Synchronizing DDoS Defense at Network Edge with P4, SDN, and Blockchain

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—**Botnet-originated DDoS attacks continue to plague the internet and disrupt services for legitimate users. While various proposals have been presented in the last two decades, the botnet still has advantages over the defenders, because botnets have orchestrated processes to launch disruptive attacks. On the other hand, the defenders use manual methods, siloed tools, and lack orchestration among different organizations. These unorchestrated efforts slow down the attack response and extend the lifespan of botnet attacks. This article presents shieldSDN and shieldCHAIN, an inter-organization collaborative defense framework using P4, SDN, and Blockchain, which extends our earlier research on microVNF, a solution of Edge security for SIP-enabled IoT devices with P4. Besides mitigating DDoS attacks, microVNF also produces attack fingerprints called Indicator of Compromise (IOC) records. ShieldSDN and shieldCHAIN distribute these IOCs to other organizations so that they can create their own packet filters. Effectively, shieldSDN and shieldCHAIN synchronize packet filters for different organizations to mitigate against the same botnet strain. Four experiments were performed successfully to validate the functionalities of shieldSDN and shieldCHAIN. The scope for the first experiment was intra-company, while the second, third, and fourth experiments were inter-company. In the first experiment, shieldSDN extracted IOCs from the source switch and installed these as packet filters on other switches within the same organization (in the U.S.). In the second experiment, the shieldCHAIN in the publishing organization (in the U.S.) shared IOCs by posting them to the Blockchain. In the third experiment, the shieldCHAIN in the subscriber organizations (in Singapore & the U.K.) retrieved these IOCs from Blockchain. Finally, in the last experiment, the shieldCHAIN in the subscriber organizations installed the retrieved IOCs as packet filters; that are identical to those in the originating organization. To the best of our knowledge, this is the first framework that uses the P4 switch, SDN controller, and Blockchain together for this use case. As SDN and Blockchain gain acceptance, this framework empowers community members to collaborate and defend against botnet DDoS attacks.**

Index Terms—**SDN, P4, DDoS, Blockchain, DAO, NFT**

I. Introduction

A. Background

GOVERNMENTS, industry, and academia are concerned about threats from IoT botnets. They recognize the threats and potential damage that IoT botnets can cause in the wrong hands. Understanding that these threats impact the whole Internet ecosystem, there are initiatives underway which involve participants from different sectors in developing recommendations, standards, and implementations that address problems imposed by the IoT botnet.

From the Government perspective, in 2017, the U.S. President issued an executive order that called for increasing resilience against botnet and other automated, distributed threats [1]. In response, the Secretary of Commerce and the Secretary of Homeland Security published the "Botnet Report" to the President [2], followed by a road map report toward resilience against botnets [3]. The National Institute of Standards and Technology (the science laboratory under the Department of Commerce) published three documents: NISTIR 8228, 8259, and 8267 that specifically address IoT cybersecurity. In June 2019, NIST published the NISTIR 8228 document to establish the management of risks posed by IoT devices [4]. Since consumer home IoT devices are part of the ecosystem, NIST published 8267 in October 2019 to improve security on these devices [5]. These documents were then followed by the publication of NISTIR 8259 in January 2020 to define baseline security capabilities for IoT device manufacturers [6].

From a joint government-industry perspective, the National Cybersecurity centre of Excellence (NCCoE) published a Special Publication (SP) 1800-15 [7] that mitigates network-based attacks using the Manufacturer Usage Description (MUD). The MUD that is introduced in RFC8520 [8] essentially allows the manufacturer to publish the intended functions of the IoT device (profile), which are then enforced by the router using packet filters when the IoT device behaves outside the profile. This MUD initiative also includes specification for its data modeling [9], data interchange format [10], access list [11], data serialization [12], and signature [13]. Other supporting documents for identifying the network behavior [14] and procedures for device on-boarding [15] have been published. All of these are to ensure that the IoT device is not hijacked to perform functions that are not originally intended by the manufacturers, (e.g., being hijacked and recruited by a botnet to launch a DDoS attack).

B. Related Work

In the industry, DDoS mitigation methods are typically performed by using techniques such as Remote Triggered Black Holing (RTBH) [16], BGP Flowspec [17], unicast Reverse Path Forwarding (uRPF) [18], Enhanced Feasible-Path Unicast Reverse Path Forwarding (EFP-uRPF) [19], DDoS Open Threat Signaling (DOTS) [20], and using the scrubbing center [21] to drop malicious packets and forward legitimate ones. With RTBH, the victim is able to send a request to upstream network providers to drop DDoS packets. BGP Flowspec improves RTBH by offering more granular filtering criteria and supported actions. The uRPF technique blocks traffic from spoofed source IP addresses and EFP-uRPF further improves on this by having more flexibility relating to directionality. Some service providers offer a scrubbing service or mitigation-as-a-service [21] where the DDoS traffic is redirected to a network which performs deep packet inspection and forwards legitimate packets to the destination via a separate path.

From an academic perspective, some investigations have been carried out in Blackholing activities such as in-depth statistical analysis [22], automatic blackhole detection [23], and the application of the advanced mitigation strategy [24]. Some investigations take a different direction where they explore the use of SDN and NFV to defend against botnet attacks that are launched from hijacked IoT devices. Some proposals [25], [26] combine SDN with MUD, or NFV with MUD at home [27] or ISP level [28]. Bull et al., proposes an SDN gateway to monitor traffic going to or from IoT devices [29]. The gateway can block, forward, or modify QoS as a response to anomalous behavior. Ozcelik, Chalabianloo, and Gur proposed Edge-Centric Software-Defined IoT Defense (ECESID) [30] that provides detection at the network edge using SDN controller and Fog computing concepts. Bhunia and Gurusamy proposed SoftThings, an SDN-based secure IoT framework to detect anomalous behavior using a support vector machine (SVM), machine learning classifier [31]. Zarca et al., proposed a policy-based framework using SDN and NFV concepts as security enablers for IoT devices and the environment [32]. Yan et al., proposed Multi-Level DDoS Mitigation Framework (MLDMF) that includes edge/fog/cloud level using SDN-based IoT gateways (SDNIGW) [33]. Yin, Zhang, and Yang proposed a general framework called Software-Defined IoT (SD-IoT) consisting of SD-IoT controllers, switches integrated with IoT gateway, and devices [34]. Shorman, Faris, and Aljarah proposed a machine learning method that uses an unsupervised evolutionary IoT botnet detection method using the Grey Wolf Optimization algorithm (WGO) [35]. Afek, et al., did a proof-of-concept for NFV-based IoT Security at the ISP level, where it uses White List Monitoring (WLM) and White List Enforcement (WLE) using MUD protocol [36]. Andalibi et al., proposes a MUD-Visualizer to ease the deployment of MUD [37].

However, while good progress and contributions have been achieved on this front, the focus is on detecting whether or not an IoT device deviates from the manufacturer's predefined behavior.

C. Research Gaps

As per National Institute of Standards and Technology's Cybersecurity Framework [38], the next phase after detection is response. With botnet attacks as prevalent and disruptive as they are today, it is arguable that efforts need to extend beyond the detection phase, into the response phase (as depicted in Fig. 1).

IDENTIFY
PROTECT
DETECT
RESPOND
RECOVER

Fig. 1. NIST Cybersecurity Framework functions: Identify, Protect, Detect, Respond, and Recover [38]

Currently substantial manual coordination efforts are required to disseminate intelligence about DDoS attacks. This intelligence needs to be manually reviewed before it is actionable. With a shortage of qualified cybersecurity workers and increasing frequency of attacks, this process is not sustainable and scalable. Augmenting staff with automation is often viewed as a sustainable method to scale and meet the increasing workload.

The emergence of the programmable control plane (SDN) [39], programmable data plane (P4) [40] [41]–[44], and programmable blockchain technologies (Ethereum smart contract) present new capabilities which make it possible to build a community-based, synchronized, and automated defense for modern networks. These capabilities empower the network operators [45] to secure their network against botnet DDoS attacks.

D. Our Approach and Contributions

Some characteristics that make botnet-originated DDoS attacks effective are their synchronized coordination and the globally distributed nature of the attacks. To counter this synchronized mode of attack, we propose an automated and synchronized defense.

Dealing with the distributed nature of botnet, our proposed approach likewise empowers the community to form a distributed defense. Software-Defined Networking (SDN) with its programmable control and data plane features presents a novel option to automate and scale these tasks within an organization.

There are two synchronization tasks involved in the proposed solution, intra and inter organization. The first task is to facilitate intra-organization synchronization that leverages SDN to synchronize packet filters on multiple switches within an organization. The second task is to facilitate inter-organization packet filter synchronization that leverages Blockchain as a threat distribution platform in a community (Fig. 2), allowing a packet filter that has been created by one organization to be leveraged by another organizations in the community.

In this article, we present shieldSDN and shieldCHAIN, a novel framework to automate and synchronize defense posture on multiple switches in multiple organizations. This framework enables a distributed defense against distributed botnet attacks. ShieldSDN is an SDN controller that controls a P4 application running on an edge switch that has packet filtering capabilities. ShieldSDN is responsible for synchronizing packet filters on all edge switches within an organization, i.e., intra-organization synchronization. ShieldCHAIN is a blockchain

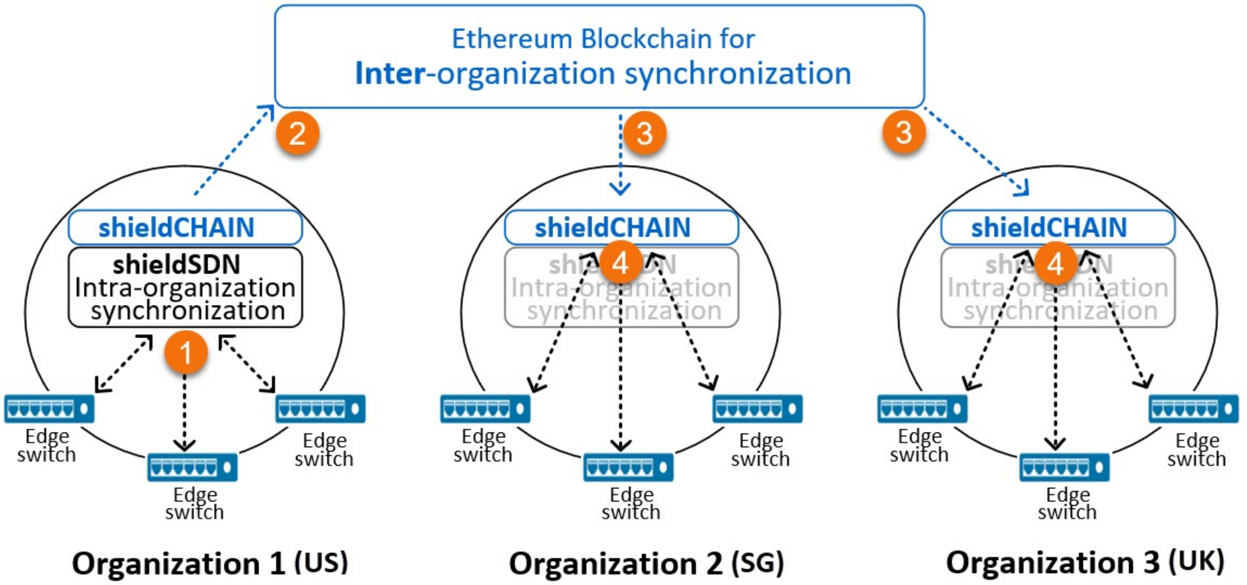


Fig. 2. Automated and synchronized defense: shieldSDN for intra-organization (1) & shieldCHAIN for inter-organization synchronization (2, 3 & 4)

distributed Application (dApp) that communicates with a smart contract deployed at Ethereum blockchain. ShieldCHAIN is responsible for publishing and subscribing IOC from other organizations, i.e., inter-organization synchronization. When a publisher-organization would like to share IOC with the community, shieldCHAIN creates a new Blockchain transaction. Similarly, when the subscriber-organizations would like to retrieve the current state of IOC data, the shieldCHAIN will retrieve it from Blockchain.

Four successful experiments were performed to validate the hypothesis that the programmable data plane, control plane, and blockchain can be orchestrated to scale DDoS defense against botnet by synchronizing packet filters on the edge networks. The first experiment was to validate that shieldSDN can distribute the IOC from one switch to multiple switches within the organization (number 1 in Fig. 2). The second experiment was to validate that the shieldCHAIN-agent in the publishing-organization can share IOC to Blockchain (number 2 in Fig. 2). The third experiment was to validate that the shieldCHAIN-agent in the subscribing-organization can receive IOC from Blockchain (number 3 in Fig. 2). The fourth experiment was to validate that the shieldCHAIN-agent in the subscribing-organization can install a packet filter on the edge switch (number 4 in Fig. 2).

A novel contribution of this article is a synchronized and automated botnet DDoS defense framework at the network edge using, P4, SDN, and Blockchain. This framework provides a synchronized defense for the network edge within an organization, as well as inter-organization. This framework empowers network operators (customers and/or service providers) to contribute their fair share of responsibilities towards defending against botnet DDoS attacks. To the best of our knowledge, this is the first framework that uses an SDN and Blockchain for these functionalities and use-cases.

II. The Proposed Solution: shieldSDN and shieldCHAIN

A. Design and considerations

The proposed solution leverages the strengths of both SDN and Blockchain technology to address the DDoS problem described in the previous section.

SDN technology proposes separation of the control and data plane, producing a network infrastructure with a programmable control and data plane. This new capability enables new breed of SDN-based applications such as: traffic engineering, optimization, tunneling, analytic, and network security use cases. However, these use cases are typically implemented within one organization or autonomous system. DDoS is a distributed problem that needs distributed solution and coordination between organizations. Additional elements are required to apply SDN across multiple organizations.

Blockchain technology offers an immutable, programmable, and distributed ledger. This capability enables a new breed of Blockchain-based applications such as: Decentralized Finance (DeFi), Non-Fungible Tokens (NFT), and Decentralized Autonomous Organization (DAO). Even though the pioneer use case for blockchains is cryptocurrency, the same technology can be leveraged for other use cases, such as sharing information across multiple organizations. This capability complements SDN allowing each organization to remain independent of each other, whilst still sharing threat intelligence amongst each other.

The proposed solution is designed as a collaboration between SDN-applications (shieldSDN), facilitated by the distributed application (shieldCHAIN). Together, shieldSDN and shieldCHAIN enable an organization to share and process threat intelligence with other organizations, which are then installed as packet filters on their respective P4 switches. By having the same threat intelligence and synchronizing their packet filters, these organizations form a herd immunity as they are capable of mitigating against the same strain of botnet attack.

B. ShieldSDN roles & functions

ShieldSDN is an SDN controller/application that automates the task of synchronizing packet filters among multiple switches in the organization. It inspects a switch for IOCs, and when found, converts the IOC into packet filters that can be installed on all switches. ShieldSDN is triggered by a scheduled job that runs at regular intervals to perform four tasks as depicted in Fig. 3:

1) Collect IOC from edge switches
2) Convert from IOC to packet filter
3) Install packet filter on other edge switches
4) Store packet filter in the database

The roles and functions that shieldSDN performed are described in more detail below:

1) IOC inspector: ShieldSDN's first task is to inspect each switch under its control for IOCs generated by the switch. The switch generates an IOC after it detects and mitigates a DDoS attack. The switch stores the IOC in its register along with two pieces of information: the attacker IP address and port number. When shieldSDN is able to find an IOC in the register, it retrieves the IOC for conversion.

There are three types of IOC that shieldSDN synchronizes among all switches, i.e., known attacker, known command-and-control (CNC) server, and known victim:

- Indicators about the **known attacker.** The receiving switch will inspect the source IP address of each packet that it receives. If the source IP matches, this packet is coming from a known attacker and the microVNF will drop the packet

- Indicators about the **known CNC server.** When the destination IP address of the packet matches a known CNC, the switch will drop the packet to prevent the botnets from registering and receiving commands from its CNC server.

- Indicators about the **known victim.** When the destination IP matches, this means that this packet is going to a DDoS attack victim. In this case, the switch will drop the packet to prevent the botnet from participating in the attack.

2) IOC converter: The second task for shieldSDN is to convert the IOC into packet filter format that is compatible and can be installed on other switches. This step is necessary because the data retrieved from the source switch is not directly actionable. The IOC data must be converted into commands that are understood by the destination switch. When shieldSDN sends these commands into the destination switch, they will get installed as packet filtering rules.

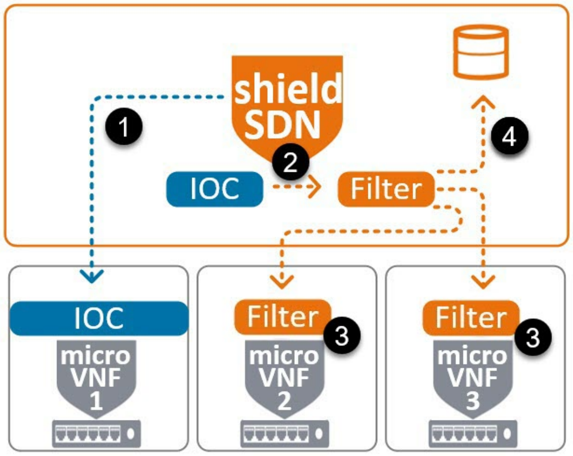


Fig. 3. ShieldSDN initiates a one-directional connection to the source switch to collect Indicator of Compromise (IOC) (1), converts it to packet filter (2), installs it as packet filtering rules on other switches (3), and stores it in the database.

3) Database feeder: In addition to installing packet filters on edge switches, shieldSDN stores the IOCs in a centralized database for long-term storage and analysis. When all the switches have installed the same packet filters, they have reached a synchronized state where each switch can take action based on the same set of information.

4) Filter Optimizer (Delete & Reinstall): Considering the finite amount of memory available on the switch's data plane, the SDN controller needs to perform memory management [46] by optimizing the number of filters that exist on a switch at a given time. The principle is to use the least amount of memory footprint necessary. This translates to managing the filters in such a way that the switch can continue to operate in the midst of a dynamic attack with the limited amount of memory available. Since the majority of attacks are short-lived, one way to conserve a switch's memory is by deleting filters that have expired so that new filters can be installed. However, when the same attackers reappear after the filter has been deleted, the switch has no recollection of previous incidents. The SDN controller has stored the previous incidents in the database and will be able to find and reinstall the filters for these repeat offenders, which are now considered as persistent threats.

ShieldSDN conserves the switch's memory by expiring old rules and storing them in a centralized database. ShieldSDN accomplished this task by initially assigning a random expiry time between 5 to 10 minutes, so that the switch only keeps rules associated with active attacks. The use of randomized expiry time is intended to make it less predictable for the attacker. The initial 5 to 10 minutes expiry time was chosen for two reasons. First, in case that it affected legitimate uses, they do not have to wait for too long before they can resume

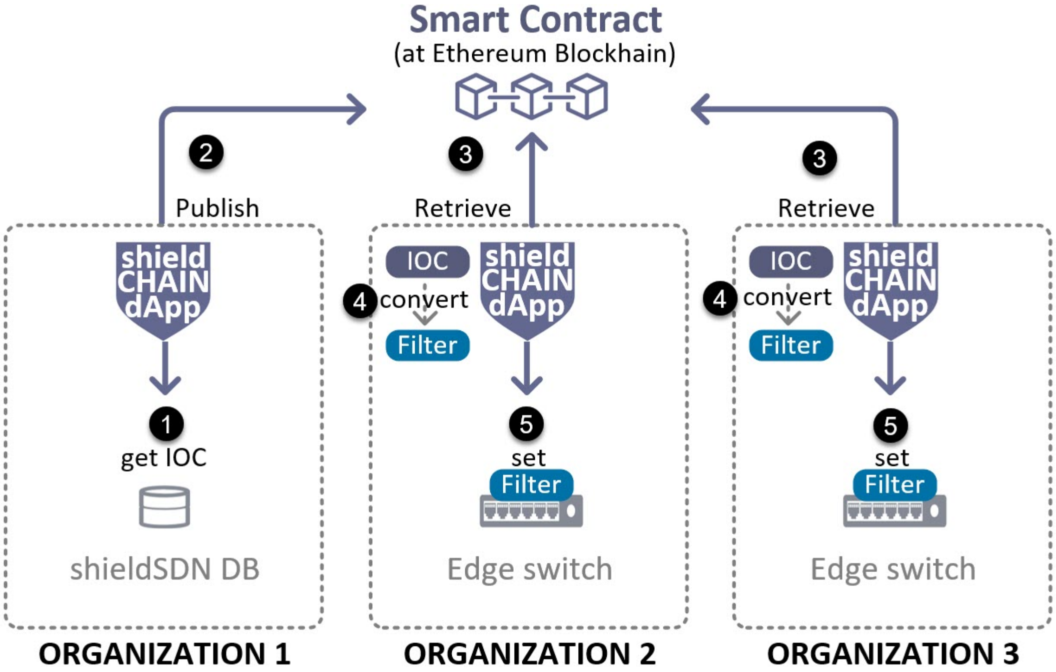


Fig. 4. ShieldCHAIN retrieves Indicator of Compromise (IOC) from shieldSDN database (1), publishes IOC to Blockchain (2), retrieves IOC from Blockchain, converts IOC to packet filter (4), and installs packet filter to edge switches (5).

their legitimate activities. Second, the assumption is that most attacks are short-lived. Kaspersky DDoS report for Q2 2020 states that 81.96% of the attacks are less than four hours [47]. Should the attackers keep attacking after the initial 5-10 minutes, shieldSDN will keep assigning new expiry time based on the attack frequency until it is longer than four hours. In the current implementation, a switch can keep 1024 records of each indicator (attacker, CNC, and victim). Since the switch is running at the edge of the network, having a space for 1024 attackers seems reasonable and can be adjusted depending on the underlying switch being used. Another reason for expiring the rules is to prevent self-inflicted DoS when an attacker tries to create entries that will drop legitimate packets.

Packet filters for Advanced Persistent Threats (APT) require a different strategy that takes into account the attack frequency. APT is often associated with well-organized attackers that are being careful and strategic in the way they carry out their attacks. APT is also persistent and may lie dormant for a long time before re-emerging to launch their attacks. APT poses an interesting challenge with the limited number of filters that can be created on a switch. One way to deal with this is for the SDN controller to create filters with longer expiry times instead of using the default. This expiry time should correspond with the frequency of attacks where it gets longer as the frequency of attack increases.

Persistent threats that reappear after being deleted will get a longer expiry time based on the frequency of the attack. There may be occasions when shieldSDN has deleted a rule, but then the same attacker reappears and launches the same attack. In this case, shieldSDN compares the previous IOC records in the database with the IOCs just retrieved. When a match is found, shieldSDN recognizes that this is a repeat offender and installs a new expiry time that is longer than the first time. The calculation is similar to the first-time offender (a random expiry time between 5 to 10 minutes). However, for the second time offender, the expiry time will be the square of the random expiry time. For example, suppose that the random time for a first-time offender is 10 minutes, when the same attacker re-offend for the second time, the new expiry time will be 100 minutes (10 to the power of 2). For the third-time, the expiry time would be 1000 minutes (10 to the power of 3). By doing it in this way, shieldSDN achieves two purposes: the expiry time is still not easily predictable, and the limited memory space is still being used efficiently. Making the expiry time random and less predictable for the attacker is important because it forces the attackers to spend more time and effort if they chose to continue.

The resiliency against repeat attacks are dependent on the amount of memory available and threat actor's persistence. The threat model considered is against an opportunistic threat actor as opposed to state-sponsored ones. With each successive mitigated attack on the same object, the filter will have longer expiry time. An opportunistic threat actor is likely to move on to easier targets to optimize their time and effort investment. In contrast, a state-sponsored actor is likely to persist in attacking the targeted object.

C. *ShieldCHAIN roles & functions*

ShieldCHAIN is a blockchain distributed application (dApp) that facilitates inter-organization IOC sharing. ShieldCHAIN provides a collaborative defence framework that is affordable, automated, and practical. This is to address common challenges identified to defend against a botnet attack such as lack of personnel, lack of standardized defense framework, and the workload required to implement such a collaboration.

ShieldCHAIN is designed to address these concerns. In terms of affordability, shieldCHAIN consists of components which are cost effective for member organizations to own. From labor requirement, shieldCHAIN automates sharing and retrieving IOCs at regular time interval, so there are no manual tasks involved. From an implementation perspective, shieldCHAIN allows the organization to start small and have incremental deployment. With low-cost hardware, an automated framework, and flexible deployment, ShieldCHAIN presents a solution for the community to collaborate against botnet attacks.

Collaboration among loosely connected organizations requires a distinctive approach to interaction. In an organization with centralized decision-making structure, an authoritative figure can prescribe and implement actions for all members in the organization. Another decision-making structure is a consortium of independent-organizations. With a consortium, each member is making decisions independently from each other. For example, Financial Services Information Sharing and Analysis Center (FS-ISAC) is a consortium of financial organizations. Health Information Sharing and Analysis Center (H-ISAC) is a consortium of healthcare organizations. In these structures, a member can share information with the community, but it is up to the individual member to decide whether to act on the information being shared. ShieldCHAIN is designed to accommodate this model of interaction, where a member can publish IOCs and the individual member can subscribe to IOCs.

ShieldCHAIN uses public Blockchain technology due to Blockchain's intrinsic features that facilitate data sharing between independent organizations. Blockchain is a public and distributed ledger technology that keep records permanently. Being a public ledger, anyone can verify the transactions that took place and the content of those transactions. The records are permanent and protected by cryptography functions such that any alteration will be noticeable. Blockchain distributes these records of transactions among Blockchain nodes such that there is no single-point-of-failure. Another important characteristic of Blockchain technology is that it is peer-to-peer, and there is no central authority figure that governs the interaction. Trust and integrity are established in the community by a combination of three factors: the first is by being transparent about the transactions and their content, the second is by the absence of a central figure that could potentially compromise the system, and the third is by using secure cryptography functions.

In terms of its architecture, shieldCHAIN can be viewed as a front-end and a back-end. The division of labor between front-end and back-end is depicted in Fig. 4 and described below:

1) ShieldCHAIN dApp (front-end). The decentralised Applications (dApps) agent serves as the front-end component of ShieldCHAIN. Each member that wants to be part of this community and participate in sharing threat intelligence will need to run an instance of the dApp in their organization. The dApp interfaces with a smart contract which has a variable that holds the current state of IOC records. The smart contract also provides functions such as create, delete, and retrieve to update this variable. While the value of this variable may change, the records of operations (add and delete) to this variable are permanent. The dApp interfaces with a SQL database to access the IOC records created by ShieldSDN. Lastly, the dApp interfaces with microVNF to install packet filtering rules.

2) ShieldCHAIN smart contract (back-end). The smart contract serves as the back-end component of ShieldCHAIN. A smart contract is code that gets deployed on the Ethereum Blockchain and that provides logic for accessing and storing the data. A smart contract provides the logic necessary to maintain the current state of threat intelligence. The smart contract also provides the functions for the dApp to add, delete, and retrieve data from variables that store IOC records and accessed by dApp agents.

   The smart contract maintains consensus of shared variables using the underlying Proof-of-Work (PoW) protocol for Ethereum nodes. There are two elements involved in maintaining the consistency of shared variables: consensus among the Ethereum nodes and shieldCHAIN dApp (the front-end agents). Among the Ethereum nodes, they use PoW as the consensus protocol. From a cross-cutting perspective, the shieldCHAIN smart contract is deployed on, or sits above, the Ethereum nodes.

   The consensus among shieldCHAIN dApp (the front-end agents) is reached through accessing the same variables on the smart contract. With dApp agents sharing variables, there is a need for concurrency control to avoid inconsistencies between records. ShieldCHAIN smart contract relies on the underlying PoW protocol as the arbiter among competing transactions so that the dApp agent will retrieve the latest version as per Ethereum transactions.

   In addition to technical consensus described above, there is also a consensus at the organizational level that occurrs offline. For this consensus, multiple independent organizations agree to create a consortium for the purpose of sharing threat intelligence. Each organization makes an independent decision to join this consortium based on their industry and requirement similarities. For example, financial institutions have Financial Service Information Sharing and Analysis

Center (FS-ISAC) and healthcare institutions have Health Information Sharing and Analysis Center (H-ISAC).

The dApp agent is triggered by a scheduled job that runs at a regular interval to perform five tasks as depicted in Fig. 4.

1) Retrieve IOC from shieldSDN Database
2) Publish IOC to Blockchain
3) Retrieve IOC from Blockchain
4) Convert IOC to packet filter
5) Install packet filter to edge switches

The dApp Agent could either play the role of publisher or subscriber. The dApp-publisher performs the first and second tasks; whereas the dApp-subscriber performs the third, fourth, and fifth tasks. Each dApp Agent is capable of performing both roles. Which role it takes depends on whether or not it has an IOC to share. Below are the details for each function:

1) Database retriever (dApp-publisher): DApp-publisher inspects shieldSDN database for IOC at regular intervals. When IOC is found, dApp-publisher retrieves the IOC.

2) Blockchain publisher (dApp-publisher): The next task for dApp-publisher is to share the IOC with the community via Blockchain. The dApp-publisher does this by calling a REST API with IOC as the payload to create a transaction in Blockchain and append a new record. At this stage, dApp-publisher has completed its task.

3) Blockchain subscriber (dApp-subscriber): The first task for dApp-subscriber is to inspect the smart contract deployed on Ethereum Blockchain. When a new IOC is found, it will retrieve the new IOC from the Blockchain, and hand it off to the next task.

4) IOC converter (dApp-subscriber): When the new IOC is retrieved, it needs to be converted before it can be deployed on the switch. The dApp-subscriber converts the IOC into a packet-filter format understood by P4-enabled switches deployed at the edges of the network. When this task is complete, the next task can commence.

5) Filter installer (dApp-subscriber): The final task for the dApp-subscriber is to install the packet filters on the edge switches. Once the packet filter is installed, it is able to defend against DDoS attacks coming from the same strain of botnet.

III. Implementation

ShieldSDN implementation involves a controller, southbound interface, and a SQL database. The shieldSDN controller provides the logic for IOC management, whereas the southbound interface provides the shieldSDN with the means to interact with microVNF. The database is used for long-term storage of IOC and analytics. In the experiments, there were four EC2 t2.micro instances used (one for shieldSDN and three microVNF). The microVNF instances used Mininet [48] with bmv2 selected as the switch type [49].

A. ShieldSDN

1) SDN Controller: The SDN controller is an application that is written in node.js [50] that interfaces with microVNFs and a database. ShieldSDN could use other programming languages, but node.js offers portability and availability on many platforms, from raspberry Pi to serverless function-as-a-service from cloud providers [51] [52] [53]. This portability ensures that the shieldSDN implementation is scalable from lab-scale to cloud production.

In these experiments, the SDN controller script was running on an AWS EC2 instance. The same instance hosts the SQLite database [54]. The instance has direct access to all microVNFs in the organization via IP. The SDN controller runs every minute and is launched by a task scheduler (cron) to communicate with the microVNFs. The SDN controller uses a command-line interface tool that is provided by the bmv2.

Collect IOC from edge switches (step 1). As depicted in Fig. 3 step 1, the SDN controller uses a command line tool (`simple_microVNF _cli register_read`) to retrieve the IOCs from a microVNF's registers. For example, to retrieve IOC information about the attacker's IP address, the SDN controller would connect to the microVNF and issue ”`register_read victimrIp_register` command.” MicroVNF would then return the complete content of attackerIp_register. For example, when the microVNF returns an entry ”`1226008369`”, it becomes ”`0100 1001 0001 0011 0110 0111 0011 0011`” when converted to binary, which is an IP address of ”`73.19.103.49`”. Besides collecting the IP address of the victim, shieldSDN also retrieves the victim's port number that is under attack with `register_read victimPort_register` command.

Convert from IOC to packet filter (step 2). The SDN controller converts the IOC to a packet filter format appropriate for microVNF. The native format of IOC is not readily usable by microVNF as a packet filter. The SDN controller needs to convert and identify the appropriate table on microVNF so that it can function as a packet filter. For example, the IP address of ”`1226008369`”, needs to be converted to ”`MyIngress.KnownVictim_table drop 73.19.103.49/32 5060`”. In this case, we want to add the IP address of ”`73.19.103.49/32 5060`” into ”`MyIngress.KnownVictim`” table.

Install packet filter on microVNF (step 3). The SDN controller inserts a packet filter on the microVNF table by using a command line tool (`simple_microVNF_cli table_add`) to insert packet filtering rules on the microVNF's tables. MicroVNF inspects every packet that it receives against `KnownVictim_table`, along with KnownAttacker_table, KnownCNC_table, and KnownVictim_table, and drops the packet when it finds a match.

The shieldSDN that creates a packet filtering rule is also responsible for deleting these entries. When a shieldSDN interacts with multiple microVNFs, it is important for the shieldSDN to keep track of the microVNF that it is interacting with. For example, supposed that shieldSDN1 is responsible for installing a packet filtering rule on microVNF1, microVNF2, and microVNF3. In this case, shieldSDN1 is also responsible for deleting the expired IOC from microVNF1, microVNF2, and microVNF3. ShieldSDN1 will also store these data in the SQLite database, which we will review next.

Store packet filter in database (step 4). The last task for the SDN controller is to store packet filters in the database. The SDN controller uses the mysql command line tool to insert packet filters in the appropriate table.

2) IOC Database: ShieldSDN stores the data in a SQL database using three tables: KnownAttacker, KnownCNC, and KnownTarget. As the name suggests, these tables are for storing information about attackers, CNCs and targets that microVNF have dealt with previously. This database stores both active and expired IOCs.

B. ShieldCHAIN

1) Smart Contract: A smart contract is code that is deployed to Blockchain and that processes, how the data is received, stored, and shared. The data is dynamically added and deleted by the members so that they reflect the current state of IOCs. When there are no activities, the smart contract will contain an empty string. However, when there is an outbreak with many known attackers, the data may contain a long list of IP addresses and ports. When a member first submits the data, ShieldCHAIN ensures that there is no duplicate data submitted previously by other members. This step ensures that the list only contains valid and unique IOCs. The design of the smart contract keeps the programmable logic at a minimum, following a minimalist contract design pattern.

TABLE I
SMART CONTRACT AND ITS ADDRESS AT KOVAN NETWORK

Smart Contract	Address at Kovan Network
Attacker	0x9FACE6372e53B5Cc61 848340AA194E709773CEa4
CNC	0x53D721ebA7Db1c8e8 0x53D721ebA7Db1c8e8
Victim	0x31C39540518B1a337 0x31C39540518B1a337

A minimalist contract design pattern leverages the strengths of Blockchain technology to perform the critical tasks while leaving other tasks outside of the Blockchain. The critical tasks in this case are for establishing trust among members and achieving synchronised state management of IOC. The trust is built based on the features that are offered by Blockchain, for example transaction transparency, peer-to-peer, and the use of cryptography to secure storage and confirm identity. Since shieldCHAIN is using a public Blockchain, the synchronised state is achieved by having the data accessible by all members in the community. This minimalist design is influenced partly by the speed and cost of operations. The cost factor is influenced by the availability of miners for the computing and storage requirement for the operations. The speed factor influences how fast the transaction is completed where high-reward transactions will get completed first by the miners. With these considerations in mind, the use of Blockchain is kept to a minimum and reserved for those tasks that only the Blockchain can do. Other tasks such as the logic to access, refresh, and sort the data are delegated to the dApp that are not bound by those constraints. With the minimalist contract design, there is a dynamic string array and five functions for data operations. Each element in the array stores an IP address and port, e.g., "43.13.34.32:5060" which indicates that a member has seen an incident that involved the IP address 43.13.34.32 at port 5060. The functions consist of: addIp(), delIp(), getLength(), deleteAll(), and getAll(). ShieldSDN calls for function addIp() to insert a new IOC, while delIp() is for deleting an IOC when it has expired. In order to retrieve all IOC in the smart contract, ShieldSDN will call the getAll() function.

ShieldCHAIN deploys three smart contracts to hold different information: KnownAttacker, KnownCNC, and KnownVictim. Table I lists the addresses of the smart contracts that are deployed on the Ethereum Blockchain. The separation of contracts allows the community to subscribe to get information that is relevant and important for their environment. They could subscribe to just one or all three smart contracts at the same time. As the name implies, KnownAttacker smart contract holds information about the source IP address and port number for attackers that have launched either SIP scanning, SIP enumeration, or a SIP brute force attack. This information is useful for the community so that they can proactively drop packets coming from these source IP addresses. The KnownVictim smart contract holds information about the IP address and port number that are known to be the target of the SIP DoS attack. The community wants to learn this information to verify that their IP address is not on the list as a target. Besides confirming that they are not the target, the list also prevents the organization from participating in the SIP DoS attack if they have infected IoT devices. The last smart contract is about blocking communication with a known CNC server so that even though they have infected IoT devices, these bots are not functioning as intended because the CNC channel is blocked. These smart contracts are deployed to Ethereum, ready for accepting calls from the dApp, which is the topic of the next discussion.

TABLE II
SHIELDCHAIN SMART CONTRACT FUNCTIONS

Smart Contract Function	Description
`addIp(IOC)`	Adds new IOC to smart contract
`delIp(IOC)`	Deletes current IOC in smart contract
`getLength()`	Returns the number of entries
`getAll()`	Returns all entries

TABLE III
END-POINTS FOR ACCESSING SHIELDCHAIN SMART CONTRACTS

URL	Description
`https://kovan.infura.io/v3/89517c9511fe4e029578d2d4c30bffb6`	End-point for Known Attackers
`https://kovan.infura.io/v3/9d000f3595c4468f9148254e920fbbd8`	End-point for Known CNCs
`https://kovan.infura.io/v3/37dcc5503a9c461cbcafd9df98b8ac03`	End-point for Known Victim

2) dApp: A dApp is a node.js script that interfaces with the smart contract, SQL database, and microVNF. Every organization that wants to participate in this collaborative defence framework needs to run a copy of dApp. This dApp allows the organization to contribute to the community by creating new IOCs that they discover in their network. The dApp also allows the members to retrieve IOCs that were generated and observed by other member organizations. Each organization is responsible for the IOC that they originated. When the IOC has expired, the original dApp that shared the IOC is also responsible for removing the expired IOC from the Blockchain. In this way, the IOC stored in the Blockchain reflects the current state of observed threats. The way the dApps interact with the Blockchain is by calling the functions that the smart contract provides.

The dApp uses these functions to access the data that is stored in the smart contract. The smart contract provides functions to facilitate access and operation for the information stored in the data structure. These functions are listed in Table II. The data structure is a dynamic array of strings that store one IOC in each element.

The dApp interacts with the SQL database to retrieve and insert IOC records that it receives from the smart contract on the Ethereum Blockchain. The dApp accomplishes this by using the command-line tool sqlite3 that allows the dApp to query the database. There are two use cases that require interaction between dApp and the SQL Database. The first is when the dApp wants to share with the community the IOC that is stored in the database. The second is when the dApp retrieves an IOC from the community and wants to store it in the SQL Database.

The dApp interfaces with the microVNF to install packet filtering rules so that the switch can drop malicious packets. The other critical function that the dApp performs involves interfacing with the microVNF that is running on the ethernet switch. Once the dApp learns about the IOC from the community, the dApp will convert these IOCs into packet filtering rules that are understood by microVNF. The dApp will install packet filtering rules by using the command line tool `simple_switch_cli` that comes with the bmv2 switch. With this tool, the dApp can install packet filtering rules so that the switch can drop the malicious packets when it encounters packets that match with the rules. The dApp also uses the same command line tool to remove expired packet filtering rules from the switch.

3) Infura: REST API for Ethereum: Instead of calling the smart contract on Ethereum Blockchain directly, the dApp interacts with the smart contract through Infura [55]. Infura is an online service that provides REST API access to the smart contract deployed on the Ethereum Blockchain. This method greatly simplifies the operation on the client-side. Instead of having to build and maintain an Ethereum Virtual Machine (EVM) node for writing and reading operation, the dApp will make a REST API call to Infura. Building and maintaining an EVM node is not a trivial task and may not be practical for most field implementations due to the skill set, systems, and networking requirements. On the other hand, calling the REST API is much more accessible as it does not take a lot of system resources. The objective of this approach is to lower the technical and logistical barriers for accessing the Blockchain so that we can maximise participation.

With three smart contracts deployed to Blockchain, each smart contract has its endpoint URL. Infura generates a unique URL for the dApp to access each smart contract. The URLs that are accessible by the dApp are listed in Table III. Besides providing the convenience of accessing Blockchain via REST API, Infura also provides a dashboard for analytics and additional security functions, e.g., API secret key, filter for contract address, user agents, and origins.

IV. EXPERIMENT DESIGN

The main objective of synchronized DDoS Defense is for a community to have a synchronized defense posture. In this state, all switches have the same packet filtering rules to contain DDoS attacks. Success criteria is defined as having an identical number of packet filters on all switches. The challenge is for these switches to reach this state while they are owned and operated by independent organizations.

The experiments are designed to share IOC from one organization to other organizations in the community through blockchain. As Fig. 5 shows, there are four experiments involved:

1) **Experiment 1: Intra-organization synchronization**. The first experiment is for shieldSDN to collect IOC from the microVNF1 and install it to microVNF2 and microVNF3. Besides installing packet filters on switches, shieldSDN also stores IOC in a SQL database. In this experiment, we can quantitatively measure the number of IOCs from the source microVNF (microVNF1) with the destination

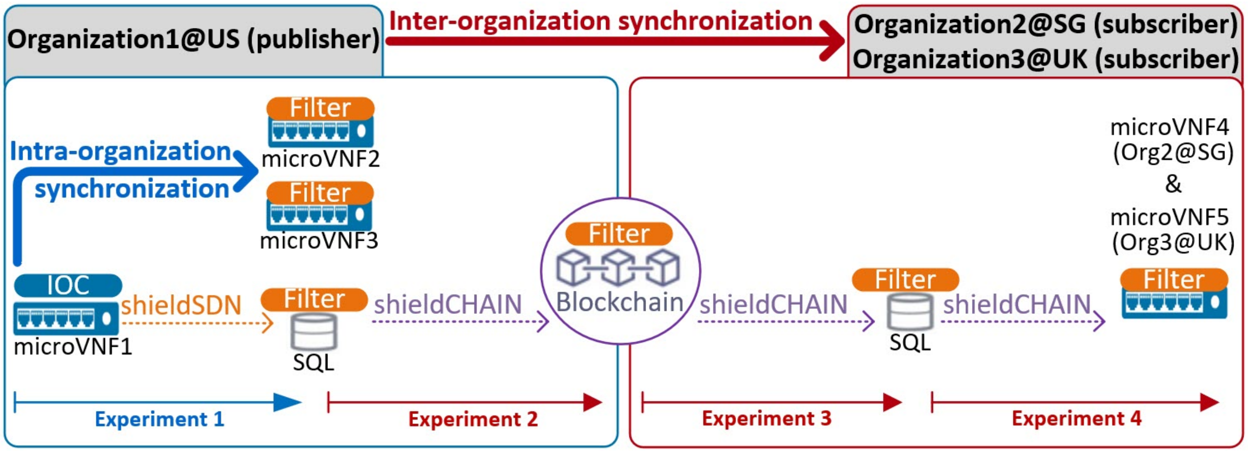


Fig. 5. Synchronized defense: ShieldSDN synchronized packet filters from microVNF1 to microVNF2 & microVNF3 in Organization1 (Experiment1: intra-organization synchronization). ShieldCHAIN synchronized packet filters from Organization1(US) to Organization2(SG) and Organization3(UK) as inter-organization synchronization (Experiment 2, 3, & 4). Together, these switches formed a synchronized edge network defense against DDoS attacks.

microVNF (microVNF2 and microVNF3) to validate that shieldSDN has correctly performed its function. A positive result is achieved when we have a synchronized state where microVNF1, microVNF2, and microVNF3 have the same number of packet filtering rules.

2) **Experiment 2: Publish (inter-organization synchronization)**. The second experiment is about one organization sharing IOC with the community via Blockchain. In this experiment, the process starts with ShieldCHAIN in organization1 making the SQL query to the database for the IOCs. Once the IOCs are retrieved, the ShieldCHAIN creates a Blockchain transaction for each record in the appropriate smart contract that is deployed on the Blockchain. There are three smart contracts deployed on the Blockchain (attacker, CNC, and victim) that the member organizations can retrieve.

3) **Experiment 3: Subscribe (inter-organization synchronization)**. The third experiment is about community members retrieving IOC from Blockchain. In this experiment, the process starts with ShieldCHAIN in organization2 and organization3 calling a function in a smart contract to retrieve all IOCs. Once the IOCs are retrieved, ShieldCHAIN inserts the IOC into the SQL database for further processing in the next experiment.

4) **Experiment 4: Packet filter installation**.The fourth experiment is about installing packet filters on the edge switches. In this experiment, the process starts with ShieldCHAIN in organization2 and organization3 retrieving IOC from the SQL database. Once the IOCs are retrieved, ShieldCHAIN inserts the IOC into the appropriate table as packet filters on the edge switch (microVNF).

The purpose for the experiment is to establish the viability of the approach rather than to generate accurate benchmarks for specific scenarios. With this objective in mind, the time required to perform these experiments will be based on the time it takes to establish a pattern. When we observe that three transactions showing a similar pattern at the time of the experiment and the variables involved are not changed, we expect that it would give similar result when we perform more transactions over a longer period of time. We consider this approach to be viable when the time taken to perform these experiments were reasonable and not excessively long to achieve the purpose.

These experiments were designed to validate the hypothesis that the proposed solution, shieldSDN and shieldCHAIN, can scale DDoS defense against botnet by synchronizing packet filters on the edge networks. The packet filters are synchronized within one organization (intra-organization synchronization) or to other organizations (inter-organization synchronization).

V. Experiments

The experiments begin by preparing the environment using a python script, this is followed by the four experiments. The python script created 100 IOC records each in 3 registers (attacker, CNC, and victim) on microVNF1. The presence of IOC records in microVNF simulates previously detected and mitigated attacks, which, the microVNF produced IOC records about the parties involved, either as the attacker, botnet CNC, or target. Our previous paper describes [56] how microVNF developed this capability of detecting, mitigating, and creating these records. This preparation step sets the stage for shieldSDN to collect these IOCs and install them as packet filtering rules on other switches within Organization1 (intra-organization) or in other organizations (inter-organization, with Organization2 and Organization3).

The script generates these IOC entries randomly at run time. To ensure that shieldSDN will work with any IPv4 addresses, the script assigns these IOCs to a random group (attacker vs. CNC vs. victim). This random assignment method also removes bias so as to explore causality. This method achieves a double-blind effect where neither the participant nor researcher knows ahead of time of group assignment. The infrastructure (microVNF, SQL database, and Blockchain) is able to handle more than 100 IOC records, but we chose 100 for ease of statistical analysis.

At the end of this stage, the registers at microVNF2 and microVNF3 are empty because they have not encountered the attack yet.

A. *Experiment 1: Intra-organization synchronization*

The first experiment is in answer to the first research question, i.e., What approaches can be used to collect an IOC from one switch and install it as packet filtering rules on other switches? If shieldSDN is working as designed, then it will create new packet filtering rules (on microVNF1, microVNF2, and microVNF3) and new records in the SQL Database.

1) Procedures: The process starts with shieldSDN retrieving the contents of microVNF1 registers for known attackers, known CNC, and known victims. Upon receiving this information, shieldSDN is supposed to install packet filtering rules in KnownVictim_table, KnownCNC_table and KnownAttacker_table on microVNF1, microVNF2, and microVNF3. In addition, shieldSDN was also supposed to create entries in the database for long-term storage. Fig. 6 shows the sequence of this experiment.

Positive outcome is demonstrated when packet filtering rules are successfully created in the KnownVictim_table, KnownCNC_table and KnownAttacker_table at microVNF1, microVNF2, and microVNF3. Negative outcome is demonstrated when these tables are empty.

2) Result: The result of this experiment is captured in Table IV below. After the experiment, the tables in microVNF1, microVNF2, and microVNF3 are populated with packet filtering rules. Out of 100 IOCs, 100 packet filtering rules were created.

Chronologically, the result is marked on Fig. 6 as point ”A” and ”B”. Point A, as depicted in Fig. 7, shows identical number of records on three edge switches. Point B, as depicted in Fig. 8, shows the number of IOC stored for each type.

The experiment took 64 seconds to run. Fig. 9 shows the CPU utilization. On average, CPU utilization on shieldSDN is 59.58% during this experiment.

3) Statistical Analysis: We have a process with a probability p of failing on each attempt. We assume that failures on successive attempts are independent. We have made 100 attempts with zero failures. From this we conclude that $p < 3\%$ at the 0.05 level of significance, and that $p < 4.6\%$ at the 0.01 level of significance.

This follows from the binomial theorem: the probability of zero failures is $P = (1 - p)^{100}$ and if we hypothesize that $p \geq 0.03$ then

$$1 - p \leq 0.97, \quad \ln(1 - p) < -0.03,$$

$$\ln P = 100 \ln(1 - p) < -3.0, \quad P < e^{-3.0} < 0.05$$

so we reject the hypothesis at the 0.05 level of significance. A similar argument leads us to reject the hypothesis that $p \geq 4.6\%$ at the 0.01 level of significance on the basis of 100 trials.

4) Discussion: This experiment demonstrated that shieldSDN was able to collect IOC records from one switch and install them as packet filtering rules on other switches. For IOC collection, shieldSDN used the command line tool `simple_switch_cli` to connect to microVNF1. For installing packet filtering rules on microVNF1, microVNF2 and microVNF3, shieldSDN used the same tool, but with different commands and parameters. Fig. 7 shows the number of entries in the KnownAttacker_table and KnownVictim_table on microVNF1, microVNF2, and microVNF3.

ShieldSDN also successfully created records for these IOCs

TABLE IV

EXPERIMENT 1: SHIELDSDN COLLECTS IOCS FROM MICROVNF1, INSTALLED THESE IOCS AS PACKET FILTERING RULES AT MICROVNF1, MICROVNF2 AND MICROVNF3, AND STORED THESE IN THE SQL DATABASE (B=BEFORE EXPERIMENT; A=AFTER EXPERIMENT).

Experiment 3: APT Round#	micro VNF1		micro VNF2		micro VNF3		SQL	
	B	A	B	A	B	A	B	A
# of IOC in attackerIp **_register**	100							
# of IOC in attackerPort **_register**	100							
# of IOC in victimIp **_register**	100							
# of IOC in victimPort **_register**	100							
# of IOC in cncIp **_register**	100							
# of IOC in cncPort **_register**	100							
# of Filters in KnownAttacker **_table**		100		100		100		100
# of Filters in KnownCNC **_table**		100		100		100		100
# of Filters in KnownVictim **_table**		100		100		100		100

TABLE V

EXPERIMENT 1: RECORD IN SQL DATABASE FOR AN ATTACKER

Field	**Value**
`attacker_id`	`1`
`time`	`2020-7-4 11:45`
`switchNo`	`1`
`expirytime`	`2020-7-4 11:50`
`frequency`	`1`
`attacker`	`167.118.98.95:5060`
`tablehandle`	`0`
`registerindex`	`20`
`VNFdeleted`	`0`
`BCNdeleted`	`0`

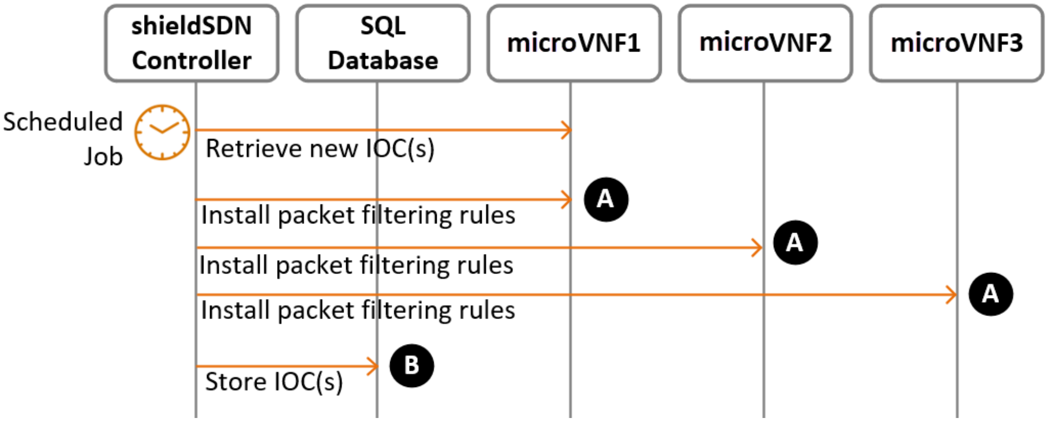
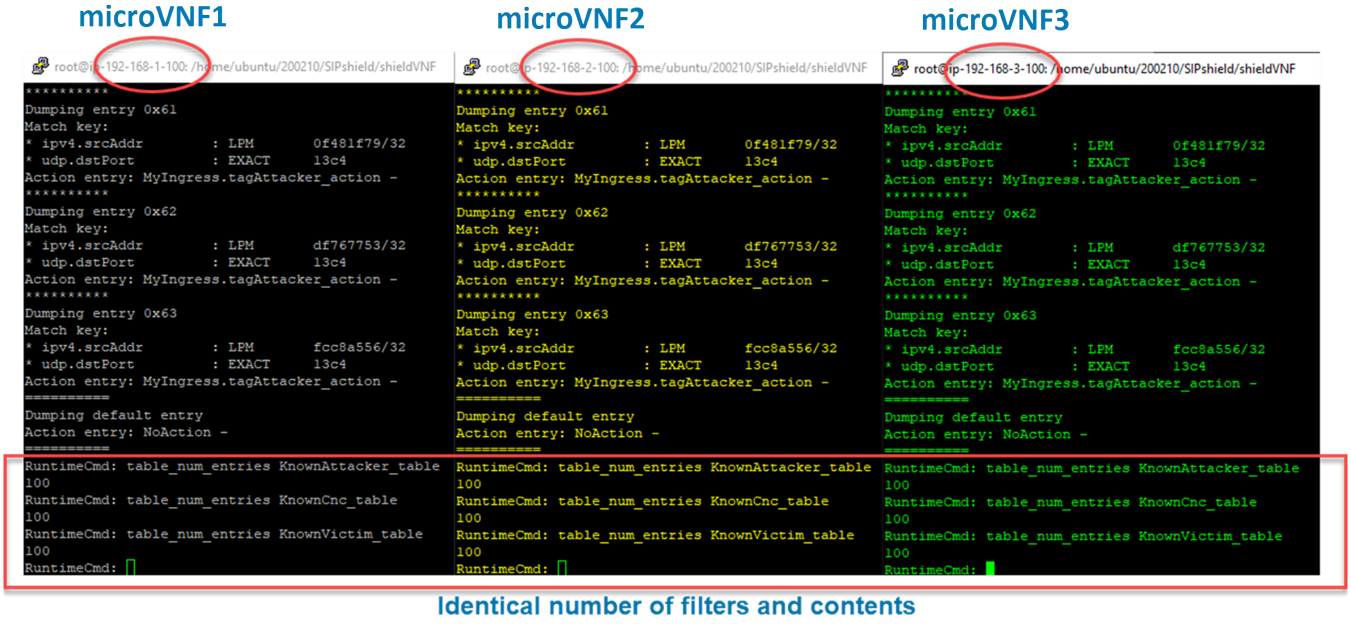


Fig. 6. **Experiment 1:** Sequence diagram for experiment 1. Snapshot of result at point ”A” is depicted in Figure 7. Snapshot of result at point ”B” is depicted in Figure 8.



root@ip-192-168-1-100: /home/ubuntu/200210/SIPshield/shieldVNF

```
**********
Dumping entry 0x61
Match key:
* ipv4.srcAddr        : LPM       0f481f79/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x62
Match key:
* ipv4.srcAddr        : LPM       df767753/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x63
Match key:
* ipv4.srcAddr        : LPM       fcc8a556/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
==========
Dumping default entry
Action entry: NoAction -
==========
RuntimeCmd: table_num_entries KnownAttacker_table
100
RuntimeCmd: table_num_entries KnownCnc_table
100
RuntimeCmd: table_num_entries KnownVictim_table
100
RuntimeCmd:
```

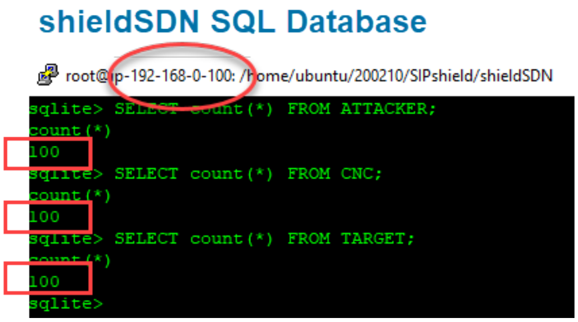
root@ip-192-168-2-100: /home/ubuntu/200210/SIPshield/shieldVNF

```
**********
Dumping entry 0x61
Match key:
* ipv4.srcAddr        : LPM       0f481f79/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x62
Match key:
* ipv4.srcAddr        : LPM       df767753/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x63
Match key:
* ipv4.srcAddr        : LPM       fcc8a556/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
==========
Dumping default entry
Action entry: NoAction -
==========
RuntimeCmd: table_num_entries KnownAttacker_table
100
RuntimeCmd: table_num_entries KnownCnc_table
100
RuntimeCmd: table_num_entries KnownVictim_table
100
RuntimeCmd:
```

root@ip-192-168-3-100: /home/ubuntu/200210/SIPshield/shieldVNF

```
**********
Dumping entry 0x61
Match key:
* ipv4.srcAddr        : LPM       0f481f79/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x62
Match key:
* ipv4.srcAddr        : LPM       df767753/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x63
Match key:
* ipv4.srcAddr        : LPM       fcc8a556/32
* udp.dstPort         : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
==========
Dumping default entry
Action entry: NoAction -
==========
RuntimeCmd: table_num_entries KnownAttacker_table
100
RuntimeCmd: table_num_entries KnownCnc_table
100
RuntimeCmd: table_num_entries KnownVictim_table
100
RuntimeCmd:
```

Fig. 7. **Experiment 1:** At the end of experiment 1, microVNF1, microVNF2, and microVNF3 have identical number of filters installed for known attackers, CNC, and victim.



root@ip-192-168-0-100: /home/ubuntu/200210/SIPshield/shieldSDN

```
sqlite> SELECT count(*) FROM ATTACKER;
count(*)
100
sqlite> SELECT count(*) FROM CNC;
count(*)
100
sqlite> SELECT count(*) FROM TARGET;
count(*)
100
sqlite>
```

Fig. 8. **Experiment 1:** At the end of experiment 1, the IOC(s) are stored in the database.

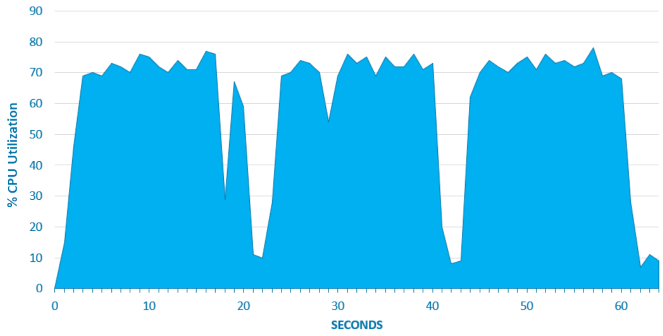


Fig. 9. **Experiment 1:** CPU utilization by shieldSDN.

in the SQL database tables. Table V shows one entry in the database table. In this particular record, it shows that this is a new attacker (167.118.98.95:5060), shieldSDN sets the `frequency` field to one, and the `expirytime` field to 5 minutes (11:45 - 11:50). The `frequency` and `expirytime` are two important fields for dealing with a persistent threats, which we shall look at shortly.

These screenshots show that the IOCs that originated from microVNF1 have been successfully installed as packet filtering rules at microVNF1, microVNF2, and microVNF3. With these rules installed, these switches will be able to drop malicious packets without having to do attack detection tasks again. Having packet filtering rules on microVNF2 and microVNF3 is significant because it proves that shieldSDN was able to replicate the success from one switch (microVNF1) to multiple switches in the organization (microVNF2 and microVNF3). Out of 100 IOCs in the registers, 100 packet filtering rules were created in the tables. This is statistically significant and demonstrates a positive outcome for this experiment.

B. Experiment 2: Inter-organization synchronization (publisher)

The second experiment is in answer to the question, What strategy could be adopted to share threat intelligence with the community? If the ShieldCHAIN in organization1 is working as designed, then it will create IOC records on the Blockchain that are accessible by the whole community.

1) Procedures: The process starts with ShieldCHAIN in organization1 making the SQL query to the database for the IOCs. Once the IOCs are retrieved, the ShieldCHAIN creates a Blockchain transaction for each record in the appropriate smart contract deployed on the Blockchain. There are three smart contracts deployed on the Blockchain (attacker, CNC, and target) that the member organizations can retrieve.

2) Result: The result of this experiment is captured in Table VI below. Before the experiment, there were 0 records in the Blockchain. After the experiment, there were 100 records present.

The experiment took 1552 seconds (25.86 minutes) to run. Fig. 10 shows the CPU utilization and Fig. 11 shows the memory utilization of ShieldCHAIN during the experiment.

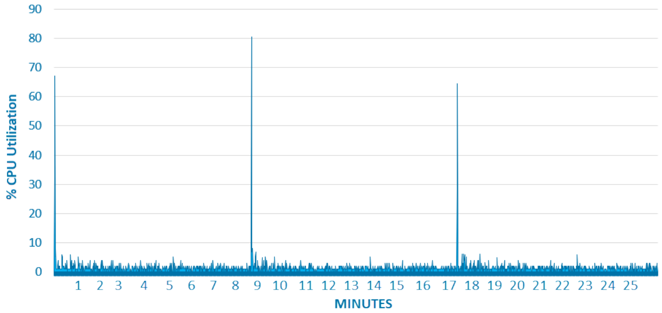


Fig. 10. **Experiment 2:** CPU utilization of ShieldCHAIN and the time it took to complete the process.

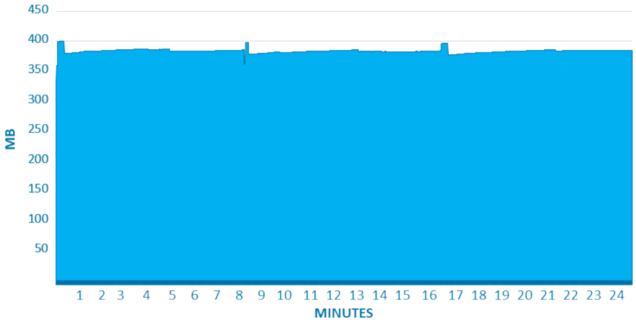
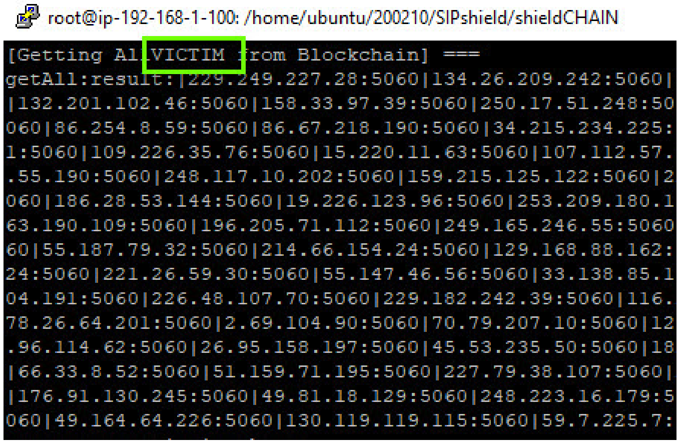


Fig. 11. **Experiment 2:** Memory utilization of ShieldCHAIN

TABLE VI
EXPERIMENT 2: SHIELDCHAIN QUERIES IOCS FROM THE DATABASE AND ADDS IT TO ETHEREUM BLOCKCHAIN.
(B=BEFORE; A=AFTER)

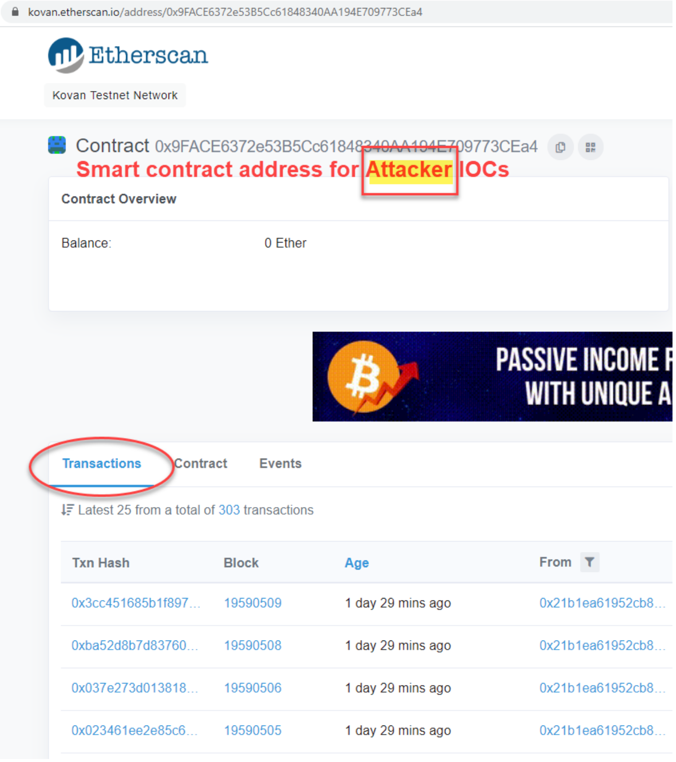
Dependent Variables	Blockchain		VNF4 @ Org2		SDN4 @ Org2		VNF5 @ Org3		SDN5 @ Org3	
	B	A	B˜	A	B	A	B	A	B	A
#IOC in **register**										
#Filters in **table**										
#IOC in **SQL**										
#IOC in **Blockchain**	0	100								


root@ip-192-168-1-100: /home/ubuntu/200210/SIPshield/shieldCHAIN

```
[Getting All VICTIM from Blockchain] ===
getAll:result:|229.249.227.28:5060|134.26.209.242:5060|
|132.201.102.46:5060|158.33.97.39:5060|250.17.51.248:50
060|86.254.8.59:5060|86.67.218.190:5060|34.215.234.225:
1:5060|109.226.35.76:5060|15.220.11.63:5060|107.112.57.
.55.190:5060|248.117.10.202:5060|159.215.125.122:5060|2
060|186.28.53.144:5060|19.226.123.96:5060|253.209.180.1
63.190.109:5060|196.205.71.112:5060|249.165.246.55:5060
60|55.187.79.32:5060|214.66.154.24:5060|129.168.88.162:
24:5060|221.26.59.30:5060|55.147.46.56:5060|33.138.85.1
04.191:5060|226.48.107.70:5060|229.182.242.39:5060|116.
78.26.64.201:5060|2.69.104.90:5060|70.79.207.10:5060|12
.96.114.62:5060|26.95.158.197:5060|45.53.235.50:5060|18
|66.33.8.52:5060|51.159.71.195:5060|227.79.38.107:5060|
|176.91.130.245:5060|49.81.18.129:5060|248.223.16.179:5
060|49.164.64.226:5060|130.119.119.115:5060|59.7.225.7:
```

Fig. 12. **Experiment 2:** ShieldCHAIN has created 100 IOC records of victims at Blockchain.

3) Discussion: ShieldCHAIN in organization1 retrieved IOC records from the database and created 100 IOCs for each smart contract in the Blockchain. Fig. 12 shows the IOC records that were created on Victim smart contracts. The public can use an online blockchain explorer (Etherscan.io) to visually verify that these IOC records were created. Fig. 13 shows a screenshot of Etherscan.io that lists the transactions created to store the Attacker records. In Fig. 14 we see a similar screenshot that shows the IOC records for CNC, whereas in Fig. 15 we have the IOC records for Target. As an



kovan.etherscan.io/address/0x9FACE6372e53B5Cc61848340AA194E709773CEa4

Etherscan

Kovan Testnet Network

Contract 0x9FACE6372e53B5Cc61848340AA194E709773CEa4

Smart contract address for Attacker IOCs

Contract Overview

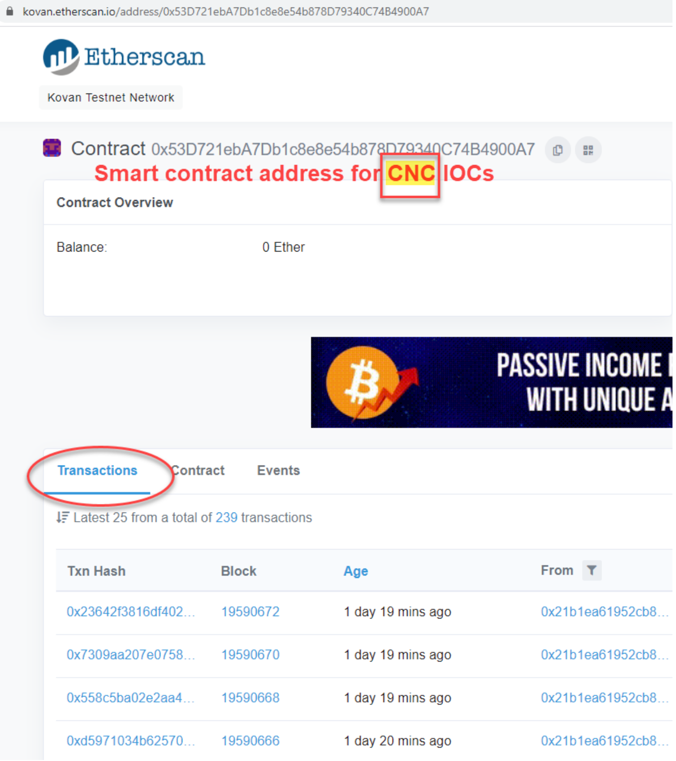
Balance: 0 Ether

Transactions Contract Events

Latest 25 from a total of 303 transactions

Txn Hash	Block	Age	From
0x3cc451685b1f897...	19590509	1 day 29 mins ago	0x21b1ea61952cb8...
0xba52d8b7d83760...	19590508	1 day 29 mins ago	0x21b1ea61952cb8...
0x037e273d013818...	19590506	1 day 29 mins ago	0x21b1ea61952cb8...
0x023461ee2e85c6...	19590505	1 day 29 mins ago	0x21b1ea61952cb8...

Fig. 13. **Experiment 2:** Attacker smart contract shows the transactions that were created for IOC records.



kovan.etherscan.io/address/0x53D721ebA7Db1c8e8e54b878D79340C74B4900A7

Etherscan

Kovan Testnet Network

Contract 0x53D721ebA7Db1c8e8e54b878D79340C74B4900A7

Smart contract address for CNC IOCs

Contract Overview

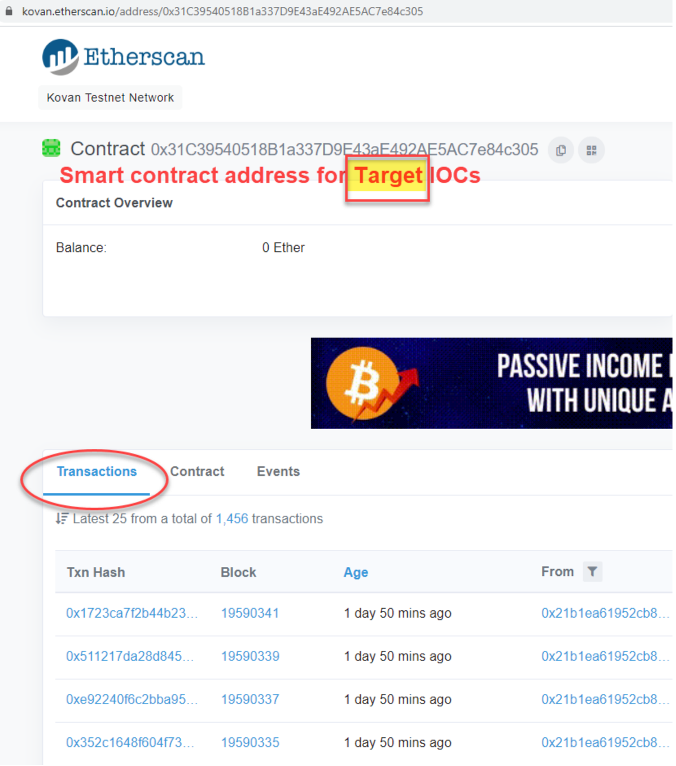
Balance: 0 Ether

Transactions Contract Events

Latest 25 from a total of 239 transactions

Txn Hash	Block	Age	From
0x23642f3816df402...	19590672	1 day 19 mins ago	0x21b1ea61952cb8...
0x7309aa207e0758...	19590670	1 day 19 mins ago	0x21b1ea61952cb8...
0x558c5ba02e2aa4...	19590668	1 day 19 mins ago	0x21b1ea61952cb8...
0xd5971034b62570...	19590666	1 day 20 mins ago	0x21b1ea61952cb8...

Fig. 14. **Experiment 2:** CNC smart contract shows the transactions that were created for IOC records.



kovan.etherscan.io/address/0x31C39540518B1a337D9E43aE492AE5AC7e84c305

Etherscan

Kovan Testnet Network

Contract 0x31C39540518B1a337D9E43aE492AE5AC7e84c305

Smart contract address for Target IOCs

Contract Overview

Balance: 0 Ether

Transactions Contract Events

Latest 25 from a total of 1,456 transactions

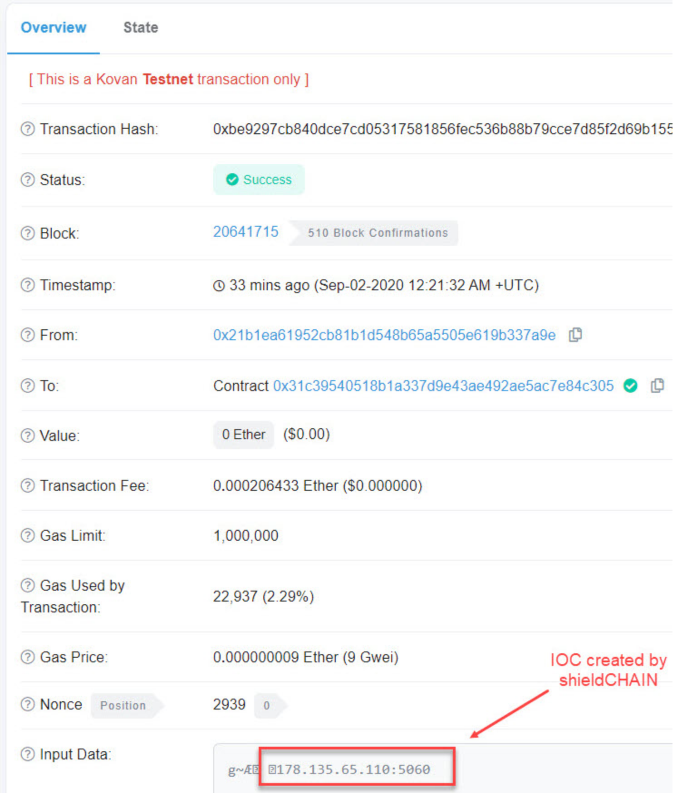
Txn Hash	Block	Age	From
0x1723ca7f2b44b23...	19590341	1 day 50 mins ago	0x21b1ea61952cb8...
0x511217da28d845...	19590339	1 day 50 mins ago	0x21b1ea61952cb8...
0xe92240f6c2bba95...	19590337	1 day 50 mins ago	0x21b1ea61952cb8...
0x352c1648f604f73...	19590335	1 day 50 mins ago	0x21b1ea61952cb8...

Fig. 15. **Experiment 2:** Target smart contract shows the transactions that were created for IOC records.

example, the content of this transaction is shown in Fig. 16 where it shows the IOC (`172.135.65.110:5060`) in the input data field.

Blockchain took about 5 seconds on average to create a transaction on Kovan Network. In this experiment, there were 300 transactions created (100 KnownAttacker, 100 KnownCNC, and 100 KnownTarget) and this corresponds with the time it took to complete the experiment (25.86 minutes). The time taken is attributed to the tasks involved for the miners to create a new transaction and append it to a block. We expect that each Ethereum network would have its own variables that affect transaction speed. The speed is considerably slower than what one might expect from a regular SQL database. As such, it may not be suitable for use cases that require instantaneous response. For storing IOCs however, the speed is acceptable and the benefits of using Blockchain outweigh the risks.

These screenshots show that the IOC records that originated from ShieldSDN in organization1 have been successfully created in the Blockchain smart contracts. Out of 100 IOC records in the SQL Database tables, 100 transactions were created on Blockchain. This result demonstrates a positive outcome for this experiment and ShieldCHAIN has provided a solution to share threat intelligence from one organization to other organizations in the community.



Overview State

[This is a Kovan **Testnet** transaction only]

Transaction Hash: 0xbe9297cb840dce7cd05317581856fec536b88b79cce7d85f2d69b155

Status: Success

Block: 20641715 510 Block Confirmations

Timestamp: 33 mins ago (Sep-02-2020 12:21:32 AM +UTC)

From: 0x21b1ea61952cb81b1d548b65a5505e619b337a9e

To: Contract 0x31c39540518b1a337d9e43ae492ae5ac7e84c305

Value: 0 Ether ($0.00)

Transaction Fee: 0.000206433 Ether ($0.000000)

Gas Limit: 1,000,000

Gas Used by Transaction: 22,937 (2.29%)

Gas Price: 0.000000009 Ether (9 Gwei)

Nonce Position 2939 0

Input Data: g~Æ 178.135.65.110:5060

Fig. 16. **Experiment 2:** Verifying new record that just created by ShieldCHAIN. This record shows the IOC (IP address: Port) that was created by ShieldCHAIN.

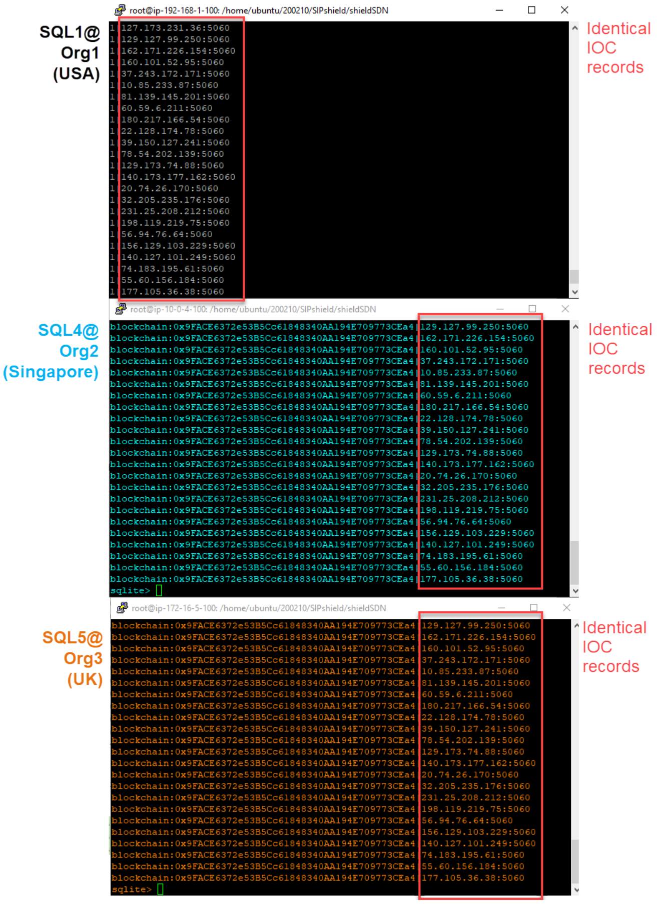
C. Experiment 3: Inter-organization synchronization (subscribers)

The third experiment is in answer to the question, What strategy could be adopted to retrieve threat intelligence shared by the community on Blockchain? If shieldCHAIN in organization2 and organization3 are working as designed, then they will retrieve IOC records from Blockchain (100 IOCs in each smart contract: attacker, CNC, and target) and create the corresponding IOC records in the SQL database tables.

1) Procedures: The process starts with ShieldCHAIN in organization2 and organization3 calling a function in a smart contract to retrieve all IOCs. Once the IOCs are retrieved, ShieldCHAIN inserts the IOC into the appropriate table in their ShieldSDN's database tables.

2) Result: The result of this experiment is captured in Table VII below. The experiment took 18 seconds to run. Fig. 18 shows the CPU utilization and Fig. 19 shows the memory utilization of ShieldCHAIN during the experiment.

3) Discussion: The shieldCHAINs running in organization2 and organization3 were able to retrieve all 100 IOC records that were produced by organization1, a member of this community. After retrieving IOC records from the Blockchain, ShieldCHAIN then inserted these IOCs in the ShieldSDN database tables (attacker, CNC, and target). Before the experiment, there were 0 records in these three tables. After the experiment, there were 100 records in each.



root@ip-192-168-1-100: /home/ubuntu/200210/SIPshield/shieldSDN

```
1|127.173.231.36:5060
1|129.127.99.250:5060
1|162.171.226.154:5060
1|160.101.52.95:5060
1|37.243.172.171:5060
1|10.85.233.87:5060
1|81.139.145.201:5060
1|60.59.6.211:5060
1|180.217.166.54:5060
1|22.128.174.78:5060
1|39.150.127.241:5060
1|78.54.202.139:5060
1|129.173.74.88:5060
1|140.173.177.162:5060
1|20.74.26.170:5060
1|32.205.235.176:5060
1|231.25.208.212:5060
1|198.119.219.75:5060
1|56.94.76.64:5060
1|156.129.103.229:5060
1|140.127.101.249:5060
1|74.183.195.61:5060
1|55.60.156.184:5060
1|177.105.36.38:5060
```

root@ip-10-0-4-100: /home/ubuntu/200210/SIPshield/shieldSDN

```
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|129.127.99.250:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|162.171.226.154:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|160.101.52.95:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|37.243.172.171:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|10.85.233.87:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|81.139.145.201:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|60.59.6.211:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|180.217.166.54:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|22.128.174.78:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|39.150.127.241:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|78.54.202.139:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|129.173.74.88:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|140.173.177.162:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|20.74.26.170:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|32.205.235.176:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|231.25.208.212:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|198.119.219.75:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|56.94.76.64:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|156.129.103.229:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|140.127.101.249:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|74.183.195.61:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|55.60.156.184:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|177.105.36.38:5060
sqlite>
```

root@ip-172-16-5-100: /home/ubuntu/200210/SIPshield/shieldSDN

```
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|129.127.99.250:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|162.171.226.154:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|160.101.52.95:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|37.243.172.171:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|10.85.233.87:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|81.139.145.201:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|60.59.6.211:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|180.217.166.54:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|22.128.174.78:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|39.150.127.241:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|78.54.202.139:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|129.173.74.88:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|140.173.177.162:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|20.74.26.170:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|32.205.235.176:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|231.25.208.212:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|198.119.219.75:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|56.94.76.64:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|156.129.103.229:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|140.127.101.249:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|74.183.195.61:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|55.60.156.184:5060
blockchain:0x9FACE6372e53B5Cc61848340AA194E709773CEa4|177.105.36.38:5060
sqlite>
```

Fig. 17. **Experiment 3:** Identical IOC records in SQL database in organization1(USA), organization2(Singapore), and organization3(UK).

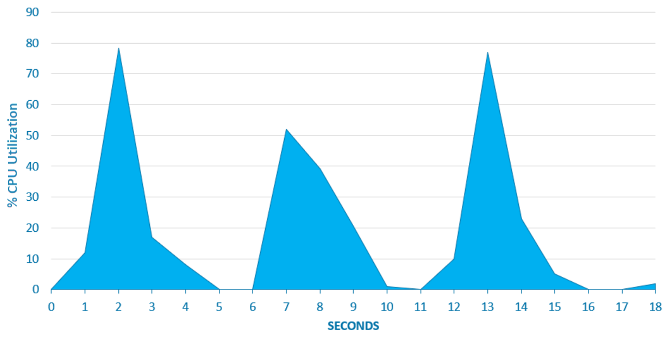


Fig. 18. **Experiment 3:** CPU utilization of ShieldCHAIN

Read operations in the Blockchain occurred much faster than write. Compared to the previous experiment where it took around 5 seconds on average for write operations, read operations took about 1 second on average. Consequently the experiment ran much faster than the previous one. Even though the speed for the read operation is still slower than a regular SQL database, the Blockchain offers unique features that are appropriate for exchanging IOCs between organizations.

This screenshot from the SQL perspective (Fig. 22) shows that the IOC records that were reported by one post of the

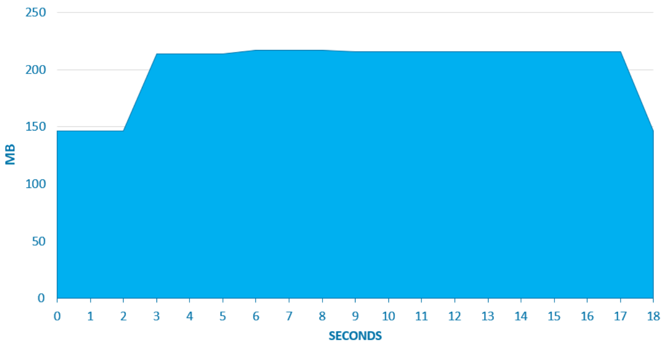


Fig. 19. **Experiment 3:** Memory utilization of ShieldCHAIN

TABLE VII
EXPERIMENT 3: SHIELDCHAIN2 AND SHIELDCHAIN3 RETRIEVES IOCS FROM BLOCKCHAIN AND INSERTS IT TO THEIR SHIELDSDN DATABASE.
(B=BEFORE; A=AFTER)

Dependent Variables	VNF4 @ Org2		SDN4 @ Org2		VNF5 @ Org3		SDN5 @ Org3	
	B	A	B	A	B	A	B	A
#IOC in **register**								
#Filters in **table**								
#IOC in **SQL**			0	100			0	100
#IOC in **Blockchain**								

community (Organization 1 in the USA) have been successfully retrieved and inserted into the database of other members (Organization 2 in Singpoare and Organization 3 in the UK). Out of 100 IOC records on Blockchain, 100 records were created in the SQL database. This result demonstrates a positive outcome for this experiment and ShieldCHAIN has provided a solution to retrieve threat intelligence from the community.

The fact that organization2 and organization3 are able to get these IOCs is significant because they are now prepared to mitigate attacks that match the IOCs. In essence, organization2 and organization3 are able to leverage the detection effort performed by organization1 for attack prevention in their respective organizations. This approach alleviates the need for organization2 and organization3 to repeat detection efforts again for the same attacks. This is how the community collaborates to defend against IoT botnet attacks.

D. *Experiment 4: Packet filter installation at microVNF*

The fourth experiment is in answer to the question, In what way could a switch leverage the community-sourced intelligence as packet filtering rules? If ShieldCHAIN in organization2 and organization3 is working as designed, then it will be able to retrieve IOC records from the SQL Database and install these as packet filtering rules on microVNF.

1) Procedures: The process starts with ShieldCHAIN queries to the ShieldSDN database for IOCs records with query parameter set as ”`switchNo` = blockchain”. This search criterion retrieves only those IOCs that originate from the Blockchain. Next, ShieldCHAIN converts these IOC records and installs these as packet filtering rules on microVNF tables.

2) Result: The result of this experiment is captured in Table VIII below. On the microVNF in organization2 and organization3, there were 0 packet filtering rules before the experiment and 100 rules after the experiment.

The experiment took 80 seconds to run. Fig. 20 shows the CPU utilization and Fig. 21 shows the memory utilization of ShieldCHAIN during the experiment.

Figure 22 shows the content of packet filtering rules on the microVNF in organization1 (USA), organization2 (Singapore), and organization3 (UK). The packet filtering rules installed on all switches are identical.

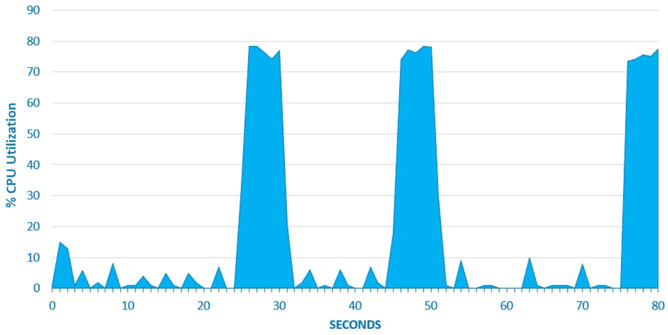


Fig. 20. **Experiment 4:** CPU utilization of ShieldCHAIN

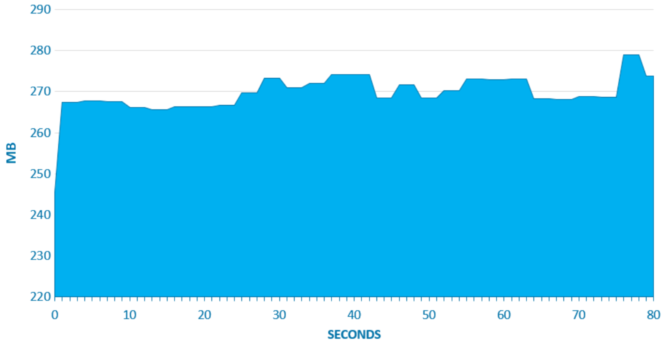


Fig. 21. **Experiment 4:** Memory utilization of ShieldCHAIN

3) Discussion: The ShieldCHAIN that was running in organization2 and organization3 was able to retrieve all 100 IOC records from the SQL database and install them as packet filtering rules on microVNF. The experiment showed that ShieldCHAIN was able to utilize threat intelligence that was generated from within the community and create packet filtering rules from this intelligence. ShieldCHAIN then installed these packet filtering rules on microVNF tables. Out of 100 records of threat intelligence in the database (which were sourced from the community), ShieldCHAIN was able to install 100 packet filtering rules on microVNF. This is

TABLE VIII

EXPERIMENT 4: SHIELDCHAIN QUERIES SHIELDSDN DATABASE TO RETRIEVE IOCS THAT ORIGINATES FROM BLOCKCHAIN, CONVERTS, AND INSERTS IT TO MICROVNF AS PACKET FILTERING RULES. (B=BEFORE; A=AFTER)

Dependent Variables	VNF4 @ Org2		SDN4 @ Org2		VNF5 @ Org3		SDN5 @ Org3	
	B˜	A	B	A	B	A	B	A
#IOC in **register**								
#Filters in **table**	0	100			0	100		
#IOC in **SQL**								
#IOC in **Blockchain**								

statistically significant and demonstrates a positive outcome for this experiment.

In terms of performance, shieldCHAIN performed as expected with high CPU utilization during packet filter insertions into three different tables as depicted in Fig. 10. For memory utilization, shieldCHAIN showing a different pattern where it is steady during the operation, as depicted in Fig. 11.

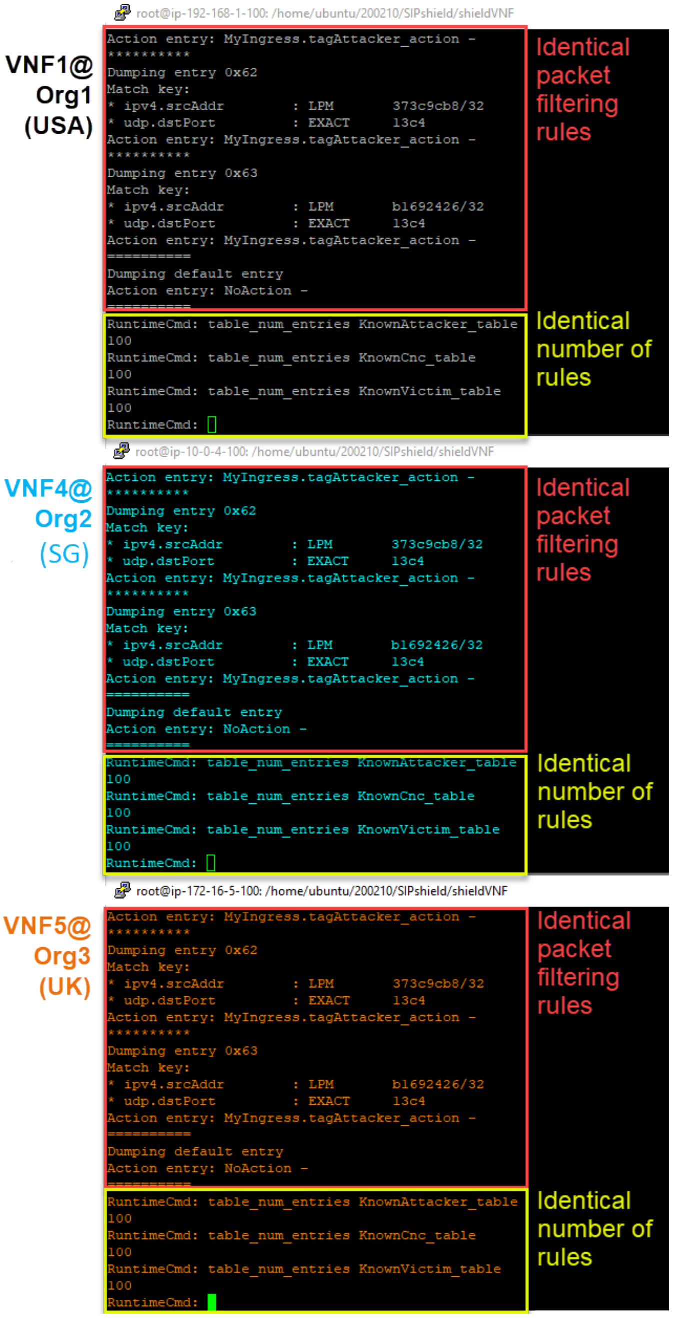
MicroVNF from organization1 (in the USA), organization2 (in Singapore), and organization3 (in the UK) have identical packet filtering rules. As shown in Figure 22, identical packet filtering rules means that these three switches are synchronised in their defence posture even though they are owned by three different organizations. The process was automated and did not involve manual process that tend to delay mitigation effort. This capability means that they will be able to mitigate against the attack should the same botnet outbreak occur in both organizations.

VI. DISCUSSION

Botnet-originated DDoS attacks are a distributed problem that requires a distributed solution. The original contribution of this paper is shieldSDN and shieldCHAIN, a framework that allows a community to synchronize their collaboration on addressing Botnet attacks using non-proprietary technology and infrastructure.

Previous siloed mitigation efforts have not effectively addressed Botnet attacks. A community-based approach is needed where independent organizations each contribute their fair share of responsibilities towards this distributed problem. This approach requires community-wide orchestration that is built on mutual trust. The emergence of new technologies such as P4, SDN, and Blockchain introduces new capabilities for the community to use.

In this paper, we have designed, implemented and tested a distributed framework that combined a programmable control plane (SDN), data plane (P4), and smart contract (Blockchain) enabling different organizations to collaborate. Together, they formed herd immunity against the same strain of botnet DDoS attack by having a synchronized defense at their respective network edge. This synchronized network edge defense approach is aligned with edge computing concepts where the intelligence is distributed and located at the network edges. It protected the smart edge infrastructure and does not allow the attacks to progress deeper into the network.


VNF1@
Org1
(USA)
root@ip-192-168-1-100: /home/ubuntu/200210/SIPshield/shieldVNF
Identical packet filtering rules
Identical number of rules

```
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x62
Match key:
* ipv4.srcAddr         : LPM       373c9cb8/32
* udp.dstPort          : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x63
Match key:
* ipv4.srcAddr         : LPM       b1692426/32
* udp.dstPort          : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
==========
Dumping default entry
Action entry: NoAction -
==========
RuntimeCmd: table_num_entries KnownAttacker_table
100
RuntimeCmd: table_num_entries KnownCnc_table
100
RuntimeCmd: table_num_entries KnownVictim_table
100
RuntimeCmd:
```

VNF4@
Org2
(SG)
root@ip-10-0-4-100: /home/ubuntu/200210/SIPshield/shieldVNF
Identical packet filtering rules
Identical number of rules

```
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x62
Match key:
* ipv4.srcAddr         : LPM       373c9cb8/32
* udp.dstPort          : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x63
Match key:
* ipv4.srcAddr         : LPM       b1692426/32
* udp.dstPort          : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
==========
Dumping default entry
Action entry: NoAction -
==========
RuntimeCmd: table_num_entries KnownAttacker_table
100
RuntimeCmd: table_num_entries KnownCnc_table
100
RuntimeCmd: table_num_entries KnownVictim_table
100
RuntimeCmd:
```

VNF5@
Org3
(UK)
root@ip-172-16-5-100: /home/ubuntu/200210/SIPshield/shieldVNF
Identical packet filtering rules
Identical number of rules

```
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x62
Match key:
* ipv4.srcAddr         : LPM       373c9cb8/32
* udp.dstPort          : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
**********
Dumping entry 0x63
Match key:
* ipv4.srcAddr         : LPM       b1692426/32
* udp.dstPort          : EXACT     13c4
Action entry: MyIngress.tagAttacker_action -
==========
Dumping default entry
Action entry: NoAction -
==========
RuntimeCmd: table_num_entries KnownAttacker_table
100
RuntimeCmd: table_num_entries KnownCnc_table
100
RuntimeCmd: table_num_entries KnownVictim_table
100
RuntimeCmd:
```

Fig. 22. **Experiment 4:** ShieldCHAIN created identical packet filtering rules on VNF in organization2 (Singapore) and organization3 (UK).

Scaling synchronized defense among independent organizations requires automation and trust. Automation is necessary to reduce labor requirements and delays. Trust is achieved through transparency and accountability. ShieldSDN uses SDN technology to automate the installation of packet filters on the edge switches. ShieldCHAIN uses Blockchain technology to automate sharing IOCs as immutable records that every member can verify. This level of transparency and accountability validates that they come from trusted sources. The same collaboration is also applicable in a private blockchain network should the need arise for a private consortium model. In this collaborative framework, one organization's attack detection efforts can be leveraged as attack prevention by other organizations in the community.

While public blockchains require high computational resources and have inherent delays in performance, the benefits still outweigh the risk of being unprepared for botnet attacks. Currently there are works on Ethereum that aim to cut energy consumption by 99 percent We anticipate that high computational requirements will be sufficiently addressed in future. With regard to delay in performance, we consider 5 seconds to write (observed in Experiment 2) and 1 second to read a transaction (observed in Experiment 3) to be sufficient for the use case of sharing threat intelligence. From the perspective of writing and retrieving information, this level of performance is good enough to inform peers about indicators of compromise. Increasing polling frequency by each peer could also compensate for this delay so that they can be informed as soon as new indicators become available.

The use of a public blockchain is a low-cost entry option to establish viability and benefit. Given the pace of development and change of private blockchains, the public blockchain infrastructure presents a low-cost option to validate the viability of the framework. When features and benefits are identified and established on a public blockchain, we would then have good criteria to select among multiple private blockchain options and make further improvements.

VII. Future Works

Future works for shieldSDN can consider exploring areas such as dynamic filtering and routing based on machine learning calculations at the network edge. For example, the raw data from community members can provide input to train a machine learning model that could predict active Botnet outbreaks and generate proactive filtering/routing rules. These rules can then be enriched with community-relevant attributes and distributed back to the community. This model would create a perpetually learning defense system specific to a particular community. Furthermore, this community-specific prediction could be tailored to serve a specific industry sector, e.g., finance, healthcare, manufacturing.

Future works for shieldCHAIN can consider adding richer social dimensions to encourage positive contribution and discourage negative influence in the community. For example, the smart contract can be programmed to enforce the reward and punishment mechanisms. Punishments are meant to prevent abuse or participation that is not helping to accomplish the tasks. For example, frequently sharing non-relevant data would only add the noise and make it challenging to extract the critical signals from the data. The punishment could be in the form of negative feedback given by other members, which will hurt their online reputations over time. On the contrary, rewards are meant to encourage and acknowledge the positive contribution from the community. For example, members can vote for meaningful IoC contributions from other community members. Over time, these votes build street credibility among their peers, and they are recognized as the most valuable members of the community.

Future works may consider enhancing shieldCHAIN with a Decentralized Autonomous Organization (DAO). Ethereum supports smart contracts that can serve as DAO. A loosely-connected community can organize itself as a shieldDAO, an autonomous organization that turns its processes into programmable code. For example, they can code their membership application, membership fee, election, and voting process, just like a typical organization. In this instance, the decision and direction of this DAO are made by a voting process. These decisions can then ensure that the functions of shieldSDN and shieldCHAIN are maintained and keep serving the community's best interest. In this model, shieldDAO provides organizational oversight for shieldSDN and shieldCHAIN operation.

Another potential future work is in the area of NFT. A well-managed shieldDAO benefits its members by delivering intelligence feeds and permission to collaborate with their community. The membership of shieldDAO can be made exclusive and could be codified as an NFT. In the physical world, there are Information Sharing and Analysis Center (ISAC) organizations where each member contributes financially to the operation of such an organization. ShieldDAO could raise funds by selling a limited number of shieldNFT to its members. This method introduced exclusivity and scarcity to shieldNFT, which could be valued and assessed like collectible items. However, shieldNFT brings real-world utility to receiving intelligence feeds and membership instead of just collectible items. Like other NFTs, shieldNFT can be sold and bought in a market. As the demand to join this shieldDAO increased or decreased, the monetary value of shieldNFT would fluctuate accordingly. In this model, the members are interested in keeping the shieldDAO community helpful and constantly delivering benefits so that the value of their shieldNFTs keeps increasing.

Future work can also enrich the IOC data to reflect the reputation of IoT device manufacturers. For example, adding the MAC address, device type, and firmware version of the IoT device to the IOC data allows the community to learn about vulnerabilities. Information on MAC addresses enables the users to trace the manufacturer. Observations about high-frequency attacks from specific IoT devices suggest that the manufacturer did not follow secure by design principles [58] when producing these devices. This information could influence customers about their purchasing decisions in the future. In addition, it could affect contracts and bids awarded to manufacturers. The idea is to introduce financial consequences to these manufacturers that produce IoT devices with a weak security posture to maximize profit. The underlying assumption is that an adequately hardened IoT device would not be easily attacked and recruited to join a botnet. Poorly designed and secured devices create a botnet and enable them to deliver powerful attacks. Introducing economic impacts is a powerful leverage that we could use to address the root cause of Botnet attacks.

VIII. Conclusion

In this study, we performed experiments in synchronizing a DDoS Defense at Network edge with P4, SDN, and Blockchain. The outcome of these experiments support the hypothesis that these components (microVNF, shieldSDN, and shieldCHAIN) can be orchestrated to scale a DDoS defense against botnet by synchronizing packet filters on the edge networks. The scope of defense and synchronization is initially within one organization (intra-organization) and subsequently extended to other organizations (inter-organization).

Four successful experiments were performed to validate the hypothesis on shieldSDN and shieldCHAIN: to validate that shieldSDN can distribute the IOC from one switch to multiple switches within an organization; that the shieldCHAIN-agent in the publishing-organization can share IOC to Blockchain; that the shieldCHAIN-agent in the subscribing-organization can receive IOC from Blockchain; and that the shieldCHAIN-agent in the subscribing-organization can install packet filter on the edge switch.

Some limitations were observed relating to the performance of the public Blockchain, as depicted in Fig. 10 which is beyond the user's control. This factor needs to be considered when building the solution. For more predictable performance and enterprise-grade features, one might consider deploying the same solution but using private and commercial blockchain platforms. In that scenario the cloud provider is offering an enterprise-grade technical support that is backed by a Service Level Agreement (SLA). Future works for ShieldCHAIN have been discussed extensively in the paper which can make ShieldCHAIN more intelligent, user-friendly, and applicable.

The proliferation of IoT brings new opportunities and threats to the community. The threat actors hijacked these IoT devices for their benefit and profit. As we all have observed through the reported incidents, the isolated defense approach is no longer sufficient to address this problem. DDoS attacks are not going away anytime soon, and they tend to get bigger and more frequent. A different approach is required, and the community needs to come together. Collectively, the community can form a synchronized defense at their network edge and contribute to the overall detection and mitigation capability. This approach creates a distributed solution to the distributed problem. Addressing the DDoS problem allows us to protect our progress in IoT so that it is working for us instead of being used against us.

References

[1] D. J. Trump, "Presidential executive order on strengthening the cybersecurity of federal networks and critical infrastructure." https://www.whitehouse.gov/presidential-actions/presidential-executive-order-strengthening-cybersecurity-federal-networks-critical-infrastructure/, 2017.

[2] The Secretary of Commerce and Homeland Security, "A Report to the President on Enhancing the Resilience of the Internet and Communications Ecosystem Against Botnets and Other Automated, Distributed Threats." https://csrc.nist.gov/CSRC/media/Publications/white-paper/2018/05/30/enhancing-resilience-against-botnets–report-to-the-president/final/documents/eo$_1 3800_b otnet_r eport_{-f} inalv2.pdf$, 2018.

[3] The Secretary of Commerce and Homeland Security, "A Road Map Toward Resilience Against Botnets." https://www.commerce.gov/sites/default/files/2018-11/Botnet%20Road%20Map%20112918%20for%20posting_0.pdf, 2018.

[4] K. Boeckl, M. Fagan, W. Fisher, N. Lefkovitz, K. N. Megas, E. Nadeau, D. Gabel, O. . Rourke, B. Piccarreta, and K. Scarfone, "NISTIR 8228 Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks," p. 44, 2019.

[5] M. Fagan, M. Yang, A. Tan, L. Randolph, and K. Scarfone, "Security Review of Consumer Home IoT Products," *Nist*, p. 41, 2019.

[6] M. Fagan, K. N. Megas, K. Scarfone, and M. Smith, "Foundational cybersecurity activities for IoT device manufacturers," tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, may 2020.

[7] E. Lear, W. C. Barker, D. Cohen, and J. Harrington, "NIST SPECIAL PUBLICATION 1800-15C Securing Small-Business and Home Internet of Things (IoT) Devices," 2020.

[8] E. Lear, R. Droms, and D. Romascanu, "RFC 8520: Manufacturer Usage Description Specification," *Internet Engineeging Task Force*, 2019.

[9] M. Bjorklund, "The yang 1.1 data modeling language," RFC 7950, RFC Editor, August 2016.

[10] T. Bray, "The javascript object notation (json) data interchange format," STD 90, RFC Editor, December 2017.

[11] M. Jethanandani, S. Agarwal, L. Huang, and D. Blair, "Yang data model for network access control lists (acls)," RFC 8519, RFC Editor, March 2019.

[12] L. Lhotka, "Json encoding of data modeled with yang," RFC 7951, RFC Editor, August 2016.

[13] R. Housley, "Cryptographic message syntax (cms)," STD 70, RFC Editor, September 2009. http://www.rfc-editor.org/rfc/rfc5652.txt.

[14] P. Watrobski, J. Klosterman, and M. Souppaya, "Draft NIST CSWP, Methodology for Characterizing Network Behavior of Internet of Things Devices," 2020.

[15] S. Symington and W. Polk, "Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management (Draft)," 2020.

[16] W. Kumari and D. McPherson, "Remote triggered black hole filtering with unicast reverse path forwarding (urpf)," RFC 5635, RFC Editor, August 2009.

[17] P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, and D. McPherson, "Dissemination of flow specification rules," RFC 5575, RFC Editor, August 2009. http://www.rfc-editor.org/rfc/rfc5575.txt.

[18] F. Baker and P. Savola, "Ingress filtering for multihomed networks," BCP 84, RFC Editor, March 2004. http://www.rfc-editor.org/rfc/rfc3704.txt.

[19] K. Sriram, D. Montgomery, and J. Haas, "Enhanced feasible-path unicast reverse path forwarding," BCP 84, RFC Editor, February 2020.

[20] C. Morrow and R. Dobbins, "Ddos open threat signaling (dots) working group operational requirements," in *Proc. IETF 93 Prague*, 2015.

[21] E. Osterweil, A. Stavrou, and L. Zhang, "21 years of distributed denial-of-service: A call to action," *Computer*, vol. 53, no. 8, pp. 94–99, 2020.

[22] M. Nawrocki, J. Blendin, C. Dietzel, T. C. Schmidt, and M. Wählisch, "Down the black hole: Dismantling operational practices of bgp blackholing at ixps," in *Proceedings of the Internet Measurement Conference*, IMC '19, (New York, NY, USA), p. 435–448, Association for Computing Machinery, 2019.

[23] V. Giotsas, G. Smaragdakis, C. Dietzel, P. Richter, A. Feldmann, and A. Berger, "Inferring bgp blackholing activity in the internet," in *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[24] C. Dietzel, M. Wichtlhuber, G. Smaragdakis, and A. Feldmann, "Stellar: Network attack mitigation using advanced blackholing," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, (New York, NY, USA), p. 152–164, Association for Computing Machinery, 2018.

[25] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting volumetric attacks on lot devices via sdn-based monitoring of mud activity," in *Proceedings of the 2019 ACM Symposium on SDN Research*, SOSR '19, (New York, NY, USA), p. 36–48, Association for Computing Machinery, 2019.

[26] A. Hamza, H. H. Gharakheili, and V. Sivaraman, "Combining mud policies with sdn for iot intrusion detection," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, IoT Samp;P '18, (New York, NY, USA), p. 1–7, Association for Computing Machinery, 2018.

[27] Y. Afek, A. Bremler-Barr, D. Hay, R. Goldschmidt, L. Shafir, G. Avraham, and A. Shalev, "Nfv-based iot security for home networks using mud," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, 2020.

[28] Y. Afek, A. Bremler-Barr, D. Hay, L. Shafir, and I. Zhaika, "Nfv-based iot security at the isp level," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–2, IEEE, 2020.

[29] P. Bull, R. Austin, E. Popov, M. Sharma, and R. Watson, "Flow based security for iot devices using an sdn gateway," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 157–163, 2016.

[30] M. Ozcelik, N. Chalabianloo, and G. Gur, "Software-Defined Edge Defense Against IoT-Based DDoS," in *17th IEEE International Conference on Computer and Information Technology*, 2017.

[31] S. S. Bhunia and M. Gurusamy, "Dynamic attack detection and mitigation in iot using sdn," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–6, 2017.

[32] A. Molina Zarca, J. Bernal Bernabe, I. Farris, Y. Khettab, T. Taleb, and A. Skarmeta, "Enhancing iot security through network softwarization and virtual security appliances," *International Journal of Network Management*, vol. 28, no. 5, p. e2038, 2018. e2038 nem.2038.

[33] 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.

[34] D. Yin, L. Zhang, and K. Yang, "A ddos attack detection and mitigation with software-defined internet of things framework," *IEEE Access*, vol. 6, pp. 24694–24705, 2018.

[35] A. Al Shorman, H. Faris, and I. Aljarah, "Unsupervised intelligent system based on one class support vector machine and grey wolf optimization for iot botnet detection," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 7, pp. 2809–2825, 2019.

[36] Y. Afek, A. Bremler-Barr, D. Hay, L. Shafir, and I. Zhaika, "Demo: Nfv-based iot security at the isp level," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–2, 2020.

[37] V. Andalibi, J. Dev, D. Kim, E. Lear, and L. J. Camp, "Is visualization enough? evaluating the efficacy of mud-visualizer in enabling ease of deployment for manufacturer usage description (mud)," in *Annual Computer Security Applications Conference*, ACSAC, (New York, NY, USA), p. 337–348, Association for Computing Machinery, 2021.

[38] Barret, M., "Framework for Improving Critical Infrastructure Cybersecurity Version 1.1, NIST Cybersecurity Framework,[online]," 2018.

[39] F. Bannour, S. Souihi, and A. Mellouk, "Distributed sdn control: Survey, taxonomy, and challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 333–354, 2018.

[40] 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.

[41] Netbergtw. https://netbergtw.com/products/aurora-710/, 2020.

[42] Netronome. https://www.netronome.com/products/agilio-cx/, 2020.

[43] Pensando, "Pensando dsc-25 distributed services card." https://pensando.io/wp-content/uploads/2020/03/Pensando-DSC-25-Product-Brief.pdf, 2020.

[44] Cisco, "Cisco silicon one." https://www.cisco.com/c/en/us/solutions/service-provider/innovation/silicon-one.html?dtid=osscdc000283/, 2020.

[45] M. Casado, A. Horowitz, N. Mckeown, and S. Shenker, "SDN History," tech. rep., 2019.

[46] C. Ünsalan, H. D. Gürhan, and M. E. Yücel, *Memory Operations*, pp. 293–340. Cham: Springer International Publishing, 2022.

[47] O. Kupreev, E. Badovskaya, and A. Gutnikov, "Ddos attacks in q2 2020," 2020.

[48] Mininet. https://mininet.org/, 2020.

[49] p4lang, "Github - p4lang/behavioral-model: The reference p4 software switch." https://github.com/p4lang/behavioral-model, 2020.

[50] OpenJS Foundation. https://nodejs.org/en/, 2020.

[51] Microsoft, "Azure functions serverless compute — microsoft azure." https://azure.microsoft.com/en-us/services/functions/, 2020.

[52] Amazon. https://aws.amazon.com/lambda/, 2020.

[53] Google, "Cloud functions — google cloud." https://cloud.google.com/functions, 2020.

[54] SQLite, "Sqlite home page." https://www.sqlite.org/, 2020.

[55] "Ethereum API — IPFS API amp; Gateway — ETH Nodes as a Service — Infura." https:/infura.io. Web. Accessed 17 Apr. 2022.

[56] A. Febro, H. Xiao, J. Spring, and B. Christianson, "Edge security for sip-enabled iot devices with p4," *Computer Networks*, vol. 203, p. 108698, 2022.

[57] P. Fairley, "Ethereum plans to cut its absurd energy consumption by 99 percent," *IEEE spectrum*, vol. 2, 2019.

[58] "Good Practices for Security of IoT - Secure Software Development Lifecycle." https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1. Web. Accessed 30 Apr. 2022.