Formal Modelling and Analysis of Man in The Middle Prevention Control in Software Defined Network

Daniel Aviz Bastos
IPT Brasil
daniel.aviz@gmail.com

Adilson Eduardo Guelfi
IP/UNOESTE
guelfi@unoeste.br

Anderson Aparecido Alves da Silva
IPT/UNIP/SENAC/USP
anderson.silva@pad.lsi.usp.br

Marcelo Teixeira de Azevedo
Universidade de São Paulo - USP
mdeazevedo@gmail.com

Alberico de Castro Barros Filho
Universidade de São Paulo - USP
alberico.castro@usp.br

Sergio Takeo Kofuji Correio
Universidade de São Paulo - USP
kofuji@usp.br

Abstract. *The Man-in-the-Middle (MITM) attack is still considered a threat to networks. On the other hand, Software-Defined Networks (SDN) has been gaining market space, precisely because of the ability to dynamically manipulate data flows through a programmable interface. Colored Petri Nets (CPN) are considered a proper tool for modeling distributed systems and security controls. The objective of this work is, using CPN, model and analysis a security control in SDN that could detect and correct the Man-In-The-Middle (MITM) attacks based on ARP spoofing. This goal is achieved modeling a SDN, a MITM attack and the proposed security control to reach a complete state space analysis. The state space analysis showed that a MITM attack is possible in a SDN environment increasing the number of possible states in 67%. The proposed security control could detect and correct the packets generated with forged data, preventing the attack, and decreasing by 96.22% the number of possible states while the SDN is under attack. Thus, this work obtained an effective security control to avoid MITM attacks based on ARP spoofing, did not falling on the main limitations presented in the previous solutions to avoid MITM attacks in traditional networks, such as: the existence*

of single point of failures or the need to update or patch the ARP protocol or system's kernel at the devices connected to the network.

Keywords: *ARP Poisoning, Colored Petri Nets (CPN), Man-in-the-Middle (MITM), Software Defined Networks (SDN).*

Resumo. *O ataque Man-in-the-Middle (MITM) ainda é considerado uma ameaça para as redes. Por outro lado, as redes definidas por software (SDN) vêm ganhando espaço no mercado, precisamente devido à capacidade de manipular dinamicamente os fluxos de dados por meio de uma interface programável. As redes de Petri coloridas (CPN) são consideradas uma ferramenta adequada para modelar sistemas distribuídos e controles de segurança. O objetivo deste trabalho é, usando o CPN, modelar e analisar um controle de segurança na SDN que possa detectar e corrigir os ataques Man-In-The-Middle (MITM) com base na falsificação do ARP. Esse objetivo é alcançado modelando uma SDN, um ataque MITM e o controle de segurança proposto para alcançar uma análise completa do espaço de estado. A análise do espaço de estados mostrou que é possível um ataque MITM em um ambiente SDN, aumentando o número de estados possíveis em 67%. O controle de segurança proposto pode detectar e corrigir os pacotes gerados com dados forjados, impedindo o ataque e diminuindo em 96,22% o número de estados possíveis enquanto a SDN encontra-se sob ataque. Assim, este trabalho obteve um controle de segurança eficaz para evitar ataques MITM baseados na falsificação de ARP, não se limitando aos principais problemas apresentadas nas soluções anteriores para evitar ataques MITM em redes tradicionais, tais como: a existência de um ponto único de falha ou a necessidade de atualizar ou corrigir o protocolo ARP ou o kernel do sistema nos dispositivos conectados à rede.*

Palavras-chave: Envenenamento ARP, Rede de Petri Colorida (RPC), *Man-in-the-Middle* (MITM), Redes Definida por *Software*.

1. INTRODUCTION

The Man-in-the-Middle (MITM) attack is considered a threat to the traditional networks until now. The attacker poisons the Address Resolution Protocol (ARP) cache table of their victims, so that all packets sent by the victims is forwarded to the attacker machine, by the network switch, which delivers the packets based only at the destination Media Access Control (MAC) address (KUMAR; TAPASWI, 2012).

Many solutions have been proposed to prevent MITM attacks, but none of them is considered a versatile solution, because they are only effective in some special scenarios (TRIPATHI; MEHTRE, 2014). For example, the proposed solutions based on cryptography, like Secure ARP and Ticket based ARP (BRUSCHI; ORNAGHI; ROSTI, 2003) and (LOOTAH; ENCK; MCDANIEL, 2007), does not have backward compatibility with any existing network infrastructure due the need to implement an updated version of the ARP protocol. Other solutions were based on:

- Patch the host kernel, like Antidote (TETERIN, 2003);

- Passive detection, like ARPWATCH (LERES, 2006);
- Centralized server, like ARP Central Server – ACS (KUMAR; TAPASWI, 2012);
- Probe packets, as proposed by Poonam Padey (PANDEY, 2013);
- Discrete event systems, like DES (BARBHUIYA et al, 2011).

Those solutions involve a probing process that could be spoofed by the attacker too, and the attacker can register all Internet Protocol (IP) addresses to himself, before the legitimate machine arrives at the network, which can make the attack effective to the environment. Other solutions listed by Tripathi and Mehtre (2014) are based on an ICMP secondary cache, as proposed by Tripathi and Mehtre (2014) and Arote and Arya (2015), which also involve a probing process that could be spoofed by the attacker, and do not allow more than one mapping per MAC address, which makes it incompatible with virtual environments.

Even a new solution proposed by Cox et al (2016), that uses an application installed at a Software Defined Network (SDN) Controller to avoid the attacker to poison the victims ARP cache table, could not be considered a versatile solution. As it uses the Dynamic Host Configuration Protocol (DHCP) server database as a trusted source for IP-MAC bindings, attacker could poison it, as it is based on the DHCP messages, which are not connection oriented and could be forged by the attacker. Another issue about that solution is that it includes an additional delay at the network, as all packets needs to be forwarded by the switch to SDN controller, to be analyzed by the proposed application, forwarded again to the switch, and finally forwarded to the packet destination.

SDN is a new network architecture designed to simplify and improve network management with high flexibility (ANAN et al, 2016). It is done by decoupling the architecture in tree layers. Firstly, the control plane, or control layer, is the system that makes decisions about where the traffic should be sent. Secondly, the data plane, or infrastructure layer, is the system that forwards the packets to its destination. The third layer called application layer, which has several SDN applications that employ an open Application Programming Interface (API) to communicate with the SDN controller.

The communication between the control layer and data layer is done with a standard protocol, called Openflow, which creates data flow rules that are installed at the switches, to forward the packets as defined by the SDN controller. Each flow rule has some packets characteristics (any field of the packet header), and one or more actions that should be applied to the packet which matches the rule. These actions can forward the packet to a specific port or change the value of any packet header field. The SDN architecture can enhance the network security enabling elevated levels of monitoring, analysis and response. The logical centralization of the control layer enables the creation of innovative security services and applications that could increase the network security level at all (MASOUDI; GHAFFARI, 2016).

The Colored Petri Nets (CPN) is a formal method for modeling, designing, specification, simulation, validation and verification of systems or process. A CPN model is made of places (circles), transitions (quadrilaterals), arcs (unidirectional or

bidirectional arrows between places and transitions) and marks that are data stored at places or being processed by transitions. Color sets (colset) are data types like integer, string, boolean, etc. Each combination of marks and places is called a state. One of the methods to analyze the CPN model is called state space analysis, and it is done with the creation of a graph, in which the nodes represents each state of the CPN model, and the arcs, between nodes, represents the changes in the marks between each state of the CPN model. After the graph is created, it is possible to analyze the properties of the CPN model and make queries to find specific states that satisfy some conditions (JENSEN; KRISTENSEN; WELLS, 2007).

The objective of this work is, using CPN, model and analysis a security control in SDN that could detect and correct the MITM attacks based on ARP spoofing. This security control acts at the Openflow switch to detect and correct the attack, while the packets are passing through the network, using data collected by an application installed in the SDN controller.

This paper is organized as follow: section 2 summarizes some theoretical references; section 3 discusses related works; section 4 presents the proposed solution and describes the CPN model; section 5 describes the initial state space analysis in some scenarios that have been used to confirm the proposed control works, and after some important results; and finally, section 6 shows our final conclusions and future works.

2. THEORETICAL REFERENCE

2.1. MAIN IN THE MIDDLE ATTACK (MITM)

The MITM attack aims to intercept the communications between two or more machines, without the victim awareness. The attacker can read, change or replace the traffic to attend its proposal. It is done using impersonation techniques, like ARP spoofing, DHCP spoofing, Domain Name System (DNS) spoofing, Border Gateway Protocol (BGP) spoofing and others.

Network devices use ARP protocol to create bindings between MAC and IP addresses at the network. This protocol is vital to any network, but it was not designed to authenticate the messages and to deal with malicious hosts. The attacker creates a MITM attack, using ARP spoofing, adding or updating the victim local ARP cache table with forged data. It sends packets with forged ARP reply messages, which creates forged bindings between the attacker MAC address and the communication destination IP address. After that, every time that the victim machine creates a packet to destination, it writes the correct destination IP address, but the attacker MAC address at the packet header. As the network devices deliver the packets based on its destination MAC address, the packet goes to the attacker machine that manipulates the data, and then forwards the packet to its correct destination. If the attacker repeats the MITM attack on both sides of the communication (source and destination), the attacker can intercept the communication in both directions.

2.2. SOFTWARE DEFINED NETWORK (SDN)

The computer networks are composed of two main layers, the control layer and the data layer. The control layer is responsible to define the best path to the packets

travel from one point to another in the network. This layer provides services like routing, switching, trunking and spanning tree, for example. The data layer is responsible to buffer the data when it arrives and delivers it to the correct device port, as the control layer defined.

In traditional networks, every network device implements both layers locally, and each manufacturer implements them in their own way, in order to provide the network services and the integration between each layer in a closed way. This architecture disallows other systems to interact with any of these network layers. The network devices made by different manufacturers cannot interact between them or even share information about the network.

The SDN split the two layers in distinct locations at the network. The control layer is centralized at one central component called SDN controller, and the data layer (also called infrastructure layer) is distributed across the network in every network device. An integration protocol was created to connect the two layers that are separated physically. Two protocols were proposed: The Internet Engineering Task Force (FORCE, 2014) proposed the Forwarding and Control Element Separation (ForCES); and the Open Networking Foundation proposed the Openflow (FOUNDATION, 2015). The powerful support by the industry and the academy makes the Openflow the standard protocol to SDN.

As depicted in Figure 1, the SDN controller (control layer) is responsible to provide all network services, like routing, switching etc. It has two interfaces, called northbound and southbound. The northbound interface allows the SDN controller to interact with any business application using APIs, allowing applications to interact and access information about both control and data layers. The southbound interface allows the SDN controller (control layer) to interact with all infrastructure layer, receiving and sending information about the network traffic to all network devices, as defined by the network services running inside the SDN controller. This communication uses the Openflow protocol to standardize the information exchange between the SDN controller and the infrastructure layer.

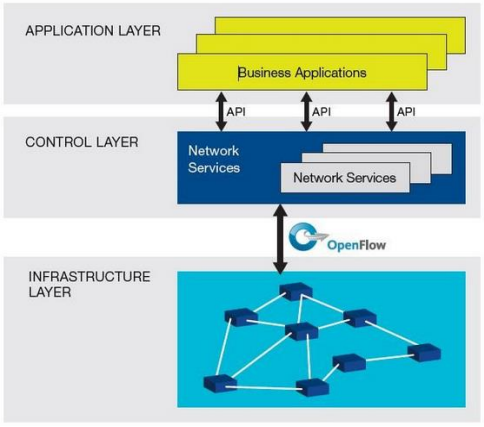

Figure 1. Openflow Architecture.
Source: Open Networking Foundation (FOUNDATION, 2015)

The control layer translates its decisions to some data flow rules, which are composed by these main fields: match fields, priority, counters, instructions, timeouts,

cookie and flags. The match fields specify which packet header fields and values should be used to choose which flow rule should be applied to a specific packet. The priority field defines the rule position inside the switch flow table. The counters field is used to count how many packets used a specific flow rule. The instructions field defines which actions should be done with the packet that matches to a specific rule. The timeouts field defines the amount of time before a flow rule is expired, and the flags field determines what should be done with the flow rule (insert, update, delete etc.). The SDN controller organizes all rules managed by it using the cookie field. These rules are sent to the infrastructure layer, which will apply them to the network traffic.

Openflow switches compose the data or infrastructure layer. This switch has only one interface, which is used to communicate with the SDN controller. Each Openflow switch has one or more flow tables that store all data flow rules received from the SDN controller. When a packet arrives, it is stored in a buffer area, and then its header is collected and compared to every data flow rule inside of the flow table, from the first until the last rule. When packet header fields match with the rule match fields, the switch stops the search and applies all actions specified by the rule instructions field. These actions include, but it is not limited to, change some header field values or delivery that packet to a specified device port. If no rules match to the packet header data, the switch can be configured to discard the packet or send the entire packet to the SDN controller, which will exanimate the packet to decide what actions should be applied. After that, the SDN controller could create a new flow rule that is distributed to one or more Openflow switches. In this case, the next time that a similar packet arrives at the network, the Openflow switch will take the actions defined without sending the packet to the SDN controller again.

3. RELATED WORKS

Tripathi and Mehtre (2014) considered five distinct factors to compare previously proposed solutions to ARP spoofing in traditional networks. The first factor defines that the proposed solution should be resistant to flood of spoofed ARP messages. The second factor defines that the proposed solution should not allow the attacker to register all unused IP address before the legitimate client ingress the network. The third factor says that the proposed solution should be compatible with the existing network infrastructure. The fourth factor identifies if the proposed solution cannot be considered a single point of failure. Finally, the fifth factor defines that the proposed solution should allow more than one IP address to each MAC address.

After defining those factors, the authors conducted a review on methods of control used to detect and prevent ARP poisoning attacks and met these methods in seven groups. This first group has solutions that use encryption to protect the ARP messages exchanged between the network clients. The second group lists the proposed solutions that patch the system kernel, so the machine can identify when it receives an ARP message with forged data. The third group has controls that use a passive detection scheme to identify when the attacker starts spoofing the packets and alert all legitimate network clients. The fourth group lists solutions that use centralized detection and validation servers, which are responsible to check if the clients can trust in the ARP messages that it receives. The fifth group has solutions that use ICMP probe packets to validate the ARP messages. The sixth group lists solutions that use host-based discrete events to observe events generated by the system and decide if the system is under a

normal or failure condition. Finally, the seventh group has solutions that use a secondary ARP cache that is updated using ICMP messages.

At the end of the work, Tripathi and Mehtre (2014) analyses every group of proposed solutions against the five factors considered at the beginning. The authors concluded that none of the solutions analyzed attend to all the factors, and cannot be considered a versatile control. The main advantage of this work was list the main MITM proposed controls. However, they analyzed them only at theoretical level, and did not test or model them.

Cox et al (2016) also proposed a control to detect and eliminate ARP spoofed packets in a SDN environment. The work achieves this goal creating an application, called Network Flow Guard for ARP (NFGA), that should be installed in the SDN controller, which inspect every DHCP and ARP packets to create a centralized ARP table, with information about IP-MAC bindings. When a host is available to the network, the NFGA only allows communication between the machine and the DHCP server. After analyzing the initial DHCP messages and update the centralized ARP table, the machine can communicate with any other machine in the network. Then, the NFGA inspects every ARP packet, discarding packets with forged data, and sending packets with correct information.

The proposed control in Cox et al (2016) is effective to prevent ARP spoofing and MITM attacks, as consequence. However, as it uses a centralized application and table to check if the packets have forged data, the NFGA can be considered a single point of failure, and increases the network latency, as every DHCP and ARP packet should be sent by the Openflow switch to the SDN controller to be analyzed and delivered back to its destination.

Sasan and Salehi (2017) proposed the creation of an application, written using Python language, called arppois, which should be installed at the SDN controller. The authors assumed they have a trusted database of IP-MAC addresses, which will be used by arppois to verify the payload of all ARP Reply messages and check if the IP-MAC binding is correct. If the binding does not match with the trusted database, the packet is discarded. As the Openflow switch cannot check information inside the packet payload, all ARP Reply messages should be forwarded by the switch to the SDN controller, where the arppois can analyze the packet payload and decide what should be done with the packet. They tested the proposal using a laboratory environment created using the MININET software to simulate the infrastructure layer, and POX software as the SDN controller.

The authors concluded that the proposed solution was efficient to prevent the ARP poisoning of the victims and the MITM attack as consequence. A second conclusion was that the SDN environment can be used to detect and prevent security attacks. However, the authors did not propose how the trusted IP-MAC bindings can be obtained, and only tested the proposed control in a specific laboratory environment. Another problem is the fact that every ARP Reply message should be forwarded to the SDN controller to be analyzed and then delivered back to the network, also increasing the network latency.

Rojas et al (2014) aims to verify if the policies applied at the SDN controller are imposed by the Openflow switches. The authors model a SDN environment with three servers (laboratory, academic and SDN controller) and two workstations (one of

students and one for teacher), using CPN. At this CPN model was created three policies, as follow:

- The student's workstation can access the laboratory server;
- The teacher's workstation can access the laboratory server;
- The teacher's workstation can access the academic server.

The authors used a tool called CPNTools to model the environment and the flow rules mentioned above. After that, the authors used a code, written in Meta Language (ML), to verify if there is at least one state at the CPN model which breaks the flow rules created. Finally, the authors concluded that the SDN controller correctly enforces the policies created in all Openflow switches, and that the CPN is a suitable tool to validate security rules in a SDN environment. However, they do not model any security attack or control, just the normal SDN operation.

The proposed solutions, at traditional networks, analyzed by Tripathi and Mehtre (2014) were not considered effective if compared to the proposed factors. The Rojas et al (2014) research shows that CPN is a valuable tool to validate security policies applied to a SDN environment. However, the solutions proposed by Cox et al (2016) and Sasan and Salehi (2017), based on SDN environment, have been proved to be effective to prevent the attack, but it increases the network latency and can be considered a single point of failure, which is one of the factors analyzed by Tripathi and Mehtre (2014). It should be very valuable a solution that can be considered effective by the Tripathi and Mehtre (2014) criteria, using a SDN environment that cannot be considered a single point of failure, and the CPN can be used to validate if the proposed control is effective in all possible states.

4. MITM PREVENTION MODULE

This paper proposes a security control to a SDN environment, which is applied at every Openflow switch to detect any packet with forged data, and correct the MAC address as the packet flow in the network. As depicted in Figure 2, paper achieves its goal assuming the trusted information about the IP-MAC bindings will be collected by an external application, as an asset management system, for example. This information should be securely delivered to an application, installed in the SDN controller, called MITM Prevention Module, which writes two flow rules per IP address, designed to correct the data at the packet header, in all Openflow switches.

The MITM Prevention Module receives trusted IPMAC address bindings from an external application, and information about the machines connected to the network (connected port, IP and MAC addresses) from an internal module, called Topology Manager. With this two information, the MITM Prevention Module has all information needed to create the two proposed flow rules.

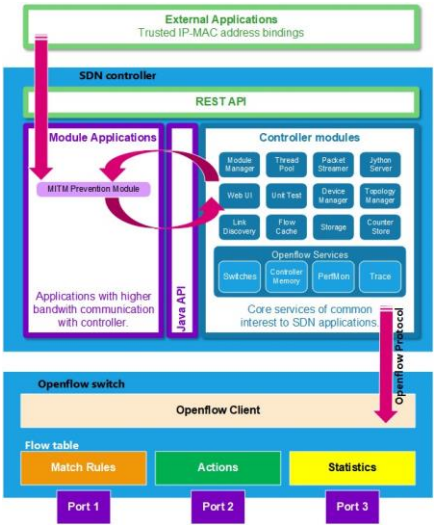

Figure 2. Proposed Architecture
Source: Adapted from Project Floodlight (FLOODLIGHT, 2018)

The first flow rule matches packets that have the correct binding of IP and MAC addresses and delivery them to the correct destination port. The second flow rule matches packets that have incorrect IPMAC bindings, updating them with the correct MAC addresses, and only after that, the packets are then delivered to the correct destination port. In that way, the first rule delivers the normal packets, and the second rule correct and delivers the packets with forged data, preventing the MITM attack to have success in intercepting packets. These rules are sent to the Flow Cache module that distributes them to all Openflow switches that will deal with the traffic for one specific machine.

The proposed solution is validated by the formal modelling of the flow rules inserted in a SDN environment composed by a client, a server and an attacker machine. In that SDN environment, the normal operation and the MITM attack, with and without the security control, were simulated.

4.1. MODEL DESCRIPTION

This subsection presents the hierarchical CPN model that is used to analyze the proposed security control. This CPN model has three levels: a top-level, which represents the network topology; the second level, which details the client, the server, the attacker machine and the switch; and the third level, which represents the details of some switch components. The CPN model color set is presented at appendix A.

4.2. TOP-LEVEL CPN MODEL DESCRIPTION

As depicted at Figure 3, this first CPN model represents a SDN topology, composed by four components: a client, a server, an attacker machine and one Openflow switch.

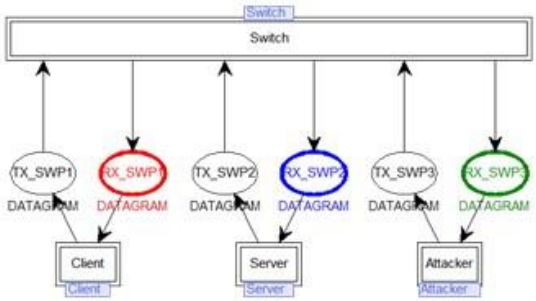

Figure 3. Network Topology (Top-level CPN Model)
Source: Author.

All these components are represented by transitions. The places represent the connection medium (network cables) between the machines and the switch. For every machine there are two places, one for sending (names beginning with TX) and another for receiving packets (names beginning with RX). At this model, there is only one communication sequence, starting with the attacker machine, follow by the client machine, and finally by the server machine. This sequence was defined to simulate an attack scenario where the attacker launches the attack before the client machine start its communication with a server. The "Enable C", "Enable S" and "Enable A" places are used to force this communication sequence. The "Enable A" place has two marks that are transferred to the attacker machine at the beginning of the simulation. The attacker machine sends its packets and when finished, the marks are transferred to the "Enable C" place. The process is repeated at the client machine, and then at the server machine. The arcs interconnect these components and determine in which direction the packets flow at the model. Every transition of this level is detailed at the second level, as follow.

4.2.1. SECOND LEVEL CPN MODEL DESCRIPTION

The second level is composed by four CPN models, one for each transition at the top-level model: client, server, attacker and switch. The first model of the second level, depicted at Figure 4, represents the client machine connected to a SDN network. This model is composed by three transitions (NextMsg, Receive Datagram and Send Datagram), and six places (DTG Count, Source, ARP Table, Transmit, Received and MsgBuffer). At this model, there are four special places (represented by double circles), which represent the points of connection between the second and the top-level CPN models. In this case, the RX-SW special places connected to the top-level place called rx-swp1; the special plac tx-sw is connected to the top-level place called tx-swp1; and the "Enable C" and "Enable S" special places are connected to the places with the same name at the top-level model.

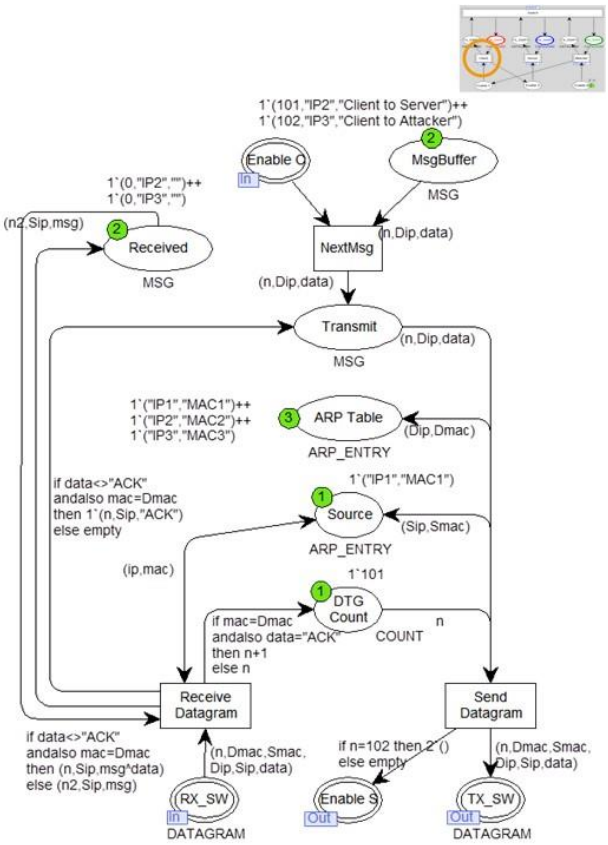

Figure 4. Client Machine (First Model – Second Level)
Source: Author.

The “DTG Count” place implements a counter that is used to create sequential number to numerate the packets sent and received by the client machine. The “Source” place storages information (mark) about the client IP and MAC addresses, which will be used to assemble packets that will be sent, and to check if the packet received by the client machine was correctly delivered. The “ARP Table” place represents the client local ARP table, and storages information about the IP-MAC bindings of all machines connected to the network. The “Transmit” place will receive marks that represent the client messages that will be sent to a remote machine at the network. The “Received” place storages the data (marks) received from remote machines at the network that were addressed to the client machine. The “MsgBuffer” place has the messages waiting to be transmitted to a remote machine.

The inscriptions at each arc define rules to determinate which data can be moved by the arc and which variables should be used to transport the data by the arc. The “Send Datagram” transition is used to assemble the packets that will be sent to the network.

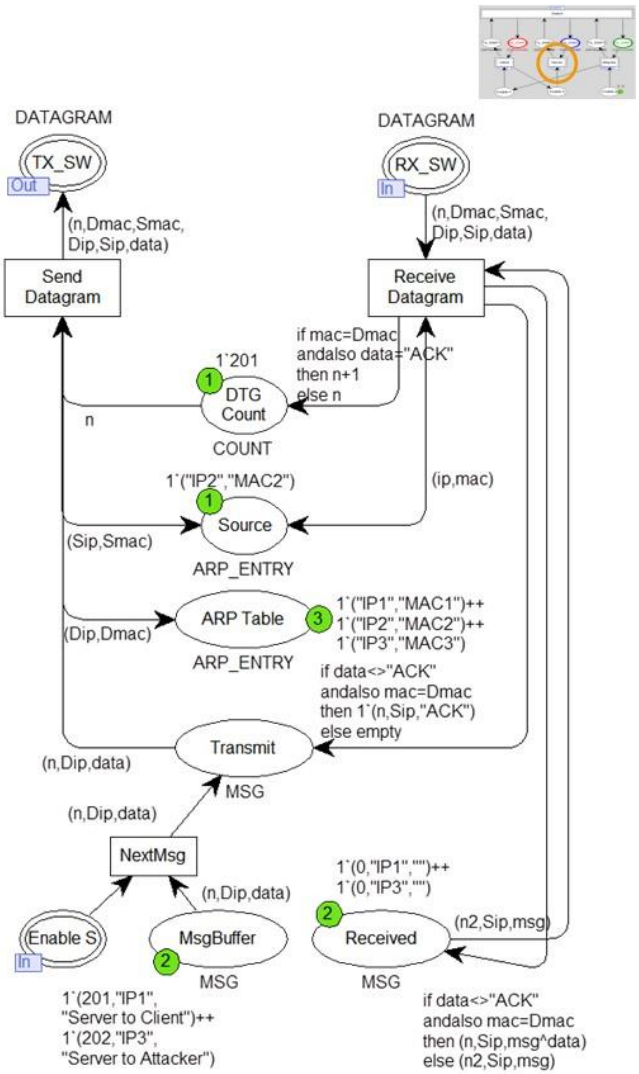

Figure 5. Server Machine (Second Model – Second Level)
Source: Author.

This transition receives the packet number from the "DTG Count" place (variable "n"), the source IP and MAC addresses from "Source" place (variables "Sip" and "Smac"), the data and the destination IP address from "Transmit" place (variables "Dip" and "data") and the destination MAC address from "ARP Table" place (variable "Dmac"). The bidirectional arrow between "Source", "ARP Table" places and the "Send Datagram" transition indicates that the data is moved from the place to the transition, and then returned to the place. The unidirectional arrows show that the data is consumed from the place to the transition.

The "Receive Datagram" transition receives the packet from the network (RX-SW special place), the client IP and MAC addresses from "Source" place, and the previous received messages from "Received" place. If the packet received is addressed to the client machine, the transition concatenates the new message with the earlier received messages from a specific IP address at the "Received" place, creates an acknowledge data at "Transmit" place to inform the sender that the packet was correctly received, and updates the "DTG Count" place to indicate that the next message can be sent if the transition receives a packet to itself with an acknowledge data. The

"NextMsg" transition receives the message to be sent from the "MsgBuffer" place and the marks from the "Enable C" special place (at the top-level CPN model), and then it makes the message available at the "Transmit" place to be sent to its destination. The goal of this transition is allowing the client machine to send its messages only when there are marks at the "Enable C" special place, as described at the top-level CPN model.

The second model of the second level, depicted at Figure 5, represents a server machine connected to a SDN network. This model is composed by the same transitions, places and arcs that compose the client machine model. There are only three differences between the client and the server CPN models. The first difference is at "Source" place initial mark, which now has information about the server IP and MAC addresses (IP2 and MAC2). The second difference is the "MsgBuffer" place initial mark, which storages different messages. The third difference is the absence of an arc between a top-level place and the "Send Datagram" transition, which was used at the client machine model to enable the transmission of the next machine messages. As the servers' machine is the last machine to communicate at the model, it is not necessary any integration between this model and the top-level model.

The third model of the second level, depicted at Figure 6, represents the attacker machine connected to a SDN environment.

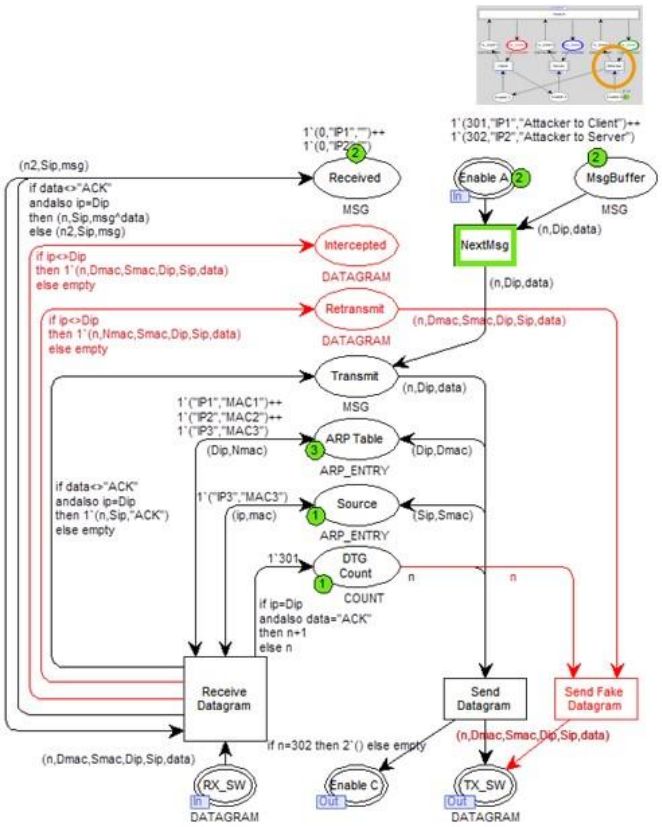

Figure 6. Attacker Machine (Third Model – Second Level)
Source: Author.

This model is composed by the same transitions, places and arcs that compose the client machine model. There are only five differences between the client and

attacker CPN models. The first difference is at “Source” place initial mark, which now has information about the attacker IP and MAC addresses (IP3 and MAC3). The second difference is the “MsgBuffer” place initial mark, which storages messages with different payloads. The third difference is the existence of the “Intercepted” place, that is used to storage all packets received by the attacker machine that are not addressed to itself. The fourth difference is the existence of the “Retransmit” place, which is used to retransmit the intercepted packets to its original destination. The fifth and last difference is the existence of a new transition called “Send Fake Datagram” that is used to reassemble the packet with the original destination IP and MAC addresses.

The fourth model of the second level, depicted at Figure 7, is the Openflow switch that has three ports, and interconnects the machines at the SDN environment. This model is composed by six transitions (Receive SW P1, Transmit SW P1, Receive SW P2, Transmit SW P2, Receive SW P3 and Transmit SW P3), and seven places (P1, P2, P3, Warehouse, DTGLIST, PreAct, and RULE Action). At this model, there are two special transitions (ChkRules and Actions) that will be detailed in the third level CPN model. There is one special place for each transition (RX-SW or TX-SW) that are the same ones described at the top-level CPN model.

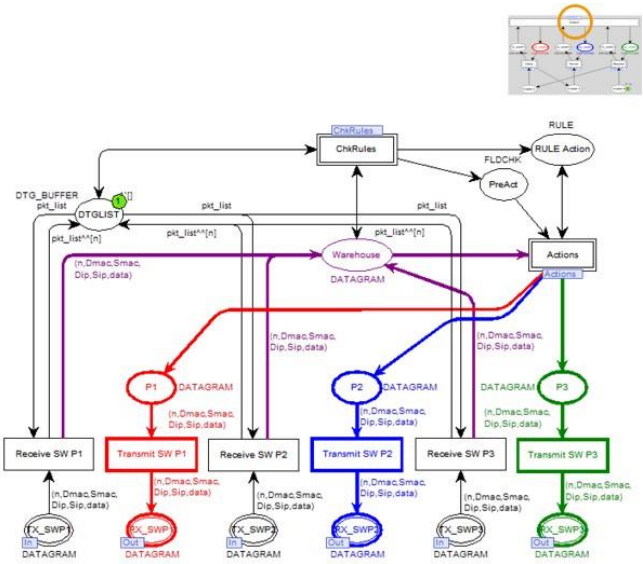

Figure 7. Openflow Switch (Fourth Model – Second Level)
Source: Author.

Each Openflow switch port has two transitions connected to it. One handles the reception of the packets from the machines connected to the switch (Receive SW P1, Receive SW P2 and Receive SW P3), and the other handles the transmission of the packets to the machines connected to it (Transmit SW P1, Transmit SW P2 and Transmit SW P3). When the switch receives a packet, it storages the packet at the Warehouse place, and append the packet number at a list that is managed by the “DTGLIST” place. The “ChkRules” special transition stores the rule table and is responsible to match the correct rule to each packet. It reads the packet number list and the packet header, compares the header fields with the flow rule table and defines which actions the switch will apply to the packet. The packet and the matched rule number are stored at the “PreAct” place, and a copy of the matched flow rule is stored at the “RULE Action” place. The “Actions” special transition reads the information storaged at the

"PreAct" and "RULE Action" places, applies the actions defined by the flow rule and delivers the packet at one of the switch port temporary buffers (P1, P2 or P3 places).

4.2.2. THIRD LEVEL CPN MODEL DESCRIPTION

The third level is composed by two CPN models, one for the "ChkRules" transition and another for the "Actions" transition, which were mentioned at the fourth model of the second level. The first model of the third level, depicted at Figure 8, represents the switchs "ChkRules" transition that is responsible for verify which flow rule should be applied to a specific packet. This model is composed by five transitions (Check Datagram, ChkDmac, ChkSmac, ChkDip and ChkSip) and six places (Rule Count, Rule Table, PreDmac, PreSmac, PreDip, and PreSip).

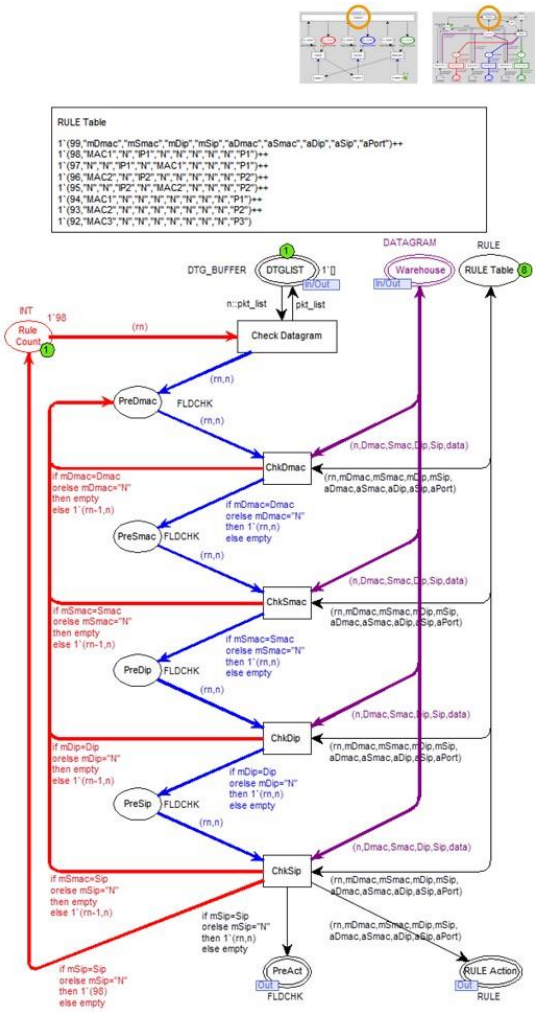

Figure 8. ChkRules Transition (First Model – Third Level)
Source: Author

The "Check Datagram" transition reads the packet number list at the DTGLIST place (Fourth Model Second Level), extracts the number of the first packet at the list, obtains the number of the first flow rule to be tested from the "Rule Count" place, and forwards these information's to the "PreDmac" place. The "PreDmac", "PreSmac", "PreDip" and "PreSip" places are used by this model as a stage area, which are used to store the flow rule number and the packet number, between each packet header field validation.

The “Rule Table” transition stores the flow rule table, and each flow rule entry is composed by a priority number (higher numbers means higher priority), the packet destination MAC address, the packet source MAC address, the packet destination IP address, the packet source IP address, the actions destination MAC address, the actions source MAC address, the actions destination IP address, the actions source IP address and the Openflow switch port the packet should be delivered. The Destination MAC, Source MAC, Destination IP and Source IP are used to match a rule to the packet, and the last four packet header fields are used to store which field values should be replaced by the Openflow switch at the packet. The model checks and acts only at four packet header fields, which are at the scope of this work, but any packet header field can be verified by the Openflow switch. The flow rule fields filled with the letter N means that rule accepts any value for this field at the packet header.

The first four rules implement the proposed control, which creates two flow rules for each IP address at the network. The first rule pair treats all packets that have the client machine as destination, and the second rule pair treats all packets that have the server machine as destination. None of these rules treats the source of the packet. The objective is to treat packets that have been assembled with forged data, by a machine that has been target of an ARP Spoofing attack. This paper does not prevent the occurrence of the ARP Spoofing attack, but the MITM attack that is the consequence of the ARP Spoofing attack.

For each rule pair, the first rule checks if the packets have the correct IP-MAC binding, and delivers the packet at the correct Openflow switch port. The second rule checks if there are any packets with the correct IP address but with an incorrect MAC address, and replaces the original packet destination MAC address with the official and trusted MAC address for the machine. As the Openflow switch stops the flow rule table processing as the first rule matches with the packet header, the packets using the correct IP-MAC binding are matched by the first rule and delivered to the correct switch port. The packets with ARP spoofed are matched by the second rule (as it has the correct IP address, but with the attacker MAC address), which correct the packet destination MAC address with the trusted MAC address, and delivers the packet with the corrected data at the correct destination switch port. The last three rules implement forwarding rules based only on the destination MAC address, as occurs in traditional networks, or in a SDN environment that operates simulating a traditional network.

The “ChkDmac” transition receives the rule number and the packet number (arcs in blue), a copy of the packet from the “Warehouse” special place (arcs in purple), which were explained at the Fourth Model - Second Level, and a copy of the chosen flow rule. If the packet destination MAC address matches with the flow rule destination MAC address, or the flow rule destination MAC address is filled with the letter N, the transition creates a mark at the “PreSmac” place with the rule and packet numbers. Otherwise, the “ChkDmac” transition decrements the rule number and creates a mark at the “PreDmac” place with a new rule number and the same packet number. The “ChkSmac”, “ChkDip” and “ChkSip” transitions implements the same logic to the packet source MAC address, destination IP address and source IP address respectively. The “ChkSip” transition also creates a mark at the “Rule Count” place with the first flow rule number, a mark with the matched rule number and the packet number at the upper level “PreAct” place, and a copy of the matched rule at the upper level “Rule Action” place.

The second model at the third level, depicted at Figure 9, represents the switch's "Actions" transition that is responsible to modify the packet header as defined by the flow rule, and to deliver the packet to the correct Openflow switch destination port. This model is composed by five transitions (ActDmac, ActSmac, ActDip, ActSip and Deliver) and four places (PosADM, PosASM, PosADI and PosASI).

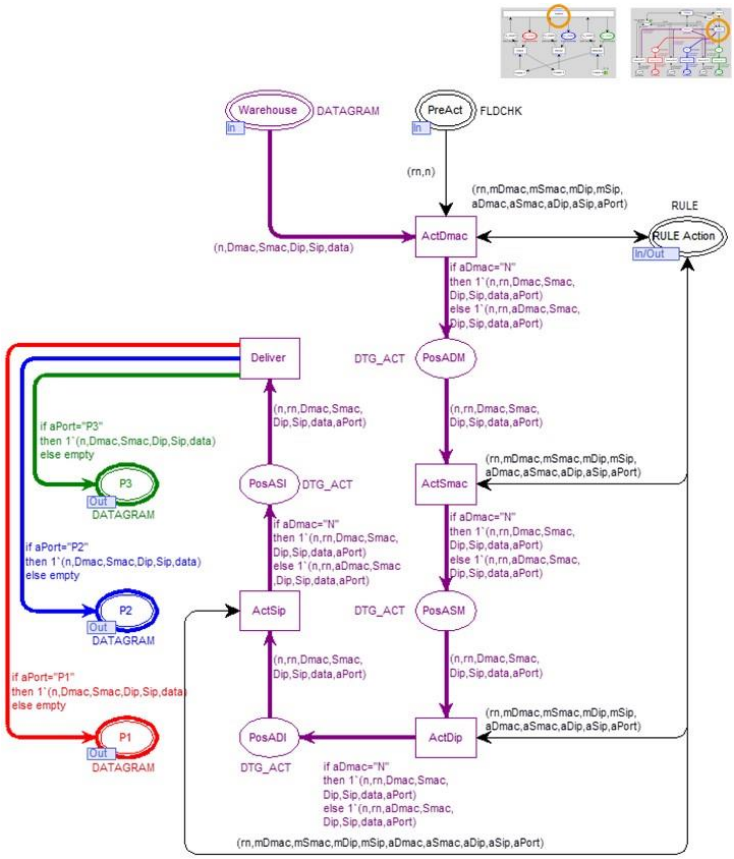

Figure 9. Actions Transition (Second Model - Third Level)
Source: Author

The "ActDmac" transition receives the rule and packet numbers from "PreAct" place, the packet that was stored at the "Warehouse" place and a copy of the matched flow rule. If the "aDmac" rule field is filled with the letter N, the transition forwards the original packet to the "PosADM" place. Otherwise, the "aDmac" rule field value overrides the packet destination MAC address, and the updated packet is sent to the "PosADM" place. The "ActSmac", "ActDip" and "ActSip" transitions implements the same logic to the source MAC address, destination IP address and source IP address, respectively.

The "Deliver" transition reads "aPort" rule field and forwards the packet to the Openflow switch port temporary buffers (P1, P2 or P3) at the Fourth Model - Second Level. The "PosADM", "PosASM", "PosADI" and "PosASI" places are used as a stage area between "ActDmac", "ActSmac", "ActDip", "ActSip" and "Deliver" transitions.

5. VALIDATION AND VERIFICATION OF THE CPN MODEL

This work uses the state space analysis to validate the CPN model and the proposed solution. The state space analysis consists in creating a graph. Each graph node is called a state, which is a combination of the CPN model marks and places. The graph arcs represent the CPN model transitions between one state to another. With this graph is possible to calculate the CPN properties, like the reachability that calculates if there is an occurrence sequence from the initial CPN model marking (node) to a specific marking (another node). With the calculation of the state space, it is possible to make queries in ML to search the graph looking for states that satisfy specified conditions. This work creates two functions, called PrivateCom and InterceptedCom whose codes can be analyzed at Appendix B. Both functions search for states which satisfy all these conditions:

- The “MsgBuffer” place, at Attacker Machine (Third Model - Second Level), is empty;
- The “MsgBuffer” place, at Client Machine (First Model - Second Level), is empty;
- The “MsgBuffer” place, at Server Machine (Second Model - Second Level), is empty;
- The “Warehouse” place, at Openflow Switch (Fourth Model - Second Level), is empty;
- The “RULE Action” place, at Openflow Switch (Fourth Model - Second Level), is empty;
- The “Received” place, at Attacker Machine (Third Model - Second Level), has the marks with the client and server messages;
- The “Received” place, at Client Machine (First Model - Second Level), has the marks with the server and attacker messages; and
- The “Received” place, at Server Machine (Second Model - Second Level), has the marks with the client and attacker messages.

These conditions identify states in which the communications among the machines has finished, and there are no packets inside the switch. The difference between PrivateCom and InterceptedCom is that the first function checks if the “Intercepted” place, at Attacker Machine (Third Model - Second Level) is empty, and the second function checks if the “Intercepted” place has a copy of the marks that represents the packets sent between the client and the server machines.

5.1. VALIDATION OF THE CPN MODEL CORRECTNESS (SDN only)

The first validation scenario aims to verify if the model acts like a normal SDN network, in which every machine communicates with each other without interception. This scenario is validated making one change at the “Rule Count” initial mark at ChkRules Transition (First Model - Third Level - Figure 8), from 98 to 94. This change disables the proposed control (represented by rules from 95 to 98) and enables only the rules that forward the packets using only the MAC address.

After this change, the state spaces analysis is calculated, and the functions PrivateCom and InterceptedCom are executed. The PrivateCom function results the existence of 216 possible states that satisfy the conditions. The InterceptedCom function does not result any possible state that satisfy the conditions. These results indicate that the SDN model is correct, the communication occurs in a private way, and the attacker machine is unable to intercept the communication between the client and the server.

5.2. VALIDATION OF THE MITM ATTACK IN THE CPN MODEL (SDN+Attack)

The second validation scenario aims to verify if it is possible to execute a success MITM attack in a SDN environment. This scenario is validated making some changes in initial marks of the previous scenario, modelling a situation in which the client and server local ARP table has been spoofed, as follows:

- At the client machine model (Figure 4), the initial mark that binds IP2 to MAC2 (server IP and MAC addresses) is changed to bind IP2 to MAC3 (server IP address and attacker MAC address);
- At the server machine model (Figure 5), the initial mark that binds IP1 to MAC1 (client IP and MAC addresses) is changed to bind IP2 to MAC3 (server IP address and attacker MAC address).

After the changes, both the state spaces analysis and the functions PrivateCom and InterceptedCom are executed again. The PrivateCom function does not result any possible state that satisfy the conditions. The InterceptedCom function results the existence of 216 possible states that satisfy the conditions. These results indicate that a SDN environment is susceptible to the MITM attack as the attacker machine could intercept the packets sent between client and server.

5.3. VALIDATION OF THE PROPOSED SECURITY CONTROL IN THE CPN MODEL, WITHOUT A MITM ATTACK (SDN+CTRL)

The third validation scenario aims to verify if the proposed control alters the SDN normal operation. This objective is achieved returning the client and server machines to its initial state (without the spoofed local ARP table), and the proposed control rules should be enabled again. The following changes are needed in initial marks of the previous scenario:

- At the client machine model (Figure 4), the initial mark that binds IP2 to MAC3 (server IP address and attacker MAC address) is changed to bind IP2 to MAC2 (server IP and MAC addresses);
- At the server machine model (Figure 5), the initial mark that binds IP2 to MAC3 (server IP address and attacker MAC address) is changed to bind IP1 to MAC1 (client IP and MAC addresses);
- The initial mark at the “Rule Count” place (Figure 8) is changed from 94 to 98. This change enables the proposed control rules from 95 to 98.

The execution of the state space analysis and the functions PrivateCom and InterceptedCom bring the same results that were obtained at the first scenario (see section 5.1). The PrivateCom returned 216 possible states and InterceptedCom does not result any possible state. These results indicate that the proposed control has no effect on the SDN normal operation.

5.4. VALIDATION OF THE PROPOSED SECURITY CONTROL IN THE CPN MODEL, UNDER A MITM ATTACK (ATTACK + CONTROL)

The fourth scenario aims to verify if the proposed control is effective to prevent MITM attacks. This scenario is validated when the machines have its local ARP table spoofed and the proposed control is kept enabled. It is done by making the following changes at the previous scenario initial marks:

- At the client machine model (Figure 4), the initial mark that binds IP2 to MAC2 (server IP and MAC addresses) is changed to bind IP2 to MAC3 (server IP address and attacker MAC address);
- At the server machine model (Figure 5), the initial mark that binds IP1 to MAC1 (client IP and MAC addresses) is changed to bind IP2 to MAC3 (server IP address and attacker MAC address).

After the changes, the state spaces analysis and the functions PrivateCom and InterceptedCom are executed once again. The PrivateCom function results the existence of 216 possible states that satisfy the conditions. The InterceptedCom function does not result any possible state that satisfy the conditions. These results indicate the proposed control is effective to prevent MITM attack as the communication occurs in a private way, and the attacker machine is unable to intercept the communication between the client and the server machines.

5.5. RESULTS

As depicted at Figure 10, the state space analysis also provides information about the total number of possible states in a CPN model, which represents all behaviors of the model. The comparison of the number of states gives some insights about the results. The comparison of scenarios shows:

- An increase of 44.904 states (67%), between the first and second scenarios, as result of a MITM attack in a SDN environment without the proposed control. These states represent the Attacker machine intercepting the packets between Client and Server machines;
- A reduction of 11.172 states (9,98%) between the second and fourth scenarios, which shows that the states representing the interception of the data sent between Client and Server machines was eliminated;
- An increase of 1.696 states (1,71%) between scenarios three and four, as results of a MITM attack in a SDN environment with the proposed control. These states represent the proposed control treating the forged packets;

- An increase of 32.036 states, between the first and third scenarios, indicating that the proposed control increased 47,79% the number of possible states in a SDN environment;
- The comparison of the impact caused by the MITM in a SDN without (44.904) and with (1.696) the proposed control shows 96,22% of reduction at the MITM impact in a SDN environment.

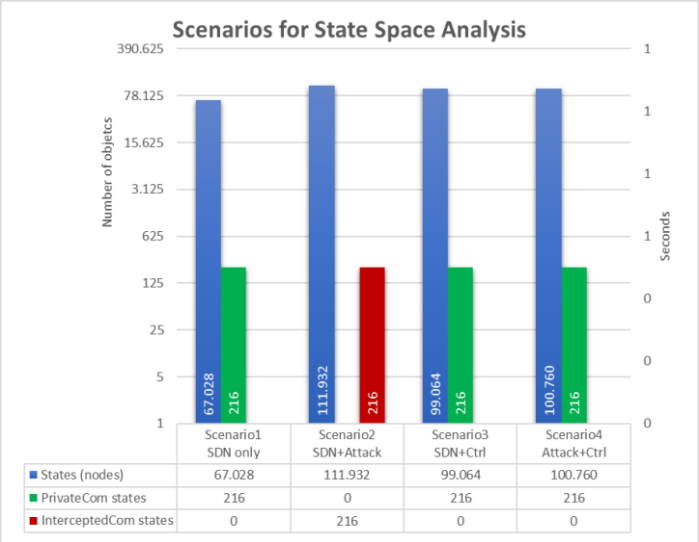

Figure 10. Scenarios for State Space Analysis.
Source: Author

It is possible to attest that the proposed control attends to the five factors, proposed by Tripathi and Mehtre (2014), to consider a control a versatile solution to MITM attacks. As the proposed solution does not analyzes ARP messages and prevent MITM attack even when the machine has a spoofed local ARP table, it can be considered resistant to flood of spoofed ARP messages (first factor). Even if the attacker registers to itself all unused IP address, before the legitimate client joins to the network, it will receive only packets addressed to it. So, it is resistant to the second factor. As the proposed control does not requires any changes at the ARP protocol, it can be considered resistant to third factor that says that the proposed solution should be compatible with the existing network infrastructure. The fourth factor identify if the proposed solution cannot be considered a single point of failure. It is not the case of the proposed control, as the rules are distributed across all Openflow switches. Finally, as the proposed control uses the IP address as a unique identity, it can be considered resistant to the fifth factor that defines the proposed solution should allow more than one IP address to each MAC address.

6. FINAL CONCLUSION

This work has proposed a preventive control against MITM attack in a SDN environment, even with machines that have its local ARP table spoofed. The SDN environment, the MITM attack and the proposed control have been modeled using CPN. The CPN parameterization allowed the creation of four scenarios, and two functions

were defined to check if the communication between client and server machines could or couldnt be intercepted in all possible CPN model conditions. The functions proved the communications occur in a private way at scenarios one, three and four. Only the second scenario has the communication intercepted. These results show that the proposed control is effective to prevent MITM attacks in a SDN environment.

The total number of states calculated to each scenario, by the state space analysis, was used to mensuration of the attack and control impact in a SDN environment. The comparison of the scenarios demonstrated that:

- The proposed security control eliminated the interception states;
- The MITM attack impact was reduced in 96,22% by the proposed security control;
- The proposed control increases the number of possible states in 47,79% on a SDN environment; and
- The analysis of the proposed security control using the five evaluation factors proposed by Tripathi and Mehtre (2014) shows the control is resistant to all five factors and can be considered an effective control against MITM attacks.

The proposed control has the following limitations:

- The proposed control is effective against MITM attacks based on ARP Spoofing;
- The proposed control does not prevent the ARP table poisoning;
- The CPN model does not represents the communication between the controller and the switch;
- The proposed control is based only on SDN that uses the Openflow protocol. So, considering another protocol, it is not possible to say about its effectiveness; and
- The proposed control needs a trusted source of IPMAC bindings, and that data is transmitted to the controller in a secure way.

As future work, we can consider the proposed security control been used to prevent types of attacks different than MITM, which would have others packet header fields.

REFERENCES

ANAN, M., AL-FUQAHA, A., NASSER, N., MU, T.Y., and Bustam, H. (2016) Empowering networking research and experimentation through software-defined networking. Journal of Network and Computer Applications, 70, 140–155.

AROTE, P.; ARYA, K. V. (2015) Detection and prevention against arp poisoning attack using modified icmp and voting. Computational Intelligence and Networks (CINE), 2015 International Conference on, pp. 136–141. IEEE.

BARBHUIYA, F. A., BISWAS, S., HUBBALLI, N., NANDI, S. (2011) A host based des approach for detecting arp spoofing. Computational Intelligence in Cyber Security (CICS), 2011 IEEE Symposium on, pp. 114–121. IEEE.

BRUSCHI, D., ORNAGHI, A., ROSTI, E. (2003) Sarp: a secure address resolution protocol. Computer Security Applications Conference, 2003. Proceedings. 19th Annual, pp. 66–74. IEEE.

COX, J. H., CLARK, R. J., OWEN, H. L. (2016) Leveraging sdn for arp security. SoutheastCon, 2016, pp. 1–8. IEEE.

FLOODLIGHT, P. (2016). The controller.

FOUNDATION, O. N. (2015) Openflow Switch Specification Version 1.5.1.

FORCE, I. E. T. (2014). Forwarding and control element separation (forces).

JENSEN, K., KRISTENSEN, L. M.; WELLS, L. (2007) Coloured petri nets and cpn tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer, 9, 213–254.

KUMAR, S.; TAPASWI, S. (2012) A centralized detection and prevention technique against arp poisoning. Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on, pp. 259–264. IEEE.

LERES, C. (2006). Arpwatch tool:Arp spoofing detector.

LOOTAH, W., ENCK, W., McDaniel, P. (2007) Tarp: Ticket-based address resolution protocol. Computer Networks, 51, 4322–4337.

MASOUDI, R.; GHAFFARI, A. (2016) Software defined networks: A survey. Journal of Network and computer Applications, 67, 1–25.

PANDEY, P. (2013) Prevention of arp spoofing: A probe packet based technique. Advance Computing Conference (IACC), 2013 IEEE 3rd International, pp. 147–153. IEEE.

ROJAS, M. A. T., UEDA, E. T., BRITO, T. C. M. (2014) Modelling and verification of security rules in an openflow environment with coloured petri nets. Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on, pp. 1–7. IEEE.

SASAN, Z.; SALEHI, M. (2017) Sdn-based defending against arp poisoning attack. Journal of Advances in Computer Research, 8, 95–102.

TETERIN, I. (2003). Antidote.

TRIPATHI, N.; MEHTRE, B. (2014) Analysis of various arp poisoning mitigation techniques: A comparison. Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on, pp. 125–132. IEEE.

APPENDIX A

The Table 1 presents the CPN model color set, which defines the data type and description of each variable.

Variable	Color (Type)	Description
pkt list	DTG BUFFER	Packet list at the Open flow switch
n and n2	INT (Integer)	Packet number
rn	INT (Integer)	Data flow rule number at the Open flow switch
ip	STRING	IP address
mac	STRING	MAC address
Smac	STRING	Source MAC
Dmac	STRING	Destination MAC
Nmac	STRING	Correct victim MAC, stored by attacker
Sip	STRING	Source IP
Dip	STRING	Destination IP
data	STRING	Packet data
msg	STRING	Data received by a machine
mDmac	STRING	Match destination MAC address
aDmac	STRING	Destination MAC that will override the packet header field
mSmac	STRING	Match source MAC
aSmac	STRING	MAC that will override the packet header field
mDip	STRING	Match destination IP
aDip	STRING	Destination IP that will override the packet header field
mSip	STRING	Match source IP
aSip	STRING	Source IP that will override the packet header field
aPort	STRING	Open flow switch port number that will be used to deliver the packet

Table 1. The CPN Model Color Set
Source: Author.

APPENDIX B

The following two ML codes are used to scan all graph, generated by the state space analysis, and count how many states satisfies the conditions described at section 5.

PrivateCom Code:

[language=C] fun PrivateCom k = ((Mark.attacker'MsgBuffer 1 k) == empty) and also ((Mark.client'MsgBuffer 1 k) == empty) and also ((Mark.server'MsgBuffer 1 k) == empty) and also ((Mark.Switch'Warehouse 1 k) == empty) and also ((Mark.Actions'RULEAction 1 k) == empty) and also ((Mark.attacker'Received 1 k) == 1'(102,"IP1","client to attacker")++ 1'(202,"IP2","server to attacker")) and also ((Mark.client'Received 1 k) == 1'(201,"IP2","server to client")++ 1'(301,"IP3","attacker to client")) and also ((Mark.server'Received 1 k) == 1'(101,"IP1","client to server")++ 1'(302,"IP3","attacker to server")) and also ((Mark.attacker'Intercepted1k)==empty);val state*list*= *PredAllNodesPrivateCom*;*valstatelistsize*=*lengthstatelist*;

InterceptedCom Code:

[language=C] fun InterceptedCom k = (Mark.attacker'MsgBuffer 1 k) == empty) andalso ((Mark.client'MsgBuffer 1 k) == empty) andalso ((Mark.server'MsgBuffer 1 k) == empty) andalso ((Mark.Switch'Warehouse 1 k) == empty) andalso ((Mark.Actions'RULEAction 1 k) == empty) andalso ((Mark.attacker'Received 1 k) == 1'(102,"IP1","client to attacker")++ 1'(202,"IP2","server to attacker")) andalso ((Mark.client'Received 1 k) == 1'(201,"IP2","server to client")++ 1'(301,"IP3","attacker to client")) and also((Mark.server'Received 1 k) == 1'(101,"IP1","client toserver")++1'(302,"IP3","attackertoserver")) andalso ((Mark.attacker'Intercepted1k)== 1'(101,"MAC3","MAC1","IP2","IP1","client to server")++ 1'(101,"MAC3","MAC2","IP1","IP2","ACK")++ 1'(201,"MAC3","MAC1","IP2","IP1","ACK")++ 1'(201,"MAC3","MAC2","IP1","IP2","server to client"));valstate*list*= *PredAllNodesInterceptedCom*;*valstatelistsize*= *lengthstatelist*;[]