
Article:
Girdler, Thomas and Vasilakis, Vasileios orcid.org/0000-0003-4902-8226 (2021) Implementing an intrusion detection and prevention system using Software-Defined Networking:Defending against ARP spoofing attacks and Blacklisted MAC Addresses. Computers & Electrical Engineering. 106990. ISSN 0045-7906

https://doi.org/10.1016/j.compeleceng.2021.106990

Implementing an intrusion detection and prevention system using Software-Defined Networking: Defending against ARP spoofing attacks and Blacklisted MAC Addresses

Article *in* Computers & Electrical Engineering · March 2021

DOI: 10.1016/j.compeleceng.2021.106990

CITATIONS
0

READS
61

2 authors:

Tom Girdler
The University of York
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Vassilios Vassilakis
The University of York
119 PUBLICATIONS **805** CITATIONS
SEE PROFILE

Implementing an Intrusion Detection and Prevention System using Software-Defined Networking: Defending against ARP Spoofing Attacks and Blacklisted MAC Addresses

Thomas Girdler, Vassilios G. Vassilakis

Department of Computer Science, University of York, York, United Kingdom

Abstract

This work focuses on infiltration methods, such as Address Resolution Protocol (ARP) spoofing, where adversaries sends fabricated ARP messages, linking their Media Access Control (MAC) address to a genuine device's Internet Protocol (IP) address. We developed a Software-Defined Networking (SDN)-based Intrusion Detection and Prevention System (IDPS), which defends against ARP spoofing and Blacklisted MAC Addresses. This is done by dynamically adjusting SDN's operating parameters to detect malicious network traffic. Bespoke software was written to conduct the attack tests and customise the IDPS; this was coupled to a specifically developed library to validate user input. Improvements were made to SDN in the areas of attack detection, firewall, intrusion prevention, packet dropping, and shorter timeouts. Our extensive experimental results show that the developed solution is effective and quickly responds to intrusion attempts. In the considered test scenarios, our measured detection and mitigation times are sufficiently low (in the order of a few seconds).

Keywords: Intrusion detection and prevention system, software-defined networking, ARP spoofing, MAC address, OpenFlow, POX controller

1. Introduction

Software Defined Networking (SDN) is a new approach to computer networking whereby the control and data planes are decoupled [1]. This allows implementation of an efficient global configuration across its entirety, making overall network management easier. The centralised control plane, which consists of one or more controllers, improves network performance and enables more efficient network monitoring [2]. SDN-enabled applications send their requests to the controller, which in turn, directly communicates them to the data plane. The approach facilitates efficient *intrusion detection*, which is the process whereby networks are monitored for malicious activities or threats. On the other hand, *intrusion prevention* actively modifies network resources or configuration to mitigate any identified threats. Our work has developed an SDN-based Intrusion Detection and Prevention System (IDPS), with the associated configuration and testing tools. The IDPS was designed to defend against Address Resolution Protocol (ARP) spoofing and Blacklisted Media Access Control (MAC) Addresses. ARP is commonly used to resolve Internet Protocol (IP) addresses into MAC addresses. However, it suffers from many weaknesses, including being a stateless protocol which does not verify the sender from which packets originate. These weaknesses can be exploited to carry out *ARP spoofing*, whereby an attacker sends fabricated ARP messages, linking their MAC address to a genuine device's IP address. Such exploits can launch Man-in-the-Middle (MITM) or Denial-of-Service (DoS) attacks [3]. A greater demand for SDN, with forecasters indicating a global market size of over $100bn by 2025 [4], will inevitability lead to an increase in these attacks. Current research into securing ARP within an SDN environment has focused on utilising data from Dynamic Host Configuration Protocol (DHCP) servers [5], switch MAC to IP address mappings [6], or 'sanitising' spoofed ARP requests with dummy values [7]. These factors led us to the requirement to produce an SDN-based IDPS to detect and alleviate ARP spoofing attacks, which has minimal input from other network services or devices.

Our work contributes the following: (1) We designed and implemented an SDN-based IDPS, which defends against ARP spoofing attacks and Blacklisted MAC Addresses. Python-based POX [8] is the SDN controller utilised, which implements the OpenFlow protocol [9] for communication with an OpenvSwitch (OvS) switch [10]. (2) Tools to conduct the attacks and customise the IDPS were devised. These are coupled to a specialist library which also validates user input. Improvements are made to the POX controller by developing a module that logs the attempted attacks and mitigates them by installing flow rules on the OvS switch. (3) The IDPS was thoroughly tested against its design goals in different scenarios. This involved four types of experiments, namely *ARP Request Attack*, *ARP Reply Attack*, *ARP Reply Destination Attack*, and *Blacklisted MAC Address*. The experiments produced a zero false positive rate and all packets sent were successfully detected. Average detection times for the first three types of attacks were very low (in the order of a few seconds), whereas they were less than a second for Blacklisted MAC Address tests. This process allowed comparisons to other SDN-based IDPS to be made.

The remainder of the paper is organised as follows: Section 2 reviews recent research on IDPS within SDN environment, focusing on ARP-related attacks. Section 3 illustrates how we developed an IDPS. Section 4 describes our testbed, the experimental setup, as well as the developed software tools for attack testing and system administration. Section 5 documents the results of our experiments. Section 6 concludes the paper by summarising the contributions that our work has made and outlining future directions for research.

2. Related Work

Traditional network devices do not include methods for detecting ARP spoofing, although many tools have been developed for such purposes. As a result, in recent years, a number of SDN-based solutions have been proposed. In this section, we review the most important and recent works in the filed. We also discuss their limitations and highlight the novelty of our work.

Khalid *et al.* [5] extend the SDN controller with a module that prevents ARP spoofing by checking every ARP packet within their network. They utilised the POX controller with the *L2 learning* module, an OvS switch, and a DHCP server which inspects DHCP offer packets. Flow rules are installed on the connected switches that forward all DHCP and ARP packets to the controller. POX also holds a main table with IP to MAC address associations for each device on the network, created from the received DHCP offer packets. POX instructs switches to drop received ARP packets with a MAC address not present in the table. A flow rule is then installed for the originating OvS port.

The solution proposed by Cox *et al.* [6] utilised the previously described ARP spoofing mitigation method. The solution is named Network Flow Guard (NFG) and employs the POX controller. DHCP and ARP packets are analysed to build a table with MAC to IP address mappings together with their associated switch port. Compared to other papers, the system has an additional feature, it accepts static MAC - IP address - port number mappings from a user inputted file. For reasons of security, entries to the table can only be made by a valid DHCP offer, with the entry being set to initialised. After that, a DHCP request can be used to assign a port to an existing MAC to IP address entry, at which point the state of the entry is moved to pending. When the DHCP server responds with an acknowledgement (ACK), the entry is moved to verified.

Alharbi *et al.* [7] introduce a system where potentially spoofed information in the SPA and SHA fields of an ARP message is prevented from coming in contact with network hosts. Their solution is based on Network Address Translation (NAT). A routine which forwards ARP requests to the SDN controller is installed on the switch, where they are stored in a list of pending requests. The SPA and SHA fields on the pending ARP requests are then validated and any potentially poisoned fields 'sanitised' with dummy values.

Birkinshaw *et al.* [11] developed an SDN-based IDPS for defending against port-scanning and DoS attacks. They implement and test a variety of mechanisms, including Credit-Based Threshold Random Walk and Rate Limiting. They also devised a new algorithm, called Port Bingo, which guards against

port-scanning attacks. Experiments were carried out on a purpose-built testbed, consisting of a OvS switch, POX controller, together with Virtual Machines (VM) used as attackers, targets, and benign nodes.

Leal *et al.* [12] implemented an architecture composed of two domains – monitoring and countermeasure. Monitoring detects the initial stages of an attack and is comprised of an sFlow agent, collector, and supervision element to visualise network behaviour. The sFlow agent collects data from different sources, including physical and virtual network devices, as well as software applications. sFlow is able to take advantage of the statistical packet sampling to produce measurements and includes an API able to configure customised measurements, metrics and set thresholds.

Abubakar *et al.* [13] integrated a flow-based IDS into the SDN architecture. This is done using an anomaly-based neural network, coupled to the NSL-KDD dataset. The virtual testbed consisted of an OpenDaylight SDN controller, coupled to a OvS switch, and Mininet simulator, with Parrot security generating attacks on Metasploitable2. Their results indicated a high detection accuracy of 97.4% with training data and 97% with testing data. The researchers cite reliability issues related to redundant records in KDD-Cup 99 dataset, from which the NSL-KDD dataset is derived [14].

A recent survey by Shah *et al.* [15] found that most proposed solutions to mitigate ARP spoofing in SDN utilise IP-MAC address bindings held by the controller; either by listening to DHCP server data or ARP requests. The researchers state that these may be subject to DHCP spoofing attacks, or the initial ARP packets themselves could be spoofed. In addition, they also remark that if large numbers of network hosts are present, construction of the IP-MAC address table may cause an additional load on the controller. We have made improvements by developing a solution that does not require an IP-MAC address table to be held by the SDN controller. In addition, our work utilises custom IDPS attack and configuration tools, built using Scapy [16], a powerful packet manipulation tool.

3. IDPS Functions Design

The purpose of our IDPS is to detect devices, within the SDN environment, actively performing ARP spoofing attacks. Furthermore, the IDPS identifies devices whose MAC address has previously been blacklisted using the IDPS Configuration Tool (presented in Section 4). A high-level operation of the IDPS is represented by a flow-chart in Figure 1. When a packet is received without an existing flow-entry, a *PacketIn* event is sent from the OvS switch to the POX controller. The IDPS is an extension to POX and listens for these *PacketIn* events. After the event is accepted, it is processed by one of four separate functions - ARP Request Spoofing, ARP Reply Spoofing, ARP Reply Destination Spoofing, and Blacklisted MAC Addresses. Packets matching set conditions are logged to screen and file, any others are ignored. To mitigate these attacks, the attacker's IP address is obtained from the *PacketIn* event, then used to generate a flow-rule which prevents IPv4 network traffic to and from that specific address. This flow rule is active on the OvS switch for a set number of seconds, known as the flow hard timeout value. The design of this flow modification process was determined by the POX controller.

Each of the IDPS functions has been designed to recognise a separate type of network attack (see Table 1). One function deals with ARP Request Spoofing, another with ARP Reply Spoofing, the other with ARP Reply Destination Spoofing. Finally, a different function detects Blacklisted MAC Addresses. As an example, a normal ARP reply has an ethernet frame with an ARP payload packet encapsulated within it. The source MAC address is identical in the ethernet frame and ARP packet, which verifies it has been sent from the same physical device. However, in a spoofed ARP reply, the ARP payload contains a different MAC address (of the attacker), to that of the frame, so victim device sends their packets to the attacker, not the intended device.

These three ARP spoofing functions have the same basic design but vary slightly dependent upon the type of attack they intend to detect. Once the *PacketIn* event is received by the function, it is transferred into a variable entitled

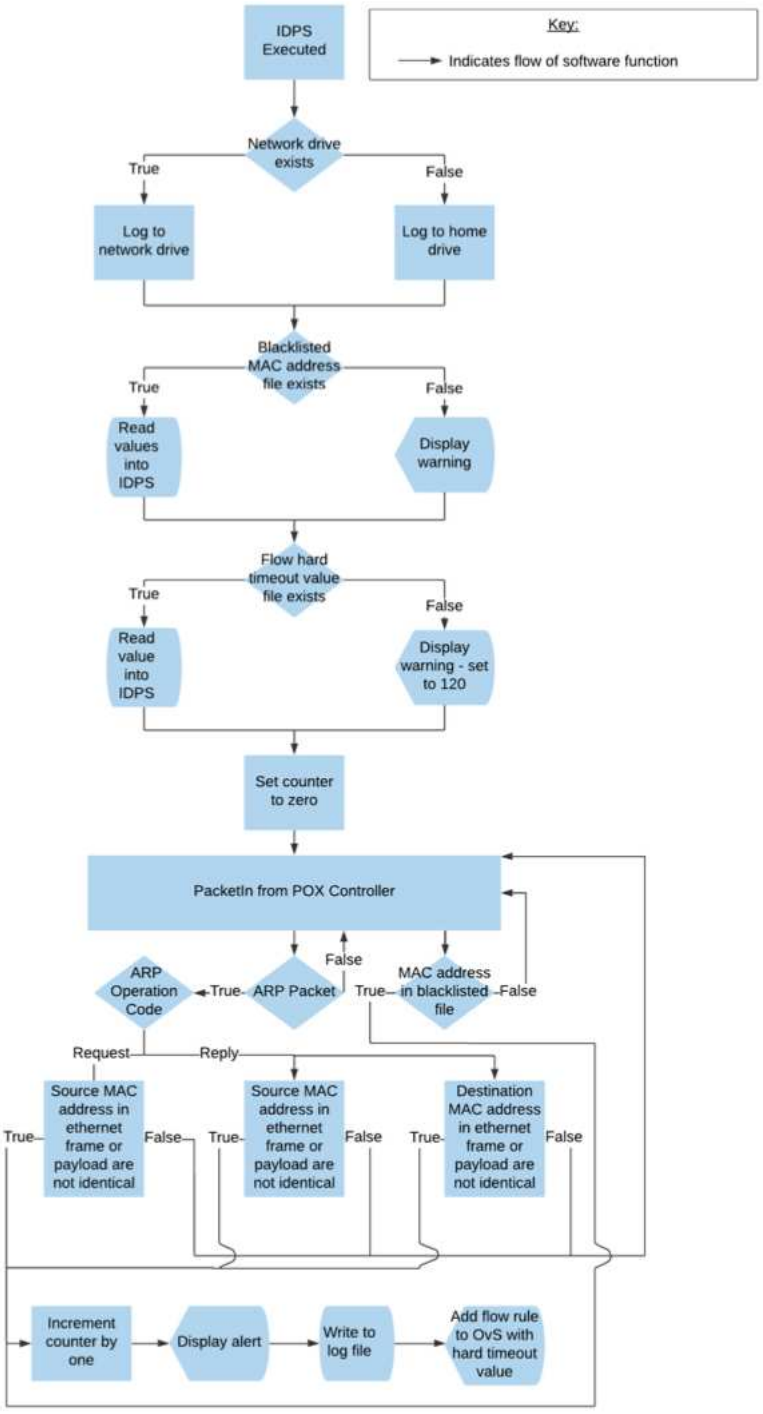

Figure 1: IDPS High-Level Operation

Table 1: IDPS Functions

Function	**Packet Conditions Matched**
ARP Request Spoofing	1. ARP Protocol 2. ARP Request 3. Source MAC Address in ethernet frame or payload are not identical
ARP Reply Spoofing ARP Reply Destination Spoofing	1. ARP Protocol 2. ARP Reply 3. Source MAC Address in ethernet frame or payload are not identical or 4. Destination MAC Address in ethernet frame or payload are not identical
Blacklisted MAC Addresses	1. IP Protocol 2. Source MAC Address of ethernet frame contained within `blacklisted_mac_addresses.json` file

`packet`. Every packet has a field entitled *EtherType* that designates the type of data contained within it. The IDPS evaluates this field, to see if its type is set to ARP. After this, the payload of the packet is assessed, which contains an ARP field entitled `opcode`. This can be set to either Request or Reply, depending on the type of ARP packet. The ARP Request Spoofing function checks if the opcode is set to Request, whereas the ARP Reply Spoofing function looks for Reply. Once it has been ascertained that the received packet contains either an ARP Request or Reply the IDPS will then assess the content of the packet, as per the conditions in Table 1. If all of the conditions are matched, a variable called *counter* is incremented by one. The *counter* references every packet detected by any IDPS function, it is set to zero when the IDPS initialises.

The other role of the IDPS is detecting devices with Blacklisted MAC Addresses. As with the ARP spoofing functions, the *PacketIn* event is transferred

into the variable `packet`. The IDPS evaluates the *EtherType* field, to see if its type is IP. A JavaScript Object Notation (JSON) file containing the Blacklisted MAC Addresses is then read into the variable `blacklisted_mac_address_file`, which is compared to the the source hardware address contained in the *packet* variable. An on-screen and file log of attacks is provided by the IDPS. The Openpyxl Python library writes these into a standard `.xlsx` spreadsheet file, which is saved onto a network drive. Each row contains the *counter* variable, time at which the packet was detected by the IDPS, type of attack, source IP or MAC address of attacker and time when the flow-rule was installed.

4. Testbed and Supporting Tools

4.1. Experimental SDN Testbed

The testbed runs the Ubuntu operating system, four VMs using the VirtualBox hypervisor, OvS, and Wireshark. OvS allows the host and four VMs to communicate with one another using the OpenFlow protocol. Each VM has 4GB of memory and a 50GB dynamically allocated virtual hard disk. Within VirtualBox, *promiscuous mode* must be set to 'Allow All' to enable the host to process all packets received at the network interface card, all other options remain at their defaults. Each of the VMs is attached to a different virtual network port. A virtual network switch (OVS-BR) was created using OvS, this connects the four VMs together and allows data on the network to be captured using Wireshark. Figure 2 gives an overview of the testbed.

Every experiment utilised the OpenVSwitch testbed host together with the POX controller VM. Each experiment also employed different combinations of the other VMs - Attacker1, Attacker2, and Node. For consistency, the IDPS Attacker Tool was always configured as follows: source IP address 10.0.0.3, destination IP address 10.0.0.4, source MAC address 11:11:11:11:11:11. Table 2 indicates the testbed configuration and the software used.

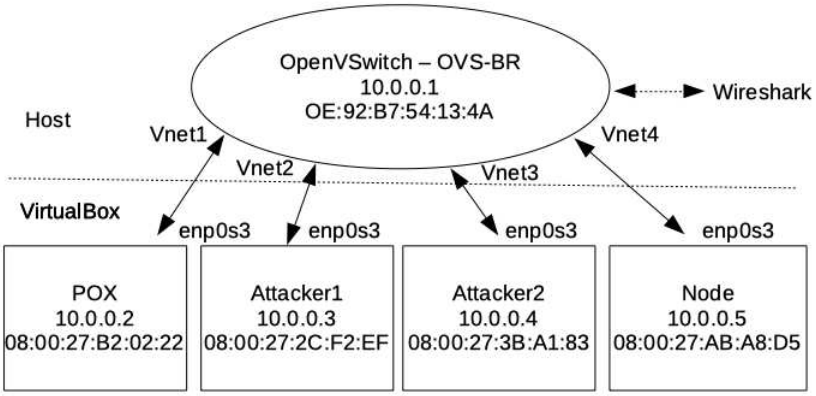

Figure 2: SDN Testbed

Table 2: Testbed Configuration and Software

Machine Name	**Network Interface**	**IP Address**	**Software**
OpenVSwitch	ovs-br	10.0.0.1	OvS, Wireshark, Virtualbox
POX	vnet1	10.0.0.2	POX
Attacker1	vnet2	10.0.0.3	Scapy, Openpyxl
Attacker2	vnet3	10.0.0.4	Scapy, Openpyxl
Node	vnet4	10.0.0.5	Scapy, Openpyxl

4.2. Sampling Points

T1 to T6 in Figure 3 are the time sampling points for a packet travelling into our IDPS. Points T1 and T2 are on the host network. The packet enters the IDS at point T3, T4 is when the IDS detects the packet, T5 is the packet entering the IPS, and T6 is when it is mitigated by the IPS. Timings within points T1 and T2 vary dependant on network traffic; so ideally they would be discounted. However, due to the testbed being on a single network segment, it is hard to detect when a packet transitions between devices. In addition,

programming limitations within the POX controller meant the only sampling points provided in practice were T4 and T6.

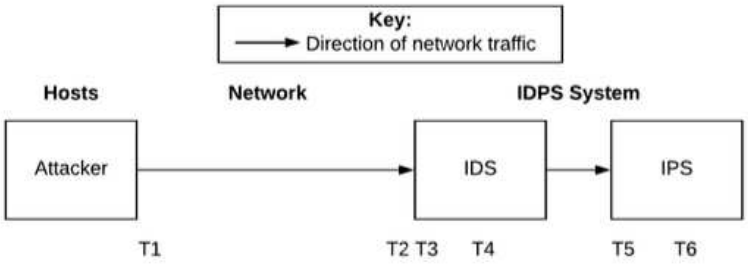

Figure 3: IDPS Network Packet Sampling Points

4.3. Software Library and Tools

Our work specifically developed a library and two software tools. Firstly, the User Input Check Library validates all user input into our software tools using Regular Expressions. Secondly, the IDPS Configuration Tool allows the IDPS flow time out value to be set, as well a list of Blacklisted MAC Addresses to be added to the IDPS. Finally, the IDPS Attacker Tool tests IDPS effectiveness by performing four separate ARP or TCP attacks.

4.3.1. IDPS Configuration Tool

Four functions are incorporated within the IDPS Configuration Tool: setting or viewing flow timeout value, adding or viewing Blacklisted MAC Addresses. The purpose of the View Flow Timeout option is to view the existing flow time out value. Measured in seconds, this value is used when a flow rule is inserted into the OvS switch and determines how long it is active. When this time has passed, the rule expires. The value is stored in a text file entitled `flow_timeout_value.txt`. By selecting the Change Flow Timeout option, the software prompts for a new flow timeout value, replacing the existing one. The value is checked by the User Input Check library, if valid it is written to `flow_timeout_value.txt`, otherwise a warning is presented. The View

Blacklisted MAC Address option is used to view the MAC addresses that are blacklisted by the IDPS and read when it is initialised. If a device appears on the network with a MAC address contained within the list, a warning is logged by the IDPS. IPv4 data from the device's IP address is prevented from entering the network. The final option - Change Blacklisted MAC Addresses - provides the ability to change the Blacklisted MAC Addresses. In this instance, the software prompts for new MAC addresses, which each being verified by the User Input Check library.

4.3.2. IDPS Attacker Tool

As part of our work, the IDPS Attacker Tool software was created to provide five distinct functions: three ARP spoofing attacks (Request, Reply, and Reply Destination), a Blacklisted MAC Address Attack and an ARP Request. These allow the operation of our IDPS to be fully tested. The ARP Request Attack function creates an ethernet packet with an ARP payload. The ARP payload contains a source MAC address and IP address as well as destination IP address. The ARP operation code is set to request. The ARP Reply Attack is identical to the Request Attack, but the ARP operation code is set to reply. The ARP Reply Destination Attack varies from the other two, in that the destination (not source) MAC address is modified. The ARP payload is set to reply, as per the previous attack. A Blacklisted MAC Address attack sends TCP packets containing random source and destination port numbers. The ARP request function creates benign ARP requests, this is used with ARP request attacks to test if the IDPS generates false positives. All of the packets sent from the IDPS Attacker Tool are individually created using the Scapy Python library [16]. Information regarding the source and destination IP address, as well as MAC address are obtained from the user and passed to Scapy. Once the required addresses have been entered by the user, each packet is simultaneously sent, displayed on screen and added to the log file in a pre-defined format. This format ensures the Attacker Tool and IDPS logs can be easily compared.

5. Experimental Results

In this section we initially explain our evaluation metrics and afterwards we document four experiments. The first experiment tests the detection system (IDS) and software tool operation utilising three different types of ARP spoofing attack. The second employs a different attack type (Blacklisted MAC Address) to further validate IDS operation. The third had the same configuration as the first, apart from employing the IDPS to certify its prevention functionality. Our final experiment simultaneously sent benign and spoofed ARP requests, simulating a real-life network environment.

5.1. Metrics

The papers discussed in Section 2 had several different ways to measure performance of their developed systems. For the purposes of our work, performance metrics used were Detection Time, Mitigation Time, False Positives, and Detection Rate. These metrics assess the effectiveness of our solution and allow comparison to other related works. *Detection Time* is defined as the time required to detect the attack, measured by the packet entering the IDS (points T1 to T4 in Fig. 3). *Mitigation Time* is defined as the time lapsed from detecting the attack to blocking the attacker (points T4 to T6 in Fig. 3). *False Positives* occur the when the IDPS erroneously detects a benign packet as being malicious, determined by subtracting the number of attacker packets sent to those detected by the IDPS. *Detection Rate* is percentage indication of IDPS performance, computed by dividing the number of packets sent by the attacker, to those detected by the IDPS then multiplying the results by 100. In the following we present our evaluation results for detection times and mitigation times in the four considered experiments. In all cases, false positives where 0, whereas the detection rates were 100%.

Other proposed IDS solutions [13, 17, 18], which employ ML-based techniques, actively create patterns from the traffic dataset and therefore can learn to differentiate attacks from normal network traffic. This necessities the utilisation of other metrics, such as: *Accuracy* a percentage of true detection over the

total traffic, *Sensitivity* the percentage of predicted attacks against all attacks, and *Precision* which is a measure of the number of attacks predicted by the IDS that actually occur. Given that we do not employ ML-based detection methods, the aforementioned metrics have not been used in our experiments.

5.2. IDS - ARP Spoofing Attack

In this experiment, we verify the operation of the IDS and the Attacker Tool, analyse attacks, and confirm virtual network functionality. To ensure reliability, our experiment was performed with three different types of ARP spoofing attack - Request, Reply, and Reply Destination. For each attack, spoofed ARP packets were sent by the IDPS Attacker Tool from the Attacker1 VM. Figure 4 is a Wireshark capture of a packet sent by the IDPS Attacker Tool. The sender MAC address in the ARP payload is different to that of the ethernet source frame.

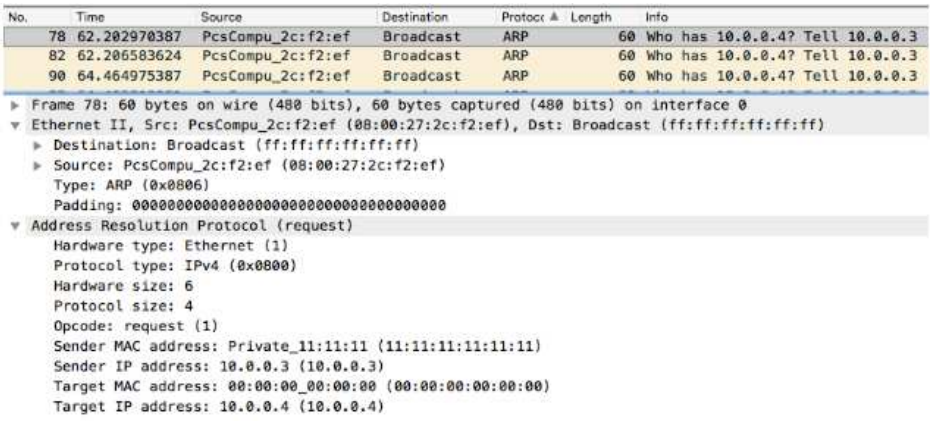

Figure 4: ARP Request Attack – Wireshark Capture

In order to check attack effectiveness, the POX ARP cache was examined by inputting the command: `arp -a`. Figure 5 depicts the output we received and confirms that the ARP cache was 'poisoned', as the network host 10.0.0.3 was mapped to the spoofed MAC address 11:11:11:11:11:11.

```
root@POX:~/pox# arp -a
usersvm0.york.ac.uk (144.32.54.4) at <incomplete> on enp0s3
? (10.0.3.3) at 52:54:00:12:35:03 [ether] on enp0s8
openvswitch (10.0.0.1) at fa:10:4f:c5:12:4f [ether] on enp0s3
5.85.222.35.bc.googleusercontent.com (35.222.85.5) at <incomplete> on enp0s3
? (10.0.0.3) at 11:11:11:11:11:11 [ether] on enp0s3
api.snapcraft.io (91.189.92.38) at <incomplete> on enp0s3
156.99.224.35.bc.googleusercontent.com (35.224.99.156) at <incomplete> on enp0s3
swhri.york.ac.uk (10.0.3.2) at 52:54:00:12:35:02 [ether] on enp0s8
root@POX:~/pox#
```

Figure 5: POX VM - 'Poisoned' ARP Cache

Broadly speaking, this experiment established that the IDS has an average detection time of around 2.2 seconds, consistent across all three types of attack. We examined the Wireshark captures during subsequent experiments, all indicated the IDPS Attacker Tool generated 'spoofed' ARP packets as intended and validated full operation of the virtual network. A check for false positives was made by comparing the Attacker and IDS Log files, none were recorded. Figure 6 and Table 3 present a summary of the results for this experiment.

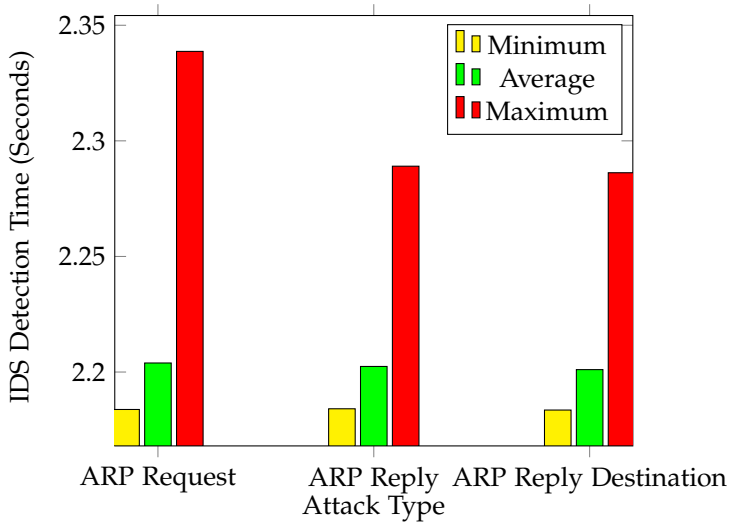

Figure 6: IDS Detection Time - ARP Spoofing Attacks

Table 3: IDS - ARP Spoofing Attacks - Detection Time (Seconds)

ARP Attack	Avg.	Min.	Max.
Request	02.203914	02.183804	02.338696
Reply	02.202416	02.184107	02.289039
ReplyDestination	02.201044	02.183559	02.286205

An anomaly was observed in this experiment, whereby the maximum (Max.) detection time is not proportional to the average (Avg.) or minimum (Min.). Figures 7 to 9 provide a breakdown of the IDS response time for each individual packet. It can be seen that for every attack there are two packets that have greater response times than the others. This results in the maximum detection time being significantly higher than the average or minimum. Nevertheless, these anomalies do no seem to have a significant impact on the overall system performance and attack detection efficiency.

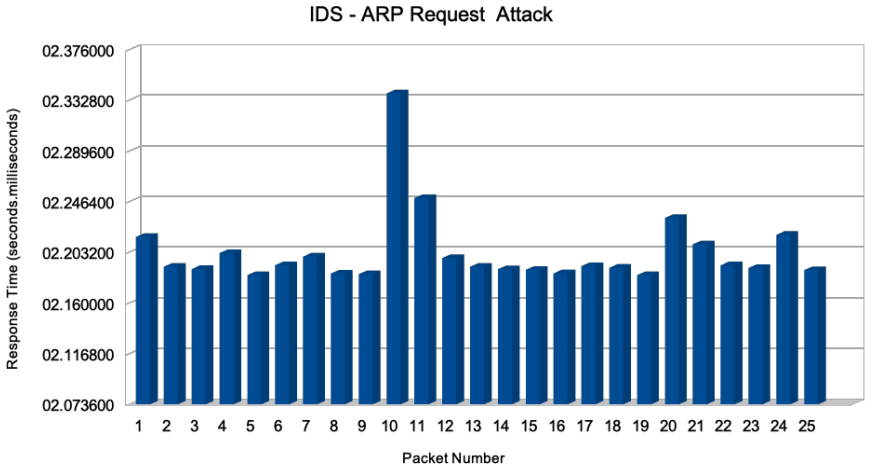

Figure 7: ARP Request Attack - Individual Packets

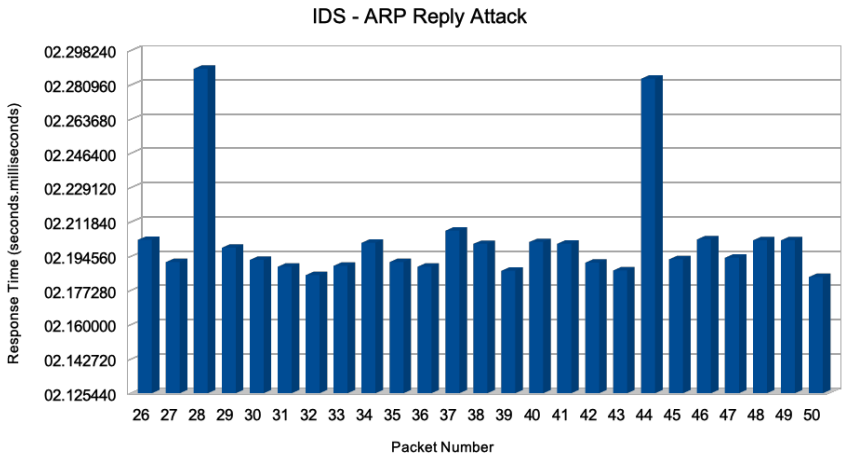

Figure 8: ARP Reply Attack - Individual Packets

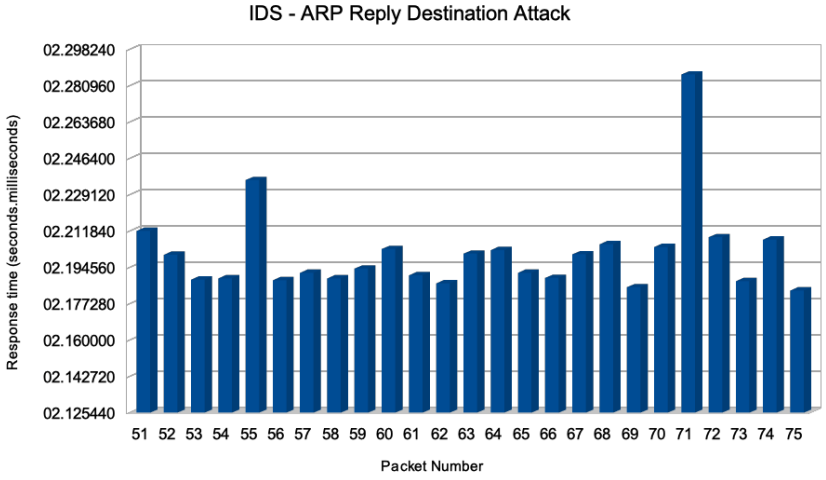

Figure 9: ARP Reply Destination Attack - Individual Packets

5.3. *IDS – Blacklisted MAC Address*

Our second experiment is built on the first, by employing a new type of attack from a different host. In order to test IDS operation, the Attacker2 VM was employed and its `enp0s3` interface was entered as a Blacklisted MAC

using the IDPS Configuration Tool. Using Wireshark, a repeating pattern of `OFPT_Packet_In` and `OFPT_Packet_Out` packets were observed during the attack, as illustrated in Figure 10. This is due to Attacker2 sending packets to the OvS switch, which encapsulates them with the OpenFlow protocol, these are then directed to POX as an `OFPT_Packet_In` packet. Inspection of the `OFPT_Packet_In` encapsulation includes an explanation for the packet being sent "Reason: No matching flow (table-miss flow entry)". The source and destination ports are randomly generated in each packet sent by Attacker2, which means POX must create a new flow-entry, as none exist on the OvS switch. To create the flow-entry, POX returns the packet header to the switch as an `OFPT_Packet_Out` packet, containing an `OPFT_Flow_Mod` packet, which allows the packet to reach its destination. This process is depicted by Figure 11, which shows a `OFPT_Packet_Out` packet, with randomly generated TCP source and destination ports.

No.	Time	Source	Destination	Protocol	Length	Info
99	43.144262270	10.0.0.4	10.0.0.2	OpenF…	144	Type: OFPT_PACKET_IN
100	43.144425376	10.0.0.2	10.0.0.1	TCP	66	6633 → 49210 [ACK] Seq=1333 Ack=2489
101	43.256174627	10.0.0.4	10.0.0.2	OpenF…	238	Type: OFPT_PACKET_OUT
102	43.256333088	10.0.0.1	10.0.0.2	OpenF…	74	Type: OFPT_BARRIER_REPLY
103	43.256429381	10.0.0.2	10.0.0.1	TCP	66	6633 → 49210 [ACK] Seq=1505 Ack=2497
104	44.233745040	10.0.0.4	10.0.0.2	OpenF…	144	Type: OFPT_PACKET_IN
105	44.233898557	10.0.0.2	10.0.0.1	TCP	66	6633 → 49210 [ACK] Seq=1505 Ack=2575
106	44.357656505	10.0.0.4	10.0.0.2	OpenF…	238	Type: OFPT_PACKET_OUT
107	44.357822491	10.0.0.1	10.0.0.2	OpenF…	74	Type: OFPT_BARRIER_REPLY
108	44.357903807	10.0.0.2	10.0.0.1	TCP	66	6633 → 49210 [ACK] Seq=1677 Ack=2583
109	45.306322932	10.0.0.4	10.0.0.2	OpenF…	144	Type: OFPT_PACKET_IN
110	45.306462310	10.0.0.2	10.0.0.1	TCP	66	6633 → 49210 [ACK] Seq=1677 Ack=2661
111	45.400576641	10.0.0.4	10.0.0.2	OpenF…	238	Type: OFPT_PACKET_OUT
112	45.400713089	10.0.0.1	10.0.0.2	OpenF…	74	Type: OFPT_BARRIER_REPLY
113	45.400785675	10.0.0.2	10.0.0.1	TCP	66	6633 → 49210 [ACK] Seq=1849 Ack=2669
114	46.379188062	10.0.0.4	10.0.0.2	OpenF…	144	Type: OFPT_PACKET_IN
115	46.379380499	10.0.0.2	10.0.0.1	TCP	66	6633 → 49210 [ACK] Seq=1849 Ack=2747
116	46.649623693	10.0.0.4	10.0.0.2	OpenF…	238	Type: OFPT_PACKET_OUT

Figure 10: OpenFlow Packets

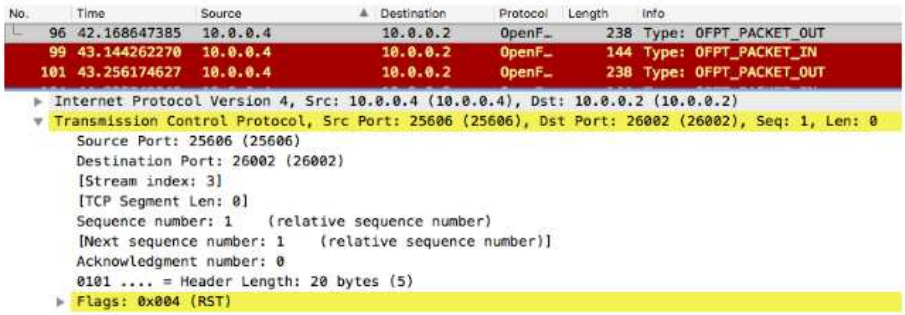

Figure 11: TCP Packet Format

The IDS was able to detect packets despite a delay being added to them by the insertion of flow rules. The delay could be observed as a potential flaw with OpenFlow-based SDN. This type of network traffic places a load on the controller, which may be unable to cope if the volume is high. The situation could be exacerbated if the SDN controller and switch are physically separate or have a high round-trip-time between themselves. The experiment confirmed OpenFlow operation on the SDN network, including insertion of flow rules, which are employed in our last experiment to prevent malicious network traffic. Our obtained detection times (in seconds) are: Avg.= 00.029930, Min.=00.068751, and Max.=00.656727. These times are significantly lower than in our first experiment. This is because the Scapy *send* command, used by the IDPS Attacker Tool, transmits all packets at Layer 3, the same layer as TCP, whereas ARP operates primarily on Layer 2. Therefore, the overhead involved in translating packets between network layers may account for the higher response time for ARP packets.

As with our first experiment, no false positives were recorded by the log files. In addition, as indicated by Figure 12, one packet has a significantly higher response time than the others, again resulting in the overall results being biased. We note that the anomaly occurs for both TCP and ARP packets and speculate the causes as being load on the POX controller or malformed

packets sent by the Scapy Python library.

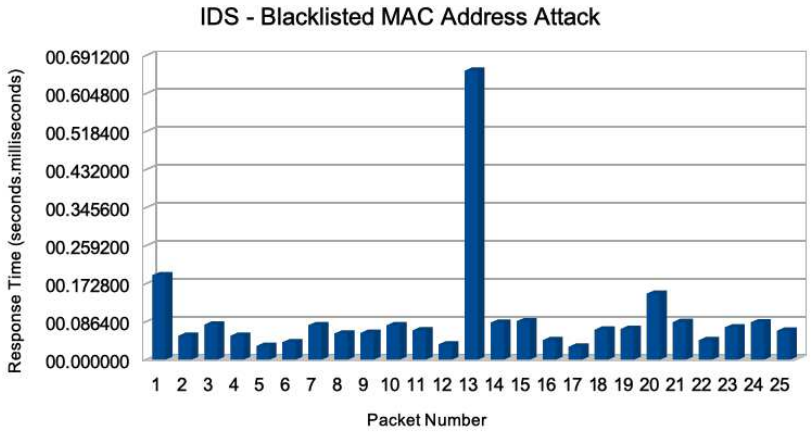

Figure 12: Blacklisted MAC Address Attack - Individual Packets

5.4. IDPS – ARP Spoofing Attacks

Previous experiments only involved intrusion detection, with no countermeasures against attacks. This experiment used an IDPS to drop traffic from a malicious host once detected. Three different types of ARP attack were used - Request, Reply, and Reply Destination. A review of the packets captured during the attack indicated the Sender MAC Address (11:11:11:11:11:11) in the ARP payload being different to the ethernet source (08:00:27:2C:F2:EF) in the ethernet frame. The IDPS successfully detected the attack and installed a flow rule preventing network traffic from the attacker. To check flow rule insertion, `ovs-ofctl dump-flows ovs-br` was input on the OpenVSwitch host, which returned: `cookie=0x0, duration=0.700s, table=0, n_packets=0, hard_timeout=0, priority=1, ip, nw_src=10.0.0.3, nw_dst=10.0.0.4, actions=drop`. This confirmed that a flow rule had been generated to block IP network traffic from the attacker. Further validation was carried out by sending a ping from the OvS host machine. This resulted in 'Destination Host Unreachable', proving IP traffic cannot reach the attacker's IP address. Prior to the attack, this ping was successful.

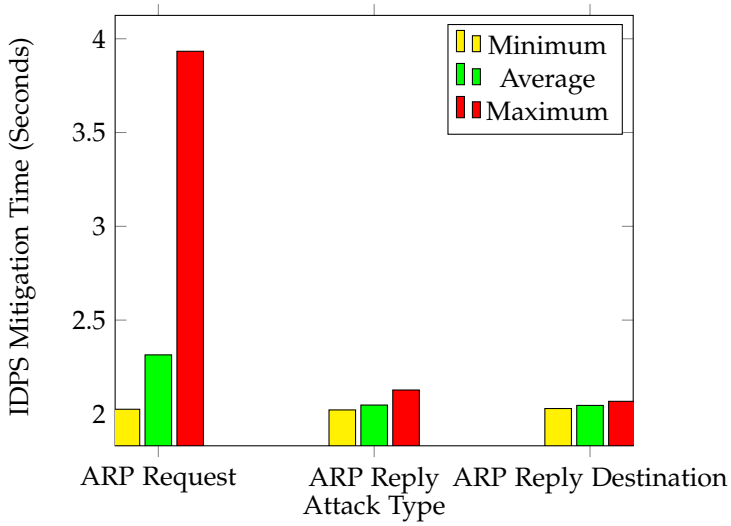

Figure 13: IDPS Mitigation Time - ARP Attacks

Table 4: IDPS - ARP Spoofing Attacks - Mitigation Time (Seconds)

ARP Attack	**Avg.**	**Min.**	**Max.**
Request	02.314543	02.024589	03.933365
Reply	02.046977	02.021009	02.127211
ReplyDestination	02.045087	02.028510	02.066943

Experiment 3 results indicate that the mitigation time was not significantly greater than in our previous experiments. General IDPS response times were similar to that of the IDS in our first experiment, with an average of 2.2 seconds across all types of ARP spoofing attack. This indicates that the intrusion prevention functionality did not add a significant overhead to our system. Figure 13 and Table 4 are summaries of our results for this experiment.

5.5. IDPS - Spoofed & Standard ARP Attacks

Our final experiment consists of 25 standard and spoofed ARP packets being simultaneously sent by the Node and Attacker1 VMs, respectively. Unlike previous experiments, a two second gap between each packet transmission

was introduced to simulate the usual network traffic an IDPS would operate in. The IDPS was successfully able to differentiate between the spoofed and normal ARP requests. Examination of the Node and IDPS Attacker Tool Log indicates that all of the ARP requests sent received a valid response. This is time taken by the IPS to mitigate the attack once it is detected by the IDS. Our measured additional response time is: Avg.= 02.01465, Min. =02.00890, and Max. = 02.03694. This difference was noted to be very small, proving intrusion prevention functionality did not add a significant overhead to the IDS. We observe that IDPS mitigation times are roughly equivalent to the previous experiment. As with previous experiments, a number of packets with higher than normal response times were detected, leading to a higher maximum packet response. The additional time taken by the intrusion prevention system to mitigate the attack is: Avg. = 0.001092, Min. = 0.000674 , and Max. = 0.002004. It was noted not to be significant.

6. Conclusion

This work has emphasised the flexible nature of SDN and its ability for customisation to meet user requirements. We have shown that SDN can satisfactorily detect as well as protect against network intrusions. More precisely, our IDPS was successful in defending a network against ARP spoofing and Blacklisted MAC Address attacks. We were able to generate log files on the console and write these to spreadsheet files. Testing was carried out by using a variety of hostile and benign network traffic from different hosts. All parts of the SDN testbed, including the specifically designed software were verified as fully operational.

Limitations of this work include the virtual nature of the network testbed and POX controller programming constraints which resulted in restricted experimental sampling points. To mitigate these limitations, the testbed would be based upon a physical network, with hardware SDN switches and multiple hosts running a variety of SDN-enabled applications. This would give a

better indication of IDPS performance under real-life operating conditions and allow additional detailed test data to be produced. Finally, by integrating Machine Learning into the IDPS, it could be trained to accurately predict attacks, which would facilitate a wider range of metrics being employed within the experiments.

References

[1] D. Kreutz, et al., "Software-defined networking: A comprehensive survey", Proc. of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015, doi: 10.1109/J-PROC.2014.2371999

[2] J. Ma, "Resource management framework for virtual data center embedding based on software defined networking", Computers & Electrical Engineering, vol. 60, pp. 76-89, May 2017, doi: 10.1016/j.compeleceng.2016.09.002

[3] W. Chen, S. Xiao, L. Liu, X. Jiang, and Z. Tang, "A DDoS attacks traceback scheme for SDN-based smart city", Computers & Electrical Engineering, vol. 81, pp. 1-12, Jan. 2020, doi: 10.1016/j.compeleceng.2019.106503

[4] A. Bhutani and P. Wadhwani, "Global SDN Market Size to Exceed 100bn by 2025", March 2019. [Online]. Available: https://www.gminsights.com/pressrelease/software-defined-networking-sdn-market. [Accessed 10 Jan. 2021].

[5] H. Khalid, P. Ismael, and A. B. Al-Khali, "Efficient mechanism for securing software defined network against ARP spoofing attack", Journal of Duhok University, vol. 22, no. 1, July 2019, doi: 10.26682/sjuod.2019.22.1.14

[6] J. H. Cox, R. J. Clark, and L. H. Owen, "Leveraging SDN for ARP security", IEEE SoutheastCon, Norfolk, USA, April 2016, doi: 10.1109/SECON.2016.7506644

[7] T. Alharbi and M. Portmann, "Securing ARP in software defined networks", IEEE 41st Conference on Local Computer Networks (LCN), Dubai, UAE, Dec. 2016, doi: 10.1109/LCN.2016.83

[8] J. McCauley, "Installing POX". [Online]. Available: https://noxrepo.github.io/pox-doc/html/. [Accessed 20 Oct. 2020].

[9] Open Networking Foundation, "Open Flow Switch Specification", April 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf [Accessed 20 Oct. 2020].

[10] Open vSwitch, "Production Quality, Multilayer Open Virtual Switch". [Online]. Available: http://www.openvswitch.org/features. [Accessed 20 Oct. 2020].

[11] C. Birkinshaw, E. Rouka, and V. G. Vassilakis, "Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks", Journal of Network and Computer Applications, vol. 136, pp. 71-85, June 2019, doi: 10.1016/j.jnca.2019.03.005

[12] A. Leal, J. F. Botero, and E. Jacob, "Improving early attack detection in networks with sFlow and SDN", 5th Workshop on Engineering Applications (WEA), Medellín, Colombia, pp. 323-335, Oct. 2018, doi: 10.1007/978-3-030-00353-1_29

[13] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks", 7th Int. Conf. on Emerging Security Technologies (EST), Canterbury, UK, Sept. 2017, pp. 138-143, doi: 10.1109/EST.2017.8090413

[14] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set", IEEE 2nd Symposium on Computational Intel-

ligence for Security and Defense Applications (CISDA), Ottawa, Canada, July 2009, pp. 1-6, doi: 10.1109/CISDA.2009.5356528

[15] Z. Shah and S. Cosgrove, "Mitigating ARP cache poisoning attack in software-defined networking (SDN): A Survey", MDPI Electronics, vol. 8, no. 10, pp. 1-26, Sept. 2019, doi: 10.3390/electronics8101095

[16] P. Biondi, "Scapy - Packet crafting for Python2 and Python3". [Online]. Available: https://scapy.net/ [Accessed 20 Oct. 2020].

[17] M. S. Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcu, "Machine-learning techniques for detecting attacks in SDN," IEEE 7th Int. Conf. on Computer Science and Network Technology, Dalian, China, Jan. 2020, pp. 277-281, doi: 10.1109/ICCSNT47585.2019.8962519

[18] L. N. Tidjon, M. Frappier, and A. Mammar, "Intrusion detection systems: A cross-domain overview", IEEE Communications Surveys & Tutorials, vol. 21, no. 4, pp. 3639-3681, June 2019, doi: 10.1109/COMST.2019.2922584

Thomas Girdler obtained a BSc (Hons) in Computer and Network Engineering from Sheffield Hallam University in 2003 and an MSc in Cyber Security from the University of York in 2019. His research focuses on using Software-Defined Networking to develop Intrusion Detection and Prevention Systems.

Vassilios G. Vassilakis received his Ph.D. degree in Electrical & Computer Engineering from the University of Patras, Greece in 2011. He is currently a Lecturer (Assistant Professor) in Cyber Security at the University of York, UK. He's been involved in government and industry funded R&D projects related to the design and analysis of future mobile networks and Internet technologies. His main research interests are in the areas of network security, Internet of things, next-generation wireless and mobile networks, and software-defined networks.