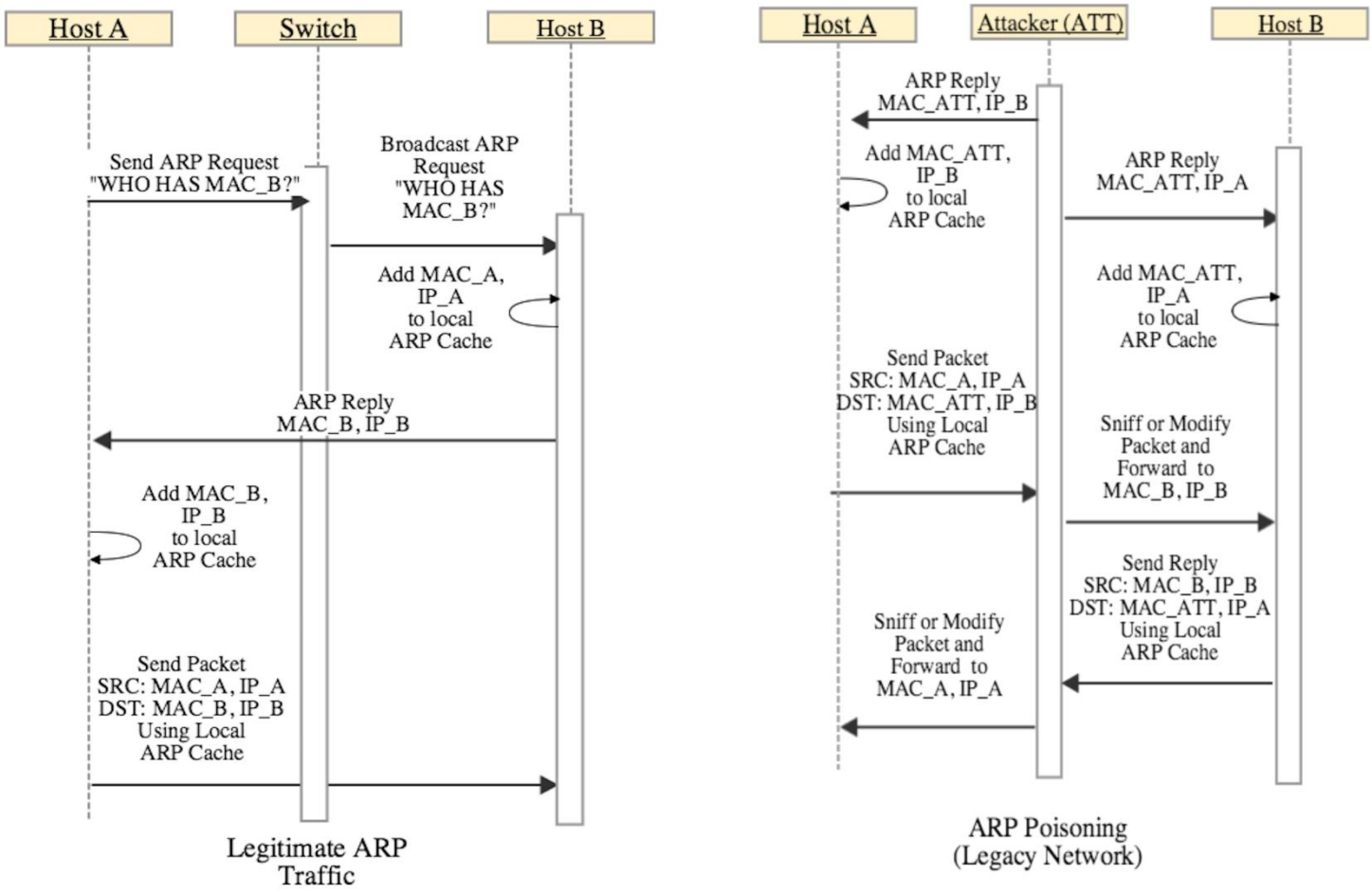

Figure 1: Normal Behavior of ARP traffic Vs. ARP Poisoning traffic scenario. While the switch is not visible in the ARP Poisoning traffic chart, all traffic is forwarded by it.

3. Related Work

Leveraging SDN to make the network more secure is an active line of research [4][5]. There are several works that discuss the issue of Controller DDoS vulnerability, but they do not discuss the possibility of an ARP Poisoning attack or its solution in SDN. As far as we know, there is currently no other research that tries to solve ARP Poisoning using SDN - we are the first to analyze this kind of attack in OpenFlow and propose a solution.

Protecting the network against an ARP Poisoning attack has been a mission for the network administrator for a long time. Possible mitigation processes against ARP Poisoning in legacy network have been proposed in [2]. The described protection mechanism uses ARP packets to validate the hosts, similar to our approach. While the writers describe the disadvantage of adding and managing a centralized server to handle the long term IP-to-MAC mappings, we can use the already existing centralized Controller to hold this mapping.

A recent survey of DDoS attacks emphasizes the contradicting relationship between SDN and DDoS: while SDN may be a great platform to defend against such attacks, SDN itself is susceptible to DDoS [6]. The writers of this paper noted that an attacker may create any number of new and unknown flows by spoofing packets, potentially overloading the switches and possibly the Controller.

An attack which overloads the Switch input buffer was described in [7], which causes the switch to send whole packets to the Controller to overload it. The writers did not implement it, though. Also, in [7], several

Denial of Service risks caused by Controller vulnerabilities were described. Similar to our approach, they have also claimed that proactive network may not be as susceptible to the DoS risks as reactive networks. However, they did not suggest any mitigation method using this observation.

Similar to our approach, the use of specially crafted packets to attack OpenFlow network was also described in [8]. However, they did not describe the packets they used. These packets were mainly used for fingerprinting an OpenFlow network, and only then to attack it with a DoS attack. This attack was not described in detail. An attack against the Controller and how to prevent it was introduced in [9]. Like us, they claim that the centralized controller quickly becomes a bottleneck, a potential Achilles heel, and that an adversary can produce a series of unique flow requests to quickly saturate the control plane. They also proposed to allow the switch itself to apply a new flow rule which drops packets when an attack is detected, without involving the controller. They did not, however, execute an actual DDoS attack against the controller - only mentioned it as a potential bottleneck.

As a side note, there are several papers that suggest that SDN has a promise to solve DDoS in the network using OpenFlow whole view and the ability to dynamically react to events [10][11].

4. Executing Layer-2 Attacks in OpenFlow

4.1 ARP Poisoning

In order for the Controller to programmatically manage the switches, it runs a Learning Switch application. It is an OpenFlow application which receives packets that did not match any flow on the OpenFlow switch, and in return installs a new flow on the switch. Two flows are installed - one that matches on the source MAC, and one that matches on the destination MAC. Installing a flow will prevent further packets to/from known hosts to be escalated to the Controller, making the forwarding process much faster as only the switch is involved. In such cases the switch are operated in a reactive mode, with result in either forwarding the packet to the correct port, or forwarding it to the Controller for further processing. This enables the creation of a MAC-to-Physical port mapping at the data plane and a whole view of the network mappings at the Controller. In legacy networks, however, it is not feasible - legacy switches learn the Layer-2 addresses of the small subset of hosts connected to them, without the centralized Controller. In legacy network, there is now centralized location that holds a mapping of the whole network.

The ARP Poisoning attack is executed by spoofing a special ARP packet which will poison the victim's ARP cache, into thinking that the attacker is his session counterpart. Both the Learning Switch application and the OpenFlow Switch do not provide any mitigation against such attack - the spoofed packet will reach the victim and poison his cache.

We have successfully executed this attack in an OpenFlow network by forging a spoofed ARP Reply packet containing the Source MAC of the attacker and the Source IP of another system. We have run our ARP Poisoning attacks in OpenFlow network using Mininet, and successfully created a MitM scenario using this attack. This implementation is exactly the same as in legacy networks.

One of the many partial solutions to mitigate ARP Poisoning in legacy networks, is to use an authentication mechanism. For example, when the RADIUS protocol is used, it will be easier to mitigate spoofing and poisoning, as the hosts are authenticated. Even so, preventing this attack in BYOD networks can be much harder, where no authentication may take place.

Ethernet Header	Arp header
Src MAC = **random MAC** Dst Mac = FF:FF:FF:FF:FF:FF (broadcast) Type = ARP Request	Src MAC = **random MAC (same as in ETH header)** Dst MAC = FF:FF:FF:FF:FF:FF (Broadcast) Src IP = Another host IP Dst IP = Another host IP (same as Src IP)

Figure 2: Spoofed ARP packet.

In order to create a DDoS on the Controller, the Src MAC can be randomly generated and thus millions of different packets can be created, each creating a matching packet-in event.

4.2 Controller DDoS

Alas, the ability of spoofing a packet to our liking and replaying it over the reactive OpenFlow network, opens the door to other Layer 2 attacks. By creating specially crafted packets (such as spoofed ARP packets), we can

create a scenario in which each host will send a large number of these packets to create multiple packet-in events in the Controller. While these packets are used by OpenFlow to create the connection between the data plane and the control plane in order for the Controller to manage the switches, they may cause harm. These packet-ins may result in an overloading of the centralized Controller - **effectively creating a DDoS attack that is unique to OpenFlow and inherent in its design** and affects the entire network

We have successfully implemented and executed such attack in an OpenFlow network, while using a commercial grade Beacon Controller. The packets that we repeatedly sent in order to generate packet-in events are spoofed ARP packets with random source MAC addresses. Figure 2 illustrates the full spoofed packet specification.

In our simulation, each host repeatedly sent this packet to the switch. In turn, when the switches observe a packet from an unknown source or destination MAC, they send a packet-in packet. In order to create a DDoS on the Controller, the Src MAC can be randomly generated and thus millions of different packets can created, each creating a matching packet-in event sent to the Controller and wait for a flow to be installed. Since we use constantly-changing random MACs, each time our packet was sent, a packet-in event was generated and sent to the Controller. We evaluate the controller performance as the number of responses it sends as the number of packet-ins. Figure 3 illustrates how well the Controller copes with the Attack traffic that simulated a large number of packet-ins. As can be seen, when reaching a certain threshold of about 1.3 Mpps (million packet-ins per second), the number of responses does not increase

correspondingly. Above this threshold a large number of packet-ins are not handled; a DDoS.

We have achieved this result using two physical machines. The first is a MacBook Pro with Intel Core i7 CPU and 16GB of RAM running the Controller. The attack traffic was simulated using Cbench on an Ubuntu machine with Intel Core i5 CPU and 4GB of RAM. Achieving this result with a single machine emphasizes the severity of this kind of attack. We note that today the number of packets that have no matching flow that the switch can handle is up to 12000 packets, per second [12]. These packets will trigger a packet-in. An attacker using this rate, will need about 110 switches to create the DDoS. Work is done to enable OpenFlow switches handle a large number of such packets, handling 12000 such packet per second, will limit drastically the number of packet that can be forwarded. In order to really support forwarding a large volume of packets (e.g. 1 Gbps throughput), we will need to also scale in the number of packets not matching any flow that the switch can handle. This process of scaling will also enable the DDoS [13].

Our observation is that the key to solving these vulnerabilities lie in the way OpenFlow handles packets which do not match any rule (table-misses), which cause a packet-in to be sent. By reducing the number of table-misses, we will be able to contain such an attack.

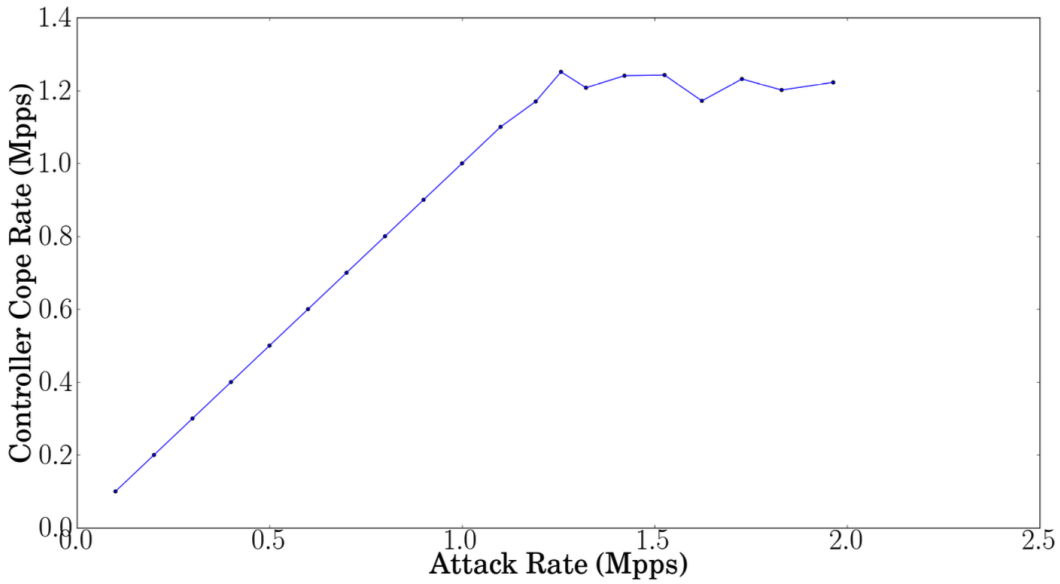

Figure 3: Controller Cope Rate as a function of the Attack Rate.

The attack traffic is simulated by creating a large set of packet-in events. For each packet-in, the Controller should send a packet-out. The Controller Cope Rate measures the packet-out rate during the attack.

5. Solution Requirements - Desired Properties

We would like to propose changes to the OpenFlow protocol, that will fully utilize its full network view, while not overloading the network or the Controller, and taking the following criteria into account, to find a satisfactory solution:

- Little load on the Controller, regarding packet-in events.
- A lesser number of flows installed on the OpenFlow Switch - a reactive approach allows the attacker to install flows as he likes by sending the above specially crafted packets, until the flow table of the switch reaches its maximum capacity. This could result in filling up the switch's TCAM or overloading it, causing any new flows arriving to be dropped.
- A reactive solution - we would like to be able to install flows on the fly, as opposed to a proactive approach. A proactive approach requires us to install many flows in advance, using our knowledge of the network topology. This can cause an overhead for the network administrator. This approach may also be irrelevant in BYOD (Bring Your Own Device) networks, where the MAC or IP addresses of the devices are not known in advance.

Furthermore, while several papers have proposed the use of multiple Controllers or distributed control-plane to mitigate DDoS [7][9], we suggest an orthogonal solution. It coerces the attacker to spend more resources, but the attack will still be feasible. Hence, we would like to propose a solution that will increase the overall security in OpenFlow,

and make the attacks not harder to execute for the attacker. We will compare our proposed solution to the Reactive approach, and the tradeoffs of using them.

6. Suggested Solution

6.1 Shifting Control-Plane Responsibilities

SDN's philosophy is that there should be a total separation between the control plane and the data plane. However, we have seen that it creates a load on the Controller. We will show that shifting some of the responsibilities of the Controller back the switch, will allow a much more robust OpenFlow architecture. This shift will enable the mitigation and scaling against several network attacks, while maintaining a fully functional network. By allowing the switch to hold the IP to-MAC mappings of the hosts connected to it, similar to all the mappings held by the Controller, we are able to shift some of the OpenFlow decision making process back to the switch.

We are not the first to suggest a mechanism that shifts the load of the controller back to the switches: in AvantGuard [9], the writers of the paper implemented an extension module for the OpenFlow switch, to enable quick reaction when a network attack starts. These simple extensions added a very minimal intelligence to the data plane, that increases the resilience to anomalous control-plane floods occurring at the data plane. Also, in DevoFlow [16], the writers suggested that the Controller doesn't necessarily need to be involved in every flow setup. Instead, their main design principal is to avoid leaving the data-plane, since the bandwidth between the Control and Data plane is limited. They have suggested a "Rule Cloning" mechanism that devolves some control back to the switch , together with the use of wildcard flows.

6.2 Algorithm

In both Layer 2 attacks, spoofed ARP packets (unknown source or destination MAC address) are used. In this case, we would like to find and validate the identity the of the new host - check if the IP matches the MAC.

Our suggested solution is named the **Switch Reactive ARP Query**: Let us assume that the network has been running and is active. We aim to retain a IP-to-MAC and a MAC-to-Physical port mappings of each host connected to the switch, at the switch level. This mapping is saved in the switch level in the form of OpenFlow flows. When a new host connects to the network, which could be cause by a valid host sending normal TCP/UDP/ARP data or the start of an attack, we can try and match the packet with our current mappings, if we have one. Of course, this is a new host, so we won't have a matching flow. In this case, we want to validate the host. The host validation is done by sending a Reactive ARP Packet. The **OpenFlow Switch itself** will send an ARP Query request packet (who-has IP, can be source or destination IP). By sending this query, we can validate the mapping further. If the host is validated correctly, we will create a packet-in event sent to the Controller so he can install a matching rule on the switch. If the hold is not validated, the drop will be dropped.

There are three apparent scenarios:

- Two ARP replies (one with expected MAC, one with other MAC).
- Invalid ARP reply (a packet is returned with a different MAC address than expected/No reply).
- Valid ARP reply (matching IP and MAC).

Generally, when speaking of Layer 2 attacks, there are two kinds of attackers: a naive attacker and a sophisticated attacker. A sophisticated attacker will reply to an ARP request for each source MAC he spoofs , where a naive attacker will not.

In the first scenario, both the attacker and the victim replied to the query. The attacker is sophisticated, but in most cases the victim is an existing host, hence he has already been authenticated before - the switch holds its IP-to-MAC mapping, so it's easy to differentiate between the victim and the attacker, so the packet will be dropped by the switch.

In the second scenario the attacker is naive, and the victim replied to the query, or not at all, so the attacker's packet will be dropped. In both cases, an attacker was discovered so the switch will drop the original incoming packet and **will not notify the Controller**, as to not overload it. A packet-in event will not be created.

In the third scenario, it may be the victim or a sophisticated attacker replying. The new host is now validated, so we should notify the Controller, which will in turn install a flow on the switch. Algorithm 1 illustrates the algorithm flow.

Handling the above three scenarios will allow the complete mitigation against ARP Poisoning for both attacker types. We assume a steady state when the network starts operating. If an attacker binds the victim's IP address to its MAC address before the victim connects, the victim's packets will be dropped. We can prevent mistaking a user for an attacker by using our solution as soon as the network starts operating or installing proactive rules for all known safe hosts ahead.

6.3 Rate Limiting

Alas, a sophisticated attacker can still create a large number of packet-ins using spoofed packets with non-existing source or destination MAC and IP addresses. A Reactive ARP query will be sent for each packet, but all will be validated.

To prevent this from happening in a large scale, a response rate-limiting (RRL) algorithm is in place, on the physical port of the sender. Only the rate of table-misses is limited, using OpenFlow Meters on the switch itself - an OpenFlow feature which allows rate-limiting by matching packet headers, such that valid traffic will not be affected. A similar method is being used in DNS DDoS Amplification attacks mitigation [3]; The network administrator will configure his desired rate of table-misses per second. Rate-limiting was also proposed as a form of defense against DoS attacks in SDN networks [7]. In [1], the writers conclude that "With regard to denial-of-service vulnerabilities, controller vendors need to emphasize the importance of rate-limiting". Also, when assessing the security of a Learning Switch Application, the writers conclude that "Even for an application as simple as this one, protecting it requires implementing MAC spoofing protection through port counters (with timers if network flexibility is required), rate limiters for traffic to unknown hosts, and switch rules for multicast". The final conclusion is that "networks that rely heavily on PacketIn messages (i.e. reactive designs) can be exposed to denial-of-service attacks against the switches and controller(s) without complex rate-limiting logic in the controller".

Note that when rate limiting is enabled, the physical port may drop packets sent from valid hosts connected to this port. To prevent this, we

use the rate-limit on the closest OpenFlow switch connected to the attacker. If the attacker is connected directly to an OpenFlow switch, and not a legacy switch, only the attacker will be limited. Also, note that using the rate-limit does in fact improve the network availability for valid hosts - not using it will enable the attacker to create as many flows per second as he would like, causing the switch or Controller to stop handling new flows at all.

While choosing the rate of the rate-limit itself is out of the scope of this paper, work is being done to use the separation of the control and data plane in OpenFlow to gather network statistics [14]. A method for collecting flows statistics in order to detect DDoS attacks in OpenFlow is described in [10]; This method, called “Flow Collector” fetches different network statistics. One of the most interesting metrics being pulled from the switch is “Percentage of Pair-flows (PPf).

The writers describe a “Pair-Flow” as a flow for which the following conditions are true:

- The source IP of flow 1 must be equal to the destination IP of flow 2
- The destination IP of flow 1 must be equal to the source IP of flow 2
- Both flows must have the same communication protocol

As a DDoS increases the number of single-flows because of spoofed packets [10], this is a very important metric for the detection of a DDoS attack. We feel that future work in this area, will allow to rate-limit the suspected port, using the statistics of the single and pair-flows that reside in the network. A switch that has host suspected as a sophisticated attacker, may use these statistics to decide if the physical port has a large number of single-flows, and limit it accordingly.

Algorithm 1 Reactive ARP Query Algorithm, Running on the OpenFlow Switch

```
1: procedure HANDLEPACKETIN(pkt)                    ▷ SRC: (MAC_A, IP_A) DST: (MAC_B, IP_B)
2:     if pkt has no matching SRC flow (MAC_A, IP_A) then
3:         pkt.expected_mac ← MAC_A
4:         SendReactiveARP(IP_A, IP_SWITCH, MAC_SWITCH)          ▷ Check SRC spoofing
5:     end if
6:     if pkt has no matching DST flow (MAC_B, IP_B) then
7:         pkt.expected_mac ← MAC_B
8:         SendReactiveARP(IP_B, IP_SWITCH, MAC_SWITCH)          ▷ Check DST Spoofing
9:     end if
10:    return
11: end procedure

12: procedure HANDLEREPLIES(reactive_arp_query_packet, original_packet)
13:    responses ← replies for reactive_arp_query_packet
14:    if no responses then
15:        DropPacket(original_packet)                  ▷ No replies - Host A is an attacker
16:        return
17:    end if
18:    for each response r in responses do
19:        if r.source_mac! = original_packet.expected_mac then
20:            DropPacket(original_packet)          ▷ Non-matching MAC, Host A is an attacker
21:            return
22:        end if
23:    end for
24:    NotifyController(original_packet)▷ Valid reply, Host A may be a valid host. The Controller installs a matching flow on
       the switch, and enables Rate-Limiting on Host A's port in case the host is an attacker.
25:    return
26: end procedure
```

Algorithm 1: Reactive ARP Query Algorithm, running on the OpenFlow Switch

6.4 The Division of Labor

While these Reactive ARP Query validations could have been performed at the Controller-level only, and not by the switch, this would have caused a packet-in to be sent from the switch to the Controller for every packet now matching a flow, which might cause an overload at the Controller in both ARP Poisoning and DDoS. Shifting this responsibility back to the switches, enabled us to scale against such attacks using the data-plane itself.

The fine line between the Control Plane and the Data Plane has never been clear to all. One of the several papers that have crossed that line is [9]. The goal of this paper is to increase the resilience of data-plane to control-plane floods. The writers introduced several extensions that adds minimal intelligence to the data-plane to achieve this goal. As said in [10], when thinking of the default component used to mitigate network attacks in an SDN network, we think of the Controller. We conclude that the Controller, by itself, cannot hold these responsibilities, as the load may cause it to stop function correctly. Instead, our approach takes the a different direction; we shift these responsibilities from the “default” component, the Controller (control plane) into the switch (data plane).

7. Evaluation

7.1 Implementation

We implemented our solution using two machines: the Controller was run on a MacBook Pro with Intel Core i7 CPU and 16GB of RAM. The endpoint hosts were run using Mininet on an Ubuntu machine with Intel Core i5 CPU and 4GB of RAM. We used an unmodified Open vSwitch 2.3.1, running OpenFlow 1.3.

To simulate our Reactive ARP Query extension module, we have used the Ryu Controller running the Reactive ARP Query Switch Application [15]. This application functions as a Reactive Learning Switch, with the new capabilities of the above Reactive ARP Query. In our tests we have compared both the mitigation of ARP Poisoning and Controller DDoS, by simulating both the Reactive ARP Switch (using the Reactive ARP Query Switch Application) and a normal, naive OpenFlow switch (using the naive Learning Switch Application). We've chosen Ryu as our Controller, as it's both a Python Controller, allowing us to use the Python ecosystem strengths, and it is one of the only Controllers that support OpenFlow 1.3. As part of future work, we hope to also implement the Reactive ARP Query extension as a totally independent data plane component.

7.2 ARP Poisoning

First, we run our ARP Poisoning test; the forged ARP request, sent by a sophisticated attacker, has been blocked by the switch because an ARP Reply was sent from the real host. We have tried this attack from different hosts connected to multiple switches, and all attacks have been blocked

by dropping the first attack packet. This means we effectively prevented all of our simulated ARP Poisoning attacks by using the Reactive ARP Query mechanism.

7.3 DDoS - Naive Attacker

We then run our DDoS tests, which include sending forged ARP packets with spoofed source MAC, simulating both naive and sophisticated attackers. A Reactive ARP Query will be sent against each such packet. A naive attacker does not reply to the ARP Query packet, so the packets will be dropped (and not raised as packet-ins to the Controller). This makes the DDoS attack not feasible for the naive attacker.

attack not feasible for the naive attacker. Figure 4 illustrates the comparison of the number of packet-ins at the Controller, as a function of the number of spoofed packets. Using a normal OpenFlow Switch, each spoofed packet requires a packet-in to be received by the Controller. Using the Reactive ARP Query Switch, no packet-ins are received at the Controller at all.

Figure 5 illustrates the comparison of the number of flows installed, as a function of the number of spoofed packets. Using a normal OpenFlow Switch, two flows will be installed for each packet-in - one matches on the source, and one on the destination. In our solution, we drop the attacking packet, without installing any flows, and without informing the Controller.

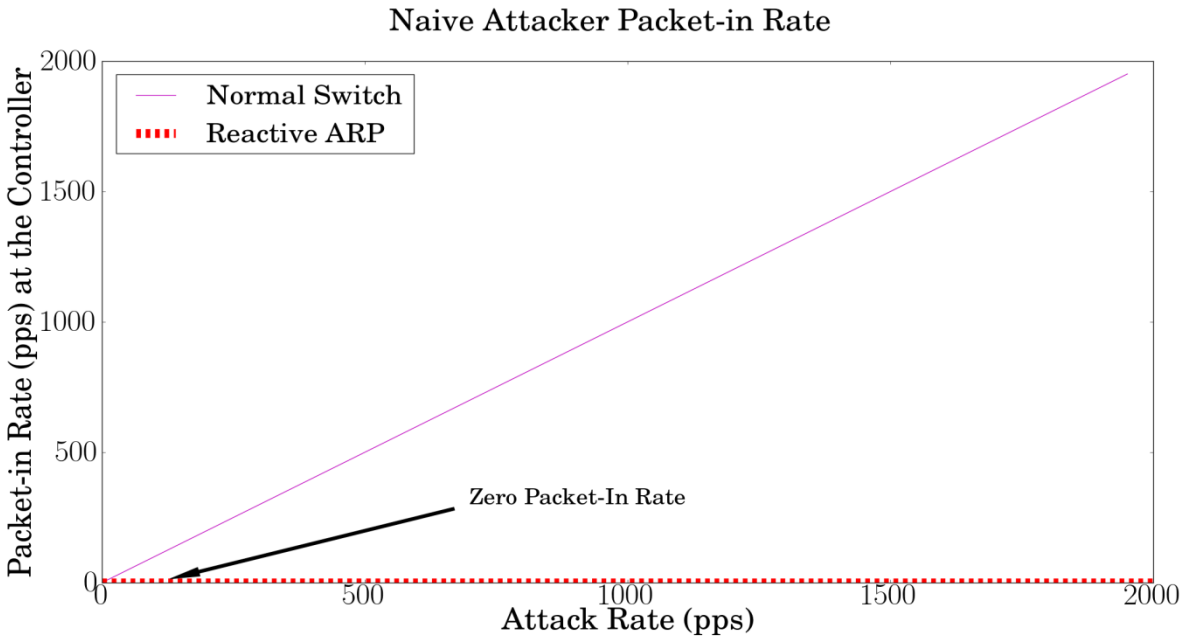

Figure 4: The number of packet-ins received at the Controller for a naive attacker. Using Reactive ARP results in a rate of zero packet-ins.

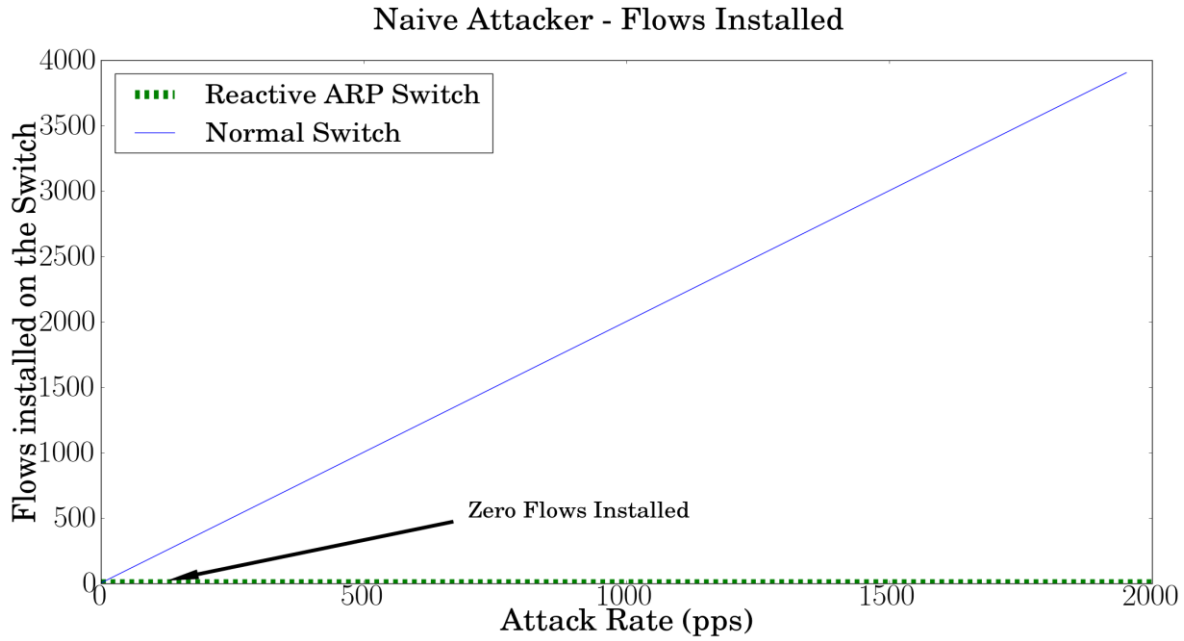

Figure 5: The number of flows installed for a naive attacker. Using Reactive ARP results in zero flows installs.

7.4 DDoS - Sophisticated Attacker

A sophisticated attacker does reply to the ARP Query packet. The first attack packet will arrive at the switch, where the host will be validated and Rate-Limiting on the match-misses will be enabled.

Figure 6 illustrates that using a normal OpenFlow Switch, each spoofed packet requires a packet-in to be generated by the switch and received by the Controller. Using the Reactive ARP Query Switch, the rate of the packet-ins are limited (in our case, to 100 packets per second, chosen arbitrarily). If the Rate-Limiting rate is N pps, then we can assure that the maximum packet-in rate at the Controller will be N pps as well.

Figure 7 illustrates that using a normal OpenFlow Switch, two flows will be installed for each packet-in - one matches on the source, and one on the destination. In our solution, we rate-limit the incoming packet-ins, so the rate of the creation of new flows depends on the rate limitation.

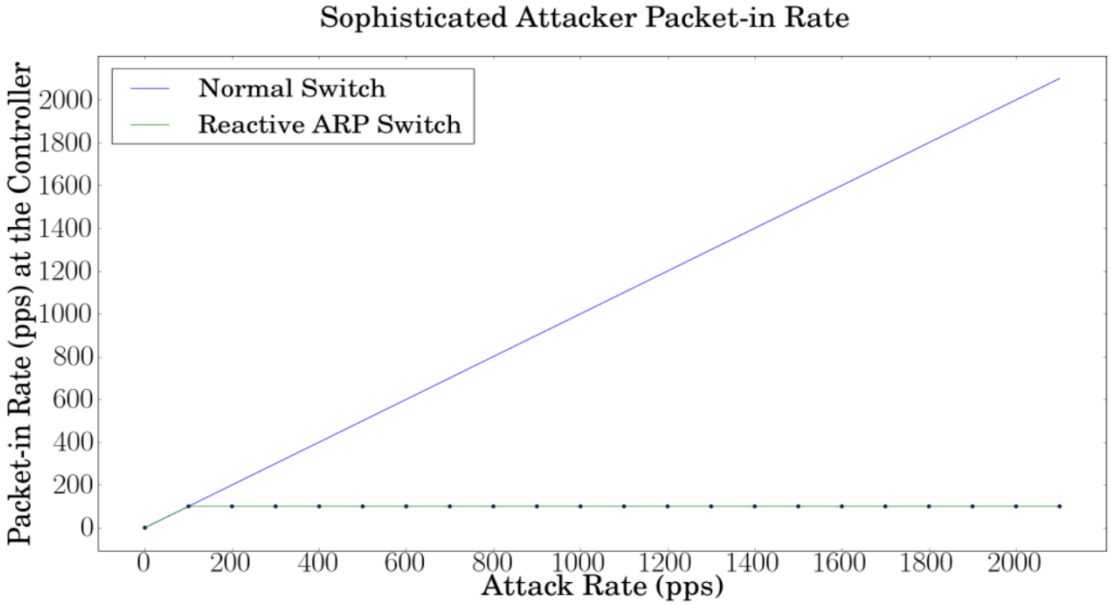

Figure 6: The number of packet-ins received at the Controller for a sophisticated attacker.

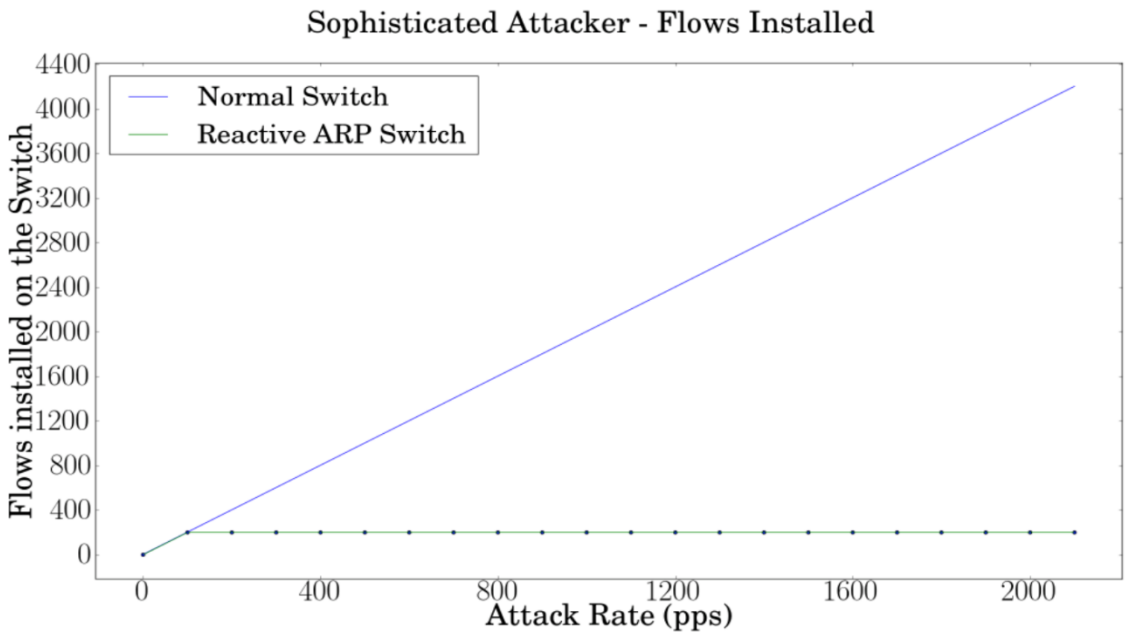

Figure 7: The number of flows installed for a sophisticated attacker.

7.5 Network Overload

Let us investigate the traffic overhead of our solution. The three possible scenarios are:

- Two ARP replies (one with expected MAC, one with other MAC).
- Invalid ARP reply (a packet is returned with a different MAC address than expected/No reply).
- Valid ARP reply (matching IP and MAC).

In the first scenario, the attacker is sophisticated. The first packet sent by him will be intercepted by the switch. A single Reactive ARP packet will be issued for the attacker's packet to be dropped. In the second scenario, the attacker is naive and a single Reactive ARP packet is used for the packet to be dropped. In the third scenario, it might be the attacker's packet or a valid host the connected to the network. Let us assume the worst case scenario - that this host is an attacker. We are sending an ARP

request to validate the attacker for the first time. No more ARP requests will be sent after he is validated, as a flow is being installed by the Controller. Thus the network overhead here is only a single packet. Overall, in order to validate a single host, we use a single Reactive ARP packet.

We should notice that this is a minor overhead for the network, as switches are built to handle large volumes of broadcast packets flowing through them. One other concern is the overhead on the switch itself, it is hard for us to bound the exact overhead, as we emulated the OpenFlow extension using Python code that runs on the Controller, rather than C/C++ code running on a physical switch. We hope that future work will allow the implementation of this extension and the evaluation of its performance while running on a physical switch.

We thus argue that we have successfully prevented ARP Poisoning and decreased dramatically the number of packets-in and flows installed.

8. Conclusions

In this paper we presented the execution of Layer 2 attacks in OpenFlow network: ARP Poisoning, which is executed exactly as in legacy networks, and a Layer 2 DDoS on the Controller, which is an inherent, new vulnerability in OpenFlow networks. While there are several works that discuss OpenFlow security, we are the first ones to implement both attacks. We have proposed an extension to **OpenFlow - Reactive ARP Query**.

While our solution has only a minor network overhead, both attacks can be mitigated successfully. We believe this is a large step taken in changing our perception about the security issues that are inherent to SDN, and what advantages these networks have over legacy ones. This is a major step in changing our approach by shifting the control plane responsibilities back to the switch, as the Controller, being the single point of failure in many networks, cannot handle the forwarding decisions load by itself.

9. References

[1] K. Benton, L. Camp and C. Small. OpenFlow Vulnerability Assessment. In Proceedings of the ACM SIGCOMM workshop on Hot topics in Software Defined Networking, 2013.

[2] S.Y. Nam, D. Kim and J. Kim Enhanced ARP: preventing ARP poisoning-based man-in-the-middle attacks In Communications Letters, IEEE (Volume:14 , Issue: 2), 2010.

[3] E. Winstead. Response rate limiting with bind. In Proceedings of the APRICOT Conference, 2014.

[4] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson and G. Gu. A security enforcement kernel for OpenFlow networks. In Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networks, 2012.

[5] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson and G. Gu. Fresco: Modular Composable Security Services for Software-Defined Networks. In Proceedings of the Internet Society NDSS, 2013.

[6] Q. Yan, R. Yu, Q. Gong and J. Li. Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges In Communications Surveys and Tutorials, IEEE, 2015.

[7] R. Kloti. Openflow: OpenFlow: A Security Analysis. In Proceedings of the Workshop on Secure Network Protocols (NPSec), part of IEEE ICNP, 2013.

[8] S. Shin and G. Gu. Attacking Software-Dened Networks: A First Feasibility Study. In Proceedings of the ACM SIGCOMM workshop on Hot topics in Software Defined Networking, 2013.

[9] S. Shin, V. Yegneswaran, P. Porras and G. Gu. AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In Proceedings of the 2013 ACM SIGSAC Conference on Computer and communications security, 2013.

[10] R. Braga, E. Mota and A. Passito. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In Proceedings of the IEEE Conference on Local Computer Networks, 2013.

[11] R. Krishnan, M. Durrani and P. Phaal. Real-time SDN Analytics for DDos mitigation. Technical report, Brocade Communications, 2013.

[12] Noviflow.com NoviFlow NoviSwitch 2128 Switch Specification. Technical Report, 2015.

[13] A. Wang, Y. Guo, F.Hao, T.V. Lakshman and S.Chen. Scotch: Elastically Scaling up SDN Control-Plane using vSwitch based Overlay. In Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, 2014.

[14] J. Mattes Traffic Measurement on OpenFlow-enabled Switches In Thesis, Department Information Technology and Electrical Engineering in ETH Zurich, 2012.

[15] Bitbucket.org, 'nsolo / reactive_arp_switch — Bitbucket', 2015. [Online]. Available: https://bitbucket.org/nsolo/reactive_arp_switch.

[16] A.R. Curtis, J.C. Mogul, J.Tourrilhes and P. Yalagandula DevoFlow: Scaling Flow Management for High-Performance Networks In Proceedings of the ACM SIGCOMM conference, 2011.

תקציר

עבודה זו מספקת סקירה עיונית ובחינה מעשית של התקפות רמה 2 ברשתות OpenFlow: התקפות ARP Poisoning והתקפות DDoS על רכיב ה- Controller. שתי ההתקפות מומשו על ידינו. בנוסף, העבודה מתארת את גישת החוקר למניעת ההתקפות, הנקראת Switch Reactive ARP Query. רעיון המפתח למניעת ההתקפות הללו הוא הסבת חלק מהאחריות שנמצאת היום ב- Control Plane חזרה אל ה- Data Plane, וזאת על מנת להוריד את העומס המוטל על ה- Controller.

ARP Poisoning היא סוג התקפה בו התוקף יכול לשנות את ה- ARP Cache של הנתקף, על מנת למנף מצב זה לטובתו וליצור התקפת MitM או Denial of Service; כאשר Distributed Denial of Service היא התקפה שבו משאבי הנתקף מתכלים במהירות, בדרך כלל על ידי כמות גדולה של תוקפים. שני סוגי התקפות אלו רלבנטיים ברשתות SDN שמנוהלות על ידי פרוטוקול OpenFlow. זהו מצב בו ישנם יחסים ניגודיים בין טווח הראייה הרחב שהרשת מאפשרת, לבין דבר היותו של ה- Controller רכיב מרכזי ברשת.

עובדה זו מדגימה כיצד הצליח החוקר למנוע את התקפות ה- ARP Poisoning ולצמצם משמעותית את כמות ה- Packet-ins, שהוא הגורם המרכזי להתקפת ה- DDoS על ה- Controller.

המרכז הבינתחומי בהרצליה

בית הספר אפי-ארזי למדעי המחשב

התוכנית לתואר שני (M.Sc.)

מניעת התקפות רמה 2:
חשיבה מחדש על חלוקת האחריות

מאת

ניר סולומון

פרויקט גמר, מוגש כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc.

במסלול הרגיל, בית ספר אפי ארזי למדעי המחשב,

המרכז הבינתחומי הרצליה

נובמבר 2015