Edge Security for SIP-enabled IoT Devices with P4

Aldo Febro
Dept. of Computer Science
University of Hertfordshire
Hatfield, UK AL10 9AB
aldo.febro@iotseclab.com

Hannan Xiao
Dept. of Informatics
King's College London
London, WC2B 4BG
hannan.xiao@kcl.ac.uk

Joseph Spring
Dept. of Computer Science
University of Hertfordshire
Hatfield, UK AL10 9AB
j.spring@herts.ac.uk

Bruce Christianson
Dept. of Computer Science
University of Hertfordshire
Hatfield, UK AL10 9AB
b.christianson@herts.ac.uk

Abstract*—The exponential growth of IoT devices poses security concerns, in part because they provide a fertile breeding ground for botnets. For example, the Mirai botnet infected almost 65,000 devices in its first 20 hours. With the prevalence of Session Initiation Protocol (SIP) phones and devices on the networks today, the attacker could easily target and recruit these IoT devices as bots. Conventional network security measures do not provide adequate attack prevention, detection, and mitigation for these widely distributed IoT devices. This paper presents microVNF, a Virtualized Network Function (VNF) that leverages the programmable data plane feature on the edge switch. Based on knowledge gained from the Mirai botnet incident and following the defense-in-depth principle, microVNF protects IoT devices against SIP DDoS attacks in two stages: before and after infection. Prior to infection, it protects against SIP scanning, enumeration, and dictionary attacks. After infection, microVNF blocks botnet registration attempts to the command-and-control (CNC) server, thereby preventing the botnet from receiving commands sent from the CNC server, and detects and mitigates botnet SIP DDoS attacks. We conducted six experiments that involved using popular attack tools against microVNF, and it successfully performed deep-packet inspection of unencrypted SIP packets so as to track anomalies from a typical SIP state-machine. In this use case, besides providing physical connectivity to the IoT devices, the edge switch containing microVNF also provides the first line of defense in stopping malicious packets from propagating upstream to the core network. In addition to securing SIP, the microVNF approach can be adapted to other text-based, application-layer protocols such as HTTP and SMTP. MicroVNF leverages the native capability of programmable data planes without depending on external devices, thereby making this approach practical for securing edge-computing environments against application-layer attacks.

***Index Terms*—SIP, DDoS, Dictionary attack, IoT, P4, VNF, SIPVicious, Edge Computing**

I. Introduction

Although the Internet of Things (IoT) brings benefits, it also poses serious challenges in securing IoT devices. It is undeniable that IoT devices are capable of producing data that brings a new level of situational awareness, which can be critical for making decisions and taking actions. However, the deployment of these devices presents a new challenge: it is relatively easy for them to be exploited by an attacker, due to a lack of resources for IoT devices to perform rigorous security controls. This vulnerability threatens the privacy and integrity of the data produced by these devices, and threatens the security of the Internet as a whole.

IoT devices, for example, have the unintended side effect of providing an ecosystem for malware to multiply and form botnets [1]. When activated, a botnet can launch attacks such as Distributed Denial of Service (DDoS). To get a sense of the magnitude, Antonakakis, et al. documented the spread of the Mirai botnet where it infected nearly 65,000 IoT devices within the first 20 hours and reached a stable population size of 200,000 - 300,000 devices [2]. At its peak, Mirai consisted of 600,000 infected devices, including IP cameras, IP phones, IP video recorders, etc. [2] [3] [4]. As a botnet of this magnitude, Mirai was used to launch multiple DDoS attacks of various types (e.g., application-layer, volumetric, and TCP state exhaustion) with some high-profile targets such as Dyn [5], Krebs on Security [6] and a large telecom operator in Liberia [7].

Securing IoT devices against botnets is challenging for the conventional approach, due to its distributed nature. The Mirai botnet can propagate itself by scanning for IoT nodes with open administrator ports and weak passwords. For traditional firewalls or intrusion-detection systems to be aware of such activities, they need to have visibility of these activities on potential targets. Unfortunately, IoT devices are installed at the network's edges and fall outside of conventional security devices' visibility. In zero-day attack scenarios, even when firewalls or intrusion-detection systems are present, they are not effective in detecting malicious botnet activities due to the absence of signatures for attacks that are unknown [8]. Kamaldeep, et al. [9], Zarpelo et al. [10], and Wazzan et al. [11] made similar observations that a signature-based approach is not effective against zero-day attacks.

Securing IoT devices against botnets is also challenging due to resource constraints. IoT devices are mass produced for a specific purpose. With economy of scale in mind, the cost of these devices is optimized, by using just enough components to perform the purpose. This approach results in devices with weak security posture and vulnerable to attacks. Survey papers [11] [12] on IoT security note where these devices are most resource constrained and therefore vulnerable. Wazzan et al. [11] suggest exploring edge-based detection, which is the next layer of defense with IoT devices being vulnerable.

Researchers often use variations of the three-layer IoT architecture (cloud, fog, and edge) to describe the characteristics that are present at each layer [13], [14], [15]. The cloud layer has the most resources (e.g., computing, storage, connectivity, reliability, etc.), typically found in data centers. The fog layer acts as an intermediary layer to fill in the gap by providing some computing resources to process the data generated by the edge devices. The edge layer contains millions of IoT devices that generate the data. Typically, firewalls and intrusion-detection systems are absent at the edge layer due to practical and financial concerns. In the Mirai attack case, the miscreants successfully exploited two IoT vulnerabilities: inadequate authentication and insufficient access control, to take over the device. These vulnerabilities are among nine IoT vulnerabilities, described in the survey paper by Neshenko, et al. [16]. Therefore, it is arguable that the edge layer needs additional security features to defend against similar attacks at this layer. The edge switch, with its programmable data plane, could be positioned in the fog layer to provide a layer of security for the edge devices as depicted in Fig. 1. Considering that the edge switch has more resources than IoT devices, it could off-load high computational tasks from edge devices. For example, cryptographic algorithms, packet inspection, and data analysis.

Edge networks that are implemented using the Software-Defined Networking (SDN) approach suffer from scalability and performance issues. Despite the advantages of using the SDN architecture that separates control from the data plane, it introduces latency that stems from control-data plane communication. In an architecture where a centralized SDN controller controls multiple edge switches, the edge switches depend on the controller for instructions. Scalability and performance issues in OpenFlow-enabled switches for stateful packet processing are discussed in [17].

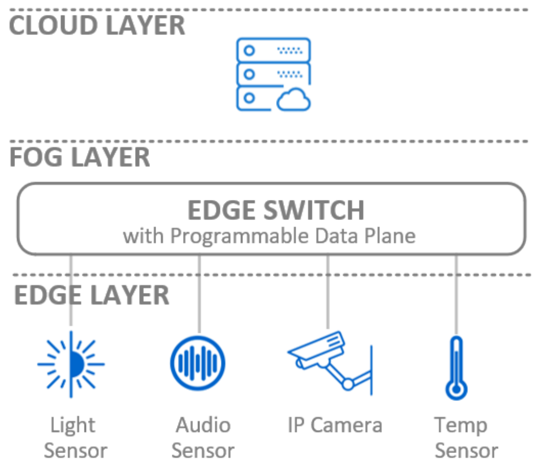


Fig. 1. Edge switch providing protection to IoT devices

This paper addresses a research gap that is found between the weak security posture of IoT devices [16] and the latency that exists in SDN's control-data plane communication [17].

The present paper proposes a new defense model, in which the edge switch is empowered for application-layer attack detection and mitigation. This model addresses the scalability and performance issues of SDN architecture as described in review paper [17]. The key enabler of our model is a new paradigm in networking, where the data plane is now sufficiently programmable to perform custom parsing, match-action pipelining, and queuing. With these capabilities, the data plane off-loads the task from a centralized SDN controller for tracking stateful packet inspection. These new capabilities are available on the new generation of switches, network interface cards, and network processors that support the Programming Protocol-Independent Packet Processors (P4) programming language [18]. Besides the powerful features already mentioned, the data plane can also perform match-action at the application-layer for protocols like SIP. For example, it could perform match-action for SIP command-and-response to track for anomalous session behavior. Instead of following a traditional approach, using a middle-box to detect and mitigate SIP attacks, we introduce microVNF, a VNF written in the P4 programming language and running on a programmable data plane.

Recent developments that we have made, extending our earlier work in [19] and [20] include the addition of new techniques to detect and mitigate different kinds of SIP attacks. In its current state, SIP scanning, SIP account enumeration, and SIP account brute-force attack can be detected and mitigated. Using the state-based attack detection method, SIP flash traffic and low-rate attack scenarios can be detected as well. With this method, SIP-state compliance is enforced regardless of how fast or slow a session is established. In a flash traffic scenario, each port on the switch is receiving compliant SIP packets from the end-point within a short period of time. Since these packets are still compliant and below the threshold, they are still allowed. In a low-rate attack scenario, each SIP session is established over a much longer time span. However, each port is still subjected to a limited number of SIP sessions allowed, regardless of how long it takes to establish each session.

A. Motivation

The motivations for this paper are threefold. The first is to perform feasibility experiments that deploy a micro VNF (application-layer firewall and intrusion-detection) on an edge switch's data plane. Our placement of this defense mechanism contrasts with that of conventional methods, which typically place defense in the data center. The aim is to scale detection and mitigation functions, so that network operators can address attacks as close to their source as possible. This method is beneficial because it minimizes collateral damage and protects the core network.

In this use case, the placement of micro VNF on a hardware-based switch rather than on a server (i.e., software-based VNF) has the potential to increase cost-effectiveness. Network administrators would typically use a per user (or per port) pricing when preparing a budget on new features or hardware upgrades. Each access-layer switch typically has 24 to 48 physical ports whereas the commercial off-the-shelf server would typically have fewer. From the hardware upgrade perspective, the old switch could be swapped with a P4-enabled switch (that has the same physical footprint) without requiring more space on the rack, rewiring, or power. With P4 being an open-source project, competition among P4-enabled switch manufacturers will drive down the cost over time and make it competitive against proprietary switches. Another advantage of using a hardware-based switch is to draw the security right to the network edge. In contrast, when using a software-based VNF server in a data center, the server would need to control remote switches, which introduces the potential of delay and failure. Taking these factors into consideration and from the cost-per-port(user) perspective, placing the VNF on a switch would allow the network administrator to provide new functions for less as compared to the server-based VNF.

The second motivation is to experiment with a solution that can defend against attacks occurring both before and after botnet infection. This approach aligns with the operation of modern botnets, which perform different sets of activities depending on the stage of infection. Prior to infection, their focus is finding and recruiting vulnerable IoT devices whereas, in contrast, after infection they connect to the CNC server and execute attacks as instructed by it. The approach adopted by our proposed method also better aligns with advanced, persistent threat types in which attacks are strategically launched and timed.

The third motivation is to investigate protection of SIP endpoints, due to their prevalence in 5G environments and services. Recent studies have provided empirical data concerning IoT device vulnerability and its growth rate [21] [22], some of which is attributable to poor configuration [23]. Another study reports an exploit on mobile phones to form a botnet to launch a DDoS attack [24]. These data suggest that the vulnerabilities of these devices to attack are likely to continue into the foreseeable future. With IoT deployment expected to enter an accelerated growth phase, the IoT security models need to be reevaluated to reduce the risks and exposure.

B. Contribution

The contribution of this paper is microVNF, a VNF that performs detection, prevention, and mitigation for SIP-based scanning, enumeration, brute force, and DDoS attacks. Our microVNF runs on the same switch that the IoT devices are connected to, and is able to function without extra hardware or software components. The concepts and methods described in this paper could potentially be implemented on a hardware-based target architecture to improve performance.

To the best of our knowledge, this is the first VNF that is designed and targeted at securing the network edge for the purpose of defending against SIP-based attacks. The same principle however, is applicable and relevant for other application-layer protocols that are similar to SIP, which is text-based and uses state machines as part of its operation, e.g., HTTP and SMTP.

Our microVNF implementation on the behavioral model architecture is intended as a proof-of-concept, and is intended to motivate further validation on hardware-based architectures. With a programmable data plane, network operators are now empowered to innovate, and quickly build proof-of-concept models with software-based architectures such as the behavioral model. With this approach, microVNF can serve as a concrete prototype for discussion with switch manufacturers or merchant silicon providers. Considering that hardware-specific implementations are still being actively researched, collaboration with these providers would validate the initial assumptions, unveil hardware limitations, and identify the best performing method available for a given hardware architecture. A list of hardware that supports the P4 programming language is given in [25].

In the remainder of this paper, related works will be discussed in section II, the proposed solution (microVNF) will be described in section III, followed by six experiments in section IV, then a discussion in section V. The conclusion of this paper is set out in section VI.

II. Related works

A. P4 and security

Sivaraman et al., [26] presented heavy hitter identification in the data plane, which is a similar concept and application that makes use of P4 functionalities for DoS detection. While similar in some aspects, our proposed solution is designed for the SIP protocol, which is at the application-layer.

B. Edge security

Y. Xiao et al., [27] published a survey on edge computing security that captures prior work in this open research area. It highlights four major attacks that are found in an edge computing environment: DDoS, side-channel, malware injection, and authentication attacks. In this paper, we propose a novel approach to two of the four attacks mentioned, namely DDoS and authentication attack.

Yan, Qiao, et al., [28] proposed a multi-level DDoS mitigation framework (MLDMF) that consists of defense at the edge, fog, and cloud computing levels. Security at the perception layer is mainly managed by an SDN-based IoT gateway, which performs security controls over the IoT devices. The experiment performed was ping of death and TCP SYN flood. In comparison, our paper experiments with application-layer (SIP) DDoS instead of the network layer.

Alharbi et al in [29] designed a framework that uses Network Function Virtualization (NFV) and edge computing for DDoS mitigation. The VNF is deployed as a virtual machine on a shared hardware appliance. The DDoS mitigation framework consists of a screener stage and service stage. The instantiation and orchestration for the NFVs is provided by a management and orchestration (MANO) element. In contrast, our paper implements the VNF on the edge switch.

In [30], Bahman and Fung propose a collaborative network called CoFence that allows several enterprises or ISPs to defend jointly against DDoS attacks, by using an NFV-enabled domain that consists of a virtual gateway and virtual IPS. In this paper, the coordination for attack mitigation is done within a single network, so prior arrangements with other organizations are not required.

C. IoT security

In [31], the authors propose antibIoTic 2.0, which relies on fog computing to secure IoT devices in a legal and controlled manner. AntibIoTic bot is installed on IoT devices to sanitize, secure, and report information to the Fog node. In comparison, the approach proposed by this paper does not require a bot or an agent to be installed on each IoT device.

In [32], Rafique, et al. propose CFADefense, which uses link selection, attack detection, and malicious flow interception modules to mitigate IoT devices from launching a Crossfire Attack (CFA). It is deployed at the application plane of the Floodlight SDN controller. In contrast, the VNF proposed in our paper is able to operate independently without the use of a centralized SDN controller.

In [33], Khosroshahi and Ozdemir propose DoS detection from an Android device, with an entropy-based detection frame-work using a Support Vector Machine (SVM) classification algorithm. Their work requires a VPN client to be installed on the Android and connected to a VPN server, which then enables data collection and attack detection. In comparison, the proposal in our paper does not require installation of any client on the end-device, as we leverage the edge switch itself for attack detection and mitigation.

In our earlier works [19] and [20], the same approach was presented but the scope was limited to DDoS attacks. In [19], an external SDN controller is required to verify the thresholds. In [20], the data plane is able to operate independently and no longer relies upon an external SDN controller. Without this external dependency, the speed is much improved. In our present paper, we include SIP scan, dictionary attacks, as well as blocking the CNC sessions which often accompany Botnet attacks.

III. The Proposed Solution: microVNF

The microVNF is a VNF that is designed to run on the programmable data plane of a P4-enabled switch deployed at the IoT network edge (Fig. 2). It provides a lightweight firewall and intrusion detection for the SIP protocol at the edge of the IoT network to protect SIP-enabled devices.

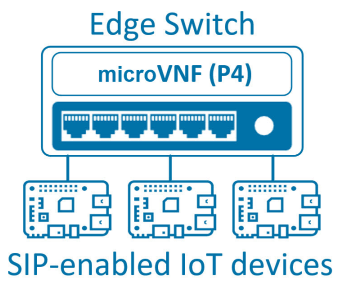


Fig. 2. microVNF running on the data plane of edge switch

Learning from the Mirai botnet incident, we now understand how it propagates and launches DDoS attacks. Armed with this knowledge, the microVNF provides SIP-based attack detection and mitigation in two distinct phases: before and after a botnet infection (Fig. 3). This approach is essential because botnet activities are different in each phase, and the defense needs to be able to adapt and respond appropriately.

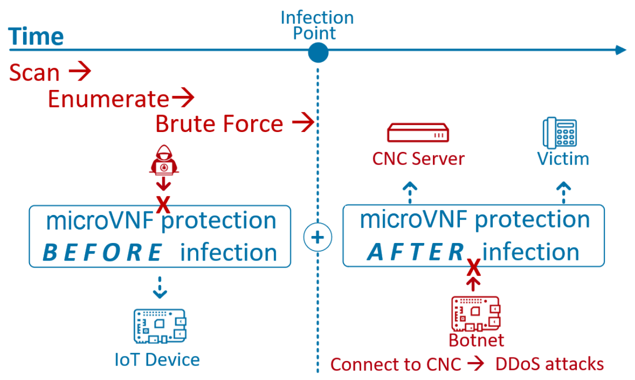


Fig. 3. microVNF provides protection for before AND after the infection

A. Workflow

microVNF uses a simple workflow that executes the following phases to process each packet: parse, inspect, verify, and count. Figure 4 depicts the workflow for microVNF processing.

1) Parse: In the **parse** phase, microVNF extracts and parses the packet headers based on the header definition. For example, 6 bytes for source and destination MAC address, 2 bytes for ethertype, 20 bytes for IPv4 headers, 8 bytes for UDP port, and 6 bytes for SIP request. When microVNF has extracted the information from these locations, microVNF has enough information to proceed and is ready to move on to the next phase, i.e., inspection.

2) Inspect: Once microVNF has parsed these fields, the packet goes through the **inspect** phase, where microVNF performs deep packet inspection. There are two main tasks for this phase: the first is to identify whether the packet is part of a known attack, and the second, to determine whether this is part of a SIP attack. microVNF accomplishes the first task by comparing the following key pieces of information and dropping those packets that match previously known records for attacks:

- The source IP address and port against the known attacker records
- The destination IP address and port against the known victim records
- The destination IP address and port against the known command-and-control records

For the second task, microVNF inspects the first 6 bytes of the SIP payload to categorize the type of SIP packet. If the packet affects the SIP state machine and is therefore important to track, microVNF will progress to the next phase. Some examples where the SIP request/response will affect the SIP session state are:

- the `OPTION` packet (it is used to verify that the server offers a SIP service)
- the `REGISTER` and `401` pair (they are used for authentication sessions)

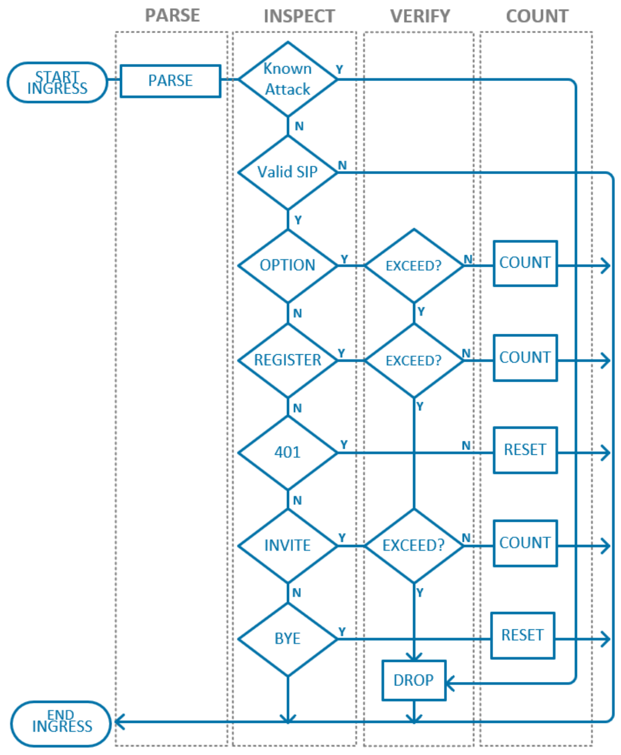


Fig. 4. microVNF workflow

- the `INVITE` and the `BYE` pair (they are used for call initiation and termination)

3) Verify: After learning about the intent of the packet, by examining the payload and how it affects the session state, microVNF performs verification in the **verify** phase. In this phase, microVNF validates whether or not the packet has exceeded the threshold that was previously set for the session. If it has, microVNF will drop the packet at the end of the process. If not, microVNF will move it to the next phase. For example, a normal SIP client-server session starts with an `INVITE` packet and ends with a `BYE` packet. When there are hundreds of `INVITE` packets without the corresponding `BYE` packet, this indicates a DDoS attack, and microVNF needs to drop those packets. With microVNF, if the threshold for open `INVITE` sessions is three, this means that every client can only initiate three concurrent SIP calls at any given time.

While the counter for each sender is still below the threshold, microVNF will simply count this packet against the sender. At the end of a normal session (and when the expected BYE packet is encountered), the counter for that particular sender is reset to zero.

The threshold value is a variable which for this study was set at 3. This value was selected as a trade-off between 'recognizing attack profile' versus 'performance'. The threshold could be set to a higher value, which would consume more CPU and memory resources to keep track of an increase in session states. For example, in a typical network-edge scenario, there are 24 IoT devices sharing a switch. In the case where each device generates 3 active SIP sessions, the switch would need to keep track of 72 active SIP sessions in the memory. Higher threshold values will add more strain to the access-layer switch which has limited computing resources. Furthermore, a higher threshold will pass more malicious packets to the next switch upstream, which may create unintentional collateral damage.

Alternatively the threshold could realistically be set to two, which would reduce resource load. While a SIP server may have many active concurrent sessions, it is extremely rare, in our experience for an edge device to have more than two active sessions simultaneously. In order to obtain a realistic, but conservative, benchmark in this proof-of-concept paper, we have set the threshold value to a compromise value of three.

Our approach on using fixed value thresholds is similar to that applied in [34] [35]. Profiling attacks is an active research field, but it is beyond the scope of this paper [36] [37] [38]. We have orthogonalized this issue by designing the trigger to be fully configurable. As profiling techniques evolve, adaptive and even non-deterministic thresholds could be incorporated into our trigger conditions.

4) Count: In the **count** phase, even when the packet has not exceeded the threshold, microVNF will count the number of packets associated with each session. The counter is increased and decreased as necessary, depending on whether or not it follows the expected operation as per the SIP protocol. For example, when a SIP-client initiates a call using the SIP `INVITE` packet, microVNF sets the counter for the first INVITE packet for this session to one. When the call ends, and the SIP-client receives a SIP `BYE` packet, microVNF will reset this counter to zero. In this instance, the SIP session followed proper call establishment protocol, and therefore the value of this counter will be zero. This counter is the one that gets verified in the previous phase. Considering that the edge Ethernet switch has a limited amount of memory, it is necessary to use a data structure that is efficient for tracking each session that is generated by the IoT devices. For this operation, bloom filter [39] provides a space-efficient data structure which will be described in detail next.

B. Data Structure

MicroVNF uses Bloom filters to keep track of packets and how they affect its session state. Keeping track of a session requires inspecting the packets that are associated with that particular session, and this operation could potentially take up much space in memory. With a limited amount of memory available in the edge switch, it is important to select a data structure that is appropriate for this context.

MicroVNF selected the Bloom filter due to its space-efficient property for the insert and lookup operations, and this array is implemented as P4 registers, which are stateful memories that can be read and written by a P4 program [40]. Since Bloom filters are implemented as P4 registers, they can leverage low-latency components that are available on the data plane of hardware-based P4 switches. The Bloom filter does not store any actual information about the object itself, but rather, it stores whether or not it has seen this object previously. Since it does not store the details about each packet

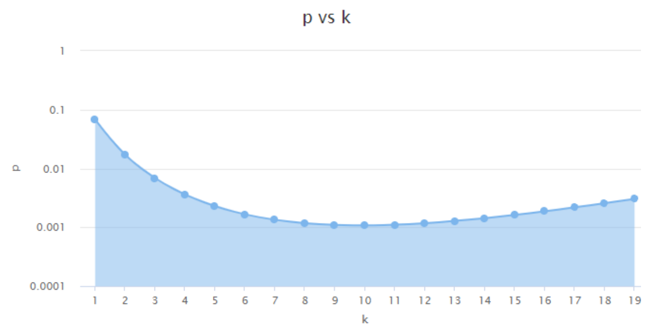


Fig. 5. Probability of false positives (p) versus the number hash functions (k) in the filter. [41].

and session, it is efficient in using memory space. Bloom Filters were originally used to check whether or not an element was a member of a given set of elements. In this context, a Bloom Filter is adapted as a Counting Bloom Filter (CBF) so that it can count the frequency of the packets that we are tracking. When the frequency exceeds a threshold, the packet is considered part of a malicious session, and it is dropped. In order to uniquely identify and track each session while preserving memory space, microVNF uses a hash function.

The hash function takes in four elements to generate a hash value that represents a particular packet. To strike a balance between being granular enough and not blocking legitimate sessions, we selected the following four pieces of information to describe a packet: the destination's IP address, the destination's UDP port, the physical port number where the packet arrived, and the first 6 bytes of the SIP header. This information is fed to a hash function to produce a hash value (10 binary digits, value range of 0 to 1023) and this is used as the index number to access the array. For example, a packet arrives with the following characteristics as shown in Fig. 6:

- Destination IP address: 49.21.4.2
- Destination Port: 5060
- Source Port: random
- Physical port on the edge switch: 12
- SIP header: INVITE

Our use of two hash functions follows the approach taken by the original Bloom Filter paper [39], and represents a trade-off between the false-positive rate and the load imposed on the switch's limited computing resources, particularly CPU and memory. While the use of more hash functions will reduce the chance of false-positives, the concern is that the additional load imposed might negatively impact the switch's normal operation. With this consideration, the number of hash function is kept at two to minimize the chance of adverse impact.

With a 1024 bit filter (`m`), 2 hash functions (`k`), and 72 evaluated items (`n`), this bloom filter will have 0.017 probability (`p`) of false positives (1 in 58) [41] as depicted in Fig. 5,which we deem acceptable for this proof of concept use case.

When these headers shown in Fig. 6 are fed to hash1 and hash2, it produced the values 231 and 489, respectively. Since

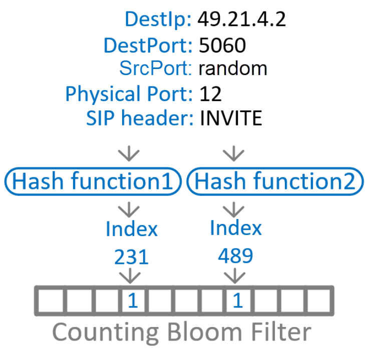


Fig. 6. Using counting bloom filter (CBF) data structure to keep track of the number of SIP packets. This packet is hashed to index 231 & 489. Since this is the first time the CBF encountered this packet, the value for index 231 & 489 is set to 1.

this is the first packet that microVNF sees, microVNF sets the value of element numbers 231 and 489 to 1. The next time microVNF sees the same packet and the same hash, microVNF will set the values to 2. In essence, microVNF uses each array element as a packet counter. In order to put them into a proper context, the packets need to be tracked together as a session.

MicroVNF tracks the session state by correlating the SIP packets' payload which indicates whether this is a SIP request or response. When inspecting a SIP payload, microVNF is checking for a SIP request and response in order to deduce the state of a SIP transaction. When a SIP client initiates a connection to the server, it goes through the following states: Calling, Proceeding, Completed, and Terminated as described in RFC3261 [42].

A SIP session starts when the client enters a calling state (i.e., the client sends an INVITE request), and is terminated when either side sends a BYE request. State tracking works by a simple algorithm: increase the counter when the IoT-device sends an INVITE request to the server, and decrease the counter when the IoT-device sends or receives a BYE request (which indicates completion of a SIP session). With a typical SIP session, the counter should be close to zero. In essence, the counter shows the number of SIP sessions that are in an open state. When a counter is increasing, it shows that the client keeps opening new SIP sessions, which is considered an anomaly from the SIP state machine perspective, and indicates an attack is taking place.

C. A walkthrough of packet and session tracking

In order to describe how these components work together, let us consider a walkthrough that demonstrates how packet counter and session tracking work together to detect and mitigate SIP DDoS attacks. In Fig. 7, an IP camera is connected to physical port number 3 on the edge switch. It sends a SIP INVITE packet to a remote SIP server, and the packet is received by the switch that has microVNF enabled. microVNF parses the headers and performs an inspection. From the inspection process, microVNF realises that this packet contains a SIP INVITE request. microVNF needs to track this packet due to the potential for a DDoS attack associated with it. The two hash functions generate two index numbers (3) and (7) for this particular packet based on four parameters: destination IP address, destination UDP port, 3 (the port number on the edge switch), and INVITE (the payload of SIP header). microVNF then sets the values of index 3 and index 7 locations to 1 to indicate that this is the first packet of this session. If the IP camera sends another packet that shares the same details (destination IP address, destination UDP port, port 3, and INVITE), the counters will be increased by one. Any variance on the four input elements will generate different index numbers. For example, a different destination IP address will generate different index numbers even though the packet is using the same destination UDP port. This process describes the 'northbound' process (from an IP camera to the SIP server). In the next section, we will look at the 'southbound' process (from the SIP server to the IP camera).

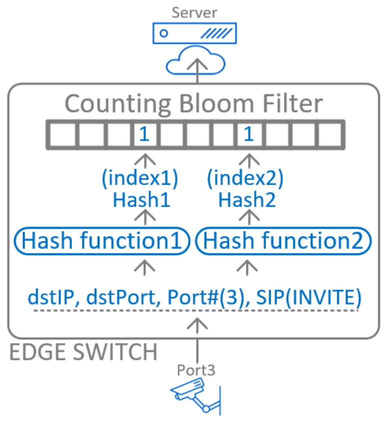


Fig. 7. Northbound: From camera to server: Counting Bloom Filter sets the number of active SIP session to 1. A botnet infection and DoS attack is suspected when this number keeps increasing.

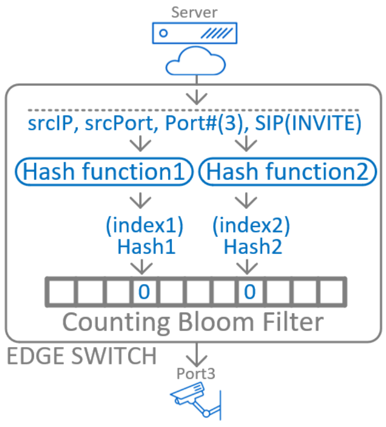


Fig. 8. Southbound: From server To camera: Counting Bloom Filter resets the number of active SIP session to 0.

When the SIP server sends a session termination request by sending a `BYE` packet to the IP camera, it will reset the counter at index 3 and 7 location, back to zero (Fig. 8). When the server sends a response to the original request, microVNF will use the same hash algorithms to generate the same index numbers. In order to generate the same index numbers as the original request, microVNF has to modify the input for the hash function so that it matches with the original request. In this process, microVNF substitutes the source IP address with the destination IP address. microVNF also substitutes the destination UDP port with the source UDP port. This step is important because we need to ensure that the hash generates the same index so that it can reset the correct counters in the CBF array.

microVNF compares these counters during the validation phase against a predetermined threshold for a pass or drop decision. As part of the validation phase, microVNF retrieves the counter from CBF to compare it against a threshold value that has been set by the administrator. For example, the administrator could have set a rule to prevent a DDoS attack, which states that for every switch port, microVNF will only allow up to 3 concurrent SIP sessions at any given time. With this rule in place, microVNF will allow the client to initiate 3 SIP sessions. When it tries to open the fourth session, microVNF detects that it has exceeded the threshold and drops the INVITE packet.

Besides tracking the SIP state for a DDoS attack as described in the previous example, the same concept applies for detecting other types of attacks such as a scanning attack, enumeration attack, and dictionary attack (also known as a brute-force attack).

D. Implementation

This section describes the way that the experiments were implemented in the Amazon Web Services (AWS) environment. The experiments used the EC2 t2.micro, which launches Mininet software to create a virtual network environment that consists of a virtual switch and virtual hosts. Mininet created several virtual hosts that were connected to a virtual switch. This level of programmability allows for a flexible and powerful way to create different networking environments for different experiments. For example, to emulate an attacker with ten potential targets, we could write a Python script that instantiates 11 virtual hosts, 1 for the attacker, and 10 for the potential target hosts, as depicted in Fig. 9.

The virtual switch instance created by mininet could either use the default soft switch (Open vSwitch), or a soft switch that supports programmable data plane (bmv2). This level of flexibility is important because we need to be able to easily swap the two switches depending on the experiment that we would like to perform. The programmable data plane is critical because it allows us to experiment with VNF. In this case, we are building a VNF to secure the edge of IoT networks against botnet attacks. In this environment, the botnets and the victims are simulated by the virtual hosts.

For the experiments that follow, microVNF is implemented purely in the P4 programming language, while the environment is launched by a Python script. Chronologically, a Python script launches a mininet environment, which consists

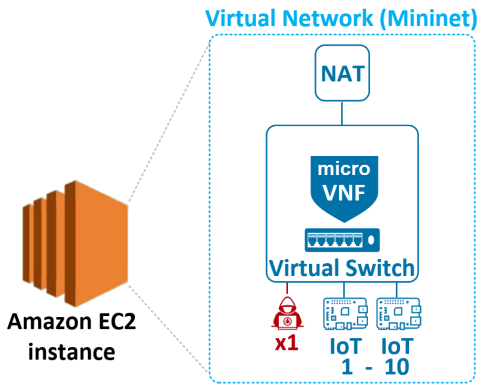


Fig. 9. A virtual network environment running on an AWS EC2 instance that consists of a virtual switch and 11 virtual hosts (1 for attacker and 10 for IoT devices).

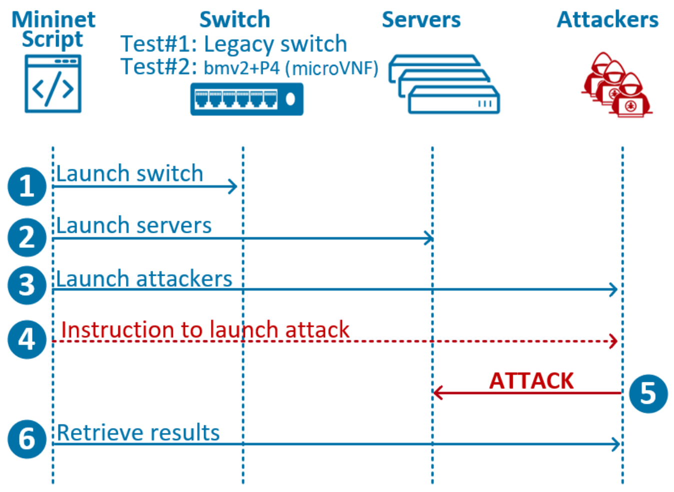


Fig. 10. Sequence and flow diagram of the experiments

of virtual switches and virtual hosts (some are designated as the attackers while some are designated as the victims). Next, the Python script sends instructions to the attacker hosts to initiate a specific attack against the victims. On the other hand, the P4 language is used to implement microVNF, which is loaded when the P4 switch is initially launched.

IV. Experiments

All of the experiments in this section followed the same sequence as depicted in the flow diagram (Fig. 10). A mininet script orchestrated the experiment by launching a virtual switch, the servers, and the attackers, it then sent the instruction to the attackers to start the attack, and then retrieved the results from the attacker's perspective. The only variable that differed was the independent variable, which in this case is the type of switch being used, i.e., Test#1: Legacy switch (ovs) vs. Test#2: bmv2+P4 (microVNF). The ovs is used to represent a non-P4 switch that does not support a programmable data plane. This experiment is designed to provide a comparison and a contrast between the performance of a non-P4 legacy switch vs. a P4 switch with microVNF. Comparing the results from the two tests provides quantitative data for evaluating the effectiveness of our proposed approach.

The experiments that follow are organized around preventive (before infection) and reactive measures (after infection). These experiments are designed to follow the way a botnet attack progresses, from discovering vulnerable devices to hijacking the device and using the device to launch DDoS attacks. Table I lists the experiments for each phase.

TABLE I
Experiments for Before and After Infection

Preventive Measures (Before recruited by a botnet)	**Reactive Measures** (After became a botnet)
1. SIP scanning attack	4. CNC registration attempts
2. SIP enumeration attack	5. SIP DoS attack
3. SIP brute force attack	6. SIP DDoS attack

A. Experiment 1: SIP scanning attack

The first experiment focuses on the question: how can an adversary be detected and mitigated from discovering SIP servers as their potential victims? If microVNF is working as designed, then the adversaries will only find relatively few potential victims.

1) Description: The first experiment performed two tests: with a legacy switch and with microVNF running on a switch with a programmable data plane as shown in Fig. 11.

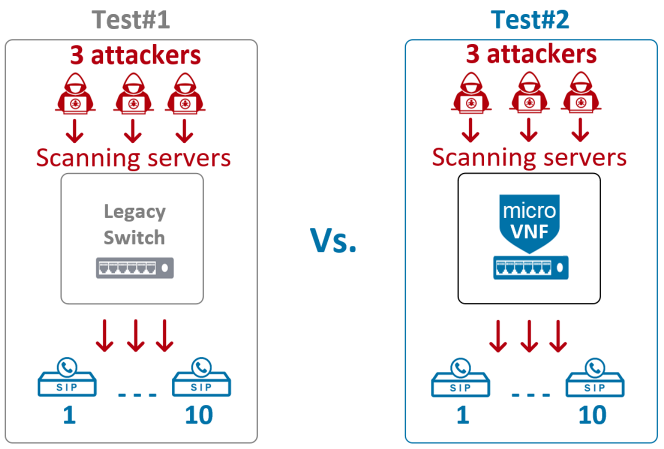


Fig. 11. **Experiment 1:** Attackers scanning for SIP servers (Test1 with legacy switch vs. Test2 with microVNF)

Both tests were conducted in a mininet virtual network environment with 13 hosts connected to a switch. As depicted in Fig. 10, the mininet script performed the following tasks in this sequence: launched a switch (1), launched SIP service on 10 servers (2), launched three attackers (3), sent a command to the attacker to initiate attack against the servers (4). The attacks were carried out in sequence, i.e., Attacker1 first, followed by Attacker2, and lastly, Attacker3. The mininet script then collected the results from each attacker (5).

The mininet script started the process by launching a virtual network environment which consisted of a virtual switch with 10 SIP servers and 3 attackers that were connected to the same switch. It assigned a random IP address to each host where all 13 hosts were on the same LAN. This method achieved a double-blind effect, where the IP addresses and the role (victim or attacker) for each IP address were not initially known. This is to ensure that the proposed solution is repeatable, accurate, and reliable for any IP addresses. Next, the script used the `net.pingAll()` function to confirm that all hosts were reachable via IP address. The script then moved on to the next task of launching the experiment.

The script started the next phase by launching a SIP-server process on all 10 servers (h1 - h10). The script did this by issuing a command "`sipp -sn uas`" [43] to launch sipp services in User Agent Server (uas) mode. At this stage, all 10 servers were ready to accept incoming SIP requests from clients.

Next, the script initiated an attack from each attacker host (h11, h12, then h13) to ensure that the results show a consistent pattern. The script triggered the attacker to start a SIP scanning attack by scanning 10.1.1.1 - 10.1.1.250 to find SIP servers that were available within this range. The attacker used "`sipvicious_svmap 10.1.1.1-10.1.1.250`" command [44] which sent a SIP OPTION packet to the servers. As per RFC3261, a SIP Server is supposed to send a response when it received this packet. In this experiment, all 10 SIP servers sent a response to the attacker. From these responses, the attackers found out that there were 10 SIP servers available on the network. To validate this result, the same process was repeated three times.

The only difference between Test#1 and Test#2 in this experiment was the switch type being used. Test#1 used the default switch that comes with mininet (OVS), whereas for Test#2, it launched a bmv2 switch and loaded the microVNF.p4 program. microVNF imposed a limit to the number of unacknowledged `SIP OPTIONS` packets that each port could send, set to three (a variable threshold that can be adjusted), as legitimate SIP clients do not typically scan a range of IP addresses with this packet. When there are three or more unacknowledged packets, microVNF recognized these packets as malicious and automatically dropped the rest of the `OPTIONS` packets coming from the same port.

TABLE II
EXPERIMENT 1: RESULT: MICROVNF BLOCKS SIP SCANNING ATTACK (3 DISCOVERED VS. 10)

Experiment 1: SIP Scanning Attack	Legacy Switch		**microVNF**	
	No. of servers available	No. of servers discovered	No. of servers available	No. of servers discovered
Attacker #1	10	10	10	3
Attacker #2	10	10	10	3
Attacker #3	10	10	10	3

2) Result: The three attacks performed by three different attackers (h11, h12, and h13) found the same number of servers for each test. During the test, each attacker operated independently without awareness of other attackers being present. The results are consistent that these attackers found the same servers. Table II shows the result from Attacker1's perspective for Test#1 with the legacy switch and Test#2 with microVNF (bmv2+P4). For Test#1, all attackers discovered 10 SIP servers, whereas for Test#2, they only discovered 3 SIP servers (Table II). Below is a screenshot of CPU Utilization from legacy switch (Fig. 12) and microVNF (Fig. 13) during the experiment.

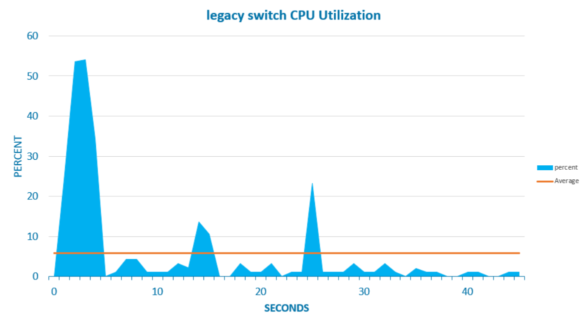


Fig. 12. **Experiment 1: Test1:** CPU utilisation of legacy switch.

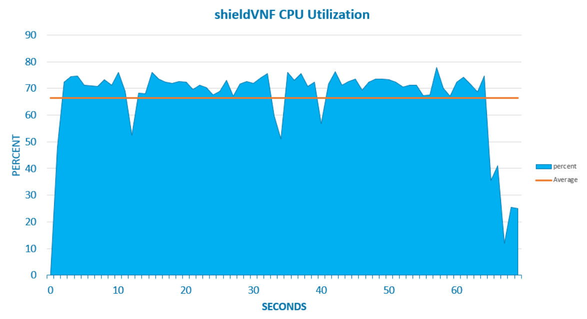


Fig. 13. **Experiment 1: Test2:** CPU utilisation of microVNF.

3) Discussion: With a legacy switch, the attackers were able to launch a SIP scan attack by sending `SIP OPTION` request to each IP address that they could find. TableII shows that the attackers were able to discover all ten SIP servers. With all the targets discovered, the attackers would be able to proceed with further attacks.

With microVNF, on the other hand, there was a rule imposed on each port where it limited the number of `SIP OPTION` packets to three. As a result, each attacker could only discover the first three SIP servers, as shown in Table II. This outcome presents evidence that microVNF was able to detect and mitigate the SIP scanning attack.

From CPU utilization perspective, microVNF consumed more CPU (Fig. 12) than the legacy switch (Fig. 13). During the attack, microVNF showed a sustained level of high CPU consumption. This is expected due to detection (deep packet inspection) and mitigation functions that were performed by microVNF on these malicious packets. In contrast, the legacy switch was not aware that these were malicious packets and processed each one as usual by forwarding it from the source port to the destination port. As such, it did not show significant CPU consumption.

B. Experiment 2: SIP enumeration attack

The second experiment focused on the question: how can an adversary be detected and mitigated from enumerating valid SIP accounts on a SIP server? If microVNF is working as designed, then the adversaries will only find some of the valid SIP accounts.

1) Description: Similar to the first experiment, the sequence follows the same steps as depicted in Fig. 10 but with a different attack scenario, i.e., a SIP enumeration attack. The objective of this test is for the attacker to discover valid SIP accounts that exist on the SIP server. The attacker achieves this by enumerating possible SIP accounts and looking for responses from a valid account, as depicted in Fig. 14.

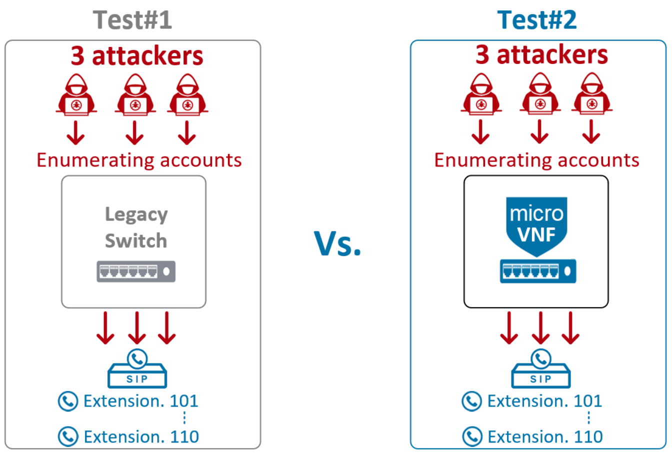


Fig. 14. **Experiment 2:** Attackers scanning for valid accounts/extension on a SIP Proxy server (Test1 with legacy switch vs. Test2 with microVNF)

The mininet script started the process by launching a virtual network environment which consists of a virtual switch with 1 Asterisk SIP server (with valid extensions ranging from 100 - 110) and 3 attackers that were connected to the same switch. The script assigned a random IP address to each host where all 4 hosts were on the same LAN. Next, the script started an Asterisk server process using the (`/usr/sbin/asterisk`) command. At this stage, the SIP server was ready to accept incoming SIP requests from clients.

The next stage is when the script instructed the attackers to start a SIP enumeration attack by probing the SIP servers for valid extensions. The attacker used the “`sipvicious_svwar -m INVITE -e100-999 <SIP server IP address>`” command [44] which sent SIP INVITE packets to the server's IP address, probing from extension 100 to 999. Since the server's IP address is randomly generated, we had to manually verify the server's IP address prior to using this command. According to RFC3261, when a SIP server receives a SIP `INVITE` request for an extension that does not exist, the SIP server is supposed to respond with a `"404 Not Found"` packet. On the other hand, when the client sends the invite to a valid extension, the server is supposed to respond with a `"401 Unauthorized"` packet. Each attacker sends invite packets to a range of extensions, e.g., 101, 102, 103, and takes note of the response (404 vs. 401). From these responses, the attackers established that there were 11 valid extensions available on this particular SIP server. To verify this result, the same process was repeated three times.

MicroVNF imposed a limit on the number of unacknowledged `SIP INVITE` packets that each port could send, to three (a variable threshold that can be adjusted). A legitimate SIP client already has an extension assigned so it does not typically probe the server for a range of extensions. Anomalous behavior such as probing for valid extensions is considered malicious and microVNF dropped the rest of the `INVITE` packets coming from the same port. In this instance, microVNF is taking a similar response as described in RFC3261, where the server may drop the request instead of responding with a ”503 Service Unavailable” response code.

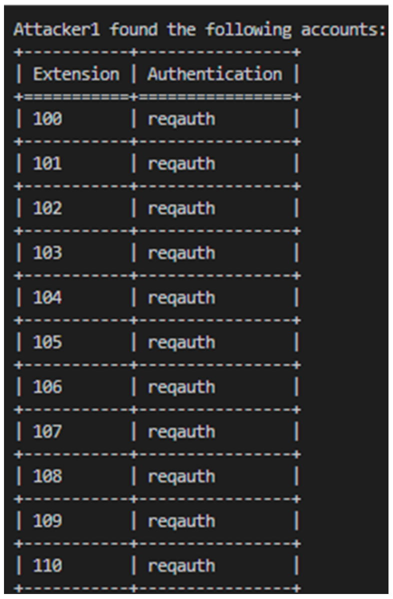
TABLE III
EXPERIMENT 2: RESULT: MICROVNF BLOCKS SIP ENUMERATION ATTACK (3 DISCOVERED VS. 11)

Experiment 2: SIP Enumeration Attack	Legacy Switch		**microVNF**	
	No. of accounts available	No. of accounts discovered	No. of accounts available	No. of accounts discovered
Attacker #1	11	11	11	3
Attacker #2	11	11	11	3
Attacker #3	11	11	11	3

2) Result: Table III shows the number of valid extensions discovered. With Test#1 (legacy switch) each attacker found 10 valid extensions while with Test#2, each attacker found 3 valid extensions. Below is a screenshot from the perspective of the attacker. Figure 15 shows the outcome with legacy switch in Test#1, while Fig. 16 shows the outcome with microVNF in Test#2. For CPU utilization during the experiment, below is a screenshot during Test#1 with legacy switch (Fig. 17) and Test#2 with microVNF (Fig. 18).

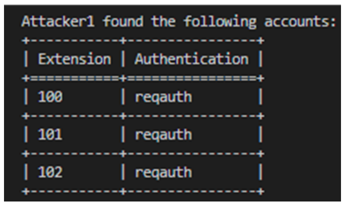
3) Discussion: With a legacy switch, the attackers were able to find extensions 100-110 that were on the SIP Proxy server (Fig. 15), whereas the attacker only found three extensions (100-102) with microVNF (Fig. 16). In this instance, microVNF detected that `INVITE` packets were received repeatedly from the same port. Even though the server has responded with either a `401` or `404`, microVNF was specifically looking for a `200` response from the server, which indicates that the request has succeeded according to RFC3261 [42]. In absence of this specific response, the attacker hit a threshold limit of three `INVITE` packets, and therefore, microVNF dropped the remaining packets. As a result, the attackers only found the first three accounts when they started the probe at extension 100.

The CPU consumption between the two tests provides evidence of this activity. With the legacy switch, the attacker



```
Attacker1 found the following accounts:
+-----------+----------------+
| Extension | Authentication |
+===========+================+
| 100       | reqauth        |
+-----------+----------------+
| 101       | reqauth        |
+-----------+----------------+
| 102       | reqauth        |
+-----------+----------------+
| 103       | reqauth        |
+-----------+----------------+
| 104       | reqauth        |
+-----------+----------------+
| 105       | reqauth        |
+-----------+----------------+
| 106       | reqauth        |
+-----------+----------------+
| 107       | reqauth        |
+-----------+----------------+
| 108       | reqauth        |
+-----------+----------------+
| 109       | reqauth        |
+-----------+----------------+
| 110       | reqauth        |
+-----------+----------------+
```

Fig. 15. **Experiment 2: Test1 Result:** SIP Enumeration attack with legacy switch. Screenshot was taken from Attacker1. Attacker2 and Attacker3 also found 11 SIP accounts.



```
Attacker1 found the following accounts:
+-----------+----------------+
| Extension | Authentication |
+===========+================+
| 100       | reqauth        |
+-----------+----------------+
| 101       | reqauth        |
+-----------+----------------+
| 102       | reqauth        |
+-----------+----------------+
```

Fig. 16. **Experiment 2: Test 2 Result:** SIP Enumeration attack with microVNF. The screenshot was taken from Attacker1. Attacker2 and Attacker3 also found 3 SIP accounts.

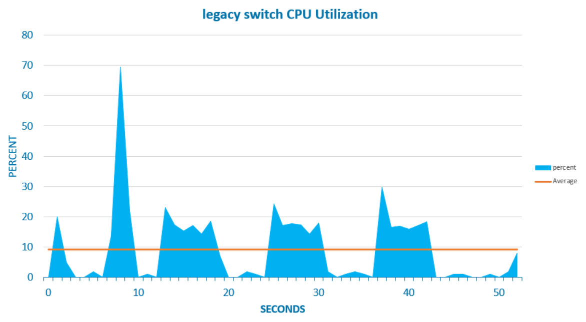


Fig. 17. **Experiment 2: Test1:** CPU utilisation of legacy switch.

was able to send more malicious packets towards the target. This activity was represented as activity that lasted for about 7 seconds for each attack (Fig. 17). In contrast, microVNF recognized malicious activities with the first few packets and dropped the rest. This is represented on the CPU utilization diagram (Fig. 18) as a slimmer peak (as compared to the peak with legacy switch), or a shorter period of activity (for about 3 seconds).

With fewer extensions discovered, microVNF reduced the attack surface and the potential for an attack. In a real-world implementation, a SIP Proxy server would typically host hundreds of extensions. Reducing the number of discoverable extensions down to three would minimise the chance of other extensions of being susceptible to a brute force attack. This outcome presents evidence that microVNF was able to detect and mitigate the SIP enumeration attack.

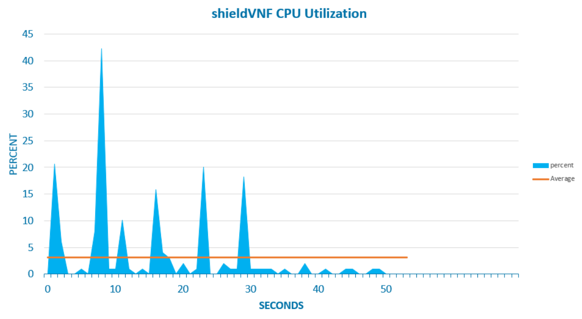


Fig. 18. **Experiment 2: Test2:** CPU utilisation of microVNF.

C. Experiment 3: SIP brute force attack

The third experiment focuses on the question: how can an adversary be detected and mitigated from brute-forcing the passwords of SIP accounts? If microVNF is working as designed, then the adversaries should not be able to crack any passwords.

1) Description: The same sequence is followed for this experiment as depicted in Fig. 10 and the attack scenario used in this experiment is a SIP brute force attack. Since valid accounts were discovered in the previous experiment, the next step is to brute force the accounts to gain entry. Similar to the previous experiment, this experiment performed two tests: the first with a legacy switch and the second with microVNF. The objective of this test is to use brute force to discover the password for an account. The attacker achieved this by using the `SIPVicious_svcrack` tool with a dictionary file that contains a list of popular passwords, as depicted in Fig. 19.

The mininet script started the same environment as in the previous experiment, which consists of an Asterisk SIP server and 3 attackers connected to the same switch. Next, the script instructed the attackers to start a SIP brute force attack by repeatedly sending `REGISTER` packets with the extension and password. For incorrect passwords, the SIP Proxy server responds by sending a `401 Unauthorized` packet. The attacker would repeat this process by trying to register again with a new password from the dictionary file until the file is exhausted. If the password is correct, the SIP Proxy server responds by sending a `200 OK` packet.

As per instruction from the mininet script, the attacker was using the "`sipvicious_svcrack -u x -d dictionary.txt y.y.y.y`" command [44] to launch the attack, where `x` was the extension (101-110) and `y.y.y.y` was the IP address of the SIP server. When attacker#1 had completed this process, attacker#2 started the brute force process. Finally, attacker#3 performed the same

TABLE IV
EXPERIMENT 3: RESULT: MICROVNF BLOCKS SIP BRUTE FORCE ATTACK (0 CRACKED VS. 10)

Exp 3: SIP Brute Force Attack	Legacy Switch		microVNF	
	# of accts with password	# of password cracked	# of accts with password	# of password cracked
Attacker #	10	10	10	0
Attacker #	10	10	10	0
Attacker #	10	10	10	0

attack against the Asterisk SIP server. The dictionary file used in this experiment is one that is commonly used by attackers, including a real password cracker tool called John the Ripper (JtR) [45], which contains 3107 commonly used passwords.

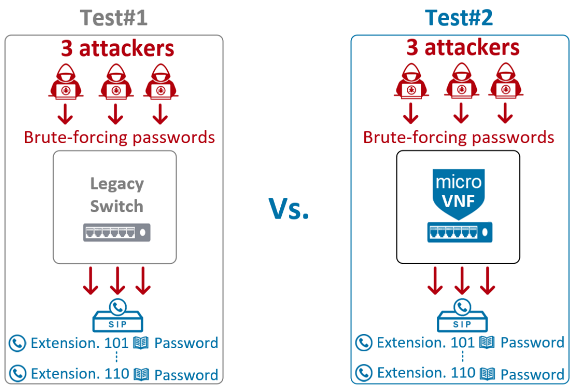
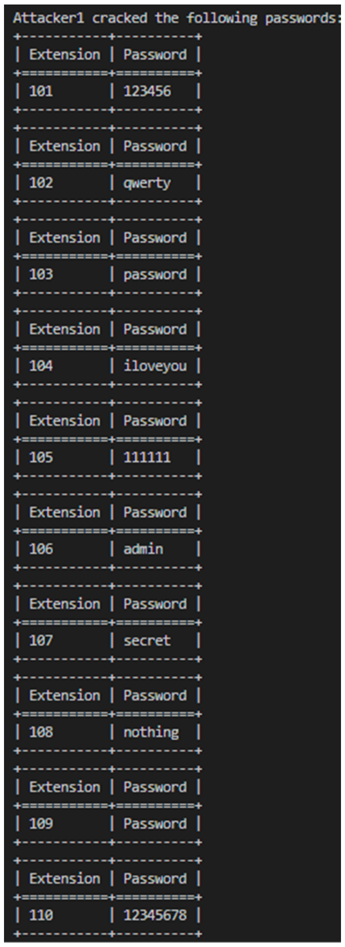


Fig. 19. **Experiment 3:** Attackers brute forcing the extensions on a SIP server for valid password (Test1 with legacy switch vs. Test2 with microVNF)

2) Result: Table IV shows the number of extensions that were cracked during the experiment. These passwords were manually assigned to these extensions to simulate a scenario where users tend to use weak passwords. Below is a screenshot from the result of the two tests. Figure 20 shows the outcome with the legacy switch, while Fig. 21 shows the outcome with microVNF. Figure 22 shows CPU utilization of the legacy switch during the experiment, while Fig. 23 shows CPU utilization for microVNF.

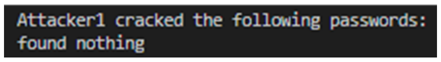
3) Discussion: With a legacy switch, the attackers were able to crack passwords for extensions 101-110 on the SIP proxy server. Figure 20 shows the 10 SIP accounts (extensions) that exist on the server, along with their current password. In contrast, when the same attack was performed using microVNF, the attacker did not crack any password (as shown in Fig. 23). This is due to microVNF recognizing the repeated registration attempts, coming from the same port, as malicious.

MicroVNF restricts multiple SIP registration attempts to three. When this threshold is exceeded, microVNF recognizes this pattern as malicious and drops the subsequent packets. Since the attacker was not successful in their first three attempts in guessing the password, they did not find anything. This outcome presents evidence that microVNF was able to detect and mitigate SIP brute force attacks.



```
Attacker1 cracked the following passwords:
+-----------+----------+
| Extension | Password |
+===========+==========+
| 101       | 123456   |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 102       | qwerty   |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 103       | password |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 104       | iloveyou |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 105       | 111111   |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 106       | admin    |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 107       | secret   |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 108       | nothing  |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 109       | Password |
+-----------+----------+
+-----------+----------+
| Extension | Password |
+===========+==========+
| 110       | 12345678 |
+-----------+----------+
```

Fig. 20. **Experiment 3: Test1 Result:** SIP brute force attack with legacy switch. Screenshot was taken from Attacker1. Attacker2 and Attacker3 also cracked 10 passwords.



```
Attacker1 cracked the following passwords:
found nothing
```

Fig. 21. **Experiment 3: Test2 Result:** SIP brute force attack with microVNF.Screenshot was taken from Attacker1. Attacker2 and Attacker3 also cracked nothing.

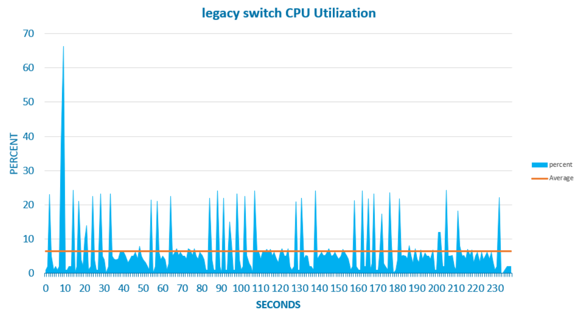


Fig. 22. **Experiment 3: Test1:** CPU utilisation of legacy switch.

The CPU utilization of microVNF in Fig. 23 shows that it was busy during the initial detection and mitigation period.

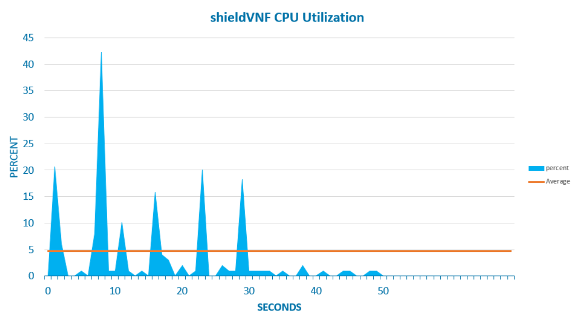


Fig. 23. **Experiment 3: Test2:** CPU utilisation of microVNF.

Afterwards, CPU utilization on microVNF did not show much activity. In contrast, CPU utilization from the legacy switch in Fig. 22 shows constant activities during the experiment as the attacker was able to continue brute forcing the password during the entire test period.

D. Experiment 4: CNC Channel Establishment

The fourth experiment considers: how can a botnet be prevented from registering with its CNC server? If microVNF is working as designed, then the botnet will not be able to register with its CNC server.

1) Description: Similar to previous experiments, this experiment performed two tests to contrast the different outcomes when using a legacy switch vs. microVNF. The objective of this test is to block bot communication with its central command and control (CNC) server, as depicted in Fig. 24. This blocking step is important to prevent the botmaster from being able to control the bot and, therefore, renders the bot non-functional for the intended purpose of launching an attack. If the bot is unable to connect to the CNC server and receive commands, it is effectively neutralised and does not pose an immediate threat.

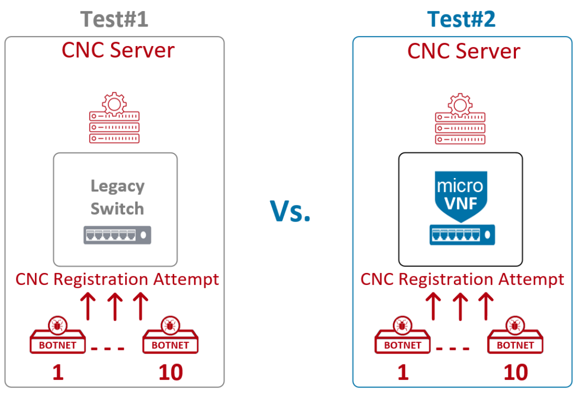
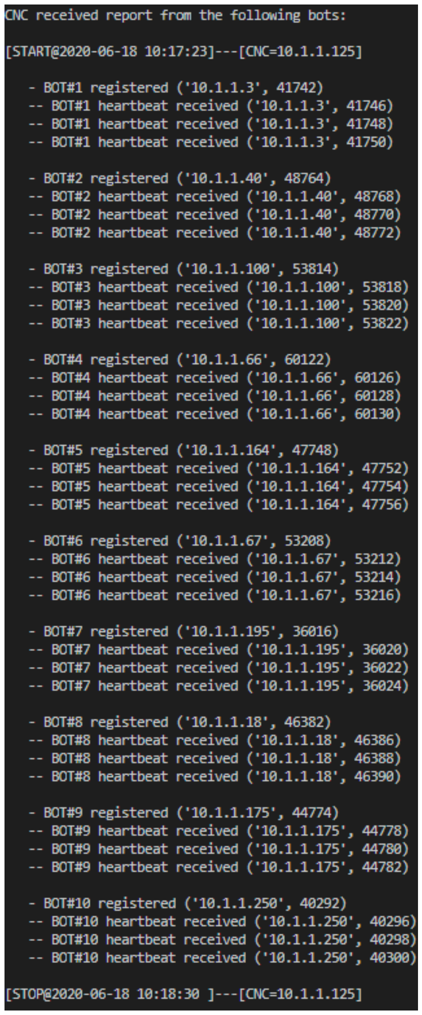


Fig. 24. **Experiment 4:** IoT-turned-to-botnet attempting to register to the centralised Command and Control (CNC) server (Test1 with legacy switch vs. Test2 with microVNF)

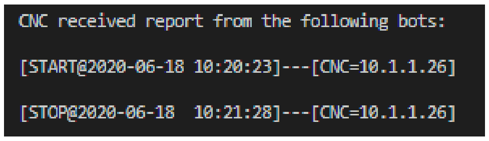
In this experiment, we will be looking at the way the Mirai botnet and its CNC operate so that we can test the hypothesis that microVNF can detect and mitigate CNC channel establishment activities. When Mirai has successfully infected an IoT device, the new bot will attempt to connect to the CNC server using Telnet (TCP/23) with a specific payload. The first packet will contain \x00000001 and the second packet will contain \x00. At this stage, the bot has registered with the CNC. From this point on, the bot and CNC will maintain a regular heartbeat every 60 seconds with the payload content of \x0000 [2].

A Python script launched a mininet environment that consists of one host as the CNC server and ten hosts as the Mirai botnet. Similar to previous experiments, IP addresses and role assignments were generated at run time to remove bias and ensure that the solution works for any IP addresses.



```
CNC received report from the following bots:

[START@2020-06-18 10:17:23]---[CNC=10.1.1.125]

   - BOT#1 registered ('10.1.1.3', 41742)
  -- BOT#1 heartbeat received ('10.1.1.3', 41746)
  -- BOT#1 heartbeat received ('10.1.1.3', 41748)
  -- BOT#1 heartbeat received ('10.1.1.3', 41750)

   - BOT#2 registered ('10.1.1.40', 48764)
  -- BOT#2 heartbeat received ('10.1.1.40', 48768)
  -- BOT#2 heartbeat received ('10.1.1.40', 48770)
  -- BOT#2 heartbeat received ('10.1.1.40', 48772)

   - BOT#3 registered ('10.1.1.100', 53814)
  -- BOT#3 heartbeat received ('10.1.1.100', 53818)
  -- BOT#3 heartbeat received ('10.1.1.100', 53820)
  -- BOT#3 heartbeat received ('10.1.1.100', 53822)

   - BOT#4 registered ('10.1.1.66', 60122)
  -- BOT#4 heartbeat received ('10.1.1.66', 60126)
  -- BOT#4 heartbeat received ('10.1.1.66', 60128)
  -- BOT#4 heartbeat received ('10.1.1.66', 60130)

   - BOT#5 registered ('10.1.1.164', 47748)
  -- BOT#5 heartbeat received ('10.1.1.164', 47752)
  -- BOT#5 heartbeat received ('10.1.1.164', 47754)
  -- BOT#5 heartbeat received ('10.1.1.164', 47756)

   - BOT#6 registered ('10.1.1.67', 53208)
  -- BOT#6 heartbeat received ('10.1.1.67', 53212)
  -- BOT#6 heartbeat received ('10.1.1.67', 53214)
  -- BOT#6 heartbeat received ('10.1.1.67', 53216)

   - BOT#7 registered ('10.1.1.195', 36016)
  -- BOT#7 heartbeat received ('10.1.1.195', 36020)
  -- BOT#7 heartbeat received ('10.1.1.195', 36022)
  -- BOT#7 heartbeat received ('10.1.1.195', 36024)

   - BOT#8 registered ('10.1.1.18', 46382)
  -- BOT#8 heartbeat received ('10.1.1.18', 46386)
  -- BOT#8 heartbeat received ('10.1.1.18', 46388)
  -- BOT#8 heartbeat received ('10.1.1.18', 46390)

   - BOT#9 registered ('10.1.1.175', 44774)
  -- BOT#9 heartbeat received ('10.1.1.175', 44778)
  -- BOT#9 heartbeat received ('10.1.1.175', 44780)
  -- BOT#9 heartbeat received ('10.1.1.175', 44782)

   - BOT#10 registered ('10.1.1.250', 40292)
  -- BOT#10 heartbeat received ('10.1.1.250', 40296)
  -- BOT#10 heartbeat received ('10.1.1.250', 40298)
  -- BOT#10 heartbeat received ('10.1.1.250', 40300)

[STOP@2020-06-18 10:18:30 ]---[CNC=10.1.1.125]
```

Fig. 25. **Experiment 4: Test1 Result:** Bot registration attempt with legacy switch. 10 Bots registered at the Command-and-Control (CNC) server.



```
CNC received report from the following bots:

[START@2020-06-18 10:20:23]---[CNC=10.1.1.26]

[STOP@2020-06-18  10:21:28]---[CNC=10.1.1.26]
```

Fig. 26. **Experiment 4: Test2 Result:** Bot registration attempt with microVNF. 0 Bot registered at the Command-and-Control (CNC) server.

Instead of using a real Mirai botnet binary, CNC and Mirai simulators (a script written in Python) were used to simulate the communication between botnet and CNC. The simulation is using the exact same payload and pattern as the real botnet communication.

TABLE V
EXPERIMENT 4: RESULT: MICROVNF BLOCKS CNC ESTABLISHMENT ATTEMPT (0 REGISTERED VS. 10)

Experiment 4: CNC Establishment Attempt	Legacy Switch	microVNF
	Number of Bots registered	Number of Bots registered
Bot #1	10	0
Bot #2	10	0
Bot #3	10	0

2) *Result:* Figures 25 and 26 show the result of the two tests. Figure 25 shows the outcome with a legacy switch where each bot was able to register with the CNC server and the CNC server was able to receive the heartbeat packets from the bot. In contrast, Fig. 26 shows the outcome with microVNF where none of the bots were able to register with the CNC server.

The CPU utilization shows different patterns of activity. With the legacy switch, each bot was able to run its complete course of action, i.e., registration, followed by heartbeat. This is shown in Fig. 27 with longer periods or ”valleys” between peaks. In contrast, microVNF recognized these malicious activities from the first few packets, shown in Fig. 28 by shorter gaps between peaks.

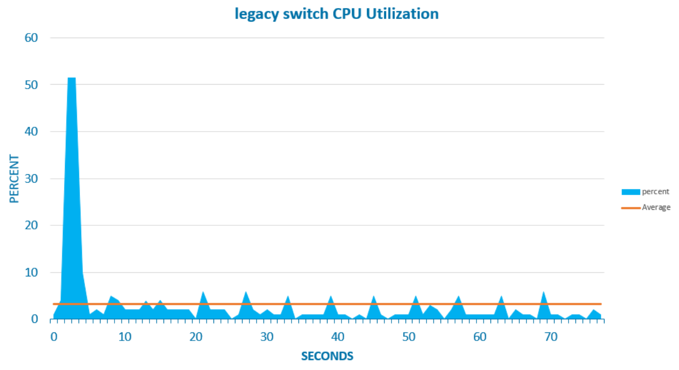


Fig. 27. **Experiment 4: Test1:** CPU utilisation of legacy switch.

3) *Discussion:* With a legacy switch, all bots were able to connect to the CNC server, as well as maintain an ongoing heartbeat. Fig. 25 shows the perspective from the CNC-side that counts how many bots were successfully registered, and the number of heartbeat packets received from each bot. With microVNF, the bots were unable to connect to the CNC, and therefore, as shown in Fig. 26, none of the bots were registered.

When the bot failed to register, the CNC would be unable to send a command for the bots to launch a DoS attack. This outcome presents evidence that microVNF was able to detect and prevent bot registration to the central CNC server.

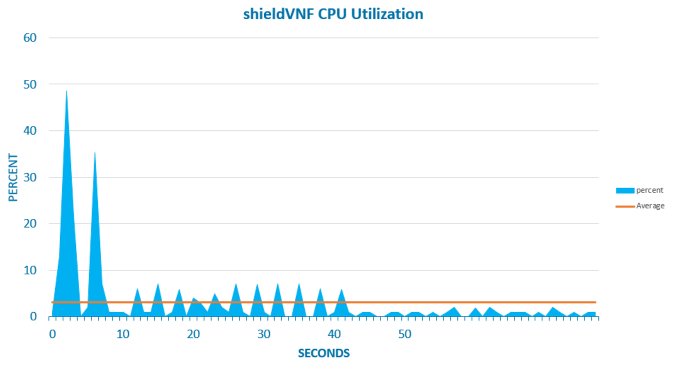


Fig. 28. **Experiment 4: Test2:** CPU utilisation of microVNF.

E. Experiment 5: SIP DoS attack

The fifth experiment considers: how can SIP DoS attacks from IoT botnets be detected and mitigated? If microVNF is working as designed, then a legitimate user is still able to make SIP calls and malicious packets are dropped.

1) *Description:* This experiment modifies the sequence slightly by adding client calls into the environment to measure whether the proposed solution can withstand a DoS attack. It is assumed that the attacker is able to get past the mitigation from previous experiments. At this stage, the attacker has successfully recruited the IoT device to be part of the botnet and is ready to launch a SIP DoS attack. The hypothesis for this experiment is that, microVNF can detect and mitigate a SIP DoS attack from a single attacker, as shown in Fig. 29.

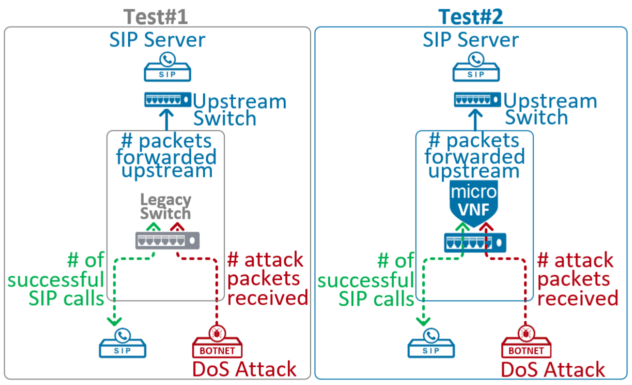

Test#1
SIP Server
SIP
Upstream Switch
# packets forwarded upstream
Legacy Switch
# of successful SIP calls
# attack packets received
SIP
BOTNET
DoS Attack
Test#2
SIP Server
SIP
Upstream Switch
# packets forwarded upstream
micro VNF
# of successful SIP calls
# attack packets received
SIP
BOTNET
DoS Attack

Fig. 29. **Experiment 5:** SIP DoS attack from a single attacker (Test1 with legacy switch vs. Test2 with microVNF)

As depicted in Fig. 30, the experiments were set up with SIP-client1 calling SIP-client2 (via the SIP Server) at one call per second. In the absence of any distraction, SIP-client1 would be able to complete 60 calls in a minute. To observe the disruption that can result from a SIP DoS attack, an attacker launched a SIP DoS attack against the same PBX that SIP-client1 was using.

The test environment had four hosts: SIP-client1, SIP-client2, PBX, and the Bot (atacker in Fig. 30). SIP-client1 represents a legitimate user that is calling SIP-client2 via the

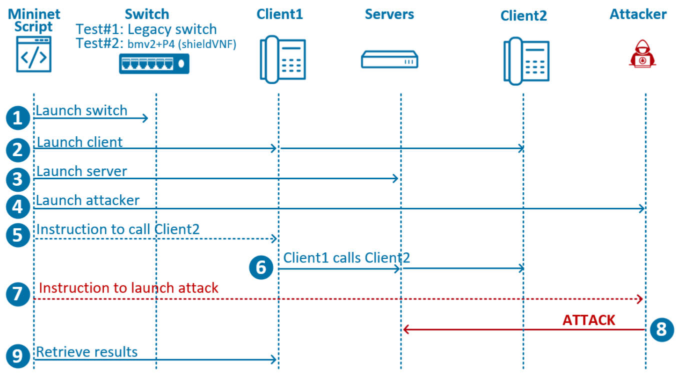


Fig. 30. Sequence and flow diagram of SIP DoS & DDoS experiment

TABLE VI
EXPERIMENT 5: RESULT: MICROVNF BLOCKS SIP DOS ATTACK (60 SUCCESSFUL CALLS VS. 20)

Exp 5: SIP DoS Attack	Legacy Switch			microVNF		
	# of attack pkts sent	# of pks fwd up stream	# of calls	# of attack pkts rcvd	# of pkts fwd up stream	# of calls
Caller	69	NA	20	184	NA	60
Attkr1	4,000,182	4,000,120	NA	4,000,073	228	NA

corporate phone system (PBX). A bot is on the same virtual-switch as SIP-client1. The process started with SIP-client1 making calls at a rate of 1 call per second to SIP-client2. At the 20th second after SIP-client1 started making calls, the bot started a SIP DoS attack against the PBX.

The experiment performed two tests for the same use case, one with a legacy switch, and the other with microVNF. The test script generated random IP addresses for the SIP-client1 and the bot at run time. Inviteflood (an attack tool) [46] generated a SIP DoS attack. This tool is popular and widely accessible as part of the Kali Linux distribution. In this experiment, at the 20th second, after SIP-client1 started SIP calls, inviteflood [46] sent 4 million `INVITE` packets to the same PBX (192.168.1.200), that SIP-client1 was using.

2) Result: The comparison between legacy vs. microVNF can be seen from three perspectives: SIP client, network switch, and CPU utilization.

From the SIP client's perspective on the legacy switch, Fig. 31 shows a total of 20 successful calls made to a SIP server. At SIP-client1, Fig. 33 shows a total of 60 successful calls made while the SIP client was connected to microVNF.

From the switch's perspective as shown in Fig. 32, the attacker generated 4,000,120 packets and these were received by the switch at port s1-eth2. The legacy switch forwarded all these packets upstream via port nat0-eth. The 62 packets that were not generated by attacker1 was attributed to switch-to-switch traffic that is not the focus of this experiment. However, Fig. 34 shows that while microVNF received about the same number of malicious packets (4,000,073), in contrast, microVNF dropped most of these packets and only forwarded legitimate packets (228) upstream via port nat0-eth.

From the CPU utilization perspective, microVNF had the highest peak reaching 80% as shown in Fig. 36, whereas for the legacy switch, the peak was at 30% (Fig. 31).

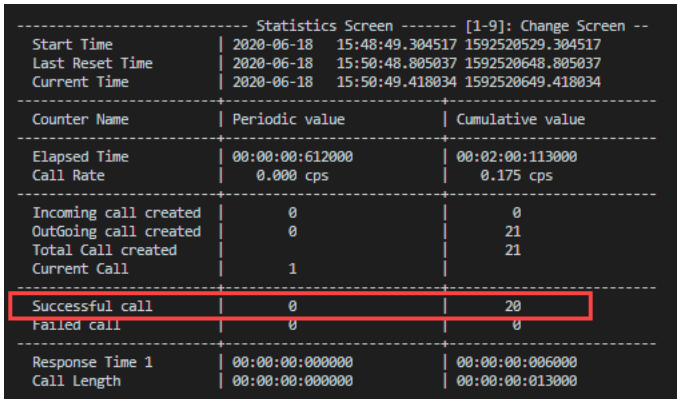

```
---------------------------- Statistics Screen ------- [1-9]: Change Screen --
  Start Time             | 2020-06-18   15:48:49.304517 1592520529.304517
  Last Reset Time        | 2020-06-18   15:50:48.805037 1592520648.805037
  Current Time           | 2020-06-18   15:50:49.418034 1592520649.418034
-------------------------+---------------------------+--------------------------
  Counter Name           | Periodic value            | Cumulative value
-------------------------+---------------------------+--------------------------
  Elapsed Time           | 00:00:00:612000           | 00:02:00:113000
  Call Rate              |    0.000 cps              |    0.175 cps
-------------------------+---------------------------+--------------------------
  Incoming call created  |        0                  |        0
  OutGoing call created  |        0                  |       21
  Total Call created     |                           |       21
  Current Call           |        1                  |
-------------------------+---------------------------+--------------------------
  Successful call        |        0                  |       20
  Failed call            |        0                  |        0
-------------------------+---------------------------+--------------------------
  Response Time 1        | 00:00:00:000000           | 00:00:00:006000
  Call Length            | 00:00:00:000000           | 00:00:00:013000
```

Fig. 31. **Experiment 5: Test1 Result:** SIP DoS attack to a SIP Server from 1 bot with legacy switch. Calls were disrupted when the attack started. 20 successful calls completed

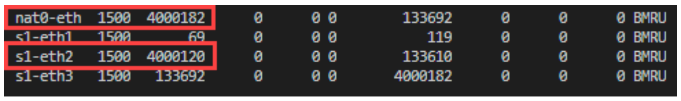

```
nat0-eth   1500  4000182     0       0 0        133692     0       0       0 BMRU
s1-eth1    1500       69     0       0 0           119     0       0       0 BMRU
s1-eth2    1500  4000120     0       0 0        133610     0       0       0 BMRU
s1-eth3    1500   133692     0       0 0       4000182     0       0       0 BMRU
```

Fig. 32. **Experiment 5: Test1 Result:** SIP DoS attack to a SIP Server from 1 bot with legacy switch. Attack packets received from `s1-eth2` interface were forwarded upstream to `nat0-eth` interface.

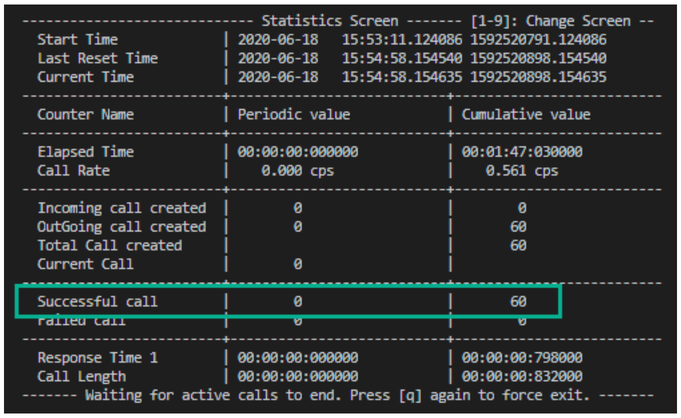
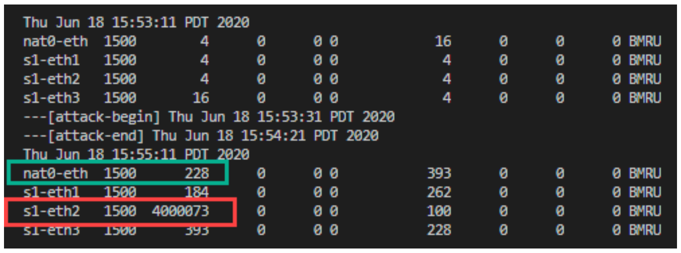

```
---------------------------- Statistics Screen ------- [1-9]: Change Screen --
  Start Time             | 2020-06-18   15:53:11.124086 1592520791.124086
  Last Reset Time        | 2020-06-18   15:54:58.154540 1592520898.154540
  Current Time           | 2020-06-18   15:54:58.154635 1592520898.154635
-------------------------+---------------------------+--------------------------
  Counter Name           | Periodic value            | Cumulative value
-------------------------+---------------------------+--------------------------
  Elapsed Time           | 00:00:00:000000           | 00:01:47:030000
  Call Rate              |    0.000 cps              |    0.561 cps
-------------------------+---------------------------+--------------------------
  Incoming call created  |        0                  |        0
  OutGoing call created  |        0                  |       60
  Total Call created     |                           |       60
  Current Call           |        0                  |
-------------------------+---------------------------+--------------------------
  Successful call        |        0                  |       60
  Failed call            |        0                  |        0
-------------------------+---------------------------+--------------------------
  Response Time 1        | 00:00:00:000000           | 00:00:00:798000
  Call Length            | 00:00:00:000000           | 00:00:00:832000
------- Waiting for active calls to end. Press [q] again to force exit. -------
```

Fig. 33. **Experiment 5: Test2 Result:** SIP DoS attack to a SIP Server from 1 bot with microVNF. All calls were successful. 60 successful call completed

3) Discussion: There are three important discussion points to cover: the number of successful calls from SIP-client, the number of packets that were forwarded upstream, and the CPU utilization.

The first discussion is on the total number of successful calls from SIP client to SIP server. With the legacy switch, as depicted in Fig. 31, there were only 20 successful calls, which means that SIP-client was able to make calls for the first 20 seconds of the test. Once the attack started at the 20th



```
Thu Jun 18 15:53:11 PDT 2020
nat0-eth  1500        4      0      0 0             16      0      0      0 BMRU
s1-eth1   1500        4      0      0 0              4      0      0      0 BMRU
s1-eth2   1500        4      0      0 0              4      0      0      0 BMRU
s1-eth3   1500       16      0      0 0              4      0      0      0 BMRU
---[attack-begin] Thu Jun 18 15:53:31 PDT 2020
---[attack-end] Thu Jun 18 15:54:21 PDT 2020
Thu Jun 18 15:55:11 PDT 2020
nat0-eth  1500      228      0      0 0            393      0      0      0 BMRU
s1-eth1   1500      184      0      0 0            262      0      0      0 BMRU
s1-eth2   1500  4000073      0      0 0            100      0      0      0 BMRU
s1-eth3   1500      393      0      0 0            228      0      0      0 BMRU
```

Fig. 34. **Experiment 5: Test2 Result:** SIP DoS attack to a SIP Server from 1 bot with microVNF. Most of the attack packets were dropped and not forwarded upstream.

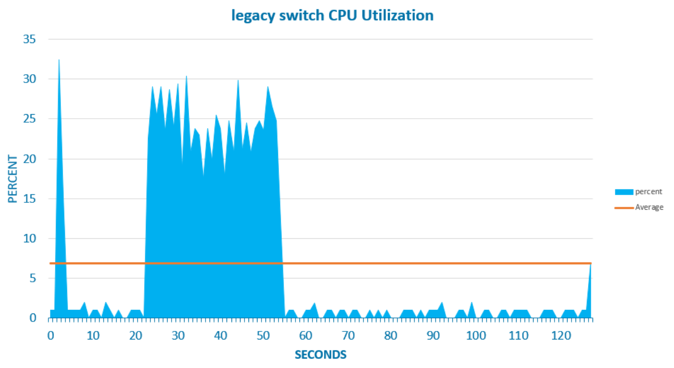


Fig. 35. **Experiment 5: Test1:** CPU utilisation of legacy switch.

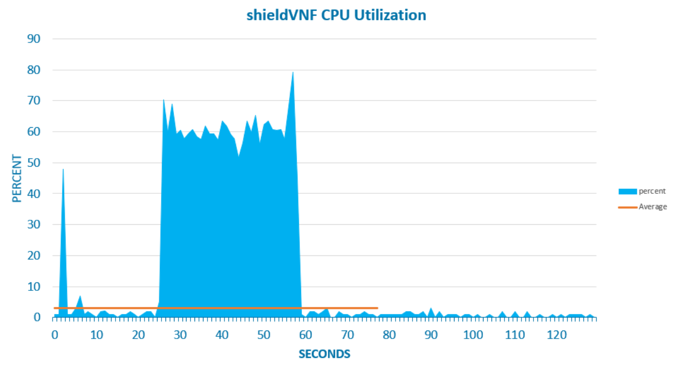


Fig. 36. **Experiment 5: Test2:** CPU utilisation of microVNF.

second, the SIP-client was no longer able to make calls. In contrast, with microVNF the SIP-client was able to make 60 calls (as shown in Fig. 33). This shows that the SIP DoS attack that started at the 20th second did not disrupt the SIP service because the attack was detected and mitigated by microVNF.

The second discussion is on the number of packets that were forwarded upstream. If the switch performs detection and mitigation functions correctly, the switch should be able to drop malicious packets and only forward the legitimate packets upstream. The legacy switch did not recognize malicious packets and forwarded 4,000,182 packets upstream via port nat0-eth as shown in Fig. 32. Out of 4,000,182 packets, the legitimate user (SIP-client1) generated 69 packets (s1-eth1) while the majority of these packets (4,000,120) were generated by the attacker that was connected to port s1-eth2 as shown in Fig. 32. In contrast, with microVNF, the attacker sent 4,000,073 packets (s1-eth2 port), yet only 228 packets were forwarded upstream (nat0-eth port) as shown in Fig. 34. Out of 228 packets forwarded upstream by microVNF, the legitimate user (SIP-client1) generated 184 packets (s1-eth1) and the remaining 44 forwarded packets were generated by the attacker before the mitigation function on microVNF was activated. In other words out of 4,000,073 packets generated by the attacker, 4,000,029 were dropped by microVNF.

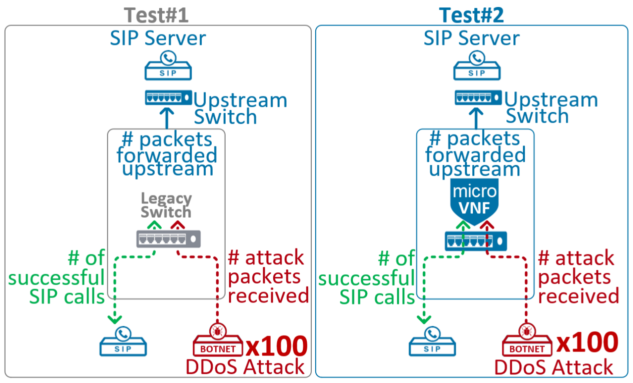
The third discussion point is on the difference in CPU utilization on the switch. As expected, microVNF consumed more CPU resources due to the detection and mitigation functions it performed, reaching a peak of 80%. In comparison, the legacy switch only performed normal forwarding functions and therefore the peak was only at 30%.

These observations present evidence that microVNF was able to keep legitimate calls up and running while successfully detecting and mitigating the SIP DoS attack from a single attacker.

F. Experiment 6: SIP Distributed DoS attack

The sixth experiment considered can SIP Distributed DoS attacks from IoT botnets be detected and mitigated? If microVNF is working as designed, then a legitimate user is still able to make SIP calls and malicious packets are dropped.

1) Description: This experiment presents a case where the majority of the IoT devices were infected and recruited by a botnet. The hypothesis for this experiment is that microVNF can detect and mitigate a SIP DDoS attack from one hundred attackers, as shown in Fig. 37.



Test#1
SIP Server
Upstream Switch
# packets forwarded upstream
Legacy Switch
# of successful SIP calls
# attack packets received
x100
DDoS Attack
Test#2
SIP Server
Upstream Switch
# packets forwarded upstream
micro VNF
# of successful SIP calls
# attack packets received
x100
DDoS Attack

Fig. 37. **Experiment 6:** SIP DDoS attack from 100 attackers (Test1 with legacy switch vs. Test2 with microVNF)

The scenario is identical to the previous experiment except for the number of bots. In this case, 100 bots participated in the attack (a SIP DDoS attack). SIP-client1 was calling SIP-client2 via PBX at the rate of one call per second. At the 20th second, 100 bots started to launch an attack against the same PBX that the SIP-client1 was using.

The test environment consisted of 103 hosts: SIP-client1, SIP-client2, PBX, and 100 bots. This experiment performed two tests, one with a legacy switch and the other with microVNF. The test script randomly generated the IP address

TABLE VII
EXPERIMENT 6: RESULT: MICROVNF BLOCKS SIP DISTRIBUTED DOS ATTACK (59 SUCCESSFUL CALLS VS. 20)

Exp6: SIP DDoS Attack	Legacy Switch			microVNF		
	# of attack packets received	# of packets fwd upstream	# of calls	# of attack packets received	# of packets fwd upstream	# of calls
Caller	65	NA	**20**	182	NA	**59**
100 Attackers	-	**512,488**	NA	4,000,073	**3061**	NA

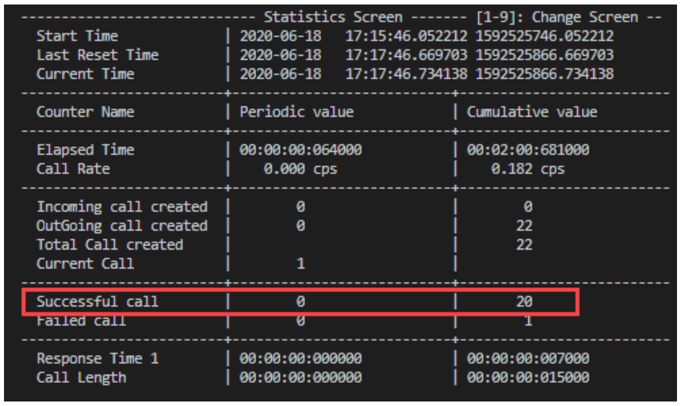
of SIP-Client1, SIP-Client2, and 100 bots at run time. The same attack tool, inviteflood [46], was used to send `INVITE` packets.

2) Result: Similar to the previous experiment, these are the three areas that are relevant: the number of successful calls from SIP client, the number of packets forwarded upstream, and CPU utilization. These results are important to indicate whether the detection and mitigation functions were effective to protect SIP service for legitimate users.

The number of successful SIP calls with legacy switch was 20 as shown in Fig. 38. With microVNF, the SIP client was able to made 59 successful calls during the experiment as depicted in Fig. 40.

The number of packets forwarded upstream with legacy switch was 512,488 (Fig. 39) compared to 3,061 with microVNF (Fig. 41). This experiment with 100 attackers quickly overwhelmed the switch and it stopped functioning at packet count 512,488. If the switch were running, the counter would show a much higher count.

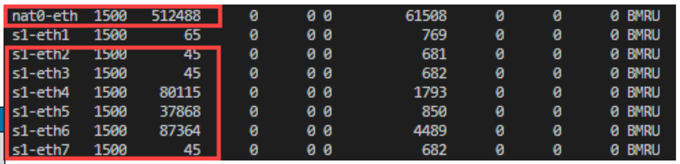
From a CPU utilization perspective, the legacy switch peaked at 51% during the DDoS attack as shown in Fig. 42, whereas for microVNF, the CPU peaked at 78% as shown in Fig. 43.



```
----------------------------- Statistics Screen ------- [1-9]: Change Screen --
  Start Time             | 2020-06-18   17:15:46.052212 1592525746.052212
  Last Reset Time        | 2020-06-18   17:17:46.669703 1592525866.669703
  Current Time           | 2020-06-18   17:17:46.734138 1592525866.734138
-------------------------+---------------------------+--------------------------
  Counter Name           | Periodic value            | Cumulative value
-------------------------+---------------------------+--------------------------
  Elapsed Time           | 00:00:00:064000           | 00:02:00:681000
  Call Rate              |    0.000 cps              |    0.182 cps
-------------------------+---------------------------+--------------------------
  Incoming call created  |        0                  |        0
  OutGoing call created  |        0                  |       22
  Total Call created     |                           |       22
  Current Call           |        1                  |
-------------------------+---------------------------+--------------------------
  Successful call        |        0                  |       20
  Failed call            |        0                  |        1
-------------------------+---------------------------+--------------------------
  Response Time 1        | 00:00:00:000000           | 00:00:00:007000
  Call Length            | 00:00:00:000000           | 00:00:00:015000
```

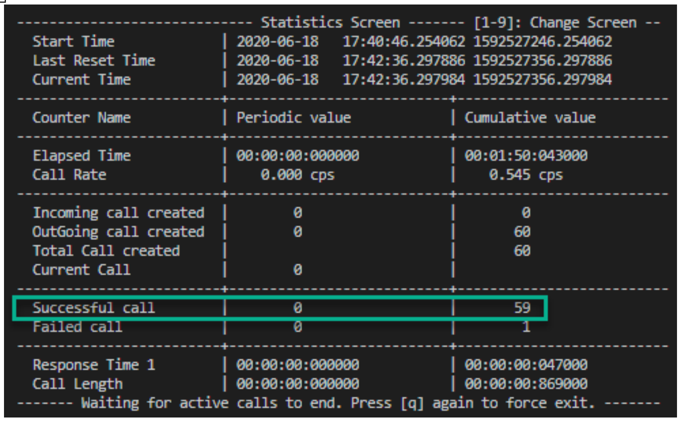
Fig. 38. **Experiment 6: Test1 Result:** SIP DoS attack to a SIP Server from 100 bots with legacy switch. Calls were disrupted when the attack started. 20 successful calls completed

3) Discussion: In this experiment we introduced 100 attackers to create a SIP DDoS attack. The use case is similar to real life where a large number of IoT devices were hijacked



```
nat0-eth  1500   512488      0      0 0        61508     0      0      0 BMRU
s1-eth1   1500       65      0      0 0          769     0      0      0 BMRU
s1-eth2   1500       45      0      0 0          681     0      0      0 BMRU
s1-eth3   1500       45      0      0 0          682     0      0      0 BMRU
s1-eth4   1500    80115      0      0 0         1793     0      0      0 BMRU
s1-eth5   1500    37868      0      0 0          850     0      0      0 BMRU
s1-eth6   1500    87364      0      0 0         4489     0      0      0 BMRU
s1-eth7   1500       45      0      0 0          682     0      0      0 BMRU
```

Fig. 39. **Experiment 6: Test1 Result:** SIP DoS attack to a SIP Server from 100 bots with legacy switch. Most attack packets from botnets were forwarded upstream to `nat0-eth` interface.



```
----------------------------- Statistics Screen ------- [1-9]: Change Screen --
  Start Time             | 2020-06-18   17:40:46.254062 1592527246.254062
  Last Reset Time        | 2020-06-18   17:42:36.297886 1592527356.297886
  Current Time           | 2020-06-18   17:42:36.297984 1592527356.297984
-------------------------+---------------------------+--------------------------
  Counter Name           | Periodic value            | Cumulative value
-------------------------+---------------------------+--------------------------
  Elapsed Time           | 00:00:00:000000           | 00:01:50:043000
  Call Rate              |    0.000 cps              |    0.545 cps
-------------------------+---------------------------+--------------------------
  Incoming call created  |        0                  |        0
  OutGoing call created  |        0                  |       60
  Total Call created     |                           |       60
  Current Call           |        0                  |
-------------------------+---------------------------+--------------------------
  Successful call        |        0                  |       59
  Failed call            |        0                  |        1
-------------------------+---------------------------+--------------------------
  Response Time 1        | 00:00:00:000000           | 00:00:00:047000
  Call Length            | 00:00:00:000000           | 00:00:00:869000
------- Waiting for active calls to end. Press [q] again to force exit. -------
```

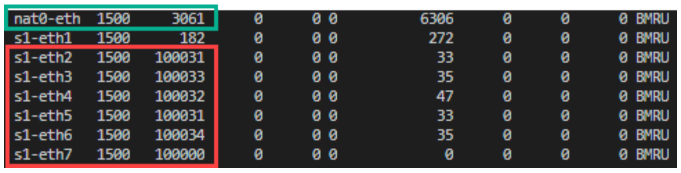
Fig. 40. [**Experiment 6: Test2 Result:** SIP DoS attack to a SIP Server from 100 bots with microVNF. 59 calls were successful.

by the same botnet and these devices were instructed to launch a SIP DoS attack by the botmaster.

From the end-users perspective, they would measure success by the number of successful calls that they can make during the attack. With the legacy switch, the SIP-client was only able to make 20 successful calls. The SIP-client was no longer able to make a call after the 100 bots started attacking the PBX. In contrast, with microVNF, the SIP-client was able to make 60 successful calls. This shows that the SIP client was still able to make calls even after the 100 bots started the attack, and therefore preserve the availability of the SIP service.

From the network integrity perspective, the network operators would measure success by the number of malicious packets that can be mitigated locally and not forwarded upstream. The legacy switch does not have the capability to recognize malicious packets and therefore it forwarded all packets upstream. In contrast, microVNF dropped the malicious packets and only forwarded legitimate packets upstream as shown in Fig. 41. Out of 3061 packets that were forwarded upstream, the legitimate user (SIP-client1) generated 182 packets (as shown by s1-eth1 row), and the remaining packets were attributed to the initial malicious packets before the mitigation function on microVNF was activated.

The CPU utilization confirms the observation that microVNF consumed more resources due to the detection and mitigation activities that it performed. With the legacy switch, the functionality was limited to forwarding packets upstream and therefore utilization peaked at 51%. In comparison, microVNF peaked at 78% as it performed deep packet inspection



```
nat0-eth  1500      3061      0      0 0         6306      0      0      0 BMRU
s1-eth1   1500       182      0      0 0          272      0      0      0 BMRU
s1-eth2   1500    100031      0      0 0           33      0      0      0 BMRU
s1-eth3   1500    100033      0      0 0           35      0      0      0 BMRU
s1-eth4   1500    100032      0      0 0           47      0      0      0 BMRU
s1-eth5   1500    100031      0      0 0           33      0      0      0 BMRU
s1-eth6   1500    100034      0      0 0           35      0      0      0 BMRU
s1-eth7   1500    100000      0      0 0            0      0      0      0 BMRU
```

Fig. 41. **Experiment 6: Test2 Result:** SIP DoS attack to a SIP Server from 100 bots with microVNF. Most of the attack packets from botnets were dropped and not forwarded upstream.

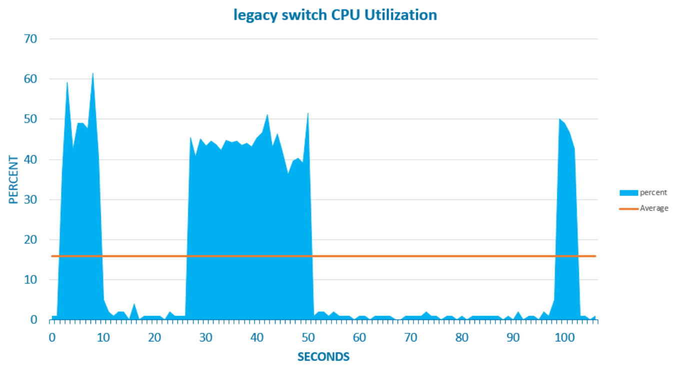


Fig. 42. **Experiment 6: Test1:** CPU utilization of legacy switch.

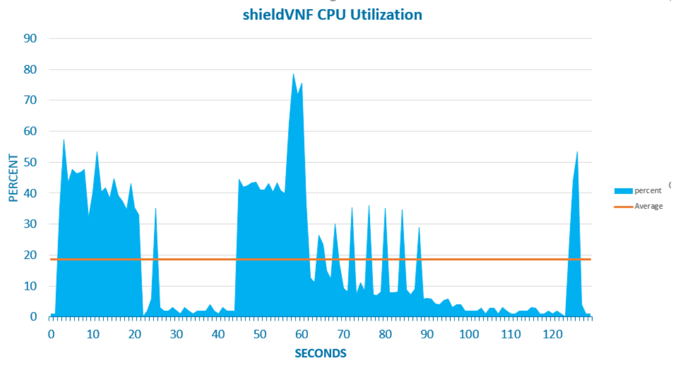


Fig. 43. **Experiment 6: Test2:** CPU utilization of microVNF.

at the SIP layer for attack detection, and dropping the malicious packets for attack mitigation.

This experiment presents evidence that microVNF is effective for detecting and mitigating SIP DDoS attacs. It was able to protect the availability of the SIP service for end-users in the midst of a SIP DDoS attack that was launched by 100 bots.

V. Discussion

The prevalence of data breaches, ransomware, and advanced, persistent threats challenge conventional defense models, which are strong with respect to network perimeters but weak in the internal networks. Given the sophistication of modern attacks, adversaries can be assumed to already be within a network, through either phishing attacks, unsecured connections, or misconfigured equipment. An alternative that has thus gained momentum is zero trust architecture [47], in which trust is never granted implicitly based on network location (i.e., externally or internally). This architecture highlights the importance of continual evaluation of network security postures.

That the internal network is more easily exploited and more vulnerable to attack is due to two contributing factors: lack of investment in internal security controls, and human nature. Adversaries take advantage of this situation by leveraging human tendencies to fall for phishing attacks by spams, e.g., sending malware-loaded attachments in emails. An email masquerading as having come from the CEO would naturally attracts clicks. Due to practical and financial reasons the zero trust model has not been as widely implemented as it should. However, network function virtualization and programmable data plane technology have progressed to the level where democratization of inspection functions for application-layer in the internal network is now possible.

With a programmable data plane, the NFV concept of separating network function from the hardware can now be extended to the edge switches as well. Virtual network functions are typically implemented on commodity servers. However, the introductory white paper believes that the NFV is applicable to any data plane packet processing and control plane function [48]. While popular implementation found in the literature is on commodity servers, the concept introduced in the white paper can be extended to the programmable data plane that exists on some modern switches today. With these switches, porting virtualized functions from the commodity servers to the network switches is now possible. This approach extends VNF deeper and wider into the network, and the network edge can now be leveraged to perform dual functionality: to provide network connectivity for end-points, and to implement the continual validation that the zero trust model advocates.

Strategic planning for VNF placement is critical in order to gain its full benefits while also reducing risks. Depending on their environment, switches are typically designed to fill such roles as access, distribution, core, spine, and leaf. Which role a switch fills defines the features that the switch must have and the functions that it must perform. For example, a core switch must switch packets as quickly as the hardware allows and disable any features that might introduce unnecessary delay. Edge and access-layer switches would typically have end point-facing features such as network access control or tagging packets with appropriate QoS settings. Given that porting virtualized network functions imposes additional workload, careful design is required so that the ported function does not negatively interfere with normal functionalities for the switch.

In the use case presented in this paper, the edge or access-layer switch was appropriate for performing application-layer inspection due to its position in the network. Edge layer switches are connected to a finite number of end-points and, under normal working conditions, the switch CPU utilization ranges from 5 to 40% [49], providing room on which to run virtualized network functions. In a worst-case scenario where the attacks degrade the performance of the entire edge switch, the risk is contained within a switch and the impact is felt only by the end-points that are connected to the same switch. When the network is designed with a hierarchical model in which the core or aggregation switch is connected to multiple edge switches, losing one edge switch would not typically affect the availability of the core switch or other edge switches. In the case of a botnet outbreak, the edge switch does not propagate malicious packets upstream and therefore protects the core and the rest of the network.

Performance degradation of an edge switch at 78% CPU utilization (Fig. 43) draws attention and signals a botnet outbreak to the network operator. The cost to the end-users would be in terms of a temporary performance hit, and the scope of this experience is confined to end-users which are connected to a particular edge switch. The impact for degraded performance is relatively less severe than having advanced persistent threat (APT) attacks. With APT attacks, there is a time delay between the exploit and its discovery, a period possibly spanning weeks or even months and so allowing exfiltration of sensitive data or the spread of the malware further into the network. In the worst case scenario, when the whole switch is degraded due to workload involving the virtualized network function, users and network operator will become aware, which prompts corrective actions required to restore service. This awareness is beneficial, in that it enables the network operator to locate the source of infection and isolate the damage to a switch, rather than allowing the infection to spread to other parts of the network. From a risk-management perspective, dealing with a known issue is more manageable than dealing with an unknown one where adversaries are quietly operating within the network.

The results of the experiments reported here support the hypothesis that placing application-layer inspection on an edge switch is effective in combating botnets. The experiments were performed to highlight relevant use cases both before and after the infection point. MicroVNF successfully defended against SIP scanning, enumeration, and brute force attacks prior to infection. After the infection, it successfully defended against connection attempts to CNC, SIP DoS, and SIP Distributed DoS attacks.

In addition to implementation on a virtual switch, microVNF can also run on hardware based P4 switches. Part of the appeal of P4 is being target agnostic, by which we mean that the same P4 program can be implemented on both software and hardware-based P4 switches, or even on a Raspberry Pi [50]. This independence is achieved through the use of a P4 compiler that translates a P4 program to a target specific implementation. The switch vendor is responsible for producing a P4 compiler that is compliant to the P4 language specification. This approach is similar to programming languages that achieve portability through a virtual machine or interpreter that is specific for particular hardware. However the scope of this proof-of-concept paper is limited to the virtual environment, with the intention to establish the viability of our approach.

In a real-world deployment environment, hardware based P4 switches would likely be deployed at the access, distribution, and core network. In addition, the P4 program and its corresponding network functions would be tailored for specific layer. For example, filtering functions are deployed at the access-layer switches, whereas switching or routing optimization functions are deployed at the core.

VI. Conclusion

This paper has presented microVNF, a novel VNF that runs on an edge switch's programmable data plane and operates at the application layer. It defends against SIP attacks such as scanning, enumeration, brute force, DoS, and DDoS, a function typically performed currently by hardware-based middle-boxes. MicroVNF is also intended to off-load the defensive workload from the data center and distribute it to spare capacity in the edge switches' data plane. Thus, microVNF scales network defense, thereby allowing network operators to have highly distributed and cost-effective SIP defense functions. Currently, microVNF is implemented on a software-based switch but implementation on a hardware-based architecture would improve the performance.

The benefits of placing VNF workload on an edge switch outweigh the risk. Unlike core switches which have high-performance requirements, edge switches have lighter workloads due to the number of end-points connected to them, making this additional workload feasible and low impact to the overall network stability. Service degradation occurring on an edge switch due to attacks launched by the end-points will allow the network operator to identify the source of the attack and contain the damage to a switch, thus protecting the rest of the network. Service degradation to an edge switch poses a lower risk profile rather than being unaware of the adversaries' presence for an extended period of time.

Besides SIP, other popular text-based, application-layer protocols such as HTTP and SMTP that share similar operational characteristics could benefit from adopting the same approach. Having application-layer inspection at edge switches would enable network operators to mitigate attacks near their sources, before these malicious packets travel farther towards the core network, use up more network resources, and reach their ultimate destination. This proposed approach would enable network operators to scale up their defense and reduce risks to their networks, particularly when vulnerable IoT devices are present on the network.

References

[1] K. Angrishi, "Turning internet of things(IoT) into internet of vulnerabilities (IoV) : IoT botnets." https://arxiv.org/abs/1702.03681, 2017.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th {USENIX} security symposium ({USENIX} Security 17)*, pp. 1093–1110, 2017.

[3] D. Goodin, "Record-breaking ddos reportedly delivered by >145k hacked cameras." https://arstechnica.com/information-technology/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/, September 2016.

[4] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoSin the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[5] C. Williams, "Today the web was broken by countless hacked devices – your 60-second summary." https://www.theregister.co.uk/2016/10/21/dyn_dns_ddos_explained, October 2016.

[6] Krebs, "Krebsonsecurity hit with record DDoS." https://https://krebsonsecurity.com/2016/ 09/krebsonsecurity-hit-with-record-ddos/, September 2016.

[7] J. Leyden, "Mirai IoT botnet blamed for smashing Liberia off the internet." https://www.theregister.co.uk/2016/11/04/liberia_ddos/, 2016.

[8] A. Khraisat and A. Alazab, "A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges," *Cybersecurity*, vol. 4, no. 1, pp. 1–27, 2021.

[9] M. Dutta, J. Granjal, *et al.*, "Towards a secure internet of things: A comprehensive study of second line defense mechanisms," *IEEE Access*, vol. 8, pp. 127272–127312, 2020.

[10] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in internet of things," *Journal of Network and Computer Applications*, vol. 84, pp. 25–37, 2017.

[11] M. Wazzan, D. Algazzawi, O. Bamasaq, A. Albeshri, and L. Cheng, "Internet of things botnet detection approaches: Analysis and recommendations for future research," *Applied Sciences*, vol. 11, no. 12, p. 5713, 2021.

[12] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on iot security: Application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[13] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, 2017.

[14] J. Portilla, G. Mujica, J.-S. Lee, and T. Riesgo, "The extreme edge at the bottom of the internet of things: A review," *IEEE Sensors Journal*, vol. 19, no. 9, pp. 3179–3190, 2019.

[15] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.

[16] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[17] S. Kaur, K. Kumar, and N. Aggarwal, "A review on p4-programmable data planes: Architecture, research efforts, and future directions," *Computer Communications*, vol. 170, pp. 109–129, 2021.

[18] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[19] A. Febro, H. Xiao, and J. Spring, "Telephony denial of service defense at data plane (tdosd@dp)," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, 2018.

[20] A. Febro, H. Xiao, and J. Spring, "Distributed sip ddos defense with p4," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019.

[21] S. Torabi, E. Bou-Harb, C. Assi, M. Galluscio, A. Boukhtouta, and M. Debbabi, "Inferring, characterizing, and investigating internet-scale malicious iot device activities: A network telescope perspective," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 562–573, IEEE, 2018.

[22] A. Mangino, M. S. Pour, and E. Bou-Harb, "Internet-scale insecurity of consumer internet of things: An empirical measurements perspective," *ACM Trans. Manage. Inf. Syst.*, vol. 11, Oct. 2020.

[23] M. Dahlmanns, J. Lohmöller, I. B. Fink, J. Pennekamp, K. Wehrle, and M. Henze, "Easing the conscience with opc ua: An internet-wide study on insecure deployments," in *Proceedings of the ACM Internet Measurement Conference*, IMC '20, (New York, NY, USA), p. 101–110, Association for Computing Machinery, 2020.

[24] M. Guri, Y. Mirsky, and Y. Elovici, "9-1-1 ddos: Attacks, analysis and mitigation," in *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 218–232, 2017.

[25] H. Singh, "p4-info." https://github.com/hesingh/p4-info.

[26] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, SOSR '17, (New York, NY, USA), p. 164–176, Association for Computing Machinery, 2017.

[27] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge Computing Security: State of the Art and Challenges," *Proceedings of the IEEE*, 2019.

[28] Q. Yan, W. Huang, X. Luo, Q. Gong, and F. R. Yu, "A multi-level ddos mitigation framework for the industrial internet of things," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 30–36, 2018.

[29] T. Alharbi, A. Aljuhani, and H. Liu, "Holistic ddos mitigation using nfv," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–4, IEEE, 2017.

[30] B. Rashidi and C. Fung, "Cofence: A collaborative ddos defence using network function virtualization," in *2016 12th International Conference on Network and Service Management (CNSM)*, IEEE, 2016.

[31] M. De Donno and N. Dragoni, "Combining AntibIoTic with fog computing: AntibIoTic 2.0," in *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–6, IEEE, 2019.

[32] W. Rafique, X. He, Z. Liu, Y. Sun, and W. Dou, "Cfadefense: A security solution to detect and mitigate crossfire attacks in software-defined IoT-edge infrastructure," (Los Alamitos, CA, USA), pp. 500–509, IEEE Computer Society, aug 2019.

[33] Y. Khosroshahi and E. Ozdemir, "Detection of sources being used in ddos attacks," 2019.

[34] E. Chen, "Detecting dos attacks on sip systems," in *1st IEEE Workshop on VoIP Management and Security, 2006.*, pp. 53–58, 2006.

[35] D. Ding, M. Savi, F. Pederzolli, M. Campanella, and D. Siracusa, "In-network volumetric ddos victim identification using programmable commodity switches," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1191–1202, 2021.

[36] W. Nazih, W. S. Elkilani, H. Dhahri, and T. Abdelkader, "Survey of countering dos/ddos attacks on sip based voip networks," *Electronics*, vol. 9, no. 11, p. 1827, 2020.

[37] M. M. Naeem, I. Hussain, and M. M. S. Missen, "A survey on registration hijacking attack consequences and protection for session initiation protocol (sip)," *Computer Networks*, vol. 175, p. 107250, 2020.

[38] M. Azrour, M. Ouanan, and Y. Farhaoui, "Survey of sip malformed messages detection," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 7, no. 2, pp. 457–465, 2017.

[39] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[40] "P416 portable switch architecture (psa) version 1.1." https://p4.org/p4-spec/docs/PSA-v1.1.0.htmlsec-registers.

[41] "Bloom filter calculator." https://hur.st/bloomfilter/. Web. Accessed 18 October 2021.

[42] J. Rosenberg, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "RFC 3261: Session Initiation Protocol (SIP)," *Internet Engineeging Task Force*, vol. 1, no. 11, pp. 1829–1841, 2002.

[43] "Sipp online help (-h)." http://sipp.sourceforge.net/doc2.0/reference.html Online+help+%28-h%29.

[44] "Sipvicious online help (-h)." https://tools.kali.org/sniffingspoofing/ sipvicious.

[45] "John the ripper usage." https://tools.kali.org/password-attacks/ john.

[46] "Inviteflood usage." https://tools.kali.org/sniffingspoofing/ inviteflood.

[47] National Institute of Standards and Technology, "Zero Trust Architecture - NIST Special Publication 800-207," *Nist*, p. 49, 2020.

[48] Chiosi, Margaret, et. al., "Network Functions Virtualisation Introductory White Paper." http://portal.etsi.org/NFV/NFV_White_Paper.pdf, 2012.

[49] Cisco, "Troubleshooting High CPU Utilization." https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3750/software/troubleshooting/ cpu_util.html54139, 2016.

[50] S. Laki, R. Stoyanov, D. Kis, R. Soulé, P. Vörös, and N. Zilberman, "P4pi: P4 on raspberry pi for networking education," *SIGCOMM Comput. Commun. Rev.*, vol. 51, p. 17–21, July 2021.