Securing the Edges of IoT Networks:

A Scalable SIP DDoS Defense Framework With VNF, SDN, and Blockchain

Aldo Kiki Febro

Department of Computer Science

This dissertation is submitted to the University of Hertfordshire
in partial fulfilment of the requirements of the degree of
Doctor of Philosophy (Ph.D.)

October 2020

I would like to dedicate this thesis to Jesus Christ, my Lord and Savior, who has given me the opportunity and means to pursue this study. Your grace has sustained me through these years. Soli Deo Gloria.

Acknowledgements

I would like to express my deep appreciation to my Supervision Team: Dr. Hannan Xiao, Dr. William Joseph Spring, and Professor Bruce Christianson. They are a wealth of knowledge and experience, which has helped me tremendously and gave me a model for thinking and writing as a world-class computer science researcher.

I would like to extend my deep gratitude to Continuant Chief Information Officer, Scott Graham, for trusting me with the balancing act between work and study. Thank you for writing a recommendation at the beginning of this program and giving me opportunities to present my research nationally and overseas.

I would like to thank my parents for instilling confidence in me, modeling a strong work ethic, and trusting the Lord. When I didn't do so well in school, my mom often said that it is not because I'm stupid, but because I wasn't paying attention. That simple comment has built my confidence over the years.

I am incredibly grateful to my wife, Cecielia, and my son, Nobel, for your unwavering support and patience. Thank you for your sacrifice and understanding during the nights, weekends, and holidays when I needed the time and space to focus on my research.

Abstract

An unintended consequence of the global deployment of IoT devices is that they provide a fertile breeding ground for IoT botnets. An adversary can take advantage of an IoT botnet to launch DDoS attacks against telecommunication services. Due to the magnitude of such an attack, legacy security systems are not able to provide adequate protection. The impact ranges from loss of revenue for businesses to endangering public safety.

This risk has prompted academia, government, and industry to reevaluate the existing defence model. The current model relies on point solutions and the assumption that adversaries and their attacks are readily identifiable. But adversaries have challenged this assumption, building a botnet from thousands of hijacked IoT devices to launch DDoS attacks. With botnet DDoS attacks there are no clear boundary where the attacks originate and what defensive measures to use.

The research question is: in what ways programmable networks could defend against Session Initiation Protocol (SIP) Distributed Denial-of-Service (DDoS) flooding attacks from IoT botnets? My significant and original contribution to the knowledge is a scalable and collaborative defence framework that secures the edges of IoT networks with Virtual Network Function (VNF), Software-Defined Networking (SDN), and Blockchain technology to prevent, detect, and mitigate SIP DDoS flooding attacks from IoT botnets.

Successful experiments were performed using VNF, SDN, and Blockchain. Three kinds of SIP attacks (scan, brute force, and DDoS) were launched against a VNF running on a virtual switch and each was successfully detected and mitigated. The SDN controller gathers threat intelligence from the switch where the attacks originate and installs them as packet filtering rules on all switches in the organisation. With the switches synchronised, the same botnet outbreak is prevented from attacking other parts of the organisation. A distributed application scales this framework further by writing the threat intelligence to a smart contract on the Ethereum Blockchain so that it is available for external organisations. The receiving organisation retrieves the threat intelligence from the smart contract and installs them as packet filtering rules on their switches. In this collaborative framework, attack detection/mitigation efforts by one organisation can be leveraged as attack prevention efforts by other organisations in the community.

Table of contents

List of figures **xv**

List of tables **xxiii**

Nomenclature **xxvii**

1 Introduction **1**

1.1 Research Context 1
1.2 Research Motivations 2
1.3 Research Aim and Objectives 3
1.4 Research Question and Thesis Statement 4
1.5 Research Challenges and Realities 4
1.5.1 Defender's Perspective: Lack of Resources and Unprepared 5
1.5.2 Attacker's Perspective: A Profitable Business Model 7
1.5.3 Government's Perspective: Regulation & Compliance 8
1.6 Research Methods 8
1.6.1 Literature Review and Research Gaps 9
1.6.2 Research and Design of Experiments 10
1.6.3 Implementation 11
1.6.4 Data Collection and Analysis 12
1.7 Contributions 15
1.8 Publications 17
1.9 Dissertation Outline 19

2 Literature Review of SIP DDoS Flooding Defence & Potential Solutions **21**

2.1 Introduction 21
2.2 Evolution of DoS attacks 21
2.2.1 DoS Attacks 22
2.2.2 SIP DDoS attacks 22

2.2.3 SIP DDoS Flooding Attack from IoT Botnet 25
2.3 Defence against DDoS Flooding Attack 27
2.3.1 Taxonomy on DDoS defence Mechanism 27
2.3.2 Previous Works on Mitigating DDoS Flooding Attack 30
2.3.3 Research Gaps 30
2.4 Defence Against SIP DDoS Flooding Attack 31
2.4.1 Previous Works on Mitigating SIP-layer DDoS Flooding Attack . . 31
2.4.2 Research Gaps 34
2.5 Defence Against IoT Botnet Flooding Attack 36
2.5.1 Previous Works on Mitigating DDoS Flooding Attack from IoT Botnet 36
2.5.2 Research Gaps 40
2.6 New Technologies For Potential Solutions 41
2.6.1 VNF 41
2.6.2 SDN 42
2.6.3 Ethereum Blockchain and Smart Contract 45
2.6.4 CTI 46
2.7 Chapter Summary 47

3 SIPshield: A Scalable SIP DDoS Defence Against Botnet Attack 49
3.1 Overview 49
3.2 Framework in Designing DDoS Defence System 50
3.3 Proposed Approach: Securing the Network Edge & Community-based Defence 51
3.3.1 Botnet Attack Demands New Defence Model 51
3.3.2 New Defence Model Involves Smart & Secured Network Edge . . . 52
3.3.3 New Defence Model Involves a Community 53
3.4 Proposed Solution: SIPshield, a Scalable and Collaborative Defence Framework 54
3.4.1 ShieldVNF 54
3.4.2 ShieldSDN 54
3.4.3 ShieldCHAIN 55
3.5 Evaluating SIPshield 56
3.5.1 By Intellectual Roots 56
3.5.2 By Addressing Research Gaps 56
3.5.3 By DDoS Defence Taxonomy and Existing Solutions 58
3.5.4 By Principles of DDoS defence 63
3.6 Chapter Summary 66

4 ShieldVNF: SIP DDoS Defence on a Switch 69
4.1 Overview . . . 69
4.2 The Proposed Solution: ShieldVNF . . . 70
4.2.1 Preventive and Reactive Defence Measures Against IoT botnet Attacks 71
4.2.2 Research Questions . . . 72
4.2.3 ShieldVNF: SIP DDoS Defence At The Network Edge . . . 72
4.3 Design of Experiments . . . 74
4.3.1 Random Sampling and Random Group Assignment . . . 74
4.3.2 Independent and Dependent Variables . . . 75
4.3.3 Data Collection and Measurements . . . 78
4.4 Instrumentation and Implementation . . . 83
4.4.1 Cloud Computing Environment . . . 85
4.4.2 Virtual Network, Virtual Switch, & Virtual Hosts . . . 85
4.4.3 VNF . . . 86
4.4.4 A Walkthrough of Packet and Session Tracking . . . 91
4.5 Experiments . . . 93
4.5.1 Experiment 1: SIP scanning attack . . . 94
4.5.2 Experiment 2: SIP enumeration attack . . . 98
4.5.3 Experiment 3: SIP brute force attack . . . 102
4.5.4 Experiment 4: Command-and-Control Channel Establishment . . . 106
4.5.5 Experiment 5: SIP DoS attack . . . 111
4.5.6 Experiment 6: SIP Distributed DoS attack . . . 116
4.6 Chapter Summary . . . 121

5 ShieldSDN: Scaling SIP DDoS Defence to Multiple Switches 123
5.1 Overview . . . 123
5.2 The Proposed Solution: ShieldSDN . . . 124
5.2.1 A Synchronised Defence Posture Approach . . . 124
5.2.2 ShieldSDN: SDN Controller for IOC Management . . . 125
5.3 Design of Experiments . . . 126
5.3.1 Sequence and Workflow . . . 126
5.3.2 Independent and Dependent Variables . . . 127
5.3.3 Data Collection and Measurements . . . 127
5.4 Instrumentation and Implementation . . . 128
5.4.1 SDN Controller . . . 128
5.4.2 Southbound Interface . . . 129
5.4.3 IOC Database . . . 131

5.5 Experiments . . . 131
5.5.1 Environment Setup . . . 132
5.5.2 Experiment 1: Automated IOC Collection and Packet Filter Installation On Multiple Switches . . . 137
5.5.3 Experiment 2: Automated IOC Expiration Management . . . 144
5.5.4 Experiment 3: Automated Persistent Threat Management . . . 149
5.6 Chapter Summary . . . 156

6 ShieldCHAIN: Scaling SIP DDoS Defence to Multiple Organisations 157
6.1 Overview . . . 157
6.2 The Proposed Solution: ShieldCHAIN . . . 159
6.2.1 Collaborative Defence Framework Approach & Its Challenges . . . 159
6.2.2 ShieldCHAIN: Blockchain Application for a Collaborative Defence Framework . . . 160
6.3 Design of Experiments . . . 165
6.3.1 Sequence and Workflow . . . 165
6.3.2 Independent and Dependent Variables . . . 165
6.3.3 Data Collection and Measurements . . . 165
6.4 Instrumentation and Implementation . . . 167
6.4.1 Smart Contract . . . 167
6.4.2 DApp . . . 168
6.4.3 Infura: REST API for Ethereum . . . 169
6.5 Experiments . . . 170
6.5.1 Environment Setup . . . 171
6.5.2 Experiment 1: Automated IOC Distribution to Blockchain . . . 174
6.5.3 Experiment 2: Automated IOC Collection From Blockchain . . . 182
6.5.4 Experiment 3: Automated Packet Filter Installation at ShieldVNF . 187
6.6 Chapter Summary . . . 192

7 Conclusion 195
7.1 Overview . . . 195
7.2 Research Findings . . . 195
7.2.1 ShieldVNF: SIP DDoS Defence At The Network Edge . . . 196
7.2.2 ShieldSDN: SDN Controller for IOC Management . . . 197
7.2.3 ShieldCHAIN: Blockchain Application for Collaborative Defence Framework . . . 197
7.3 Contribution to Knowledge . . . 198

7.3.1 Research Significance and Impact 199
7.3.2 Potential Contributions to Other Research 200
7.3.3 Current Limitations 202
7.4 Future Work 203
7.4.1 ShieldVNF 204
7.4.2 ShieldSDN 205
7.4.3 ShieldCHAIN 206
7.5 Closing 207

References **209**

Index **221**

List of figures

1.1 Three research objectives: (1) Protection for a single switch, (2) Scale the protection to multiple switches in the same organisation, (3) Scale the protection to multiple organisations. 4
1.2 Literature review moving from broad to specifics. 9
1.3 Data generated from the legitimate caller's perspective: The number of successful calls during the test. 13
1.4 Data generated from the attacker's perspective: The number of attack packets sent. 14
1.5 Data generated from the switch's perspective: The number of packets from/to a caller at `(s1-eth1)`, a botnet at `(s1-eth2)`, and an upstream switch at `(s1-eth3)`. 15
1.6 SIPshield and its components (shieldVNF, ShieldSDN, & ShieldCHAIN). Each participating organisation has these components in their network . . . 16
1.7 Dissertation Map 19

2.1 SIP Attacks and Threats Model as per RFC3261 (Rosenberg et al., 2002) . . 23
2.2 Taxonomy on SIP DDoS Attack (Ehlert et al., 2010) 24
2.3 Taxonomy of DDoS defence Mechanisms based on the action, deployment location, layers, and throughput. Adapted from (Agrawal and Tapaswi, 2019; Mirkovic and Reiher, 2004; Zargar et al., 2013). 28
2.4 Taxonomy of DDoS defence Mechanisms based on where the solution is deployed: Source, Network, Destination, and Hybrid. The source-end defence solution is deployed in Autonomous System 1 (AS#1). The destination-end defence solution is deployed in AS#5. The network-based defence solution is deployed in the intervening networks AS#2, AS#3, AS#4. The hybrid-based defence solution is a combination of these. 28
2.5 Taxonomy of DDoS defence Mechanisms based on the timing: before the attack (prevention), during the attack (detection), and after the attack (mitigation) 29

2.6 The programmable data plane is gaining momentum in academia and industry. Products are readily available in the market from chipssets, and switches, to NIC cards. 44

3.1 SIPshield and its components (ShieldVNF, ShieldSDN, & ShieldCHAIN). Each participating organisation has these components in their network . . . 55
3.2 Intellectual genealogy of SIPshield. 57
3.3 Mapping SIPshield capabilities (in orange) to DDoS defence Mechanism Taxonomy. 58

4.1 ShieldVNF mitigates attacks and generates IOC about the attack. 70
4.2 Different stages of infection require different measures. Before the infection, it requires defence against scan attacks, enumerate attacks, brute force attack, and blocking against malware download. After the infection (when the IoT device has turned into a botnet), it requires blocking against connection attempts to the CNC server and mitigate attacks against the victim. 71
4.3 The only difference between control vs. experimental group is the use of ShieldVNF as the independent variable. 75
4.4 Three dependent variables involved during SIP DoS flooding attack experiment: # of successful SIP calls made, # of attack packets received, and # of packets sent upstream. Table 4.3 shows the relationships between these variables in a failed vs. successful experiment. 77
4.5 sipvicious_svmap tool discovered 2 SIP servers in the target range. 79
4.6 sipvicious_svwar tool discovered 12 valid extensions on a SIP server. 10 extensions require authentication and 2 extensions do not require authentication. 80
4.7 sipvicious_svcrack tool cracked 1 password for extension 101 on SIP server 10.0.0.1. 80
4.8 Mirai bot registration (taken from a real Mirai session dataset) 81
4.9 One Bot has successfully registered with the simulated CNC server. 82
4.10 SIPp shows 60 successful calls (4) and other details like start time (1), elapsed time (2), and call rate, which is 1 call per second (3) 83
4.11 Netstat tool shows the number of packets sent and received by each host. In this example, h1 and h10 received 119 packets. 83

4.12 By looking at the number of packets sent/received, we can see whether ShieldVNF has successfully mitigated an attack. In this example, ShieldVNF received 1000 attack packets from botnet. The ShieldVNF on the left failed to mitigate the attack and forwarded 1000 malicious packets upstream, whereas the ShieldVNF on the right has successfully mitigated the attack and forward only legitimate packets to the upstream switch. 84
4.13 Three AWS Virtual Private Cloud (US, UK, and SG) and the EC2 virtual machine instances within each Virtual Private Cloud. 84
4.14 A virtual network environment running on an AWS EC2 instance that consists of a virtual switch and 11 virtual hosts (1 for attacker and 10 for IoT devices). 86
4.15 ShieldVNF workflow has 4 phases: Parse, Inspect, Verify, and Count. . . . 87
4.16 Using counting bloom filter (CBF) data structure to keep track of the number of SIP packets. This packet is hashed to index 231 & 489. Since this is the first time the CBF encountered this packet, the value for index 231 & 489 is set to 1. 90
4.17 Northbound: From camera To server: Counting Bloom Filter sets the number of active SIP session to 1. A botnet infection and DoS attack is suspected when this number keeps increasing. 91
4.18 Southbound: From server To camera: Counting Bloom Filter resets the number of active SIP session to 0. 92
4.19 **Experiment 1:** Attackers scanning for SIP servers (Test1 with legacy switch vs. Test2 with ShieldVNF) 94
4.20 **Experiment 1: Test1 Result:** SIP Scan attack with legacy switch. Attacker1 (left), Attacker2(middle), Attacker3(right) found 3 servers. 96
4.21 **Experiment 1: Test1:** CPU utilisation of legacy switch. 96
4.22 **Experiment 1: Test2 Result:** SIP Scan attack with ShieldVNF. Attacker1 (left), Attacker2(middle), Attacker3(right) found 10 servers. 96
4.23 **Experiment 1: Test2:** CPU utilisation of ShieldVNF. 97
4.24 **Experiment 2:** Attackers scanning for valid accounts/extension on a SIP Proxy server (Test1 with legacy switch vs. Test2 with ShieldVNF) 98
4.25 **Experiment 2: Test1 Result:** SIP Enumeration attack with legacy switch. Attacker1 (left), Attacker2(middle), Attacker3(right) found 11 accounts. . . 99
4.26 **Experiment 2: Test1:** CPU utilisation of legacy switch. 100
4.27 **Experiment 2: Test 2 Result:** SIP Enumeration attack with ShieldVNF. Attacker1 (left), Attacker2(middle), Attacker3(right) found 3 accounts. . . . 100
4.28 **Experiment 2: Test2:** CPU utilisation of ShieldVNF. 101

4.29 **Experiment 3:** Attackers brute forcing the extensions on a SIP server for valid password (Test1 with legacy switch vs. Test2 with ShieldVNF) 102
4.30 **Experiment 3: Test1 Result:** SIP brute force attack with legacy switch. Attacker1 (left), Attacker2(middle), Attacker3(right) cracked 10 passwords. 104
4.31 **Experiment 3: Test1:** CPU utilisation of legacy switch. 105
4.32 **Experiment 3: Test2 Result:** SIP brute force attack with ShieldVNF. Attacker1 (left), Attacker2(middle), Attacker3(right) found nothing. 105
4.33 **Experiment 3: Test2:** CPU utilisation of ShieldVNF. 105
4.34 **Experiment 4:** IoT-turned-to-botnet attempting to register to the centralised Command and Control (CNC) server (Test1 with legacy switch vs. Test2 with ShieldVNF) 106
4.35 **Experiment 4: Test1:** CPU utilisation of legacy switch. 107
4.36 **Experiment 4: Test1 Result:** Bot registration attempt with legacy switch. 10 Bots registered at the Command-and-Control (CNC) server. 109
4.37 **Experiment 4: Test2:** CPU utilisation of ShieldVNF. 110
4.38 **Experiment 4: Test2 Result:** Bot registration attempt with ShieldVNF. 0 Bot registered at the Command-and-Control (CNC) server. 110
4.39 **Experiment 5:** SIP DoS attack from a single attacker (Test1 with legacy switch vs. Test2 with ShieldVNF) 111
4.40 **Experiment 5: Test1 Result:** SIP DoS attack to a SIP Server from 1 bot with legacy switch. Calls were disrupted when the attack started. 20 successful calls completion. 113
4.41 **Experiment 5: Test1 Result:** SIP DoS attack to a SIP Server from 1 bot with legacy switch. Attack packets received from `s1-eth2` interface were forwarded upstream to `nat0-eth` interface. 113
4.42 **Experiment 5: Test1:** CPU utilisation of legacy switch. 114
4.43 **Experiment 5: Test2 Result:** SIP DoS attack to a SIP Server from 1 bot with ShieldVNF. All calls were successful. 60 successful call completion. . 114
4.44 **Experiment 5: Test2:** CPU utilisation of ShieldVNF. 115
4.45 **Experiment 5: Test2 Result:** SIP DoS attack to a SIP Server from 1 bot with ShieldVNF. Most of attack packets were dropped and not forwarded upstream. 115
4.46 **Experiment 6:** SIP Distributed DoS attack from 100 attackers (Test1 with legacy switch vs. Test2 with ShieldVNF) 116

4.47 **Experiment 6: Test1 Result:** SIP DoS attack to a SIP Server from 100 bots with legacy switch. Calls were disrupted when the attack started. 20 successful calls completion. 117
4.48 **Experiment 6: Test1 Result:** SIP DoS attack to a SIP Server from 100 bots with legacy switch. Most attack packets from botnets were forwarded upstream to `nat0-eth` interface. 118
4.49 [**Experiment 6: Test2 Result:** SIP DoS attack to a SIP Server from 100 bots with ShieldVNF. 59 calls were successful. 118
4.50 **Experiment 6: Test1:** CPU utilisation of legacy switch. 119
4.51 **Experiment 6: Test2 Result:** SIP DoS attack to a SIP Server from 100 bots with ShieldVNF. Most of the attack packets from botnets were dropped and not forwarded upstream. 119
4.52 **Experiment 6: Test2:** CPU utilisation of ShieldVNF. 120

5.1 ShieldSDN collects IOC from the source switch (ShieldVNF1) and installs it as packet filtering rules on other switches (ShieldVNF1, ShieldVNF2 & ShieldVNF3). 124
5.2 ShieldSDN components: SDN controller, southbound interface to Shield-VNF, and SQL database for long-term storage of IOC records. 129
5.3 ShieldSDN retrieves the content of `victimIp_register` 130
5.4 **Setup:** The setup script registers 100 IOCs on ShieldVNF1, getting ready for ShieldSDN to distribute, convert, and install it on ShieldVNF1, ShieldVNF2 and ShieldVNF3 as packet filtering rules. 135
5.5 **Setup:** The content of attackerIp_register on ShieldVNF (100 IOC records). 136
5.6 **Experiment 1:** Sequence diagram for experiment 1. Snapshot of result at point "A" is depicted in Figure 5.13. Snapshot of result at point "B" is depicted in Figure 5.14. 137
5.7 **Experiment 1:** CPU utilisation by ShieldSDN. 139
5.8 **Experiment 1:** Memory utilisation by ShieldSDN. 139
5.9 **Experiment 1:** CPU utilisation of ShieldVNF1. 140
5.10 **Experiment 1:** CPU utilisation of ShieldVNF2. 140
5.11 **Experiment 1:** CPU utilisation of ShieldVNF3. 141
5.12 **Experiment 1:** ShieldSDN collected IOCs from ShieldVNF1, converted them, and installed them as Packet Filtering Rules on ShieldVNF1, Shield-VNF2, and ShieldVNF3. ShieldVNF drops the packets that match with these filters. 141

5.13 **Experiment 1:** At the end of experiment 1, ShieldVNF1, ShieldVNF2, and ShieldVNF3 have identical filters installed. 142
5.14 **Experiment 1:** At the end of experiment 1, the IOC(s) are stored in the database. 142
5.15 **Experiment 2:** Sequence diagram for experiment 2. Snapshot of result at point "A" is depicted in Figure 5.20. Snapshot of result at point "B" is depicted in Figure 5.21. 144
5.16 **Experiment 2:** IOCs were assigned random expiry time between 5-10 minutes. As the IOCs expired, they are removed from the VNFs and the record in the database is set with `vnfdeleted=1`. 145
5.17 **Experiment 2:** CPU utilisation by ShieldSDN. 146
5.18 **Experiment 2:** Memory utilisation by ShieldSDN. 146
5.19 **Experiment 2:** ShieldSDN retrieved expired filters from SQL database and removed it from ShieldVNF1, ShieldVNF2, and ShieldVNF3. In SQL database, the deleted field is set to 1. 147
5.20 **Experiment 2:** Expired rules were deleted from the KnownAttacker, KnownCnc, and KnownVictim table. The number of entries is 0. 148
5.21 **Experiment 2:** In SQL tables, there are 100 entries that were marked as deleted (i.e., `vnfdeleted=1`). 148
5.22 **Experiment 3:** Sequence diagram for experiment 3.Snapshot of result at point "A" is depicted in Figure 5.23. Snapshot of result at point "B" is depicted in Figure 5.26. 150
5.23 **Experiment 3: Round 2:** Repeat offender has Frequency column greater than 1 and expiry time is set to 25 minutes. 151
5.24 **Experiment 3:** CPU utilisation by ShieldSDN. 152
5.25 **Experiment 3:** Memory utilisation by ShieldSDN. 152
5.26 **Experiment 3: Round 3:** ShieldSDN encountered repeat offenders (APT), reactivated the packet filtering rule (deleted=0) and sets longer expiry time (49 minutes for 2nd repeat offence) 153
5.27 **Experiment 3: Round 1-3:** ShieldSDN assigns repeat offenders with longer expiry time as the attack frequency increases. 155

6.1 ShieldCHAIN in organisation1 sends IOC to Ethereum Blockchain. ShieldCHAIN in organisation2 and organisation3 retrieves and installs it as packet filtering rules on their switches. 158
6.2 ShieldCHAIN consists of smart contracts deployed at the Ethereum Blockchain and dApp deployed by each participating organisation. 160

6.3 **Setup:** The setup script created IOC entries in ShieldSDN database tables. These IOC entries set the stage for ShieldCHAIN to retrieve them from SQL database and share them with the community via the smart contract on the Ethereum Blockchain. 172
6.4 **Setup:** Packet filtering rules are loaded to the tables on ShieldVNF. 173
6.5 **Setup:** SQL database tables are loaded with IOCs, ready to be distribution by ShieldCHAIN. 173
6.6 **Experiment 1:** ShieldCHAIN queries IOCs from the Database and adds it to Ethereum Blockchain. 175
6.7 **Experiment 1:** CPU utilisation of ShieldCHAIN for the experiment 1 . . . 176
6.8 **Experiment 1:** Memory utilisation of ShieldCHAIN on for the experiment 1 176
6.9 **Experiment 1:** ShieldCHAIN has created 100 IOC records for Attacker and Victim at Blockchain. 177
6.10 **Experiment 1:** Attacker smart contract shows the transactions that were created for IOC records. 178
6.11 **Experiment 1:** CNC smart contract shows the transactions that were created for IOC records. 179
6.12 **Experiment 1:** Target smart contract shows the transactions that were created for IOC records. 180
6.13 Verifying new record that just created by ShieldCHAIN. This record shows the IOC (IP address: Port) that was created by ShieldCHAIN. 181
6.14 **Experiment 2:** Identical IOC records in SQL database in organisation1(USA), organisation2(Singapore), and organisation3(UK). 183
6.15 **Experiment 2:** CPU utilisation of ShieldCHAIN for the experiment 2 . . . 184
6.16 **Experiment 2:** Memory utilisation of ShieldCHAIN for the experiment 2 . 184
6.17 **Experiment 2:** ShieldCHAIN2 and ShieldCHAIN3 retrieves IOCs from Blockchain and inserts it to their ShieldSDN database. 185
6.18 **Experiment 3:** ShieldCHAIN queries ShieldSDN database to retrieve IOCs that originated from Blockchain, converts, and inserts it to ShieldVNF as packet filtering rules. 188
6.19 **Experiment 3:** CPU utilisation of ShieldCHAIN on for the experiment 3 . 189
6.20 **Experiment 3:** Memory utilisation of ShieldCHAIN on for the experiment 3 189
6.21 **Experiment 3:** ShieldCHAIN created identical packet filtering rules on VNF in organisation2 (Singapore) and organisation3 (UK). 191

7.1 SIPshield: A Scalable SIP DDoS Defence Framework With VNF, SDN, and Blockchain 199

List of tables

1.1 Research Question and Sub Questions 10
1.2 Experiments organised by objectives. (IOC = Indicator of Compromise) . . 11
1.3 Publications *(the number of citations is as of 28 October 2020).* 18

2.1 By Timing 30
2.2 By Deployment Locations 30
2.3 By ISO Layer 30
2.4 By Throughput 31
2.5 Previous works on SIP DDoS Flooding defence 32
2.6 Previous works on Mitigating DDoS Flooding Attack from IoT Botnet . . . 39
2.7 Summary of gaps highlighted in the literature that this study seeks to address 48

3.1 By Timing: SIPshield supports prevention (before), detection (during), and mitigation (after) of SIP DDoS flooding attack 59
3.2 By Deployment Locations: SIPshield is deployed in the source network . . 59
3.3 By ISO Layer: SIPshield operates at application-layer (SIP) 59
3.4 By Throughput: SIPshield uses attack detection based on SIP state-machine anomaly. Therefore, high and low-rate attack detection is supported. 59
3.5 SIPshield evaluation by DDoS defence Principles 64

4.1 Experiment-level research questions for preventive and reactive measures . 73
4.2 Independent and Dependent Variables by Attack type 76
4.3 Failed vs. Successful experiments criteria 78
4.4 List of tools used in experiments 78
4.5 Experiments for ShieldVNF following the attacker's journey 93
4.6 **Experiment 1: Result:** ShieldVNF blocks SIP scanning attack (3 discovered vs. 10) 95
4.7 **Experiment 2: Result:** ShieldVNF blocks SIP enumeration attack (3 discovered vs. 10) 99

4.8 **Experiment 3: Result:** ShieldVNF blocks SIP brute force attack (0 cracked vs. 10) . . . 103
4.9 **Experiment 4: Result:** ShieldVNF blocks CNC Establishment attempt (0 registered vs. 10) . . . 107
4.10 **Experiment 5: Result:** ShieldVNF blocks SIP DoS attack (60 successful calls vs. 20) . . . 112
4.11 **Experiment 6: Result:** ShieldVNF blocks SIP Distributed DoS attack (59 successful calls vs. 20) . . . 117

5.1 Independent and Dependent Variables by Experiments . . . 127
5.2 Database schema for KnownAttacker table . . . 131
5.3 Database schema for KnownCnc table . . . 132
5.4 Database schema for KnownVictim table . . . 132
5.5 Experiments for ShieldSDN: scaling the defence from one switch to multiple switches in the organisation. . . . 133
5.6 **Environment Setup:** 100 IOCs were created in preparation for experiment 1, 2, and 3 (B = Before setup; A = After setup) . . . 134
5.7 **Experiment 1:** ShieldSDN collects IOCs from ShieldVNF1, installs these IOCs as packet filtering rules at ShieldVNF1, ShieldVNF2 and ShieldVNF3, and stores these in the SQL database (B=Before experiment; A=After experiment). . . . 138
5.8 **Experiment 1:** Attacker record in SQL database . . . 143
5.9 **Experiment 2:** IOC Expiration (Before and After the 10th Minute). (B = Before; A = After experiment). . . . 147

6.1 Considerations for back-end storage options . . . 162
6.2 Independent and Dependent Variables by Experiments . . . 166
6.3 Smart contract and its address at Kovan network . . . 168
6.4 ShieldCHAIN smart contract functions . . . 169
6.5 End-points for accessing ShieldCHAIN smart contracts . . . 170
6.6 Experiments for ShieldCHAIN: Sharing IOC to multiple external organisations 171
6.7 **Environment Setup:** Preparing the environment for IOC distribution to external organisations. (B=Before; A=After) . . . 173
6.8 **Experiment 1:** ShieldCHAIN queries IOCs from the Database and adds it to Ethereum Blockchain. (B=Before; A=After) . . . 174
6.9 **Experiment 2:** ShieldCHAIN2 and ShieldCHAIN3 retrieves IOCs from Blockchain and inserts it to their ShieldSDN database. (B=Before; A=After) 182

6.10 **Experiment 3:** ShieldCHAIN queries ShieldSDN database to retrieve IOCs that originates from Blockchain, converts, and inserts it to ShieldVNF as packet filtering rules. (B=Before; A=After) 190

Nomenclature

Acronyms / Abbreviations

3GPP The 3rd Generation Partnership Project

ACM Association for Computing Machinery

API Application programming interface

APT Advanced Persistent Threats

AWS Amazon Web Services

bmv2 behavioral model version 2

botnet Robot Network

CAM Content Addressable Memory

CNC Command-and-Control

CPU Central processing unit

CTI Cyber Threat Intelligence

CybOX Cyber Observable Expression

dApp distributed Application

DB Database

DDoS Distributed Denial of Service

DMZ Demilitarized Zone

DNS Domain Name System

DoS Denial of Service

DSC Distributed Services Card

EC2 Elastic Cloud Compute

EDoS Economic Denial of Sustainability

EENA European Emergency Number Association

ENISA European Union Agency for Cybersecurity

ETSI European Telecommunications Standards Institute

FPGA Field-programmable gate array

HTTP Hypertext Transfer Protocol

IDS Intrusion Detection System

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IMS IP-Multimedia Subsystem

IOC Indicator of Compromise

IoT Internet of Things

IPS Intrusion Prevention System

ISO International organisation for Standardization

ITU-T International Telecommunication Union - Telecommunication Standardization Sector

MUD Manufacturer Usage Description

NAC Network Access Control

NAT Network Address Translation

NCCoE National Cybersecurity centre of Excellence

NENA North America Emergency Number Association

NFV Network Function Virtualization

NG112 Next-Generation 112

NG911 Next-Generation 911

NIC Network Interface Card

NIST National Institute of Standards and Technology

ODM Original Device Manufacturer

P4 Programming Protocol-Independent Packet Processors

PBX Private Branch Exchange

RAM Random-access memory

REST REpresentational State Transfer

RFC Request for Comment

RFP Request For Proposal

RTP Real-time Transport Protocol

SDN Software-Defined Networking

SIP Session Initiation Protocol

SMTP Simple Mail Transfer Protocol

SQL Structured Query Language

STIX Structured Threat Information Expression

TAXII Trusted Automated Exchange of Intelligence Information

TDoS Telephony Denial-of-Service

VNF Virtual Network Function

VoIP Voice over IP

VPC Virtual Private Cloud

Chapter 1

Introduction

1.1 Research Context

The public has a justified expectation that they will get connected to an operator immediately when they call 999 in a medical emergency. However, they may hear a busy tone when the telecommunication system is under Denial-of-Service (DoS) attack from IoT botnet.

In the last two decades, telecommunication service providers have been migrating from circuit-switching to packet-switching technology, and implemented SIP as the signaling protocol. This migration allows the service provider to offer audio, video, and data services to the public at a lower cost and higher speed. Despite its benefits, packet-based technology and SIP protocol are still vulnerable to DDoS flooding attacks which prevent legitimate users from making or receiving phone calls. The potential impact of such attacks are ranging from loss of revenue for businesses to endangering public safety, especially when the attackers are targeting hospitals or public emergency numbers.

SIP DDoS flooding attack from IoT botnet is significant and it presents a big challenge to overcome. IoT devices are inherently insecure due to limited security features available on the device. This characteristic makes IoT devices vulnerable and easily exploited by adversaries. Since the same vulnerabilities are present on multiple IoT devices, they all are susceptible to the same exploit. Furthermore, multiple organisations might install the same IoT device on their networks which makes this vulnerability present in multiple organisations. The adversary could use the same exploit to take control of these IoT devices to form a botnet. A botnet might consist of thousands of IoT devices from different parts of the world. There are two main reasons why dealing with the botnet attack is more challenging than a typical attack. First, botnet attacks are more disruptive due to their collective power to send an attack. Second, they use spoofed source IP address that makes it hard to trace the origin of the attack (Zargar et al., 2013). Modern botnets compound the challenge with multi-vector attacks

where the same botnet is capable of launching different attacks at different time. Another important factor to remember is that botnet membership is not static. New members are joining as they were hijacked, and leaving as they were recovered by their rightful owner. With these dynamics, botnet is constantly changing in terms of its source locations, attack vectors, and capability. With the aggressive global deployment of IoT, it is a race against the time to find a better solution than what we have today.

Recognising the risk and impact of attack from IoT botnet, the President of the United States issued an Executive Order (Trump, 2017) to strengthen the cybersecurity of critical infrastructure. The President specifically called out the need for resilience against botnets and other automated, distributed threats. In response to this Executive Order, the Secretary of Commerce and Homeland Security collaborate with stakeholders from industry, academia, and civil society to address this concern. In their report, they set a goal to promote innovation at the edge of the network to prevent, detect, and mitigate automated, distributed attacks (The Secretary of Commerce and Homeland Security, 2018a). This is significant because the stakeholders recognized that the network edge is where they need to focus their efforts to defend against the IoT botnet attack.

1.2 Research Motivations

There are three motivations for this research. The first motive is to empower the community to protect against SIP DDoS flooding attacks from IoT botnet. Currently, only a few organisations have the means to defend against botnet attacks, and yet, it is not a sustainable long-term solution. Cloud service providers like Amazon Web Service (Amazon, 2020a), Microsoft Azure (Microsoft, 2020a), Google Cloud Armor (Google, 2020b) have the means and capacity to absorb attacks from botnet. Even for these providers, they could only absorb these attacks temporarily before they reached an Economic Denial of Sustainability (EDoS) situation (Sotelo Monge et al., 2019). EDoS is a situation where it is no longer sustainable economically for the service owner to absorb the cost for offering the service. This limitation indicates that a new approach is necessary to solve the problem.

The second motive is to explore the source-end defence approach against DDoS attack. The typical defence approach is to build defence mechanisms at or near the victim's end. However, when the attack volume has grown and accumulated near the victim, the damage is already done, and not much that the victim can do besides waiting until the attack stops. For example, in August 2020, the New Zealand Stock Exchange halted trading for three consecutive days due to a DDoS attack (NZ Herald, 2020). This example highlights the deficiency of current defence model and the needs for early mitigation before the attacks get

unmanageable. In the light of this situation, the motivation is to evaluate a defence approach that is near the source where mitigation is done as close to the source as possible, so that the attack does not get the chance to accumulate and reach a destructive level. This approach has been proposed previously (Mirkovic et al., 2003a), but this time, this study will incorporate new technologies like VNF, SDN, and Blockchain that were not available at that time.

The third motive for this research is to participate in this rare opportunity where the research community is working together to innovate and secure the network edge against IoT botnet attacks, as called out by the President (The Secretary of Commerce and Homeland Security, 2018a; Trump, 2017).

1.3 Research Aim and Objectives

The aim of this research is to develop a scalable and collaborative defence framework for SIP DDoS flooding attack from IoT botnet. The perspective is that these IoT devices were directly accessible by the adversaries from the Internet, have been hijacked, and leveraged as a botnet to launch SIP DDoS flooding attacks against SIP servers.

In order to achieve this aim, the research is organised into several objectives. The process starts with securing one switch (which represents the network edge), then extend the success to multiple switches, and finally, to multiple organisations, as illustrated in Figure 1.1. The objectives are as follows :

1. To develop mechanisms for prevention, detection, and mitigation of SIP DDoS flooding attacks on a single switch.

2. To develop mechanisms to scale the defence from one switch (Objective 1), to multiple switches in the same organisation.

3. To develop mechanisms to scale the defence from one organisation (Objective 2), to multiple organisations in a community.

The following topics are **not** the objectives of this research as they cover much wider open research areas and deserve dedicated research efforts on their own:

- Novel detection algorithms for SIP-based attacks
- SIP DDoS attack that is caused by non-flooding methods, e.g., message tampering or flow tampering (Ehlert et al., 2010)
- Mitigating DDoS attack against VNF, SDN, and Blockchain
- Mitigating DDoS attack against IoT devices

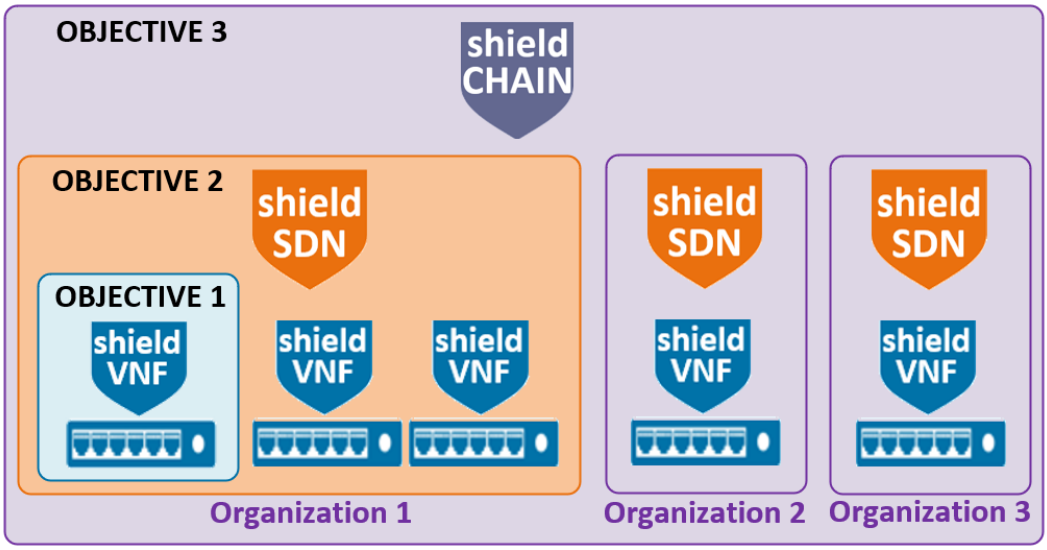


Fig. 1.1 Three research objectives: (1) Protection for a single switch, (2) Scale the protection to multiple switches in the same organisation, (3) Scale the protection to multiple organisations.

1.4 Research Question and Thesis Statement

The research question that guides this investigation is: "In what ways programmable networks could defend against SIP DDoS flooding attacks from IoT botnets?".

The thesis of my dissertation is that the SIP DDoS flooding attacks from IoT botnets can be prevented, detected, and mitigated by securing the edge of IoT networks using a scalable and collaborative framework that uses VNF, SDN, and Blockchain technology.

1.5 Research Challenges and Realities

The following realities explain why research into DoS attacks is difficult and remains an active research area after more than three decades. These factors are complex, multi-faceted, and interconnected. It is complex and multi-faceted because DDoS attacks involve multiple independent parties with different goals, priorities, and financial incentives. But yet, because they are interconnected, the decision made by one organisation will affect other organisations in the community. We will look at these challenges from different perspectives to gain a better understanding from the point of view of the defender's, the attacker's, and the government's.

1.5.1 Defender's Perspective: Lack of Resources and Unprepared

From the perspective of businesses that own the IoT devices, they are lacking resources to properly secure, maintain, and defend against adversaries. Their constraints stem from three main factors: economic, staffing, and technology.

Economic factors

The trend towards digitising business processes is driving the need to deploy IoT devices. For example, businesses are installing various sensors to produce digital data to train artificial intelligence models to analyse and improve their products or services. These organisations perceive digitisation as a key to compete in their industry, and as part of the digitisation process, the businesses are deploying IoT devices in various places from the suppliers' locations to the customers' locations. With businesses that operate globally, they need to purchase IoT devices in large volume for deployment in multiple countries and continents.

The cost of these IoT devices is influencing the buying decision over security concerns. Just like other business processes, a typical procurement process would involve sending a request for proposals (RFP) to the IoT device manufacturers, and the manufacturers would submit a bid based on the customer's request. While cost is not the only consideration in the process, it plays an essential role in getting the project awarded to the winner. With price being an essential factor, the IoT device manufacturer keeps the cost low by building a device with just enough hardware, software, and functionalities to perform the core functions as requested in the RFP. This process produced IoT devices that do not come with robust and strong security postures. With weak security postures, IoT devices become a fertile breeding ground and vulnerable to get recruited by the adversaries as part of their botnet.

The businesses are exercising financial constraint to maintain a good security posture for IoT devices. From a business's perspective, they consider the cost for security as an expense or overhead since it is not part of their main line of business that directly generates revenue for the company. As such, businesses are often reluctant to invest in security measures and lack the motivation beyond what is legally required. While most businesses agree that security is important, oftentimes it is not adequately budgeted for, especially in organisations that are low on the security practice maturity scale.

Staffing factors

Businesses are experiencing a lack of qualified personnel to plan, implement, and maintain a proper cybersecurity infrastructure. Proper risk management process requires staff with knowledge and experience to conduct risk assessment, implement recommendations, and

maintain compliance. Most businesses do not have adequate numbers of staff with this level of security knowledge and aptitude to perform and maintain a good information security program within the company.

The global deployment of IoT devices has significantly increased the attack surface for the staff to manage. With a large number of IoT devices added to a network, that also increases the attack area that the staff needs to secure. Furthermore, besides simply counting the number of devices, the staff also need to consider that each device may have more than one vulnerability. This compounding challenge presents significant additional workload for the existing staff. To keep staff-related expenses low, the businesses tend not to add new staff when they add IoT devices. With the same number of staff, they deployment of new IoT devices presents a vast attack surface for the existing staff to defend.

A tight labour market presents a challenge for businesses to recruit and keep qualified staff. A labour market shortage of qualified cybersecurity candidates is well documented (Kappelman et al., 2018) and makes it challenging for businesses to recruit and keep qualified staff. Supply-and-demand makes it a job-seekers' market and businesses are finding it hard to justify a salary beyond the prevailing market rate. Beyond recruiting, another challenge is to keep the staff motivated to stay within the company and not be recruited by other companies.

Technology factors

Lack of patch management systems for IoT devices necessitates a manual process for firmware updates. With minimum resources powering these IoT devices, they often lack patch management features which are critical to keep these devices updated against the latest exploits. Without this system, the administrators would have to do this task manually which presents logistical challenges. To add to the challenge, producing a security patch is not a sales-generating activity for the device manufacturer and, therefore, tends not to get prioritised. With these challenges, this task is often neglected and, as the result, IoT devices are often hijacked to launch DDoS attacks.

An IoT botnet DDoS attack generates a significant attack volume that is too much for the end-user to absorb. With thousands of devices sending packets against a target, it generates traffic in the Terabits, which is too much for a typical enterprise firewall and their internet connection to absorb. The previous three biggest DDoS attacks were 1.3 Tbps against Github in February 2018 (Kottler, Sam, 2020), 1.7 Tbps against a Service Provider in March 2018 which was mitigated by Netscout (Netscout, 2020), and 2.3 Tbps in Quarter 1 of 2020, which was mitigated by AWS Shield (AWS Shield, 2020) . These incidents demonstrate the level of attacks that DDoS can deliver and how the end-users are not prepared to handle attacks of this magnitude.

VoIP requires a strict response time that limits the options available for attack detection and mitigation. VoIP is a time-sensitive application and does not tolerate loss, delay, and jitter. Unlike other protocols that are transporting data (e.g., web, email, file transfer), the ITU-T standard prescribes a 150 millisecond, one-way delay for voice packets, or they risk voice quality degradation. There are two dynamics at work here: the first is the requirement to perform a deep packet inspection for VoIP packets to detect attacks; the second is the expectation that deep packet inspection will be performed in the shortest possible time so that it is still within the delay specification. Performing deep packet inspection at the application-layer will introduce additional processing delays, but yet, this delay should not reach the point where it will degrade the service. A typical solution would be to install an application-layer IDS/IPS, but it is not cost-effective to deploy this appliance at every location. On the other hand, if this appliance is located at the centralised data centre to minimise the cost, it will incur much delay since it has to travel from the network edge to the data centre, and back to the edge.

Current defence tools are point-solutions and suffer from lack of alignment with the Attacker's journey. The adversaries are following a predictable path or journey in mounting their attacks. For example, first, they will study the target; second, they will plan the attack; and third, they will execute the plan to achieve their objectives. The Cyber Kill Chain framework (Hutchins et al., 2011) describes the steps that the adversaries would typically take to accomplish the mission. Within each step, they produce traces and observables, which could provide some clues as to what their next course of action might be. Unfortunately, the typical defence model is a point-solution or episodic which only provides a snapshot of activity at that particular time. In other words, current defence tools do not stitch the observable events together for longitudinal analysis (Hutchins et al., 2011) and do not provide a complete picture along the adversary's journey. Today, the observables are captured by different tools and stored in different places. Since the data are stored in many different places, it forces the security analyst to stitch these events manually in order to identify the current status of the adversary in their journey. This task is manually intensive and requires a high degree of sophistication on the security analyst side. With high number of alerts received every day, the workload of security analyst is increased significantly and not scalable.

1.5.2 Attacker's Perspective: A Profitable Business Model

There is a market demand for DDoS, and the adversaries are meeting this demand by offering DDoS-as-a-Service. The buyers find DDoS-as-a-Service as a convenient way to inflict damage when they lack the means to do so. For the adversaries, this market demand presents an opportunity for them to monetise their botnet. For example, a disgruntled online-game

player wishing to knock off another online player off-line would hire the botnet for a fee. Another example is when a business hires a botnet to take down a competitor's online business.

The adversary is making a profit from this service as their direct cost is low. For the adversary, the ongoing cost to keep the botnet online and launching the attack is low because the legitimate owner of the IoT devices unknowingly subsidises the adversary. The direct cost to the adversary is mainly in his time and effort on recruiting vulnerable IoT devices and maintaining the command and control (CNC) server.

The rate of successful criminal prosecution is low and presents an acceptable level of business risk for the adversary. Making a criminal case against the adversary is a lengthy and costly process as the victim would need to gather evidence and sustain a long process of legal proceedings. To further complicate the case, adversaries often originated from foreign countries that make inter-country legal proceedings even more complicated. Unless it is a highly publicised case, often, the case is not pursued further. With profitable business, market demand, and low rate of law prosecution, these activities would likely continue.

1.5.3 Government's Perspective: Regulation & Compliance

Government regulations and compliance requirements limit the options available for the defenders. Some industries are heavily regulated to protect consumer rights, e.g., finance, healthcare, telecommunication. The government requires a regular audit and compliance process to ensure that these industries are performing their due diligence. Some countries introduce data residency and data sovereignty laws where the data storage and management are subject to the law of the country. These regulations place restrictions for the defenders who would need to perform deep packet inspection for attack detection purposes.

Privacy concerns about inspection of call records are adding further complexity. People are expressing concern over who has access to their call records as it could potentially infringe on their privacy. There are laws and regulations which limit what the network operator can or can-not do. These restrictions present an additional challenge because to inspect SIP packets, the operator would have access to personally identifiable information which is covered under the privacy law.

1.6 Research Methods

This research begins with a literature review to identify research gaps on this topic, followed by research and design of experiments to test the hypothesis, then with an implementation so

that we can closely observe the outcome when the attacks are launched against the proposed solution. From the experiments, we then collect the data for analysis.

1.6.1 Literature Review and Research Gaps

The literature review is organised around DDoS flooding attacks, SIP DDoS flooding attacks, and the IoT botnets that are leveraged to launch the DDoS attacks. The literature review starts with DDoS because this is where it all started. As the attacks grew more and more sophisticated, it becomes more specific, and targeting a specific application-layer of the OSI, in this case, the SIP protocol. The last area that we will review is IoT botnet since the most destructive DDoS attacks are those that were launched by IoT botnet. The literature review progresses from broad to specific as depicted in Figure 1.2.

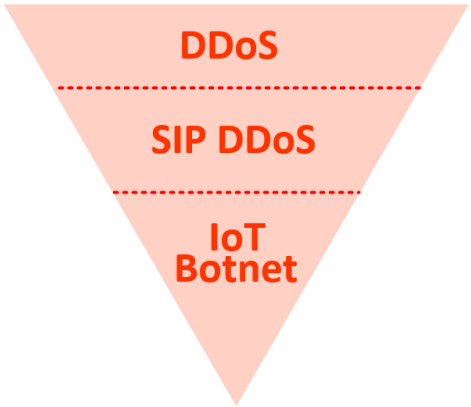


Fig. 1.2 Literature review moving from broad to specifics.

From reviewing the literature, the researchers highlighted several gaps that can be categorised under several themes. The themes that emerged are:

1. The needs for collaborative defence because there is no single entity that can effectively mitigate the attack (Zargar et al., 2013)
2. The needs to inspect and secure SIP software implementations with weak security posture (Keromytis, 2012)
3. SIP defence solutions that are not practical for implementation and protection for narrow use cases (Hussain et al., 2015)
4. The use of non-standard IoT platforms that makes security solutions non-portable among devices (Xiao et al., 2019)
5. Limited access-control and defence mechanisms on IoT devices (Xiao et al., 2019)

1.6.2 Research and Design of Experiments

This research is designed so that the findings can be deduced by analysfing the quantitative data relevant to the research question. In general, a Denial-of-Service attack is caused by the depletion of resources required to provide a service. The variables are quantifiable and measurable by looking at the capacity versus demand. From the user's perspective, the outcome is binary, whether or not the service is available when they requested it. In the context of SIP protocol specifically, the service is considered available when the user is able to make a successful call. A positive outcome is achieved when the users are still able to make successful SIP calls during the DoS attack. Qualitative research methods were not used in this research because the outcome is independent of qualitative measures, and therefore, not considered as a reliable objective predictor of the outcome. A more consistent indicator and repeatable process can be obtained through feasibility experiments (Tedre and Moisseinen, 2014). This is aligned with a view in the computer science community where researchers are encouraged to do more experiments (Denning, 2005; Tichy, 1998).

The main research question is divided into three sub-questions and objectives that contribute towards answering the main question as listed in Table 1.1.

Table 1.1 Research Question and Sub Questions

Research Question: In what ways programmable networks could defend against SIP DDoS flooding attacks from IoT botnets?
Sub Questions
Chapter 4: In what ways a programmable network could provide the first line of defense against IoT botnet attacks?
Chapter 5: How can the success of one switch be replicated to multiple switches in the organisation?
Chapter 6: 3. How can the success of one organisation be replicated to multiple organisations in a community?

For each sub question, experiments were designed to validate the functionality and objective of each question. Table 1.2 list the experiments organised by objectives.

The objective of these experiments is to establish causality (between the independent variable and dependent variables) and to produce data for analysis. The dependent variables are the variables that we will measure to evaluate whether or not the experiment was successful. For example, during a DDoS attack, we will measure two dependent variables: the number of

Table 1.2 Experiments organised by objectives. (IOC = Indicator of Compromise)

Experiments		
Research Objective 1 (Single Switch defence)	**Research Objective 2 (Multiple Switches defence)**	**Research Objective 3 (Multiple organisations defence)**
1. SIP scanning attack	1. IOC collection, storage, and packet filter installation on multiple switches.	1. IOC sharing to Blockchain
2. SIP enumeration attack	2. IOC expiration	2. IOC retrieval from Blockchain
3. SIP brute force attack	3. Persistent threats	3. Packet filter installation on a switch
4. CNC establishment		
5. SIP DoS attack		
6. SIP Distributed DoS attack		

successful SIP calls and the number of attack packets. If the VNF (the independent variable) was working correctly, then we should see a high number of successful calls (dependent variable 1) despite a high number of attack packets (dependent variable 2). These experiments produced data that are useful for analysis and establishing findings.

1.6.3 Implementation

A virtual testbed environment was set up in Amazon Web Service (AWS) to perform these experiments and to test dependent and independent variables (Amazon, 2020b). The environment consists of components that closely resemble what one would find in real life. For example, the environment has a virtual host, virtual switch, virtual firewall, network connectivity between internal switches, access to the Internet, etc.

Amazon Elastic Cloud Compute (EC2) instance hosts a virtual switch and virtual hosts. Each EC2 hosts a virtual network instance (mininet) that supports a virtual switch with multiple virtual hosts. There are two kinds of virtual switch that are involved: Open vSwitch (Linux Foundation, 2020) to represent a legacy switch and bmv2 (p4lang, 2020) to represent a switch that supports programmable data plane. Each virtual host is running on its own network namespace, so it has its interface, an IP address, routing table, etc. We can compare and contrast the result from running the DoS attack with a legacy switch vs. a switch that supports a programmable data plane.

Amazon Virtual Private Cloud (VPC) provides network segmentation and Internet connectivity (Amazon, 2020a). The networking environment for EC2 instances are provided by VPC, which provides components for a private network, e.g., subnets, routes, internet gateways, NAT, etc. In this setup, we have one VPC to represent an organisation. To simulate three autonomous organisations, we set up three VPCs (one in the US region, one in the UK region, and one in Singapore). Each region has its own Internet gateway for the hosts to access the Internet.

VNF, SDN, and Blockchain applications were implemented using programming languages that are suitable for the tasks. The VNF was developed using the P4 programming language (P4 Language Consortium, 2020a) as, at the time of writing, it is the only programming language that is supported on the bmv2 switch. The SDN application was developed using the Node.js programming language (OpenJS Foundation, 2020) as it offers lightweight implementation. Node.js is also able to scale from running on Raspberry Pi (Foundation, 2020) to Function-as-a-Service in AWS Lambda (Amazon, 2020b) or Azure cloud (Microsoft, 2020b). The Blockchain distributed application was also developed using Node.js, and the smart contract was built using the Solidity programming language (ethereum, 2020b).

Well-known and open source tools were used to run SIP call generators and launch SIP attacks. For SIP call generators, SIPp (SIPp, 2014) was used to simulate legitimate users making SIP calls. For attacking tools, tools that are available from Kali Linux distribution were used (Offensive Security, 2020b). SIPVicious (Enable Security, 2020) was used to launch a SIP scanning attack and SIP brute-force attack. For the DoS attack, a tool called inviteflood was used (Offensive Security, 2020a).

1.6.4 Data Collection and Analysis

The tools described in the previous subsection produce data from three different perspectives: the legitimate caller, the attacker, and the switch. These quantitative data were produced during the experiment and collected for analysis after the experiment. From the analysis, we

can establish the findings based on this evidence. In the following sections, we will look at each perspective and the data that were collected.

From the legitimate caller's perspective, the SIPp call generator tool provides statistics about the number of successful calls during the experiment. This is important because the results are taken from the application-layer, which represents user experience. When the tool reports 0 successful calls, it is very likely that the real user will not be able to make a call if they were to try it manually. Other relevant data indicate that SIPp produced are the number of calls made, the number of calls that failed, the timestamps, etc.. Figure 1.3 depicts a sample of a SIPp report that shows the data points generated after each experiment.

```
------------------------------ Statistics Screen ------- [1-9]: Change Screen --
  Start Time              | 2020-06-05   19:23:44.679821 1591410224.679821
  Last Reset Time         | 2020-06-05   19:25:44.301734 1591410344.301734
  Current Time            | 2020-06-05   19:25:44.782807 1591410344.782807
--------------------------+---------------------------+--------------------------
  Counter Name            | Periodic value            | Cumulative value
--------------------------+---------------------------+--------------------------
  Elapsed Time            | 00:00:00:481000           | 00:02:00:102000
  Call Rate               |    0.000 cps              |    0.175 cps
--------------------------+---------------------------+--------------------------
  Incoming call created   |       0                   |       0
  OutGoing call created   |       0                   |      21
  Total Call created      |                           |      21
  Current Call            |       1                   |
--------------------------+---------------------------+--------------------------
  Successful call         |       0                   |      20
  Failed call             |       0                   |       0
--------------------------+---------------------------+--------------------------
  Response Time 1         | 00:00:00:000000           | 00:00:00:008000
  Call Length             | 00:00:00:000000           | 00:00:00:016000
```

Fig. 1.3 Data generated from the legitimate caller's perspective: The number of successful calls during the test.

From the attacker's perspective, SIP attack tools produced data showing the number of attack packets sent. For example, inviteflood that is used to emulate a SIP DoS attack reported how many packets it sent during the experiment, as depicted in Figure 1.4.

From the switch's perspective, it produces important data about the number of packets received and forwarded. This is important because the defence mechanism (shieldVNF) is hosted on the switch. If the defence is successful, then the malicious packets should be dropped right away and not forwarded to the upstream switch. The data points that provide this information are: the number of packets received (from each user) and the number of packets forwarded (to the upstream switch). These data are provided by the Linux system

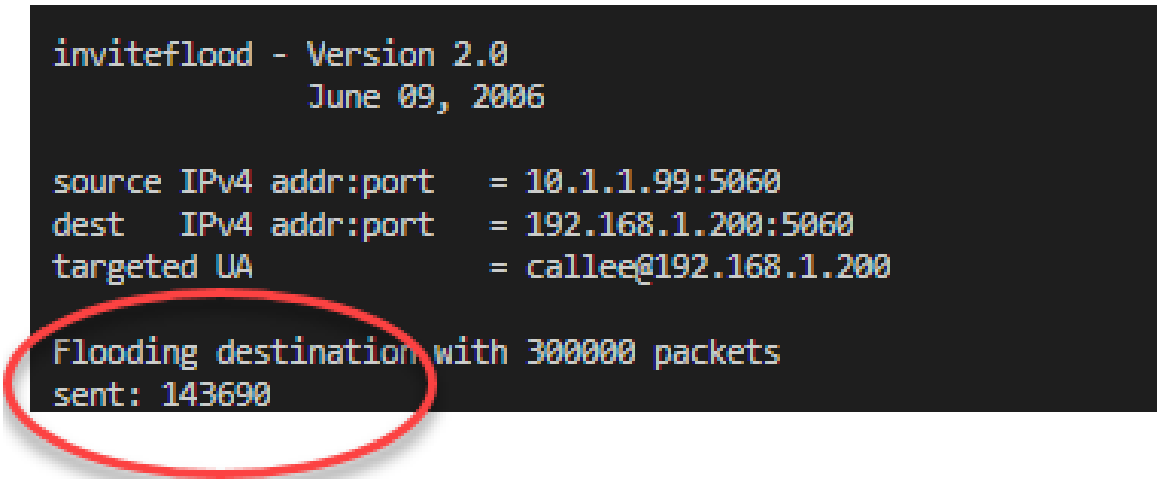


Fig. 1.4 Data generated from the attacker's perspective: The number of attack packets sent.

commands, `netstsat -i`. The number of packets received provides an indicator of whether the sender is an attacker or a legitimate user. The attacker would send a high number of packets compared to the legitimate user. The number of packets forwarded upstream provides an indicator of whether shieldVNF considers these packets as legitimate. Legitimate packets are forwarded to the upstream switch via the NAT port, whereas malicious packets are dropped.

For example, Figure 1.5 shows the switch's perspective. A legitimate caller (1) is connected to port `s1-eth1`, an attacker (2) is connected to port `s1-eth2`, and an upstream switch (3) is connected to port `s1-eth3`. This tool reports that we had `69` packets from the legitimate users, and from the attacker, we had `4,000,118` packets. In this particular scenario, the switch perceived these packets as legitimate and forwarded `4,000,180` packets to the upstream switch. When shieldVNF is enabled, we should see only the legitimate packets being forwarded to the upstream switch. Figure 1.5 shows the network diagram for this perspective.

These data were then gathered and correlated for analysis. From various data points generated by each experiment, data were gathered and correlation was performed to establish the cause-and-effect between variables. For example, we can establish that the defence is successful when we have a high number of successful calls even when the server is under attack (demonstrated by the number of attack packets that were present).

For the second research objective (scaling the defence to multiple switches), the data is the number of IOC entry that is present on a given switch or in the SQL database table. For example, when switch1 produced 100 IOC entries, at the end of a successful experiment, all switches (switch2 and switch3) should have 100 packet filtering rules in their packet filtering

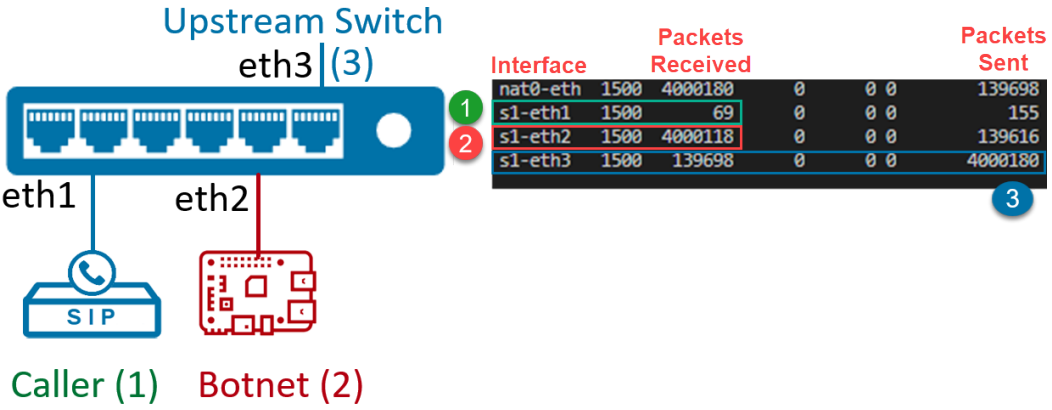


Fig. 1.5 Data generated from the switch's perspective: The number of packets from/to a caller at `(s1-eth1)`, a botnet at `(s1-eth2)`, and an upstream switch at `(s1-eth3)`.

table. These IOC and packet filtering table entries are collected using the command line tool for the bmv2 switch.

For the third research objective (scaling the defence to multiple organisations), the data is also in the form IOC (just like the second research objective), but this time, these IOCs are stored in the Blockchain and retrieved from the Blockchain. For example, when switch1 at organisation1 produced 100 IOC entries, at the end of a successful experiment, there should be 100 IOCs created in the Blockchain and 100 packet filtering rules created on the switches in organisation2 and organisation3. These data are collected and verified using the distributed application and command line tool for the bmv2 switch.

1.7 Contributions

The thesis's significant and original contribution to knowledge is a scalable and collaborative defence framework that secures the edges of IoT networks with VNF, SDN, and Blockchain technology to prevent, detect, and mitigate SIP DDoS flooding attacks from IoT botnet.

This framework is called the SIPshield (Figure 1.6), which is an integrated approach that applies to a few research areas in computer science such as cybersecurity, networking, virtualization, and IP telephony. The SIPshield framework consists of three components: shieldVNF, ShieldSDN, and ShieldCHAIN components that work together and complement each other as a scalable defence framework. This framework provides a response to modern attacks that use thousands of hacked IoT devices as a botnet to launch a DDoS attack. SIPshield is scalable, so that it allows participation from multiple parties to form a distributed defence network. This is aligned with the nature of the attack itself, i.e., distributed attack calls for distributed defence. This is significant because the defence mechanism is typically

outnumbered by the attacker in terms of computing, bandwidth, staffing, and financial resources. SIPshield levels the playing field by allowing the defenders to band together, sharing resources, and workload, to solve common problems of defending SIP DDoS flooding attack from IoT botnet. The description of each component is as follows:

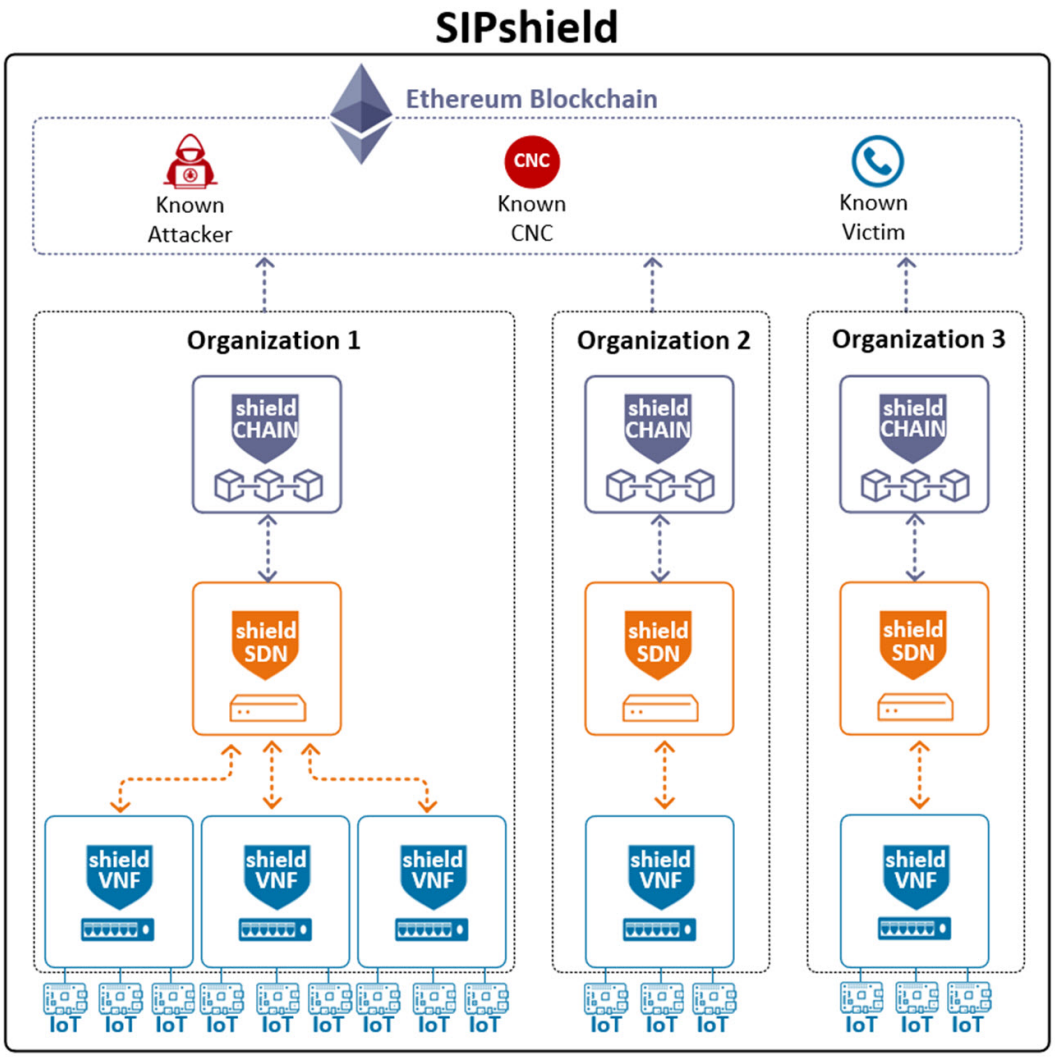


Fig. 1.6 SIPshield and its components (shieldVNF, ShieldSDN, & ShieldCHAIN). Each participating organisation has these components in their network

- **shieldVNF** is a VNF that is running on the programmable data plane of a switch. This contribution allows the switch's data plane to provide virtualised SIP Firewall functions. SIP attack detection is achieved through tracking the SIP state machine where deviation from the normal pattern is perceived as an attack,- and the packet is dropped. Upon successful mitigation, shieldVNF produces three types of threat intelligence or Indicator of Compromise (IOC): KnownAttacker, KnownCNC, and

KnownVictim. When shieldVNF successfully mitigates an attack, information about the attacker is stored in KnownAttacker. Whereas, when shieldVNF detected and blocked connection attempt to CNC, the CNC's IP address, and port number is stored in KnownCNC. When shieldVNF successfully detected and mitigated a SIP DDoS flooding attack, shieldVNF stores the destination's IP address and port of the targeted victim in KnownVictim.

- **ShieldSDN** is an SDN controller that is responsible for collecting and storing threat intelligence that was generated by shieldVNF. ShieldSDN also responsible for installing this intelligence as packet filtering rules on multiple switches within the same organisation so that these switches can prevent the same attack. Considering that shieldVNF has limited capacity to store packet filtering rules, ShieldSDN keeps track and deletes old rules so that it frees up memory space on shieldVNF. With persistent threats that reappear after being deleted, ShieldSDN installs a packet filtering rule with an expiry time that is set to be exponentially longer, based on the attack frequency. With this approach ShieldSDN allows for optimised rules management on shieldVNF.

- **ShieldCHAIN** is a Blockchain distributed application and smart Contract that are runs on the Ethereum blockchain network. ShieldCHAIN allows Ethereum Blockchain to be leveraged as a distributed ledger for storing immutable records of threat intelligence. In this use case, Ethereum Blockchain is leveraged as a Threat Intelligence Platform so that multiple ShieldCHAIN from multiple organisations can be connected to share and retrieve the current state of threat intelligence.

1.8 Publications

During the course of this thesis, some papers have been published and presented. These papers contain the initial findings of the earlier experiment, which is improved with subsequent experiments. These papers contain the initial findings of this thesis, and they are listed in Table 1.3.

- Febro, A., Xiao, H., and Spring, J. (2019). Distributed SIP DDoS defence with P4. In 2019 IEEE Wireless Communications and Networking Conference (WCNC), pages 1–8. IEEE.

- Febro, A., Xiao, H., and Spring, J. (2018). Telephony Denial of Service defence at Data Plane (TDoSD@DP). In NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, pages 1–6. IEEE.

- Febro, A., Xiao, H., and Spring, J. (2019). SIPchain: SIP defence cluster with blockchain. In 2019 Principles, Systems and Applications of IP Telecommunications (IPTComm), pages 1–8. IEEE.
- Febro, A., Xiao, H., Spring, J., and Christianson, B. (2020). Edge computing security: SIP DDoS and dictionary attack defence on the edge switch. Computers and Security Journal. Elsevier. (Submitted).
- Febro, A., Xiao, H., Spring, J., and Christianson, B. (2020). Synchronized & Automated Botnet DDoS Defense with SDN and P4. IEEE Transactions on Network and Service Management. IEEE. (Submitted).

Table 1.3 Publications *(the number of citations is as of 28 October 2020).*

Title	Type	Cited by	Year
Edge Computing Security: SIP DDoS and Dictionary Attack defense on the Edge Switch. Computers and SecurityJournal (Submitted)	Journal article		2020
Synchronized & Automated Botnet DDoS Defense with SDN and P4 IEEE Transactions on Network and Service Management (Submitted)	Journal article		2020
SIPchain: SIP Defense Cluster with Blockchain	Conference paper		2019
Distributed SIP DDoS Defense with P4	Conference paper	2	2019
Telephony Denial of Service defence at data plane (TDoSD@DP)	Conference paper	2	2018

1.9 Dissertation Outline

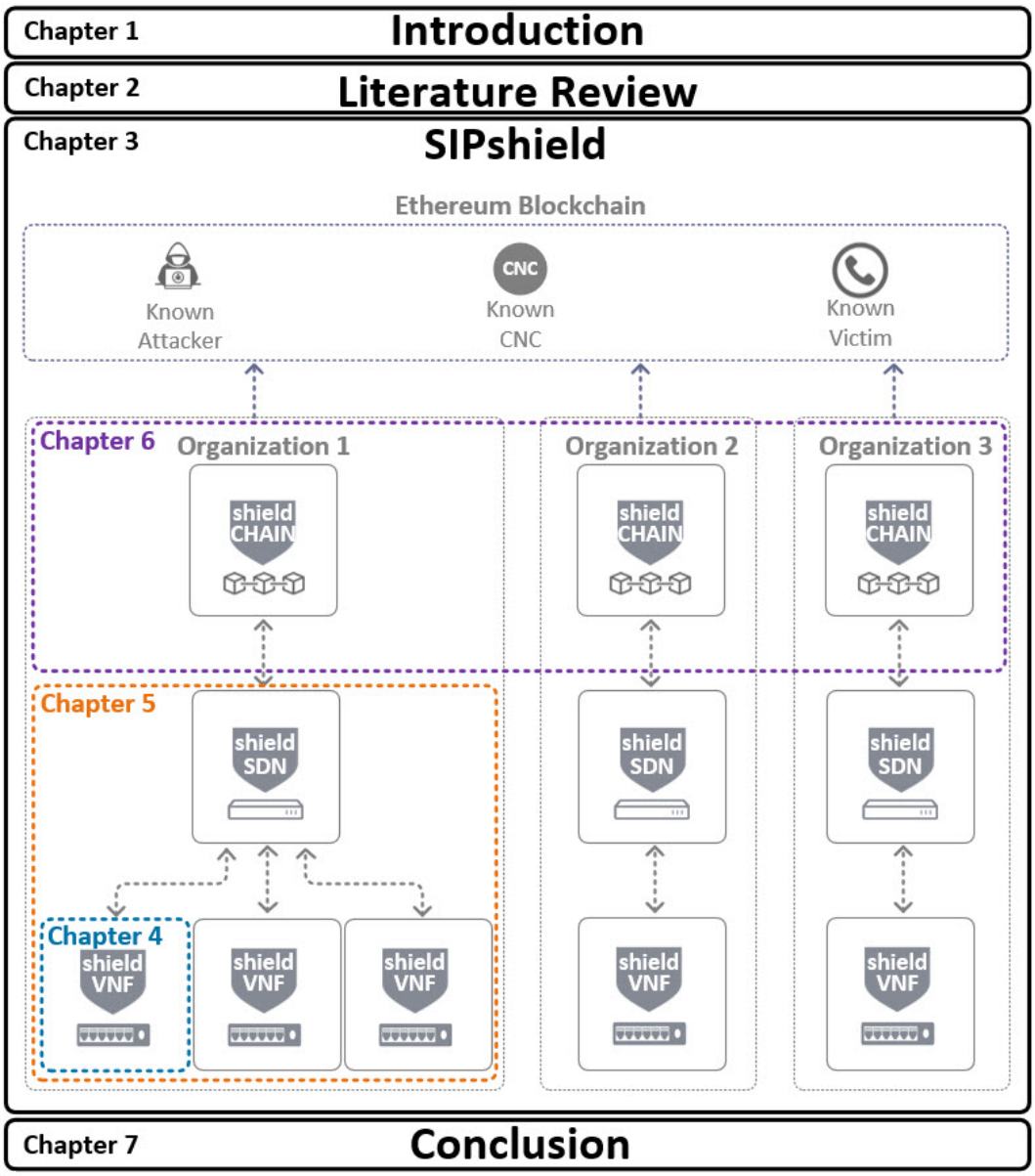


Fig. 1.7 Dissertation Map

The dissertation consists of the following chapters, and Figures 1.7 depicts the relationship between chapters.

- **Chapter 1** sets the stage by introducing the research context, motivation, aim, objectives, research question, thesis statement, methods, and contributions.
- **Chapter 2** reviews the literature to identify what has been done previously and locate this research within the broader context.

- **Chapter 3** presents the theoretical framework and the proposed approach, SIPshield: a scalable and collaborative SIP DDoS defence framework with VNF, SDN, and Blockchain.
- **Chapter 4** covers the shieldVNF component of the SIPshield framework, a VNF running on a programmable data plane that secures the edges of IoT networks and produes threat intelligence.
- **Chapter 5** discusses the ShieldSDN component of the SIPshield framework, an SDN controller, and application that orchestrates the threat collection and distribution among multiple shieldVNF(s).
- **Chapter 6** introduces the ShieldCHAIN component of the SIPshield framework, an Ethereum Blockchain smart contract, and distributed Application (dApp) that shares and retrieves the current state of KnownAttacker, KnownCNC, and KnownVictim among multiple organizations.
- **Chapter 7** concludes this dissertation with discussion and recommendations for future work.

Chapter 2

Literature Review of SIP DDoS Flooding Defence & Potential Solutions

2.1 Introduction

Denial of Service (DoS) attacks have been identified in computer science literature for more than three decades and have grown in their variety, complexity and destructive impact. This chapter begins with reviewing the literature that relates to DoS attacks, followed by the previous proposals for DDoS defence. After reviewing the problem and previous proposals, we will review the literature on new technologies that could be integrated and applied towards a solution. In the previous works sections, the literature review progresses from broad to specifics, i.e., defence against DDoS attacks, defence against DDoS attacks at SIP protocol, and defence against DDoS attacks that are launched by IoT botnets. The new technology section introduces VNF, SDN, Ethereum Blockchain, and Cyber Threat Intelligence (CTI). This chapter ends with a summary of research gaps that this thesis seeks to address.

2.2 Evolution of DoS attacks

DoS attacks have evolved a lot over the years from a theoretical concept to the disruptive force that it is today. What started as simple flooding attacks aimed at the network-layer, have grown to DoS attacks with amplification, by abusing application-layer servers like Connection-less Lightweight Directory Access Protocol (CLDAP) that generated a 2.3 Tbps attack (AWS Shield, 2020), memcached servers that generated a 1.7 Tbps attack that was mitigated by Netscout in 2018 (Netscout, 2020), and a 1.3 Tbps attack against Github in

2018 (Kottler, Sam, 2020). These incidents show that DDoS attacks evolve as the technology progresses and adversaries continue to create new variants.

2.2.1 DoS Attacks

In the literature, the first mention of Denial-of-Service in Computer Networks was by Gllgor (1986) where the author described service-oriented DoS for service-sharing using remote procedure calls. Moving from a more general concept to a specific protocol, Bellovin (1989) published an article specifically on DoS vulnerabilities with the TCP/IP protocol suite, regardless of the correctness of the protocol implementation. This is significant because of the global adoption of the TCP/IP protocol at that time and the application-layer protocols that rely on TCP/IP. This condition also implies that popular services like SMTP, HTTP, and SIP have inherited the same vulnerabilities.

The concept of DoS by flooding was first mentioned by Satyanarayanan (1989) where the attacker floods the network with packets which caused bandwidth depletion, leading to a DoS situation. In February 2000, DoS by flooding was no longer just conceptual because a teenager launched Distributed DoS (DDoS) attacks using botnets, targeting high profile websites like Yahoo, Amazon, and CNN (Lau et al., 2000; Xianjun Geng and Whinston, 2000). In an aftermath report, Yahoo reported that they received attack traffic at 1 Gbps (Harrison, 2020). This event is significant because this was the first DDoS recorded and also the first use of botnet. The trend of using botnet continues and has grown in its intensity. In 2020, Amazon Web Service reported that they mitigated a 2.3 Tbps DDoS attack (AWS Shield, 2020). Besides the intensity, botnet DDoS attacks also have evolved from targeting the network-layers to application-layers where attacks are more visible and have a direct impact on the public.

2.2.2 SIP DDoS attacks

SIP is vulnerable to DDoS attacks and has the potential to disrupt critical telecommunication services for the public. The vulnerability began when the industry made a technology migration from circuit-switched time-division multiplexing technology to packet-switched technology using SIP protocol. The major driver for this migration was to decouple the transport network from services. This decoupling allows multiple services to share the same underlying transport network. Previously, voice services ran on the voice network, whereas data services ran on the data network. The migration to IP network allows voice, video, and data services to share an IP network. This migration had a significant impact on security for two reasons. The first reason has to do with the closed nature of the circuit-switched

network where the circuits stay within the telecommunication provider's boundary and are not exposed to external organisations. With IP, there is no clear boundary and the network is exposed to external organisations. The second reason has to do with the inherent vulnerability of TCP/IP and SIP for DoS attacks.

Another critical service that has adopted SIP for their next generation infrastructure is emergency services. The North America Emergency Number Association (NENA) released Next-Generation 911 (NG911), whereas in Europe, the European Emergency Number Association (EENA) has released Next-Generation 112 (NG112) architecture. These architectures adopted SIP as their signaling protocol. With these critical services relying on SIP, the stakes are now higher since it affects public safety. It has become increasingly important to examine SIP vulnerabilities to ensure that this critical service is highly-available and resilient.

SIP protocol specification RFC3261 by Rosenberg et al. (2002) provides a threat model which includes: registration hijacking, server impersonation, message tampering, tearing down sessions, amplification, and Denial-of-Service as depicted in Figure 2.1.

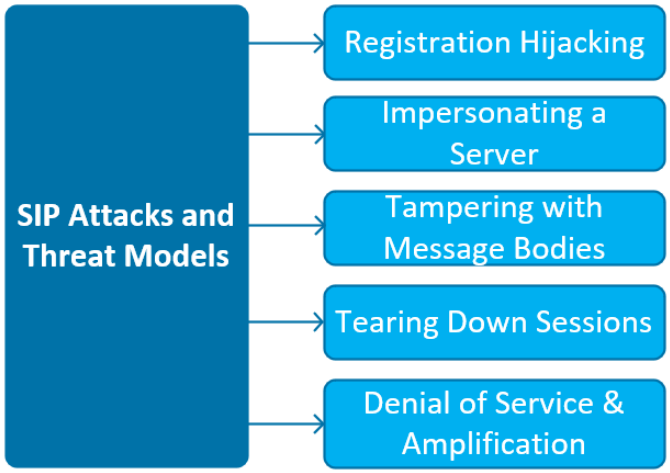


Fig. 2.1 SIP Attacks and Threats Model as per RFC3261 (Rosenberg et al., 2002)

Specific to the denial-of-service attack, Ehlert et al. (2010) defines three types: payload tampering, flow tampering, and message flooding, as depicted in Figure 2.2. With payload tampering, the attacker manipulates SIP payload that causes denial-of-service for legitimate users. With flow tampering, it redirects the session to an unauthorized endpoint. With message flooding, as the name suggests, the attacker floods the victim with packets to exploits the finite resources, e.g., bandwidth, CPU, and RAM of the victim. To gain a better understanding of SIP DDoS attack, it is helpful to analyse the actors and their motives of launching SIP-based DDoS attacks.

An analysis of the actors that are involved in SIP DDoS event, shows that there are four distinct groups of people: the perpetrators, the mercenary, the unknowing accomplice,

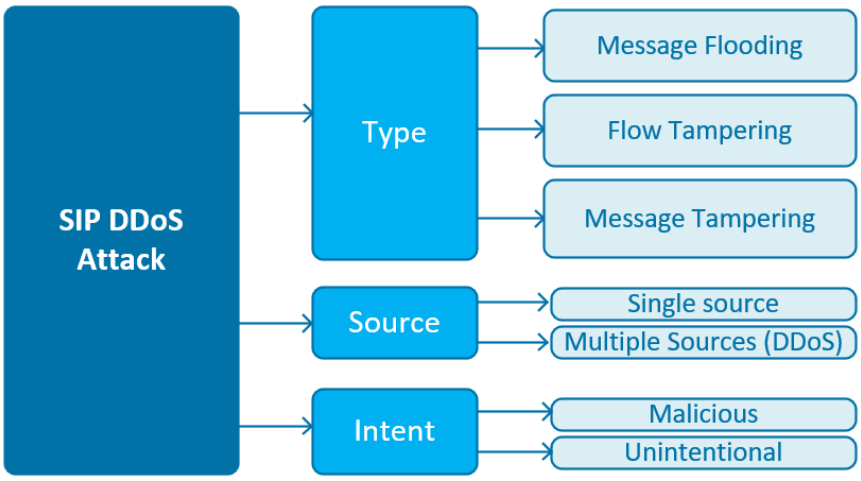


Fig. 2.2 Taxonomy on SIP DDoS Attack (Ehlert et al., 2010)

and the victim. The perpetrators are the individuals with the intent to inflict harm on the victim. The mercenaries are those who offer their service to inflict harm to the victim. The unknowing accomplices are the owners of hijacked devices that carry out attacks against the victim. The victim is the intended target of the SIP DDoS attack. Common motives that caused the perpetrators or mercenaries to launch Telephony Denial-of-Service () attack are as follows:

- For financial profit. Perpetrators who are motivated by profit promote their services for hire. They offer this as DDoS-as-a-Service, which appeals to the buyers who have the intent but without the means to carry out the attack themselves.
- For ransomware operation. In this operation, they threaten businesses or institutions with TDoS attacks unless the victim meets their demand for monetary reward. SIP DDoS attack against a call centre or main business number would directly impact the operation of that business. This kind of attack causes loss of business or inability to serve clients, which affects customer satisfaction ratings. Feeling threatened by the consequences, the business considers paying the perpetrators to be the less expensive option with minimal collateral damage, and therefore they gave in to their demand.
- Diversion tactic. Attackers also used SIP DDoS attacks as diversion tactics. In this instance, the attacker floods the victim's phone, so that the victim's bank cannot contact the victim to verify the money transfer that the hacker has initiated from the victim's bank account to the attacker's.
- Hacktivism. For example, politically motivated activism resorted to TDoS to attack FBI call centres and local law enforcement offices.

- Business competition. In this instance, the perpetrator hires the attacker to overwhelm the competitor's phone system or call centre so that their customers are not able to contact them, and as a result, will contact the business that hires the attacker.

These actors exploit the root cause of SIP DDoS, which is the depletion of server resources that makes it incapable of serving legitimate clients. The defence requirement is, therefore, to identify and block malicious requests before the server runs out of resources. While the root cause is easily understood, it is non-trivial to defend against this attack because the global nature of the problem and the attackers have an advantage over the defenders. The attackers further complicate the matters by using IoT botnet to launch SIP DDoS attack.

2.2.3 SIP DDoS Flooding Attack from IoT Botnet

The arrival of IoT brings to modern society both benefits and challenges. IoT has revolutionised modern society with insights that allow us to make better decisions based on empirical data, evidence, and reasons. It allows society to make better use of scarce resources to preserve them and improve our current living conditions for generations to come. Behind all these benefits that IoT can bring, there lies a hidden danger when the responsibilities to keep these devices secure are ignored. In the hands of adversaries, these useful tools could potentially be used as tools against the society that it is supposed to serve.

It does not take much sophistication on the attackers' side to hire and launch DDoS attacks from IoT botnet. IoT botnet are capable of disrupting any online services that are reachable over the Internet. The Botmaster monetises their botnet by offering it for rent, which makes it very convenient for people with bad intentions to launch DDoS attacks. Another characteristic of a modern bot is that it is extendible and allows the renter to launch attacks with a specifically crafted payload. This is similar to giving a developer a software development kit that allows each developer to build new software out of standard software libraries. This feature makes it a lot easier for malicious users to be very creative in crafting application-layer DDoS attacks. In turn, this will make it a lot harder for the telecommunication providers, businesses, and public emergency services to defend against these attacks.

Incidents of DDoS against telecommunication providers are real and projected to grow in the upcoming years. Telecommunication Providers in European Union countries are required by law to report cybersecurity incidents that they experienced (Art. 13a, of the Directive 2009/140 EC). In its annual report of the telecom security incident 2018 (Koukounas et al., 2019), ENISA (European Union Agency for Cybersecurity) reported an incident of DoS attack against a mobile provider that caused internet outage for a million of users. In its

2019 report (Bafoutsou et al., 2020), ENISA reported a DDoS attack incident against a VoIP service provider that caused 13 hours of outage for 400,000 users. With these real incident reports, the threat of DDoS against telecommunication providers is no longer theoretical. Coupled with the ease of launching an attack as described previously, this type of incident is expected to rise.

Businesses are vulnerable to SIP DDoS Flooding attacks, which could prevent legitimate customers from purchasing or getting support from the business. Despite the prevalence of online shopping, people from certain age groups still prefer to call when making a purchase or getting customer support. Businesses invested millions of dollars in building contact centres and toll-free numbers. However, there are a finite number of staff/agents and phone lines that these call centres have, which can easily be overwhelmed by SIP DDoS flooding attacks from IoT botnets. The adversary could easily use the SIP DDoS attack to extort ransom from the business. They could demand payment from the business, or else they would launch a SIP DDoS attack against the main toll-free number. This attack would prevent legitimate customers from being able to make transactions with the business, and therefore there is an immediate financial impact on the business. The businesses could compare the ransom amount versus the loss of revenue when the call centre is not operational. When the adversary is successful with the ransom, they could make similar threats to the next business.

Even more concerning are attacks against public emergency services like 999 (112 in the EU or 911 in the US). The impact of the DDoS attack is more severe when it is aimed against public emergency services compared with other targets. It is not an exaggeration that when the public cannot access the service, it is a matter of life-or-death. An incident in October 2016 arrested a teenager that caused mobile phones to continually dial 911 and overwhelmed the emergency service (Paley, 2016). A study done in 2017 (Guri et al., 2017) shows that a botnet that consists of less than 6000 mobile phones can launch an anonymised DDoS attack on 911, which blocks emergency services in an entire state for days.

As the attacks have evolved over the years, various solutions have been proposed to defend against DDoS flooding attacks. With a wide variety of DDoS flooding attacks, the proposed defence proposals have been varied as well, depending on where they focus the solution. The proposals differ in areas such as the deployment location, ISO layers, action, and throughput. The next section provides a taxonomy of previous defence proposals to capture and organise these proposals. We will then look at the review sections which move from a generic defence against DDoS flooding, to SIP-specific flooding, and finally, botnet-generated flooding attacks.

2.3 Defence against DDoS Flooding Attack

Researchers worldwide from academia, industry, and government have studied Denial-of-Service (DoS) extensively over the years. However, these threats are still prevalent today due to the variety and complexity of DoS attacks that still prevent us from having an effective mitigation strategy. Multiple survey papers have been produced in the last decade covering the DDoS defence landscape. Reviewing sources like IEEE Xplore, ACM Digital, Scopus, Google Scholar for "DDoS" as the keyword and sorted by the number of citations, it returned the works that have been influential in this topic. The earliest paper with the highest citation count focusing on DDoS attacks and defence taxonomies was written by Mirkovic and Reiher (2004). The most recent papers in 2020 demonstrate how DDoS has evolved and target modern environments like IoT-related (Srivastava et al., 2020) and low-rate attack (Zhijun et al., 2020). These survey papers show that DDoS is still an open research topic since it is still prevalent even today and for the foreseeable future. The victims/targets also have evolved with time as well, from typical computers to modern IoT devices that are part of smart cities, vehicular networks, and medical devices.

2.3.1 Taxonomy on DDoS defence Mechanism

Taxonomies of DDoS defence mechanisms have been useful in providing clarity for categorising and analysing various efforts to combat DoS attacks. Survey papers by (Mirkovic and Reiher, 2004), (Zargar et al., 2013), and (Agrawal and Tapaswi, 2019) uses a similar taxonomy where the authors have collectively used four criteria to categorise DDoS defence Mechanisms as depicted in Figure 2.3. These surveys classify the attacks by action, deployment location, ISO layer, and attack throughput. Classification by deployment location is depicted in Figure 2.4 and by action (or time) is depicted in Figure 2.5.

With classification by deployment location, the survey categorised these defence mechanisms as source-end, destination-end, network-based, and hybrid defence. Each network forms an independent, autonomous system (AS), so in Figure 2.4 the attack is coming from AS#1 and reaching the victim in AS#5. In this instance, the attack has to traverse three independent networks (AS#2, AS#3, and AS#4). As the name implies, the source-end defence deploys the defence mechanism in the source network, and prevents the malicious packets from leaving the source network. On the opposite side, the destination-end defence deploys the defence mechanism in the destination's network, which is near the attack target or victim. The malicious packets have travelled from the source networks to the destination network, and, eventually, get dropped at the destination network. The network-based defence deployed the defence mechanism in the cloud between the source and destination. This

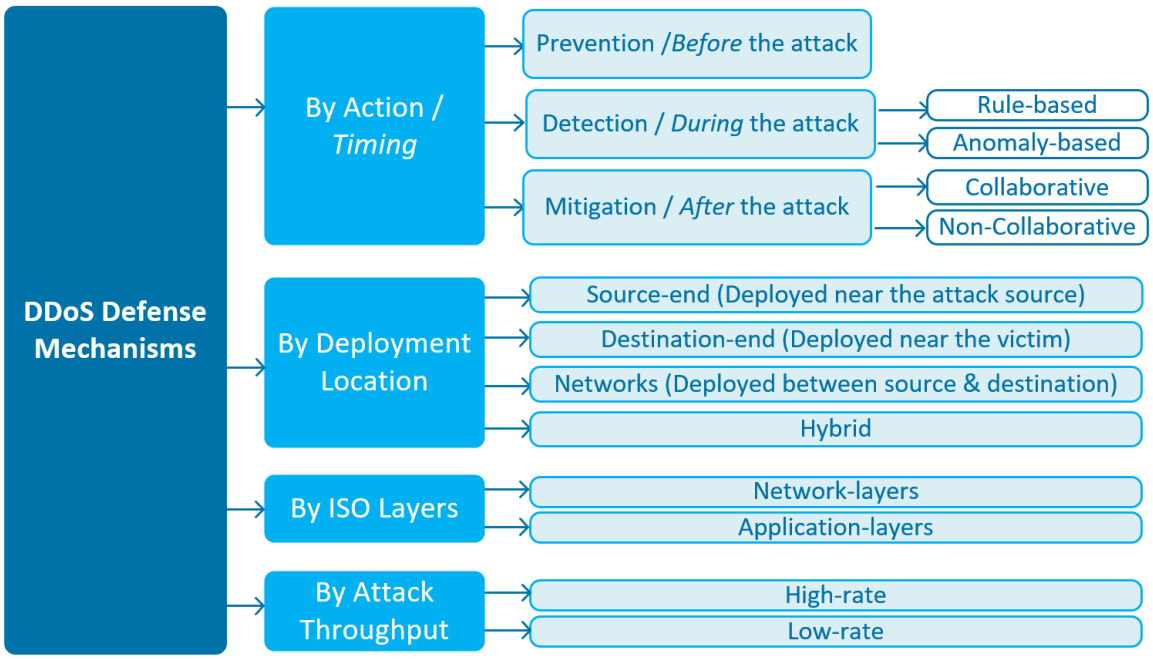


Fig. 2.3 Taxonomy of DDoS defence Mechanisms based on the action, deployment location, layers, and throughput. Adapted from (Agrawal and Tapaswi, 2019; Mirkovic and Reiher, 2004; Zargar et al., 2013).

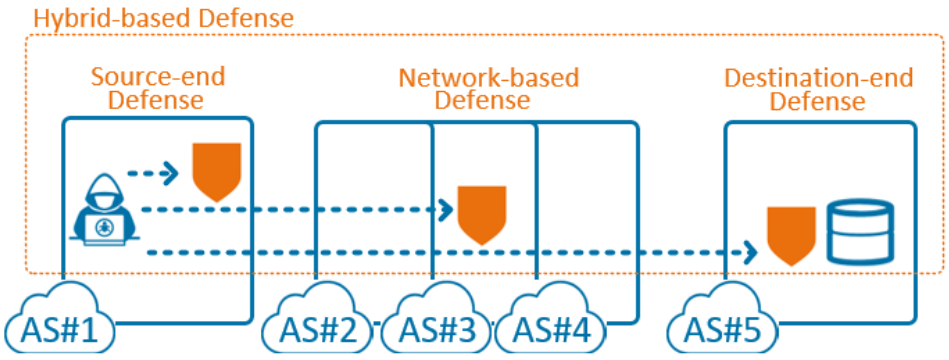


Fig. 2.4 Taxonomy of DDoS defence Mechanisms based on where the solution is deployed: Source, Network, Destination, and Hybrid. The source-end defence solution is deployed in Autonomous System 1 (AS#1). The destination-end defence solution is deployed in AS#5. The network-based defence solution is deployed in the intervening networks AS#2, AS#3, AS#4. The hybrid-based defence solution is a combination of these.

type of defence drops malicious packets in transit. The hybrid model combines two models (e.g., destination-end and network-based). This model uses the destination network to signal the network defence mechanism about the attack. The network then reroutes the malicious packets towards the scrubbing centre, where it will drop the malicious packets, and forward only the legitimate packets to the destination.

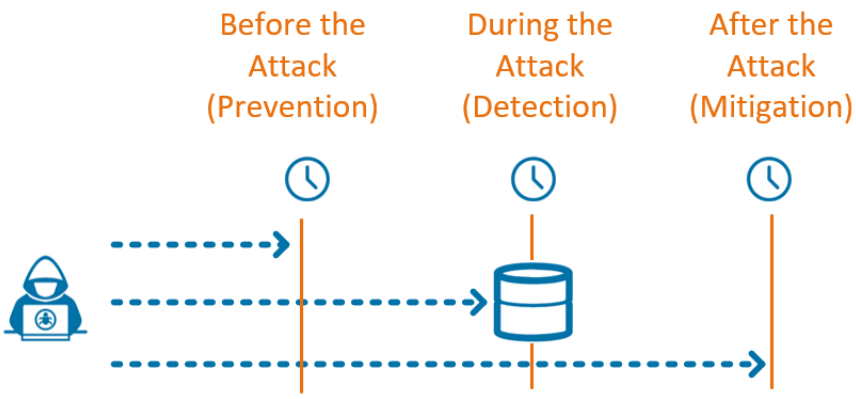


Fig. 2.5 Taxonomy of DDoS defence Mechanisms based on the timing: before the attack (prevention), during the attack (detection), and after the attack (mitigation)

With classification by timing, the survey classifies the mechanisms as prevention (i.e., before the attack), detection (i.e., during the attack), and mitigation (i.e., after the attack). The prevention mechanism is a proactive measure where the attack is stopped before it can reach the potential victim. The detection mechanisms refer to those mechanisms that are activated when the attack takes place and becoming aware of the attacks. The mitigation mechanisms are those that get activated as a response to the attack. In the next section, we will look at previous proposals that fall under these classifications.

The surveys further sub-divide the detection mechanism into signature-based and anomaly-based. The signature-based group detects the attack based on pattern matching or fingerprinting, whereas the anomaly group detects the attack based on deviation from a typical pattern. Multiple methods fall under the anomaly group e.g. filtering-based, entropy-based, change point detection, data mining, feature selection, etc. (Agrawal and Tapaswi, 2019)

The surveys also further sub-divide the mitigation mechanism into collaborative and non-collaborative. With a collaborative approach, the defence elements are working together to detect-then-mitigate the attack, e.g., by using rate-limiting and pushback. The non-collaborative uses techniques like OpenFlow, Netflow, redirection, and reconfiguring services, network, or defence.

With classification by ISO layers, the survey classifies the defence mechanisms as a network or application layer. For the network layer, this label would typically cover several layers, i.e., data link, network, and transport layers. On the other hand, the application layer covers the layers above the transport layer, e.g., HTTP, DNS, SIP, SMTP.

With classification by attack throughput rate, the survey recognises the defence mechanisms as high-rate or low-rate. defence against a high-rate attack is preparation to absorb or deal with floods of packets. The defence against low-rate, on the other hand, focusses not so much on a spike in network traffic but on being more sensitive towards an attack that is

low volume, low throughput, but still achieves the same objective, i.e., depleting the server's resources.

2.3.2 Previous Works on Mitigating DDoS Flooding Attack

In this section, we will look at previous works using the classification described in the previous section. Table 2.1 contains the works grouped by the timing/action of the attack, whereas Table 2.2 grouped the works by the deployment location. Table 2.3 arranged the works by ISO layer and Table 2.4 grouped the works by throughput.

Table 2.1 By Timing

By Timing		
Before	During	After
Geng and Whinston (2000)	Huang and Pullen (2001)	Mirkovic et al. (2003b)
Cabrera et al. (2001)	Peng et al. (2003)	Yaar et al. (2003)
Keromytis et al. (2002)	Kim et al. (2006)	Liu et al. (2008)

Table 2.2 By Deployment Locations

By Deployment Locations			
Source-end	Destination-end	Network	Hybrid
Gil and Poletto (2001)	Park and Lee (2001)	Cabrera et al. (2001)	Mahajan et al. (2002)
Mirkovic et al. (2003a)	Mizrak et al. (2008)	Wang et al. (2007)	Chen and Park (2005)
Abdelsayed et al. (2003)	Gonzalez et al. (2011)	John and Sivakumar (2009)	Chen et al. (2006)

Table 2.3 By ISO Layer

By ISO layer	
Network	Application
Cabrera et al. (2001)	Kambourakis et al. (2007)
Wang et al. (2007)	Ranjan et al. (2008)
John and Sivakumar (2009)	Rahul et al. (2012)

2.3.3 Research Gaps

Despite the works that the research community has proposed so far, there are still open research areas due to the intricate complexity that this problem entails. Some of the areas that the researchers pointed out are as follows:

Table 2.4 By Throughput

By Throughput	
High-rate	Low-rate
Behal and Kumar (2017)	Al-Haidari et al. (2017)
Boro and Bhattacharyya (2017)	Sahoo et al. (2018)
Chen et al. (2018)	Wu et al. (2019)

- There is a need for more nodes to get involved in preventing, detecting, and mitigating the attack (Zargar et al., 2013)
- Effective defence requires collaboration and cooperation among defensive points (Zargar et al., 2013)
- As the threat evolves, there is a need to be able to update the defence mechanism as well (Mirkovic and Reiher, 2004)
- Need mechanism to protect IoT devices from being compromised (Agrawal and Tapaswi, 2019)
- There is a need to standardise the syntax and protocol of threat exchange (Steinberger et al., 2020)
- The solution needs to be scalable, adequate speed, low overhead, accurate and yet still permit legitimate traffic (Praseed and Santhi Thilagam, 2019)

2.4 Defence Against SIP DDoS Flooding Attack

2.4.1 Previous Works on Mitigating SIP-layer DDoS Flooding Attack

Over the years, researchers have produced a number of proposals to address the SIP DDoS flooding attack. Like proposals for network-layer DDoS attacks, the proposals for countering SIP DDoS attacks vary greatly according to the action taken (prevention, detection, mitigation), the intended deployment location, and throughput (high vs. low-rate SIP flooding attack). In this section, we will look at several papers that inspire this thesis or take a similar approach (but different implementation). Some of the works that have gone into this particular area are listed in Table 2.5

Some notable trends have emerged which necessitate a change of mitigation strategy from a software-based solution to hardware to boost the performance. For example, during the early deployment of SIP services, while call volume was not as high as today, the solutions

Table 2.5 Previous works on SIP DDoS Flooding defence

Author	Contribution
Tas et al. (2020)	Mr. SIP, SR-DRDoSSR-DRDoS attack defence
Semerci et al. (2018)	Monitor and discriminator using Mahalanobis distance
Tsiatsikas et al. (2015)	Machine Learning-based SIP DDoS detection
Liu et al. (2014)	Cloud SIP Firweall Cloud SIP FireWall with SDN Controller
Ehlert (2009)	Denial-of-service detection and mitigation for SIP
Huici et al. (2009)	SIP Defender: source destination collaboration
Geneiatakis et al. (2009)	Bloom filter for SIP flooding attack detection
Ormazabal et al. (2008)	Carrier-class hardware-based solution: Secure SIPSecure SIP
Fiedler et al. (2007)	VoIP Defender: Filter, analyser, Decider modules
Chen (2006)	Attack detection with Finite State Machine and threshold

tend to be software-based and deployed near the SIP Proxy servers. As SIP has gained wider deployment and call volume has increased, performance and optmisation have become important considerations. One strategy to boost performance was to port the solution to hardware-based in order to meet the needs of telecommunication providers. For example, Ormazabal et al. (2008) introduced a carrier-class, hardware-based content addressable memory (CAM) solution for SIP DoS defence called "Secure SIP." It proposes the use of two filters (dynamic pinhole filter and SIP-specific filter) to protect the SIP proxy. This work is significant because it was the first proposal for a hardware-based solution that was able to get a high throughput-rate suitable for deployment in the carrier environment. This is relevant because this thesis can also be implemented as a hardware-based solution to gain high-speed performance, e.g., on a programmable NIC (Netronome, 2020b; Pensando, 2020; Yan et al., 2020), NetFPGA (NetFPGA, 2020), or switches (netberg, 2020a) that come with a Tofino (Barefoot Networks, 2020) chipset.

Attack detection based on the state-machine method enforces compliance with SIP protocol specifications while minimising false-positives. In general, the rule-based detection method suffers from static configuration and is not able to detect a new strain of attacks. On the other hand, the anomaly-based detection method suffers from false-positives due to multiple variables that affect the outcome. Researchers found that state-machine based detection is a middle-ground that minimises false positives but yet is able to cover more situations than static rule-based method. The SIP protocol, as part of normal operation, prescribes certain states that the client and server need to follow for proper session establishment or tear-down. The state-machine method tracks these state changes and flags those transactions that do not follow the expected state changes. SIP implementations are required to follow the RFC3261 (Rosenberg et al., 2002) in order to be compatible with other SIP entities. Malicious implementations, however, likely would not follow the prescribed method. In a

way, this method enforces the protocol by allowing legitimate transactions to go through while filtering malicious transactions.

Chen (2006) used a modified SIP protocol finite state machine (FSM) and four thresholds/counters as a detection mechanism for invalid message flooding and distributed reflection DoS (DRDoS). This was the first proposal in the literature that used SIP's internal state machine as a validation mechanism. This thesis also takes a similar approach where it uses SIP's built-in state machine as the detection mechanism because legitimate SIP implementations are supposed to code their software according to SIP protocol specifications. Any deviation from the prescribed behaviour as per RFC3261 becomes suspect. Ehlert (2009) proposes VoIP Defender that uses Filter, analyser, and Decider modules to protect against the SIP DDoS Flooding attack. It uses state machine-based detection (similar to Chen, 2006) and integer counters for measurement. The threshold is set at ten times higher than the normal rate to avoid a high false-positive rate. While this thesis also uses a state machine for detection, this proposal is designed to be deployed at the destination-end, near the victim.

Bloom filter data structure (Bloom, 1970) is appealing for researchers due to its speed and efficient use of storage space. Network appliances typically have to process a high-volume of packets; but, at the same time, have a limited amount of memory for storage. With these specific requirements, the solutions need to strike a balance between speed and storage. The researchers have used the Bloom filter over the years for many different network applications (Broder and Mitzenmacher, 2004) due to its space-efficient and fast performance. The proposal from Geneiatakis et al. (2009) used a bloom filter as counters and introduced a new metric called session distance which was used for threshold. The use of bloom filters in this paper (Geneiatakis et al., 2009) is appropriate since it offers a space-efficient data structure and fast performance. This study also uses bloom filters as the underlying data structure and counters. The difference between the proposal (Geneiatakis et al., 2009) and this study is on the intended deployment location. This work is designed to be deployed at the destination-end, whereas this study is about the source-end.

The theme of collaborative defence has also emerged to address the fact that there is no one component that can mitigate the attack. In this model, different components of the solution work together to mitigate the attack. For example, the sensor that was deployed near the victim would signal to the firewall near the source to start performing rate-limiting, or dropping malicious packets. The researchers implemented this collaborative defence in a few different ways, which fits under the hybrid defence model that was described earlier in the chapter. Huici et al. (2009) introduced "SIP Defender" where they proposed source-end attack mitigation coupled with destination-end attack detection. The author assumes that an IDS exists at the destination, which could send a filter request to the source-end to mitigate

the attack. This work is relevant to this thesis because it incorporates a collaborative workflow where the detection is done at one location, and mitigation is done at the other.

Liu et al. (2014) proposed a Cloud SIP FireWall (Cloud SFW) and SDN controller to protect the IP Multimedia System (IMS) from untrusted peer networks. The proposal uses queuing theory to construct appropriate low-priority and high-priority queues at Cloud SFW. Cloud SFW monitors bandwidth usage and notifies SDN controllers when it hits a certain level. During an attack, SDN controllers will coordinate mitigation efforts with either local Cloud SFW or remote Cloud SFW installed at the untrusted peer network. The author, however, did not elaborate on the SDN controller that was used during the experiment. This work is important because it first introduced an SDN controller as a component for the SIP DoS defence.

Machine learning techniques have become popular lately and were applied to this problem, but they still face some implementation challenges. For example, Tsiatsikas et al. (2015) investigated the use of Machine learning against SIP DDoS attack, for both high and slow-rate attacks. This work is beneficial because it contextualises machine learning in SIP DDoS defence. While not meant for real-time detection due to the overhead involved, the author argues that Machine Learning is beneficial for examining high-volume log files. The challenge, however, would be to apply this method for real-time detection and to minimise the overhead.

In 2020, researchers identified a new attack variant by combining and leveraging vulnerabilities of less-known features of SIP in a real SIP network. A recent paper by (Tas et al., 2020) presents a new variant of SIP DDoS called SR-DRDoS (SIP Request Based - Distributed Reflection Denial-of-Service), along with its defence mechanism. SR-DRDoS uses IP-spoofing, reflection, and DDoS attack logic that quickly depletes SIP Proxy resources. The defence mechanism uses a dynamic threshold value to activate the filtering action. This kind of spoofed and reflection attack vectors would become more prevalent for years to come because of the global deployment of IoT in many different applications.

2.4.2 Research Gaps

Specific to the SIP DDoS Flooding attack, the researchers pointed out the following gaps that require further investigation:

- Comparatively, little research is going to address the problem of denial of service (Keromytis, 2012). The author further adds that since the overwhelming majority of the vulnerabilities were found on the endpoint implementations, the research efforts

should consider the internal structures and implementations of these endpoints instead of taking a black-box view of the endpoints.

- The previous solutions are not practical and costly to implement (Hussain et al., 2015). The author states that most solutions are difficult to deploy, require an extensive change to existing endpoints, and are costly to implement. Further, the author states that existing solutions are considered point-solutions, which solve a specific issue under a specific condition.

- There is a need to monitor internal users in the event that they turn to become attackers (Hussain et al., 2015). The author made an interesting remark about this.

The first gap points out the widespread use of insecure codes in SIP endpoints. The root cause of this could be attributed to a lack of awareness or pressure for time-to-market. With commercial software, they are under pressure to release code to the market as soon as possible. With this pressure, the code is often shipped without going through a thorough inspection of its security, as it could potentially introduce delay and therefore miss the deadline. With open-source software, they are at the other end of the spectrum, where they lack developer resources to perform a rigorous quality assurance process to produce secure code.

On the other hand, with the network edge are now being programmable, affordable, and powerful, it is arguable that the network edge could be leveraged to observe and enforce SIP protocol specification, so that only compliant transactions and dialogues are allowed to enter the network. In other words, the role of the network edge is elevated from merely providing connectivity for the endpoints to a tool to enforce SIP-compliance for the endpoints.

The second gap highlights the fact that these attacks are varied and require different detection methods. On the other hand, proper SIP implementations follow the SIP protocol to ensure that they are interoperable with each other. Since SIP attacks largely occurr due to not following the prescribed protocol, it is arguable that focussing on enforcing SIP compliance is a viable solution. This is an alternative approach to tackling every unique non-compliance situation. With a compliance-based approach, while we do not have all the attack use cases covered, we have established the boundaries or criteria as to how a proper SIP implementation should behave.

The last observation is particularly relevant for SIP endpoints that have been recruited and turned as part of a botnet. In effect, these SIP endpoints are taking on a new personality (as a botnet) while retaining their old personality (as a legitimate IoT device). These devices need to be closely monitored to make sure that they are not hijacked by the adversaries and only perform the intended activities. This threat would continue to grow in light of

the aggressive deployment of IoT devices. In an article, Ang and Seng (2019) provide a taxonomy of application-specific IoT, e.g.,

- IoBT (Battlefied, e.g., weapons, munitions, etc.)
- IoMT (Medical, e.g., wearable ECG, heart rate, etc.)
- IoAT (Animal, e.g., cattle collars, ear tags, etc.)
- IoWasteT (Waste Things, e.g., smart garbage bins)
- IoUWT (Under Water, e.g., smart buoys)
- IoUGT (Under Ground, e.g., soil sensors)
- IoNT (Nano, e.g., sensors and actuators)
- IoMobT (Mobile, e.g., vehicles)

With this magnitude, in theory, they could be misused by adversaries to form a powerful botnet that would make it very difficult for traditional defence mechanisms to keep SIP services available. This is the topic that will be discussed next.

2.5 Defence Against IoT Botnet Flooding Attack

2.5.1 Previous Works on Mitigating DDoS Flooding Attack from IoT Botnet

Governments, industry, and academia are concerned about the threats from IoT botnets. They recognise the threats and potential damage that IoT botnet could incur when they fall into the wrong hands. Understanding that the threats and impacts are across the board and involve the whole Internet ecosystem, isolated efforts from one sector would not be sufficient to adequately address the magnitude of the problem. As a result, there are initiatives underway which involve participants from different sectors to build recommendations, standards, and implementations so that the problem imposed by the IoT botnets could be addressed effectively and adequately. In this effort, the government is acting as a catalyst that brings different sectors to work together towards progress.

Government efforts

Government efforts and initiatives to reduce threats from botnets are a direct order from the US President. In 2017, President Trump issued an executive order that called for increasing resilience against botnet and other automated, distributed threats (Trump, 2017). This order empowered the Secretary of Commerce and the Secretary of Homeland Security to promote collaboration among a wide range of stakeholders through initiating workshops, inquiries, and requests for comments for the purpose of developing a recommendation. In 2018, they published the "Botnet Report" to the President. (The Secretary of Commerce and Homeland Security, 2018a)

In their report, the Secretaries summarised six themes which are important to reduce threats from a botnet. The six themes are: (1) automated, distributed attacks are a global problem, (2) effective tools exist, but are not widely used, (3) products should be secured during all stages of the life cycle, (4) awareness and education are needed, (5) market incentives should be more effectively aligned, and (6) automated, distributed attacks are an ecosystem-wide challenge.

The Secretaries presented five goals that are mutually supportive actions that would dramatically improve the resilience of the ecosystem. The five goals are: (1) identify a clear pathway towards an adaptable, sustainable, and secure technology marketplace, (2) promote innovation in the infrastructure for dynamic adaptation to evolving threats, (3) promote innovation at the edge of the network to prevent, detect, and mitigate automated, distributed attacks, (4) promote and support coalitions between the security, infrastructure, and operational technology communities domestically and around the world, (5) increase awareness and education across the ecosystem.

The Secretaries also produced a road map report toward resilience against botnets (The Secretary of Commerce and Homeland Security, 2018b). In this report, the Department of Commerce and Homeland Security outlines the five lines of efforts to build resilience: (1) IoT, (2), Enterprise, (3), Internet infrastructure, (4) Technology development and transition, (5) Awareness and education. They also identify topics which are important to raise the bar for IoT security. For example, building a robust market for trustworthy IoT devices (home, industrial, and federal level), building standards, and extending risk management for IoT.

The National Institute of Standards and Technology (the science laboratory under the Department of Commerce) publishes 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 and risks of IoT devices (Boeckl et al., 2019). Since consumer home IoT devices are part of the ecosystem, NIST published 8267 in October 2019 to improve security on these devices (Fagan et al., 2019). These documents were then

followed by the publication of NISTIR 8259 in January 2020 to define baseline security capabilities for IoT device manufacturers (Fagan et al., 2020).

The National Cybersecurity centre of Excellence (NCCoE) published an interesting Special Publication (SP) 1800-15 that mitigates network-based attack using Manufacturer Usage Description (MUD). The MUD that is introduced in RFC8520 (Lear et al., 2019b) essentially allows the manufacturer to publish the intended functions of the IoT device (profile), which are then enforced by the router using packet filtering when the IoT device behaves outside of the profile. It is released in four documents: Executive Summary (Dodson et al., 2019a), Architecture (Dodson et al., 2019b), How-to Guide (Lear et al., 2019a), and Test Results (Dodson et al., 2019c). This is a method 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.

Industry efforts

As one of the industry efforts, the Council to Secure the Digital Economy (CSDE) released two documents: Security Baseline (Council to Secure the Digital Economy (CSDE), 2019) and Botnet Security Guide (Council to Secure the Digital Economy (CSDE), 2020). These documents prescribe the capabilities that each IoT device should have, e.g., unique identifiers, secured access, encryption for data-at-rest and data-in-motion, event logging, updates, and reprovisioning. It also covers the topic of product lifecycle management, e.g., vulnerability management, end-of-sales, end-of-support update, etc.

An industry forum called Global Platform published the technical specification IoTopia for IoT security and to ensure interoperability (GlobalPlatform, 2019). IoTopia is built on four pillars which support each other to secure IoT devices and services: (1) secure by design, (2) device intent, (3) autonomous, scalable, secure onboarding, (4) device life cycle management. For the device intent, it uses the MUD protocol (Lear et al., 2019b).

A non-profit organisation called Global Cyber Alliance offers free Automated IoT Defence Ecosystem (AIDE) (Global Cyber Alliance, 2020). AIDE perform the following tasks: collection, analysis, distribution, and display of attacks on IoT devices and striving for automated defence for compatible devices. The attack data are collected through honeypots with 1200 devices and feeds from partners.

Efforts in Academia

Compared to efforts from the government or industry, efforts from academia are generally more forward-looking with a longer-term implementation timeline. Academics work at the

forefront of knowledge and provide research proposals that are valuable for government and industry. With input from academia, the government could build policies that are forward-looking and relevant. The industry would then be able to bring products and services to the market that operate within government guidelines. From this perspective, the proposals presented by academia would not be implemented in public in the short-term.

Common recurring themes from the proposals are around the use of SDN and NFV to secure IoT networks. Bull et al. (2016) proposes an SDN gateway to monitor traffic going to or coming from IoT devices. The gateway could block, forward, or modify QoS as a response to anomalous behaviour. Ozcelik et al. (2017) proposed Edge-Centric Software-Defined IoT defence (ECESID) that provides detection at the network edge using SDN controller and Fog computing concepts. Bhunia and Gurusamy (2017) proposed SoftThings, an SDN-based secure IoT framework to detect anomalous behaviour using a support vector machine (SVM) machine learning classifier. Molina Zarca et al. (2018) proposed a policy-based framework using SDN and NFV concepts as security enablers for IoT devices and the environment. Yan et al. (2018) proposed a Multi-Level DDoS Mitigation Framework (MLDMF) that includes edge/fog/cloud level using SDN-based IIoT gateways (SDNIGW). Yin et al. (2018) proposed a general framework called Software-Defined IoT (SD-IoT) consisting of SD-IoT controllers, switches integrated with IoT gateway, and devices. Al Shorman et al. (2019) proposed a machine learning method that uses an unsupervised evolutionary IoT botnet detection method using the Grey Wolf optmisation algorithm (WGO). Afek et al. (2020) 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.

Table 2.6 Previous works on Mitigating DDoS Flooding Attack from IoT Botnet

Author	Contribution
Afek et al. (2020)	White List Monitoring (WLM) and White List Enforcement(WLE) at ISP-level
Al Shorman et al. (2019)	Grey Wolf optmisation algorithm (unsupervised)
Yin et al. (2018)	Software-Defined IoT (SD-IoT)
Yan et al. (2018)	Multi-Level DDoS Mitigation Framework (MLDMF)
Bhunia and Gurusamy (2017)	SoftThings: SDN-based IoT framework (supervised)
Ozcelik et al. (2017)	Edge-Centric Software-Defined Iot defence (ECESID)
Bull et al. (2016)	SDN gateway for IoT devices

2.5.2 Research Gaps

IoT is still in its infancy, and there are wide-ranging issues that are considered open research areas. The literature identifies the following factors as grand challenges in securing an edge computing environment where IoT devices are located (Xiao et al., 2019):

- Lacking consideration of security-by-design
- Security frameworks that are not easily portable from one device to the next
- Fragmented and coarse-grained access control
- Isolated and passive defence mechanisms

A common recurring theme that the literature highlights are the lack of adequate access controls. While access control is an essential area for improvement, in practice, we did not observe much progress. Schulzrinne (2018) observed a phenomenon that, while networking research has reached "the middle years" stage, the capabilities of today's networking technology are not that different from those in the 1990s. Rexford (2019) suggests not to waste the middle years and to adopt an ambitious pragmatism outlook to seek opportunities for improvement. In the age of SDN, NFV, and VNF, there are opportunities to introduce pragmatic changes to improve access control at the edge of IoT networks.

An IoT botnet attack is a distributed attack that requires distributed defence. IoT botnet exacerbates DDoS attack problems since it uses a fleet of hijacked IoT devices to deliver a powerful attack. IoT botnet turns the power of distributed computing into destructive power. The community could use a similar approach where they band together and do their part to neutralise the destructive power of IoT botnet. With the progress in programmable network technology, the community could work together with their Ethernet switches to mitigate against IoT botnet attacks. This approach spreads the workload for the benefit of every member of the community. In this community-driven and collaborative fashion, the attack could be defused early at the network edge and not get a chance to grow to a destructive level.

With the gaps that have been identified, technologies have made progress in recent years to a level where they present new possibilities to address these gaps. For example, there is technology that enables programmable infrastructure, technology that addresses the issue of trust in the digital transactions, and technology that facilitates closer collaboration for the community. This progress opens up new approaches that have not been explored previously, which we will review in the next section.

2.6 New Technologies For Potential Solutions

Recent research from adjacent areas have produced breakthroughs that could potentially provide solutions to the problem of addressing SIP DDoS flooding attacks from IoT botnets. These adjacent areas are VNF, SDN, Blockchain, and CTI. In the following sections, we will review each area to gain an understanding of the problem the research community in this area is trying to solve, the proposed solutions, the lessons learned, and the implications. In the end, we will look at how we can synthesise these seemingly separate technologies to formulate an approach for solving the research question.

2.6.1 VNF

The global trend is moving from hardware to software-centric solutions in order to increase agility and innovation. In 2012, thirteen global network operators (AT&T, British Telecom, CenturyLink, China Mobile, Colt, Deutsche Telekom, KDDI, NTT, Orange, Telecom Italia, Telefonica, Telstra, Verizon) published a Network Functions Virtualization white paper (Chiosi, 2012). In this paper, network operators highlight the issues of running networks that rely on a plethora of hardware-based network appliances (also known as a middlebox), where each middlebox is providing a specific function, e.g., transcoder, gateway, firewall, cache engine. This hardware-centric solution slows down the process of rolling out new services and innovations. Owning these devices has become a burden for the network operators because it consumes much time and effort to manage these devices during their life cycle. There is an endless cycle of procuring, designing, integration, operating, updating, and eventually decommissioning of these appliances.

Network operators are aiming to use Network Functions Virtualization (NFV) to increase time-to-market agility and minimise cost-of-ownership. The idea is to use standard IT virtualisation technology to consolidate these appliances to industry-standard servers, switches, and storage. These solutions are flexible, and the network operators could install them at either central data centres, remote data centres, or even at the end-user premises. The authors argued that this approach offers benefits such as reduced cost, increased speed of time-to-market, sharing resources to support multi-tenants, scalability, and synergy from openness in the eco-systems.

ETSI published NFV reference architecture that introduces VNF. ETSI's NFV reference architecture consists of three main blocks: Network Function Virtualization Infrastructure (NFVI), VNF, and Management and Orchestration (MANO). In essence, NFVI consists of hardware and software elements that provide a virtual environment for the VNFs, which are managed by MANO. In this architecture, the VNF provides security functions that are

traditionally performed by middleboxes. Middleboxes are defined by RFC3234 (Carpenter and Brim, 2002) as an intermediary box (between source and destination) that perform non-routing function. For example, application-layer security firewalls provide inspections that are specific to a particular application. A web cache is a box that improves the performance of web browsing tasks. Other functions that are typically provided by middleboxes are deep packet inspection and intrusion detection systems. In a VNF environment, these two functions are virtualised and run on a virtual machine or container.

In this study, the idea is to take the concept of VNF a step further by running SIP firewall VNF on an Ethernet switch instead of on a virtual machine. If the SIP firewall can be ported to a general-purpose server, the SIP firewall could also be ported to a programmable Ethernet switch. In this model, the ethernet switch would be the host, while SIP VNF would be the guest. This is one of the hypotheses that we would explore in the experiments that follow.

2.6.2 SDN

One transformative concept of SDN is that it transfers the control from network device manufacturer to network owners. In the mid-2000, routers are using a monolithic firmware that is bloated with protocols/features that are complex and rarely used (Casado et al., 2019). These are expensive, hard to change and innovate. Improvements and innovations are limited and filtered by device manufacturers. A new approach would be to start with a clean slate to improve the way we do networking. Understandably, the device manufacturers perceive this as a threat and have no incentive to support the cause. Casado et al. (2007) published a paper to propose a new approach towards networking that allows the network owners/operators to control or program how the data plane and forwarding plane operates. In essence, this approach empowers network owners/operators, so they are not solely dependent on network device manufacturers for features and functionalities needed in their environment.

The first step is to decouple the control-plane from the data-plane that allows centralised decision-making and programmability. With monolithic architecture, each router has both a control and data plane. The control plane makes routing decisions, while the data plane executes the decision. SDN decouples control-data plane so that the control plane can be done external to the data plane. This decoupling approach opens up an opportunity for a centralised decision-making process where a centralised controller can install routing decisions to remote data planes. Besides centralisation, another benefit is in terms of programmability. Being programmable, the SDN controller could be programmed to react in a certain way when specific criteria occurred. For example, when a particular link is congested, the backup link can be activated by installing a new route that prefers the backup link

The second step is to create a programmable data plane that allows new levels of innovation and control. Besides the programmable control plane, another breakthrough is happening at the data plane. The data plane is responsible for processing the packet, from parsing and deparsing packet headers, modifying packet headers value, queuing, and dequeuing the packets. Previously these features were burnt-in to the hardware or chip that is used by the router or switch. As such, network owners and operators do not have access to modify these. As a result, they have to ask device manufacturers for patch or feature requests for anything that requires a change to the firmware. The breakthrough comes in the form of a programmable data plane, where the end-users are now able to use the P4 programming language to program how to process the packet. For example, the end-user can now specify instructions on how to parse or modify packet headers. This level of programmability of the data plane empowers the network owner/operators to build and run VNF the data plane. In essence, the switch's data plane provides a run time environment for the VNF. In contrast to previous VNF implementation (which is running on a server using hypervisor or container) technology, this study proposed to run VNF over data-plane (VoD). This capability of running VNF on top of the data plane allows the network operator to push the VNF function to the extreme edge of the network.

The programmable data plane that first appeared in 2016 continues to gain popularity and support from industry and academia. The breakthrough of building and running VNF on programmable data plane started in 2016 with the release of a Barefoot Tofino chipset (Barefoot Networks, 2020) and the P4 programming language (Bosshart et al., 2014) to program the chipset. This chipset allows the original device manufacturers (ODM) to build an Ethernet switch that supports programmable data plane and runs up to 6.5 Tbits/s. In the same year, Netronome released Agilio SmartNIC (Netronome, 2020a) that allows a P4 program to run on a programmable data plane on the Ethernet Network Interface Card (NIC). In 2017, an ODM, Netberg, released Aurora switch (netberg, 2020b) that uses a Tofino chipset. In 2018, another ODM, Edge-core, released Wedge switch (edge-core, 2020) that also uses the Tofino chipset. In 2019, Cisco released a new chip called Q100 that supports programmable data plane (Cisco, 2020). In 2020, Pensando released a NIC card called Distributed Services Card (DSC) (P4 Language Consortium, 2020b) that supports P4 and programmable data plane. This level of industry acceptance benefits the end-users with competitive and cost-effective options and promotes wide adoption for programmable data plane. In turn, this programmable data plane makes the effort to secure the network edge feasible.

Researchers continuously innovate with the programmable data plane and use P4 for various network functions. The fact that the data plane is now programmable promotes

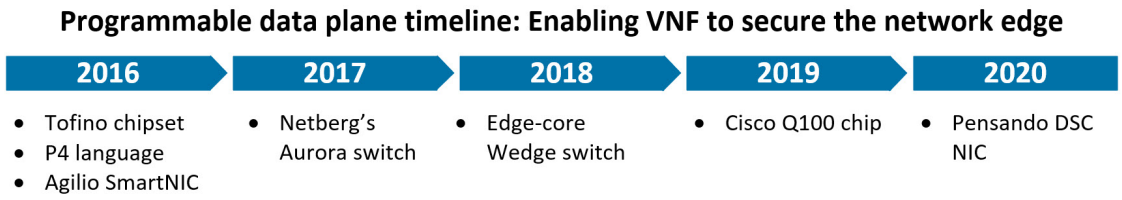


Fig. 2.6 The programmable data plane is gaining momentum in academia and industry. Products are readily available in the market from chipssets, and switches, to NIC cards.

adoption and adaptation beyond networking functions. Researchers use P4 for typical middlebox functions such as Layer 4 load balancing (Miao et al., 2017), congestion control (Handley et al., 2017), network telemetry (Kim et al., 2015). They even use P4 for some creative applications like key-value data storage (Jin et al., 2017), running consensus protocol (Dang et al., 2015), and aggregation for MapReduce (Sapio et al., 2017). These show the amount of interest in the research community and the versatility of use cases.

Combining centralised SDN with distributed and programmable data planes provides a powerful network infrastructure. With a centralised and programmable control-plane using SDN, the network owner/operator can install/remove routing rules for all switches in the organisation. With a programmable data plane, the network owner/operator can write custom processing for each packet. Combining these two powerful features, the SDN can now install/remove custom commands for the data plane to perform custom packet processing. For example, the centralised SDN controller can install a custom command for the data plane to look for a certain byte pattern in the packet payload. As demonstrated by this example, the use cases go beyond simple centralised routing rules. The network is now fully programmable and customisable by the network operators, for network operators. This level of programmability gives network operators a powerful capability that they never had before.

In this study, the idea is to use the programmable control plane to install and remove packet filtering rules on demand. The SDN controller is an application that can connect to the switch's control plane to program the routing table. Similar to the traditional routing table, the switch is using a control plane to determine which route the packet should be forwarded to next. For mitigating attacks, the control plane would have rules that act as packet filtering to drop malicious packets. By using an SDN controller, the network operators would be able to create, edit, and remove packet filtering rules for these switches on demand. For example, when Switch-A found an attacker, the SDN controller can install a packet filtering rule on Switch-B and Switch-C to drop packets from the same attacker. In essence, these proactive rules would serve as an attack prevention mechanism.

For the programmable data plane portion, the idea is to use the programmable data plane to implement a VNF that provides SIP Firewall functions as described in the previous section.

The hypothesis is that if the data plane is programmable, then the network operator could implement SIP Firewall functions that inspect all the SIP packets passing through the data plane. For example, the network operators can specify how to parse SIP packets and inspect the payload of that packet. Then, from the payload, the operator would be able to know whether this is a SIP request or response packet. With these building blocks, the network operator could build a more complex session tracking feature to determine the intent of that packet. These ideas are going to be explored and validated further in the experiments that follow.

2.6.3 Ethereum Blockchain and Smart Contract

Satoshi Nakamoto invented the Blockchain (Nakamoto, 2019) to solve the double-spending problem with digital currency. In contrast to paper money, digital currency is prone to double-spending, where the owner could spend the money twice. Bitcoin solves this problem by using a Blockchain that works according to the consensus protocol between Blockchain nodes in order to create immutable records that are publicly verifiable. The nodes consist of servers that are running Blockchain Virtual Machine. These nodes hold a complete copy of the ledger and are inter-connected in a peer-to-peer fashion. Through the consensus protocol, they can arrive at a consensus where all the nodes agree on the status of a particular transaction and therefore avoid the double-spending problem. Cryptography protects these records and provides a way for users to identify whether the records have been tampered with or altered. In this environment, the community builds trust by being transparent (and without having a central authority figure). While the inventor designed this infrastructure for digital currency, other non-financial use-cases can benefit from the same infrastructure as well.

Ethereum (Wood et al., 2014) enhances Blockchain with smart contracts that provide the programmable logic to process the data. A smart contract provides programming language for Blockchain so that the contract owners can define the logic, data storage, event, and function. Smart contract developers write a smart contract in either Solidity or Vyper programming language. For example, one could create a smart contract for voting applications where each member can only vote once and the total number of votes that each candidate received is then tallied. This use case is an example of using Ethereum for non-financial applications. A smart contract is significant because it provides a computing environment for code execution. In essence, Ethereum not only provides a distributed ledger, but it now also provides a distributed computing platform.

A smart contract deployed on the Ethereum Blockchain can be leveraged to maintain the current state of an IOC. As described earlier, Ethereum provides features like distributed ledger, decentralisation, immutable records, anonymity, and auditability of transactions,

which makes it appropriate for a community to share information. The community can keep track of the current state of the information, preview the previous state, and tell which ethereum account changed the transaction. These qualities are useful for keeping track of IOC because the community can use Ethereum as a distributed ledger for IOCs. In this thesis, the community can retrieve the current state of IOCs so that they can be aware of the threats that are going on currently. The community can verify which ethereum account committed a transaction, whether adding or deleting IOC. At the same time, since the public could only see ethereum addresses, it provides anonymity when the public views these records. However, for the members, they could associate the address with a person or organisation by sharing it by other methods, e.g., personal address book or notes.

In this study, the hypothesis is that multiple organisations can use Ethereum Blockchain as a threat intelligence platform. Each member of the community has the ability to share IOC with the community by writing a new IOC record for each smart contract that is deployed on the Ethereum Blockchain. Correspondingly, the community can retrieve the current state of IOC by reading from the smart contract. The smart contract can contain different kinds of IOCs for different purposes. For example, IOC about the known attackers will contain indicators about attackers that had previously been encountered. The IOC about the known CNC servers will have information about the CNC server that a botnet was trying to connect to. The IOC about the known victims will have the indicator of IP addresses that were targeted during the DDoS attack. From these indicators, the community is able to evaluate whether or not they are:

- involved in an attack (having their IP address listed as the known attacker)
- hosting a botnet command-and-control (CNC) server in their network (having their IP address listed as the known CNC)
- being targeted by the attack (having their IP address listed as the known victim)

2.6.4 CTI

CTI helps the defenders to collaborate to prevent attacks and increased awareness. In the era where attacks are dynamic, multi-vectored, and multi-staged, attack detection that relies on static information is no longer adequate. Time is of the essence to minimise the impact of a security breach. For these reasons, organisations are considering sharing cyber threat information (CTI) within their community to increase situational awareness of such attacks. Gartner Research (2013) defines CTI as any evidence-based knowledge about threats that can inform decisions. The objective is to shorten the time gap between compromise and

detection. Schlette et al. (2020) state that CTI information should be relevant, actionable, and valuable.

It is a balancing act between getting timely information about indicators of compromise, but yet; not getting overwhelmed by it. With attacks that are so dynamic, the information loses its value and accuracy as time goes by. For example, the adversaries change their tactics or their CNC IP address frequently in order to evade detection, so there is a narrow time window when this intelligence is valid. Once it has expired, intelligence is no longer useful for threat prevention. On the one hand, organisations need timely information to be able to react to the attack, but at the same time, it is easy to get overwhelmed by the volume of feeds that one can receive. The key is to get the relevant, actionable, valuable information fast and to filter out the noise.

Standardisation plays an essential role in defence coordination and collaboration. In order to get timely information and to appropriately react to it, there needs to be standardisation so that the collaboration can be orderly and productive. First, the IOC itself needs a standard definition so that all members can understand it. For example, CybOX (Cyber Observable Expression) helps to standardise the object Barnum et al. (2012). STIX (A structured language for CTI) helps to construct the threat intelligence using objects that are defined by Cybox (**?**) . Whereas TAXII (Trusted Automated Exchange of Intelligence Information), helps to communicate threat intelligence over HTTPS so that it is simple but yet scalable (Connolly et al., 2014). With the object, semantic, and transport defined, it allows the community to collaborate against the attack.

The current threat intelligence-sharing process still requires manual efforts. One common drawback of threat-sharing efforts is that the intelligence is not directly actionable due to differences in the environment. An attack indicator needs to be manually reviewed and analysed as to how to implement this in one's environment. An optimum solution for this effort would automate not only the way attack indicators are produced and consumed, but also the way they are implemented so that it is fully automated. This method would reduce the manpower required so that it does not need to involve highly skilled staff. By removing the manpower requirement, this would lower the barrier for organisations that would like to join the community, and as a result, more participation from members in the community.

2.7 Chapter Summary

The literature presents multiple open research areas on multiple fronts. Table 2.7 provides a brief description of these gaps that this study seeks to address.

Table 2.7 Summary of gaps highlighted in the literature that this study seeks to address

Domain	Gap
DDoS	Need for collaboration and cooperation among defensive nodes (Zargar et al., 2013)
DDoS	Need to protect IoT nodes that are directly accessible from the Internet from being compromised (Agrawal and Tapaswi, 2019)
DDoS	The solution needs to be low overhead, accurate, and minimise impact on legitimate packets (Praseed and Santhi Thilagam, 2019)
SIP DDoS	Most vulnerabilities are at the end-point implementations (Keromytis, 2012)
SIP DDoS	Most solutions are difficult to deploy, costly, require extensive change to existing end-points (Hussain et al., 2015)
SIP DDoS	Needs to monitor internal nodes in case it became rogue (Hussain et al., 2015)
IoT botnet	Nonmigratability of security frameworks (Xiao et al., 2019)
IoT botnet	Fragmented and coarse-grained access control (Xiao et al., 2019)
IoT botnet	Isolated and passive defence mechanisms (Xiao et al., 2019)

Adjacent research fields have progressed and produced new technologies that could be integrated together to address the gaps highlighted above. Adjacent research areas like virtualised network functions, programmable networks, and collaborative modes of defence, present powerful paradigms on their own. The idea is to integrate and apply these promising technologies to produce a significant and original contribution to knowledge that addresses the research question. The next chapter will introduce the proposed approach and solution in more detail.

Chapter 3

SIPshield: A Scalable SIP DDoS Defence Against Botnet Attack

3.1 Overview

In the last chapter we reviewed the literature related to the evolution of DoS attacks, previous proposals on DoS defences, and a summary of research gaps that this study seeks to investigate. In this chapter, the objective is to provide a theoretical framework, introduce the proposed approach, evaluate the technologies that enables the proposed solution, and finally to evaluate the proposed solution qualitatively against previous solutions using the DDoS defence taxonomy introduced in the previous chapter (in 2.3.1).

The framework provides principles in designing an ideal solution that inspires the hypothesis to emerge. Implementing the hypothesis is made feasible by the use of emerging technological enablers like VNF, SDN, Blockchain, and CTI. These technological advances already offer powerful features on their own. However, the real power comes when these enablers are integrated, making it a novel approach to defending against IoT botnet attacks. These enablers are the technology that makes the proposed solution possible and takes it beyond a theoretical concept.

The proposed solution section introduces SIPshield, a scalable SIP DDoS defence framework that allows the network operators community to collaborate for preventing, detecting, and mitigating SIP DDoS Flooding attacks from IoT botnets. We will close this chapter by evaluating this proposed solution against the existing solutions based on the taxonomy introduced in the last chapter. This evaluation allows us to trace the intellectual heritage of SIPshield, as well as to locate and position SIPshield among the previous works.

3.2 Framework in Designing DDoS Defence System

In a paper titled "Botnet in DDoS Attacks: Trends and Challenges" (Hoque et al., 2015), the authors include five issues or requirements that a DDoS solution should consider:

1. **DDoS is a distributed problem that needs to be solved by a distributed and coordinated solution**. Robinson et al. (2003) observe three types of networks: source, intermediate, and victim. The authors state that the source network is effective for attack response. In contrast, the victim network is effective for attack detection, and the intermediate network is suited to constrain attacks from legacy source networks. The solution needs to coordinate these activities for effective defence.

2. **The solution should not interfere with legitimate traffic**. It needs to have a graceful way to handle failures so as not to interfere with legitimate traffic. The solution should still allow legitimate activities to operate in the event of the defence system's failure.

3. **The solution needs to have protection against external and internal threats**. External threats are those threats that come from external to the system. On the other hand, internal threats are threats from compromised members that attack the system from within. In this instance, the solution needs to detect rogue members and isolate them from the system.

4. **The solution needs to provide practical strategies for adoption**. As different organisations have their policies, budget, and preferences, the solution needs to provide practical strategies that accommodate various implementation and adoption phases. For example, there is a need for partial deployment, non-contiguous deployment, incremental deployment, scalable deployment, dynamic membership, and substantial economic incentives.

5. **The solution needs to be modular blocks that can be a part of the complete solution over time.** With the complexity surrounding DDoS attacks and multiple players involved, the solution should work with other systems in a constructive and complementary manner.

This framework implies that the solution needs to be loosely coupled, able to operate independently, or coordinate as a group. The hypothesis that emerges is that if the proposed solution can secure a network edge and share IOC with the community, then the community members can take action based on these indicators to defend against the same botnet attack in their network. In later chapters, this study will design and perform experiments to validate this hypothesis.

3.3 Proposed Approach: Securing the Network Edge & Community-based Defence

A botnet-originated attack presents a unique challenge that shows the deficiency of the legacy network security model. DDoS attacks from IoT botnet are becoming more frequent, getting more convenient to launch, and the impact is getting more severe. To defend against SIP DDoS Flooding attacks from IoT botnets, network operators need to reevaluate their traditional network security model, perform application-layer inspection at the network edge to detect attack, and as a community, come up with a collaborative defence model to deal with the common enemies.

3.3.1 Botnet Attack Demands New Defence Model

Legacy network defence models often clearly delineate between internal, external, and demiliterised zone (DMZ) networks. In this classification, the organisation considers the internal network as trusted, the external network as untrusted, and the DMZ network as trusted, but under close monitoring. This model breaks down with the IoT botnet since the adversary is now able to hijack and control the devices in the internal network. The Botmaster (the adversary that owns the botnet) can use the bots to either perform advanced persistent threats on the organisation that has been infected, or make it available for rent to other adversaries as a tool to launch a DDoS attack. With the adversary gaining a foothold in the internal network, the notion of "internal devices can be trusted" is challenged and no longer holds true. Therefore, the legacy network defence model is outdated and no longer effective against modern IoT botnet DDoS attacks.

An IoT botnet is challenging to defend since it is capable of launching multi-vector and dynamic threats to the network. Modern botnets give the botmaster a generic platform for the adversaries to craft, customise and launch their attacks. Depending on the botnet, it comes with a library of ready-made attacks that the botmaster could launch. However, the botnet is also customisable and extensible so that the botmaster can craft specific packets to launch the attacks. With this level of programmability, botnets presents dynamic threats that do not fall neatly within previously known attacks and this makes them harder to detect.

An IoT botnet provides a foothold for the adversary to launch advanced persistent threats (APT). Once a botnet is present in the network, the Botmaster is in no hurry and waiting for the most opportune time to achieve maximum gain when launching attacks. This delay between infection and launching attacks makes the botnet harder to correlate security incidents for accurate analysis. The botnet also enables the adversary to launch

APT where sometimes the objectives are beyond immediate financial gain. For example, a state-sponsored group is using botnets for political gain, a competitor is using a botnet to steal strategic intellectual property (corporate espionage) or to disrupt critical services.

These challenges are not slowing down as IoT deployments are on the rise. The mass and global deployment of IoT devices are just going to increase in the near future and beyond. This trend drives an unintended effect of providing a fertile breeding ground for the botnet. These challenges present the need to critically reevaluate the current defence model so that it is effective against new challenges that the IoT devices bring.

3.3.2 New Defence Model Involves Smart & Secured Network Edge

The community needs to be empowered to inspect SIP packets and reject malicious packets at the edge of the network. A common approach is to allow internal endpoints to send packets indiscriminately just because the endpoint is owned by the organisation, connected to the internal network, and therefore considered as a trusted source. With the case of internal IoT devices being hijacked by adversary, the network operators need to seriously reconsider this assumption. With application-layer DoS being on the rise, the network operators need to perform deep packet inspection for application-layers at the network edge so that the edge can differentiate between malicious versus legitimate sessions. This inspection capability would validate SIP packets and permit only those that do not exhibit malicious patterns. Network operators can recognise malicious patterns when they deviate from a typical pattern for legitimate activities. For example, to initiate a SIP call, RFC3261 states that the caller sends a SIP INVITE to the callee, and wait for a response. When the caller sends hundreds of INVITE packets to the callee, it raised suspicion on the intent of the caller.

SIP inspection at the edge enables the network operators to set up attack prevention, detection, and mitigation use cases. Deep packet inspection at SIP level allows for attack prevention where the network edge drops malicious packets even before reaching the victim, and thus, preventing the attack. Deep packet inspection can also detect an attack when it observes an anomaly or deviation from normal behaviour. Finally, attack mitigation naturally follows the detection where the network edge drops the malicious packets after suspicious activity has been detected. With prevention, detection, and mitigation use cases covered, the network edge is equipped to deal with DDoS attacks at the application layer.

Inspecting SIP packets at the network edge does not cause service interruption. Performing deep packet inspection takes an additional step which presents a valid concern where it could exceed the maximum time allowed for a real-time service like Voice over IP (VoIP). The ITU-T specification stipulates that end-to-end one-way delays should not exceed 150 milliseconds. To address this concern, it matters where the inspection is performed. Since the

inspection occurs at the network edge while the volume is low and the workload is distributed among multiple switches, packet inspection at this stage would not overwhelm the switch and would only introduce a minimal delay. As a result, it would not exceed the specification or interrupt the service.

3.3.3 New Defence Model Involves a Community

The networks are interconnected, and everyone that is connected to the network is vulnerable to IoT botnet attacks. The community has gained tremendous values by being interconnected and it has become essential in modern society to be interconnected. These values, however, comes with the risk. When a member is vulnerable, the rest of the community is also vulnerable just by being connected to that vulnerable member. The botnet itself is a network of thousands of devices from different parts of the world and industries. In this environment where the attackers and targets could be anybody, a botnet is a common threat for the community.

Community collaboration is essential to defeating the common enemy. The adversaries are taking advantage because the victims are working in a silo and not communicate with each other. In a silo model, everyone has a limited perspective and does not have situational awareness. There are two-sides of awareness: being aware that they are vulnerable to the attack that is going on and, in the case of IoT device owners being hijacked, being aware that they are part of the problem. Should the community collaborate by sharing their limited perspective, the community can gain broader awareness and stitch together a complete picture that highlights the activities of the adversary. This greater situational awareness can then empower the community as a whole, to be more proactive against the adversary.

A low barrier-of-entry is essential to maximise participation in the community. The tools selected for the collaboration need to be accessible and affordable by the members so that it does not deter potential members from joining the community. The tools should automate the tasks so that there is no need to rely on manual intervention for day-to-day operations. It is essential to minimise manual tasks considering that most members are short of personnel and resources. The tool also needs to be a low footprint so that it does not impose a significant burden for the member to implement.

A common framework helps the community to collaborate effectively. Working together as a community would require a common framework that governs how the members should collaborate. The framework would prescribe the scope, rights, and responsibilities of each member and provide guidelines as to how each member can participate and contribute. Together, the tool and framework are required for the community to collaborate effectively.

3.4 Proposed Solution: SIPshield, a Scalable and Collaborative Defence Framework

SIPshield is a scalable and collaborative defence framework against a SIP DDoS flooding attack from an IoT botnet that adopts and implements the proposed approach. It integrates components from a few research areas in computer science such as cybersecurity, networking, virtualisation, and IP telephony. The SIPshield framework consists of three components: ShieldVNF, ShieldSDN, and ShieldCHAIN components that are designed to work together as a scalable defence framework. Each component has specific functions and each organisation that wants to participate in this framework will need to have all three components as depicted in Figure 3.1.

3.4.1 ShieldVNF

ShieldVNF is a VNF that is running on the programmable data plane of an Ethernet switch that enables the switch to provide virtualised SIP Firewall functions. When IoT devices are connected to a switch, this switch represents the edge of the IoT network. With the next generation of switch that offers a programmable control and data plane, this smart network edge can be programmed to run VNF that secures the network by filtering malicious packets and preventing them from even entering the network. With this capability, SIPshield secures the edges of IoT networks from SIP DDoS flooding attacks from IoT botnets. SIP attack detection is achieved through tracking the SIP state machine where deviation from the normal pattern is perceived as an attack, and the packet is dropped. Upon successful attack detection and mitigation, ShieldVNF produces an IOC. There are three types of IOC that ShieldVNF is capable of producing: KnownAttacker, KnownCNC, and KnownVictim. These IOCs are then leveraged and managed by ShieldSDN.

3.4.2 ShieldSDN

ShieldSDN is an SDN controller that is responsible for extracting IOC that was generated by ShieldVNF earlier, and installing it as packet filtering rules on other switches. As the result, all switches in the organisation keep identical packet filtering rules and are able to provide the same level of protection against the same attacks. Considering that ShieldVNF has limited capacity to store packet filtering rules, ShieldSDN keeps track and deletes old rules so that it frees up memory space on ShieldVNF. When dealing with persistent threats that reappear after the rule being deleted, ShieldSDN re-installs the rule with an expiry time that is exponentially longer. In essence, ShieldSDN replicates the success of one switch to

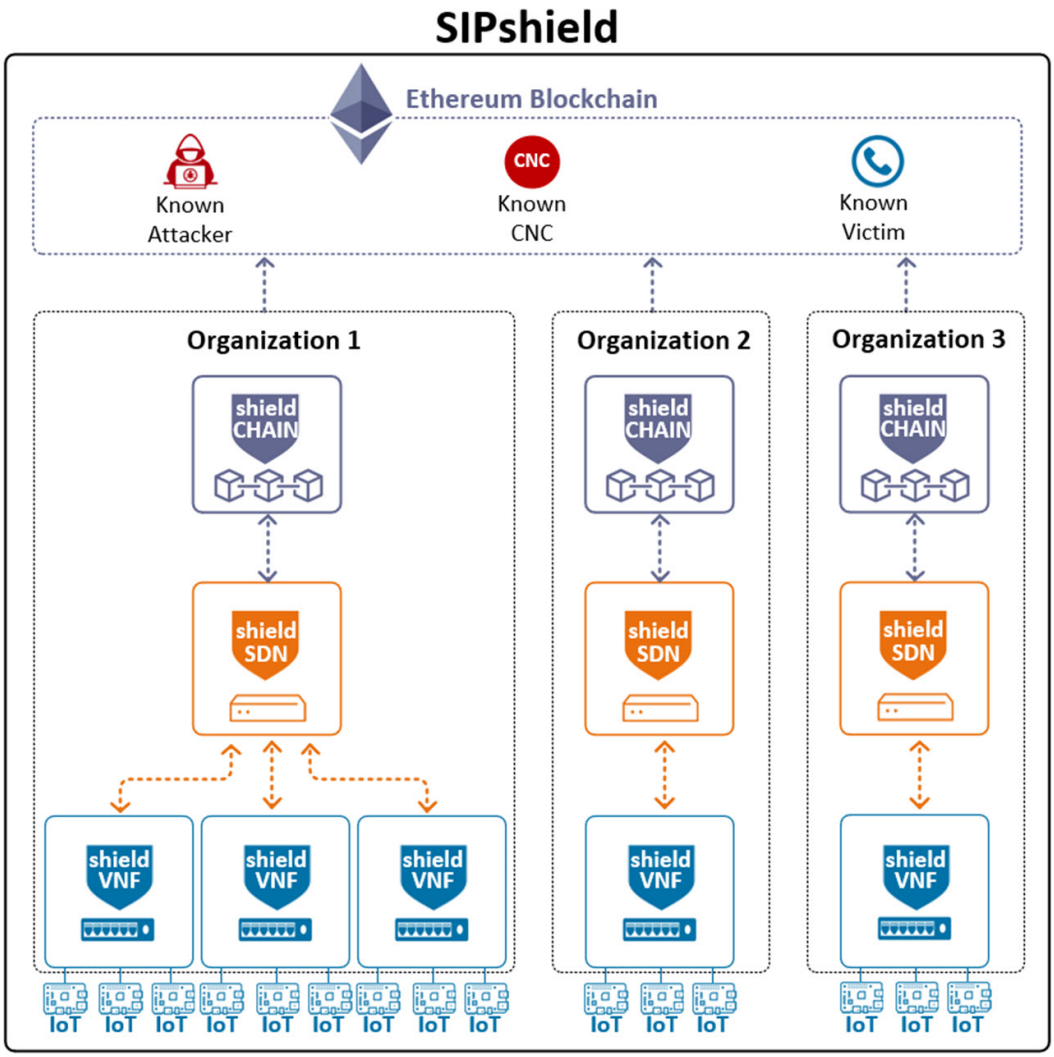


Fig. 3.1 SIPshield and its components (ShieldVNF, ShieldSDN, & ShieldCHAIN). Each participating organisation has these components in their network

other switches to achieve a synchronised defence posture in an organisation. As the last component in the framework, ShieldCHAIN extends this success one step further.

3.4.3 ShieldCHAIN

ShieldCHAIN is a distributed application (dApp) and has smart contracts that are running on the Ethereum blockchain network. A collaborative defence framework model allows community member to share and receive IOC(s) so that as a community, they can be prepared to mitigate against DDoS attack from IoT botnets. ShieldCHAIN enables this framework by allowing the source-organisation to share IOC(s) with the community, as well as, allowing the receiving organisation to retrieve IOC(s) from the community. From these

IOCs, ShieldCHAIN then generates packet filtering rules and installs it on the switches in the receiving organisations. In essence, ShieldCHAIN replicates the success of one organisation to other organisations to achieve a synchronised defence posture in a community.

3.5 Evaluating SIPshield

SIPshield presents a novel, scalable, and collaborative defence framework that is appropriate for modern IoT botnet-generated attacks. SIPshield framework draws inspiration from the literature and in this section we will analyse SIPshield from a few perspectives: by intellectual roots, by research gaps that it is addressing, by locating it in the existing DDoS defence taxonomy, and by how it maps to the principles of DDoS defence.

3.5.1 By Intellectual Roots

The intellectual roots of SIPshield come from multiple research fields. SIPshield and its components (ShieldVNF, ShieldSDN, and ShieldCHAIN) inherit features and concepts from areas such as networking, virtualisation, distributed ledger, and threat intelligence. The intellectual genealogy of SIPshield is depicted in Figure 3.5.2 and the list below shows how existing literature relate to each component of SIPshield:

- ShieldVNF is influenced by NFV, VNF, Bloom filter data structure, and source-end DDoS defence approach. The use of these technologies allows ShieldVNF to secure a single switch against a SIP DDoS Flooding attack from an IoT botnet.
- ShieldSDN is inspired by networking research, specifically the concept of a programmable control and data plane. These powerful features allow ShieldSDN to scale the solution beyond one switch and synchronise packet filtering rules on all switches in the same organisation.
- ShieldCHAIN leverages the distributed ledger research and the concept of threat intelligence sharing to synchronise the defence posture among multiple independent organisations.

3.5.2 By Addressing Research Gaps

SIPshield is addressing several gaps in DDoS, SIP DDoS, and IoT botnet research. Table 2.7 lists the gaps that SIPshield is seeking to address.

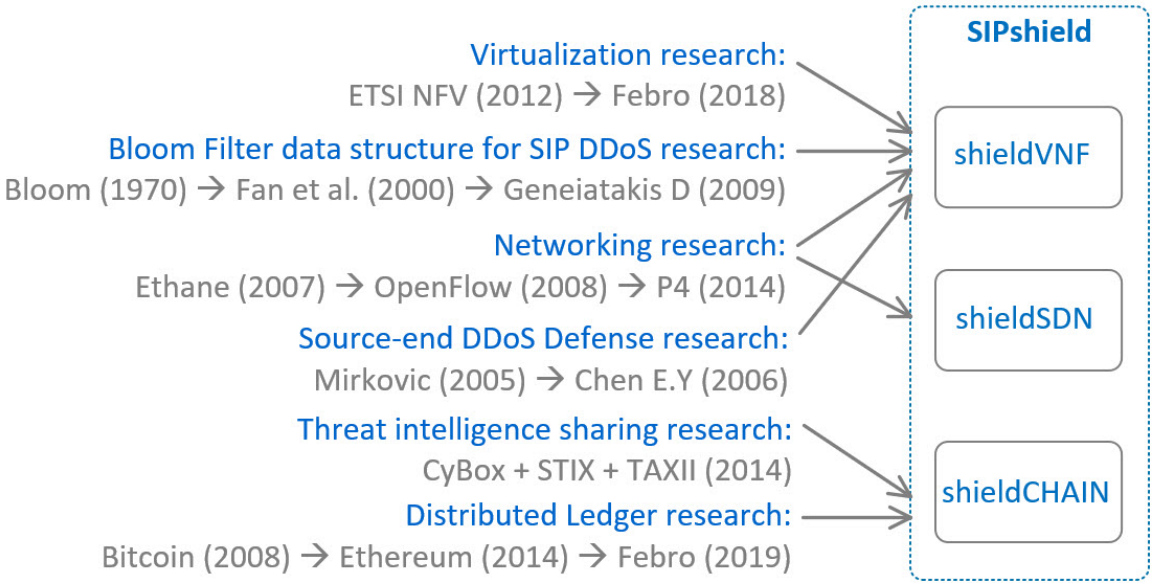


Fig. 3.2 Intellectual genealogy of SIPshield.

- **DDoS.** SIPshield is addressing the needs for collaboration and cooperation among defensive nodes (Zargar et al., 2013) by providing a framework and tools (ShieldVNF, ShieldSDN, and ShieldCHAIN) so that the community can collaborate to detect, mitigate, and prevent against attacks from IoT botnets.
- **DDoS.** SIPshield is addressing the need to be low overhead and, accurate, and to minimise impact on legitimate packets (Praseed and Santhi Thilagam, 2019) by distributing the workload across multiple edge switches so that the overhead is kept low on each device. Accuracy is achieved by each switch performing deep packet inspection for the tracking session state as prescribed by the RFC so that it is accurate, and at the same time, does not impact legitimate and RFC-compliant sessions.
- **SIP DDoS.** SIPshield is addressing the end-point vulnerabilities (Keromytis, 2012) by hardening the network edge and using ShieldVNF to provide protection for the SIP end-points. ShieldVNF is able to perform inspection at the SIP layer and keep track of SIP requests and responses to differentiate malicious vs. legitimate SIP sessions.
- **SIP DDoS.** SIPshield is addressing the concerns over practical implementation (Hussain et al., 2015) by using non-proprietary technology and commodity hardware so that it is accessible and affordable by most organisations. In addition, SIPshield does not require modification or changes on existing SIP software implementations.
- **SIP DDoS.** SIPshield addresses the need to monitor internal nodes (Hussain et al., 2015) by enabling detection, mitigation, and prevention features on all ports on the

switch so that the internal nodes, like IoT devices, are also monitored for malicious activities.

- **IoT botnet.** SIPshield is addressing the needs to protect IoT nodes from being compromised (Agrawal and Tapaswi, 2019), by using non-portable security frameworks, coarse-grained access control, and isolated defence mechanisms (Xiao et al., 2019) and by securing the edges of IoT networks using ShieldVNF in order to detect, mitigate, and prevent scanning attacks, enumeration attacks, and brute force attacks.

3.5.3 By DDoS Defence Taxonomy and Existing Solutions

For analysis and evaluation of SIPshield, we would use the DDoS defence Taxonomy introduced in the previous chapter to compare and contrast SIPshield against existing works. Evaluating with a taxonomy is helpful to locate where SIPshield stands in the defensive spectrum and to understand its capabilities. In order to evaluate SIPshield's capability, Figure 3.3 highlights the areas that SIPshield provides.

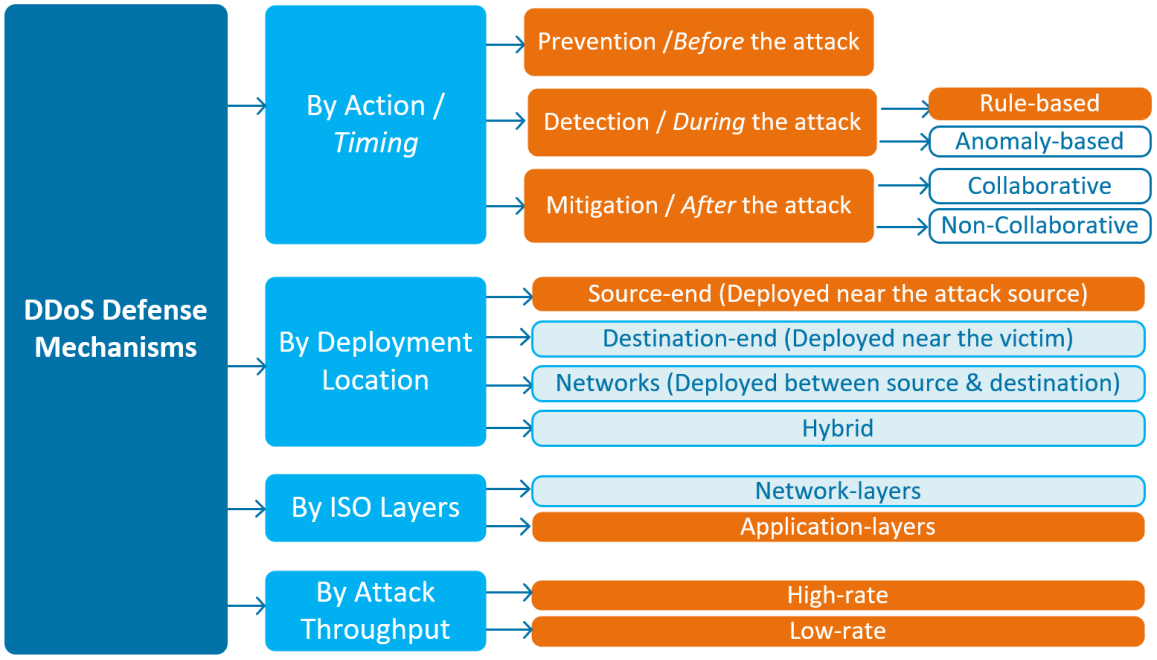


Fig. 3.3 Mapping SIPshield capabilities (in orange) to DDoS defence Mechanism Taxonomy.

Table 3.1 By Timing: SIPshield supports prevention (before), detection (during), and mitigation (after) of SIP DDoS flooding attack

By Timing		
Before	During	After
Geng and Whinston (2000)	Huang and Pullen (2001)	Mirkovic et al. (2003b)
Cabrera et al. (2001)	Peng et al. (2003)	Yaar et al. (2003)
Keromytis et al. (2002)	Kim et al. (2006)	Liu et al. (2008)
SIPshield	SIPshield	SIPshield

Table 3.2 By Deployment Locations: SIPshield is deployed in the source network

By Deployment Locations			
Source-end	Destination-end	Network	Hybrid
Gil and Poletto (2001)	Park and Lee (2001)	Cabrera et al. (2001)	Mahajan et al. (2002)
Mirkovic et al. (2003a)	Mizrak et al. (2008)	Wang et al. (2007)	Chen and Park (2005)
Abdelsayed et al. (2003)	Gonzalez et al. (2011)	John and Sivakumar (2009)	Chen et al. (2006)
SIPshield	-	-	-

Table 3.3 By ISO Layer: SIPshield operates at application-layer (SIP)

By ISO layer	
Network	Application
Cabrera et al. (2001)	Kambourakis et al. (2007)
Wang et al. (2007)	Ranjan et al. (2008)
John and Sivakumar (2009)	Rahul et al. (2012)
-	SIPshield

Table 3.4 By Throughput: SIPshield uses attack detection based on SIP state-machine anomaly. Therefore, high and low-rate attack detection is supported.

By Throughput	
High-rate	Low-rate
Behal and Kumar (2017)	Al-Haidari et al. (2017)
Boro and Bhattacharyya (2017)	Sahoo et al. (2018)
Chen et al. (2018)	Wu et al. (2019)
SIPshield	SIPshield

Evaluation by Deployment Location

Under this classification, the DDoS defence mechanism is assessed based on the location where it is deployed. There are four sub-categories: source-end, destination-end, network-based, and hybrid.

1. **Source-end.** In this sub-category, the defence mechanism is deployed near the source of the attack to prevent the devices from generating DDoS attacks. Previous implementations have it deployed on the routers by using methods such as ingress filtering, heuristic rate traffic, and reverse firewall.

   SIPshield falls under this category since ShieldVNF is deployed on the ethernet switch. The difference is on the device that performs the defence. Previous implementations have it on the router, whereas SIPshield has it on the switch. Having the defence implemented on the switch gives more granular filtering and coverage. A router is typically shared by the hosts on the same subnet, whereas with the switch, it has a one-to-one ratio where each device is directly connected to the switch port. With this one-to-one coverage, SIPshield is able to perform deep packet inspection for each packet that the device sends or receives.

2. **Destination-end.** In contrast to the previous sub-category, a destination-end defence mechanism is deployed near the victim of the attack. There have been multiple proposed solutions over the years using techniques like IP traceback, Management Information Base (MIB), packet marking, packet filtering, hop-count filtering, etc. One of the drawbacks of this mechanism is that the damage has typically already been done, so that limits the options available for effective response.

   This defence mechanism is not considered for SIPshield because the attack volume that is generated by the botnet is too much for an end-user to absorb. SIPshield's objective is to address the issue as close to the source as possible so that these malicious packets do not get a chance to accumulate upstream, causing issues in the network. Within the SIPshield framework, there is no component that needs to be deployed near the DDoS victim.

3. **Network-based.** In this sub-category, the defence mechanism is built into the network, which is located between the source and destination. The network mainly consists of routers of different Autonomous Systems (AS) that work together to detect and mitigate DDoS attacks at these intermediary networks. However, this defence mechanism places a lot of processing burden on the routers that have to perform security functions on top of packet routing, which is its primary task.

   This approach is not considered for SIPshield because the objective is to prevent malicious packets from entering the network in the first place. Considering that the prevention, detection, and mitigation tasks are already performed by the Ethernet switch at the network edge, it removes the need for further processing in the intermediary networks.

4. **Hybrid.** In the hybrid defence mechanism, there is a tight collaboration between source, destination, and intermediary networks to defend against DDoS attacks. For example, detection is performed at the victim's side, where it signals other nodes to perform mitigation.

   SIPshield is not designed to have a defence component at the destination-end or network, but it could be arranged in layers to provide multiple layers of protection. For example, within an organisation, the network operators could install ShieldVNF at the access-layer, distribution-layer, and core-layer. In this instance, when an outbreak has occurred at the access-layer, the ShieldVNF at the distribution-layer switch would provide a layer of protection so that the damage does not get escalated to the core-layer. Another example would be in an ISP environment with multiple subscribers. Each subscriber could install ShieldVNF, and at the ISP level, they would have ShieldVNF installed as well. In this use case, an outbreak from one subscriber would be contained by ShieldVNF at the ISP-level.

Evaluation by the Timing of Attack (Before vs. During vs. After)

Under this classification, the DDoS defence mechanism is assessed based on the timing related to the attack. There are three sub-categories: before the attack (prevention mechanism), during the attack (detection mechanism), and lastly, after the attack (mitigation mechanism).

1. **Before the attack (Prevention).** In this classification, the defence is to prevent the attack from occurring, for example, by hardening the systems, protocol, resource allocation, firewall, IDS, etc. SIPshield provides attack prevention by installing the relevant packet filtering rules for known attackers, known CNC, and known victims in the control plane.

2. **During the attack (Detection).** Detection algorithms can be broadly classified by rule-based vs. anomaly-based. Rule-based detection uses signatures for attack detection, and when a matching condition is encountered, the system generates an alert. The anomaly-based, on the other hand, observes the behaviour of normal sessions, and when a deviation (anomaly) is detected, the system will raise an alert.

   SIPshield uses rule-based categories where rules are defined based on a tracking state-machine with simple statistics. For example, to initiate a SIP session, a client would send an INVITE packet to the server. The next state in the protocol is for the client to wait for the server to respond with 200 OK. If a client sends hundreds of INVITE packets, then it is considered malicious. SIPshield is not using the anomaly-based

model for two primary reasons: The first is to keep the foot-print low so that it can be implemented in many different platforms (virtual or physical switch, NIC, general-purpose server, etc.). The second is to consider the performance impact so that the inspection does not negatively impact real-time sensitive sessions like SIP and RTP.

A low foot-print for implementation encourages adoption and optimises participation from the community. When the system requirement is low, it is easier to be accommodated and eventually implemented. With the objective of building a defend community, it is important to set the requirement low enough to do the job. With that in mind, the rule-based detection method is relatively easier to implement than the anomaly-based method.

A simple detection method offers better performance and does not interfere with delay-sensitive sessions. Since the rule-based detection method is simpler to implement, it can be deployed on the ethernet switch at the network edge. With deployment at the edge, the detection can be done at the edge, which offers faster processing rather than having to process it at the data center. With edge-level processing, it does not introduce a significant delay that impedes delay-sensitive sessions like VoIP.

3. **After the attack (Mitigation).** After the attack is detected, the next course of action is to locate the source and formulate a proper response. With attacks that are carried out using spoofed source IP addresses, locating the source of attack could be challenging and not effective. When responding to an attack, some use several methods like applying rate-limit, packet filtering, history-based IP filtering, hop-count filtering, etc.

   In the case of SIPshield, ShieldVNF drops the malicious packets, which followed by ShieldSDN installing the packet-filtering rules. ShieldVNF performs both functions where it detects malicious packets based on thresholds, then as a response when these thresholds are exceeded, drops the packet. At the end of this process, ShieldVNF classifies these as either known attackers or known victims (in the case of DDoS target). ShieldSDN picks up this information, distributes it to other ShieldVNFs in the organisation, and installs it as a packet-filtering rule so that all switches are synchronised in their defence posture against known attackers or known victims.

   SIPshield does not use source IP address filtering as the main criteria because it is spoofed and does not provide a reliable indicator. Being installed at the network edge, SIPshield has the advantage of being close to the attack source and therefore does not require traceback. Path tracing based on source IP address information is also not reliable because, in a DDoS attack, the IoTs often spoof the IP address to cover their tracks, and therefore it is not a reliable indicator.

SIPshield does not use rate-limiting (throttling) as it introduces additional complexity for the algorithm, which may negatively impact the performance. Rate-limiting or throttling is not currently implemented in SIPshield due to the concern that it adds complexity without getting much value out of it. Since these are malicious packets, SIPshield would immediately drop these without doing rate-limiting so that it frees up the resources for the legitimate packets. However, SIPshield implements an aging mechanism where repeat offenders are banned for a longer period of time.

Evaluation by the ISO layer (Network vs. Application)

Under this classification, the DDoS defence mechanism is assessed based on which ISO layer that is being protected. There are two sub-categories: network layers and application layers.

1. **Network layer attacks.** In this classification, the defence is to protect the ISO layers that are involved in packet delivery, e.g., physical, data link, network, and transport layers. SIPshield does not provide protection at these layers as it is not the focus of the work.

2. **Application layer attacks.** In this classification, the defence is to provide application-layer specific protection; and in this case, it is SIP.

Evaluation by the Attack Throughput (High vs. Low)

Under this classification, the DDoS defence mechanism is assessed based on the attack throughput that it is designed to handle. Researchers made this distinction because the detection and mitigation requirements are different. The low-rate attack is more elusive and harder to detect because it blends in with legitimate traffic.

1. **High-rate attacks.** SIPshield provides protection for high-rate SIP DDoS Flooding attacks from one or multiple attackers.

2. **Low-rate attacks.** SIPshield provides protection for low-rate SIP DDoS Flooding attacks from one or multiple attackers.

3.5.4 By Principles of DDoS defence

The principles were first introduced in section 3.2 where the five principles about DDoS defence were outlined. In this section, we will evaluate how SIPshield addresses the principles introduced earlier, as summarised in Table 3.5.

Table 3.5 SIPshield evaluation by DDoS defence Principles

DDoS defence Principles	(Y/N)
1. Distributed and Cooperative Solution	Y
2. The solution should not interfere with legitimate activities	Y
3. The solution should be secured against internal and external threats	N
4. The solution should have practical strategy to promote wide adoption	Y
5. The solution should be autonomously modular	Y

1. **Distributed and Cooperative Solution.** SIPshield is designed with community co-operation in mind to defeat SIP DDoS attacks from a botnet. Each member of the community participates by installing one or more ShieldVNF in their network that empowers them to prevent, detect, and mitigate SIP attacks independently. After they take care of an attack, it collaborates with other members by sharing threat intelligence via ShieldSDN and ShieldCHAIN.

   Members of this community monitor and retrieve threat intelligence from Blockchain so that they, too, can be prepared for the same attack. Retrieving a threat intelligence report and automatically installing it in their ShieldVNF prepares them to deal with the known attackers. With this workflow, SIPshield meets the criteria of being distributed but yet cooperative in its operation.

2. **The solution should not interfere with legitimate activities**. SIPshield attack detection algorithm does not modify packets nor interfere with legitimate activities. SIPshield detection works by correlating SIP packets to track session states and performing simple threshold checks. This operation does not involve modifying packets or changing the way the standard SIP operates. This method transparently enforces compliance with the standard described in RFC3261.

   A state-based tracking mechanism allows legitimate activities to pass-through. Since SIPshield is not using an anomaly-based detection mechanism, SIPshield minimises the chance of false-positives due to errors with statistical or machine-learning models. SIPshield allows sessions that adhere to a protocol specification to operate. As long as the clients are following the standard, SIPshield operation is transparent to them and does not interfere with legitimate activities.

   State-based tracking mechanism mitigates low-rate attacks, while at the same time, it allows for flash crowd situations. By tracking the session state, SIPshield decouples session quantity from session compliance. As long as the session still complies with the protocol, it does not put a restriction on the number of calls. This is important when dealing with Low-rate DDoS attacks where the botnet sends legitimate SIP sessions

at such a low-rate to evade detection. Even though the quantity of the packet is low, when it is non-compliance with the expected state, it will get dropped by SIPshield. On the opposite end of the spectrum, a flash crowd situation can occur when legitimate users are suddenly making a call at the same time. A sample of the flash crowd case would be a situation where a lot of people report a traffic accident by dialing 999 at around the same time. These calls make a surge in traffic that resembles a DDoS attack. SIPshield is able to determine that these are technically compliant calls and allow these packets to go through.

3. **The solution should be secured against internal and external threats**. At this time, SIPshield has limited security mechanisms against internal and external threats. The assumption is that ShieldVNF, ShieldSDN, and ShieldCHAIN are deployed inside a private network (local area network) and protected behind corporate firewalls. This would be a topic for further research to secure SIPshield against DDoS attacks.

4. **The solution should be practical in order to promote wide adoption**. SIPshield is made up of components that are accessible by most organisations. The basic components of SIPshield are made of VNF, SDN, and Blockchain, which will become more popular in the years to come. As network operators are replacing their old switches, one recommendation would be to purchase switches that offer programmable data planes and are compatible with the P4 programming language so that they can run ShieldVNF. ShieldSDN is node.js applications that can scale from running on small systems like raspberry pi to virtual-machines. If it needs to be scaled further, it can be run on function-as-a-service from cloud providers like Amazon or Azure as part of serverless architecture. Public Blockchain is already accessible by the public, and it does not have special requirements or a license for the public to use it.

   SIPshield is using standard-based, commodity, and non-proprietary components so that it is affordable for the community. Besides being accessible, another important factor for wide adoption is affordability. By using non-proprietary components, many vendors would be able to produce the switches, which, in turn, keeps the cost down due to market competition among vendors. This condition would benefit the customer as they have options and can avoid being locked-in by a particular vendor. The same logic goes for servers to host ShieldSDN, where it can be hosted by general-purpose computers, whether it is running as a virtual machine or in the cloud.

   Adoption is also driven by the value it brings from being a member of the community. It is envisioned that SIPshield is implemented by consortiums where like-minded organisations that use similar systems would get together to form an Information

Sharing and Analysis Center (ISAC) for a specific industry vertical, e.g., finance, manufacturing, government, etc. When the peers are part of the consortium, individual organisations would like to take part in order to benefit from being part of a larger group.

5. **The solution should be autonomously modular**. SIPshield is backward compatible and can operate autonomously. Since ShieldVNF is running on a typical Ethernet switch, the users are able to use the same cable and connectors as the legacy switch. It is backward compatible with legacy computers and interoperates with older switches. ShieldVNF is autonomous in that it can operate as a single switch and an SDN controller combination, or in a setting where one SDN controller manages hundreds of switches.

   SIPshield can be deployed in a gradual and modular fashion. ShieldVNF can be deployed as and when the old Ethernet switch needs to be replaced. Some organisations have hardware refresh schedules where they routinely replace old hardware every n-number of years with the new one in order to keep the maintenance cost down. Timing-wise, ShieldVNF deployment can be scheduled to coincide with hardware refresh schedules. When purchasing new generation Ethernet switches, the network operators would want to purchase switches that offer modern features such as programmable control and data plane so that the new switches could participate in the SIPshield framework.

3.6 Chapter Summary

In this chapter, we have proposed SIPshield, and reviewed the proposed approach and solution for defending against SIP DDoS attacks from IoT botnets. DDoS is a distributed problem which requires a distributed solution. With that in mind, the proposed approach and solution involves collaboration from the community to secure the edge of their IoT networks.

Analysis and rationale of the proposed solution were described, along with the consideration for other alternatives that are available. A taxonomy of Application-layer DDoS defence was used to locate the proposed solution within the literature, as well as justifying why certain approaches are taken. Principles of DDoS defence were reviewed and used to identify which factors were covered and which were left out for further research.

The thesis at this stage is that a SIP DDoS attack from IoT botnet can be mitigated with a scalable framework built on VNF, SDN, and Blockchain technology. In the following

chapters, each component of SIPshield will be discussed individually, along with experiments, results, and discussion, to validate this research.

Chapter 4

ShieldVNF: SIP DDoS Defence on a Switch

4.1 Overview

The previous chapter introduced the SIPshield framework (a scalable and distributed SIP DDoS defence framework that consists of ShieldVNF, ShieldSDN, and ShieldCHAIN). In this chapter we will explore ShieldVNF that is runs on a commodity Ethernet switch. In this case, the switch provides physical network connectivity for the IoT devices and, at the same time, serves as the first line of defence against botnet attacks.

The objective of this chapter is to describe the development of mechanisms for prevention, detection, and mitigation for SIP DDoS Flooding attacks on a single switch. To achieve this objective, we will investigate six sub-questions which will be validated by six experiments:

- How can an adversary be detected and mitigated from discovering SIP servers as their potential victims?
- How can an adversary be detected and mitigated from enumerating valid SIP accounts on a SIP server?
- How can an adversary be detected and mitigated from brute-forcing the passwords of SIP accounts?
- How can a botnet be prevented from registering with its CNC server?
- How can SIP DoS attacks from IoT botnets be detected and mitigated?
- How can SIP Distributed DoS attacks from IoT botnets be detected and mitigated?

Achieving these objectives are significant because it serves as the foundation where the success obtained at this stage will be replicated to other switches and external organisations as well. The solution needs to be able to mitigate the attacks and generate threat intelligence data. Figure 4.1 depicts the before and after of this process.

In section 4.2, we will look at the proposed solution (ShieldVNF) along with the features that are required to build the first line of defence against IoT botnet attacks. Section 4.3 describes a true experiment design method to validate the hypothesis. To ensure reliability and validity of the experiments, random sampling and random assignment were used, independent-and-dependent variables were identified, followed by descriptions of the ways in which the data are going to be collected and measured. Section 4.4 describes how these experiments are going to be implemented and section 4.5 describes the procedures, results, and discussion for each experiment. This chapter concludes with section 4.6 that highlights the key findings and subsequent questions that arise from the findings.

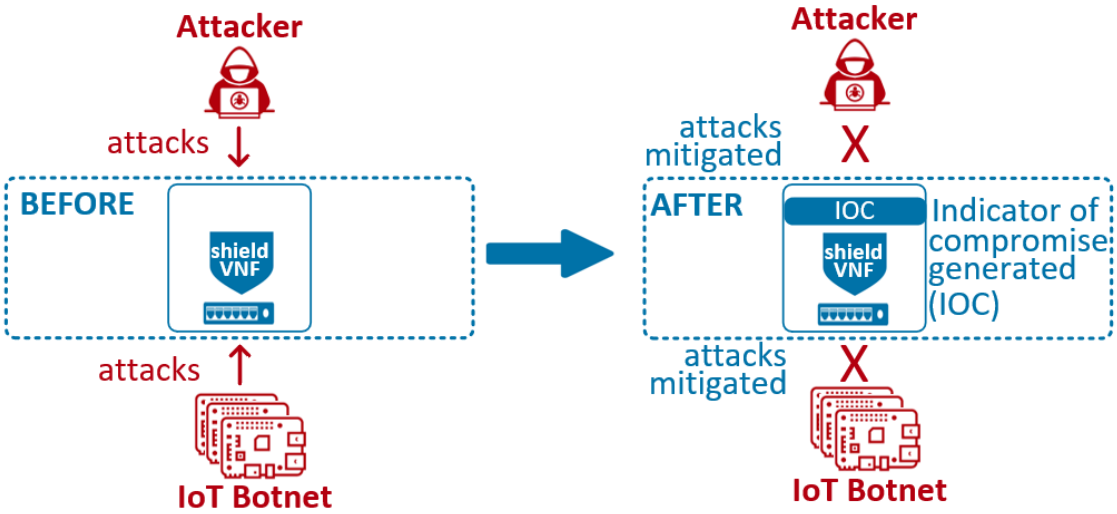


Fig. 4.1 ShieldVNF mitigates attacks and generates IOC about the attack.

4.2 The Proposed Solution: ShieldVNF

The first research objective is to develop a mechanism that provides preventive and reactive defence measures against IoT botnet attack on a single switch. Before introducing the proposed solution, we would look at the nature of the challenge and the reasons for incorporating preventive and reactive measures into the proposed solution. After gaining an understanding of the tasks at hand, we will look at the research questions that are relevant for each stage, and followed by a discussion of how the proposed solution can achieve the stated objective.

4.2.1 Preventive and Reactive Defence Measures Against IoT botnet Attacks

A comprehensive first line of defence needs to provide both preventative and reactive defence measures against IoT botnet attacks. While IoT botnet attacks have received much attention in the media, there is a precursory question that did not get much attention, i.e., how did the hundreds of thousands of IoT devices get recruited by a botnet in the first place. This question is the focus for preventive measures. The reactive measures are relevant after these IoT devices have turned into a botnet and the question becomes how to detect and mitigate attacks from an IoT botnet. The preventative measures are required before botnet infection occurred, whereas the reactive measures are required after the IoT devices have turned into a botnet. The activities involved at different stages are depicted in Figure 4.2.

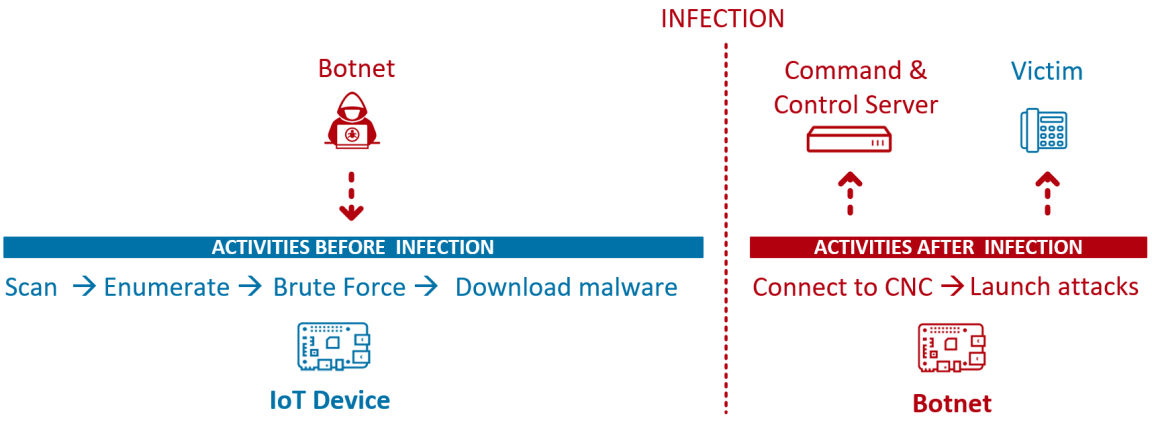


Fig. 4.2 Different stages of infection require different measures. Before the infection, it requires defence against scan attacks, enumerate attacks, brute force attack, and blocking against malware download. After the infection (when the IoT device has turned into a botnet), it requires blocking against connection attempts to the CNC server and mitigate attacks against the victim.

The preventative measures consist of several distinct, but yet related activities. From the Mirai bot incident a few years earlier, we learned from the released source code and experimentation in the lab how the Mirai botnet operates and spreads. When it first started, Mirai would scan random IP address range for vulnerable systems. Once a target was located, it would then enumerate the system to identify vulnerable accounts. When the accounts were identified, it would then brute force the accounts with weak passwords to gain entry. Once access was obtained, it would download malware that then connected to a centralised CNC server. This event registered the device as a new bot and it would start sending a regular heartbeat. At this stage, the new bot would be ready for operation and waiting for further instructions. The instructions would be sent either by the botmaster or someone that hires

botnet-as-a-service from the botmaster. These instructions were typically a command to launch a DoS attack against a victim.

The reactive measures focus on detection and mitigation of activities after these IoT devices have come under the adversary's control. First, the botnet will connect to its command and control server so that the bot can receive further commands from the botmaster. When the botmaster sends a command to the bot to launch an attack, the botnet will start to launch a DoS attack against a specific target. In this chapter, the focus is on SIP protocol as it has public safety and financial implications. We will look at three types of events: a CNC establishment attempt, a SIP DoS attack with a single attacker, and a SIP DoS attack with one hundred attackers. A single attacker assumes a situation where there is only one IoT device that has been infected. The scenario with one hundred attackers represents the case where there are multiple IoT devices that were infected and they were all sending attacks concurrently towards a victim.

4.2.2 Research Questions

With the objective of building preventive and reactive measures on an Ethernet switch, the research questions can be divided into experimental-level questions to validate the hypothesis. Table 4.1 captures lists of these experiment-level questions which follow the typical steps that the attacker would take. These questions apply the defence-in-depth principles where each question builds upon the previous one. When the attacker fails at a particular step, they are not able to proceed further to inflict further damage. Even when they succeed at one step, they need to overcome the next level of defence in their path. These layers of defences increase the amount of time and effort that the perpetrator needs to spend when attacking ShieldVNF. With this defence-in-depth principle, the adversaries might consider that it is more rewarding for them to attack easier targets and decide to move on to the next target. In the next section, we will look at the proposed solution to these questions.

4.2.3 ShieldVNF: SIP DDoS Defence At The Network Edge

The key requirements to answer the questions above lie with the ability to inspect the application-layer payload and co-relate these packets to deduce the session state. In a typical legacy Ethernet switches operation, the switches' task is to provide network connectivity. To properly defend against attacks from IoT botnet, however, we would need the Ethernet switch to expand this task to also include the application-layer inspection. The inspection needs to span multiple packets to derive the session state information. Knowing what a legitimate session state should look like based on the RFC specification is critical in order to

Table 4.1 Experiment-level research questions for preventive and reactive measures

Preventive Measures (Before recruited by a botnet)	**Reactive Measures** (After became a botnet)
1. How can an adversary be detected and mitigated from discovering SIP servers as their potential victims?	1. How can a botnet be prevented from registering with its CNC server?
2. How can an adversary be detected and mitigated from enumerating valid SIP accounts on a SIP server?	2. How can SIP DoS attacks from IoT botnets be detected and mitigated?
3. How can an adversary be detected and mitigated from brute-forcing the passwords of SIP accounts?	3.How can SIP Distributed DoS attacks from IoT botnets be detected and mitigated?

differentiate between legitimate versus malicious sessions. This level of capability is beyond what is offered by legacy Ethernet switches. However, with the emergence of programmable control and data plane from the new generation of Ethernet switches, it is now technically and economically feasible to build a VNF on top of the data plane to provide deep packet inspection at the application layer.

ShieldVNF inspects SIP and Telnet payload to track session states and enforce protocol compliance. With the VNF running on the switch data plane, it has visibility into all packets that are sent and received by each port on the switch. This level of visibility allows ShieldVNF to inspect packets related to each SIP and Telnet session to ensure that they do not violate client-server interaction as prescribed by the RFC specification. In addition to having visibility into each packet, ShieldVNF is able to correlate the packets with a session to understand the context of this packet.

With deep packet inspection capability, ShieldVNF is able to prevent, detect, and mitigate attacks from IoT botnet. For attack prevention, ShieldVNF is able to prevent the IoT devices from being hijacked and turned into bots. For example, ShieldVNF recognises when the attacker is performing scanning attacks looking for vulnerable servers to attack, or when the attacker is brute-forcing an account to gain unauthorised entry to the system. When this condition is detected, ShieldVNF is able to drop these malicious packets as an attack prevention measure. For attack detection and mitigation, ShieldVNF is able to detect an ongoing attack and mitigate the attack. For example, a compliant SIP client will send a SIP INVITE request to the SIP Proxy server and will wait for a response from the server. When hundreds of SIP INVITE packets arrive at a particular port on the switch without waiting for

a response from the SIP Proxy server, it deviates from a normal client-server interaction as prescribed by the RFC3261. ShieldVNF would detect this event as an anomaly and mitigate the attack by dropping the packet.

In addition to the above capabilities, ShieldVNF is also capable of producing an IOC that is useful for other switches. The IOCs are useful for other switches so that they too can learn from what has been done previously in order to get themselves prepared for the same attack. There are three kinds of IOC that ShieldVNF is capable of producing: KnownAttacker, KnownVictim, and KnownCNC. KnownAttacker contains information about attackers that have launched scanning, enumeration, and brute-force attacks. KnownVictim contains information about DoS target, and KnownCNC contains information about the CNC servers that the botnet has tried to connect.

The functionalities described in previous paragraphs enables ShieldVNF as the first line of defence to prevent, detect, and mitigate against a SIP DDoS Flooding attack from IoT botnets. In the next section, we will look at the design of experiments, which will provide the data for analysis and validate the hypothesis outlined in this section.

4.3 Design of Experiments

True experimental design is used in this chapter to ensure the reliability and validity of the experiments. With this design, it involves the use of random sampling, random group assignment, experimental and control groups, and the independent and dependent variables. This is necessary so that we can establish the causality and minimise the potential of alternative explanations. The tools used during experiment produced the quantitative data and it is collected after each experiment for analysis. Figure 4.3 depicts the control group vs. experimental group with ShieldVNF as the independent variable.

4.3.1 Random Sampling and Random Group Assignment

With true experiment design, random sampling and random group assignment ensures that the difference in outcome is caused by the proposed solution. In this context, the samples are the IP addresses which are dynamically and randomly assigned to the attackers and the victims. There are two groups that are involved in each experiment: the control group that uses a legacy switch and the experimental group that uses a ShieldVNF. We will conduct two experiments for each attack scenario and compare the effect. For example, in an experiment that involves 10 attackers and 10 victims, the script will generate and assign random IP address to these attackers and victims, then connect those nodes to either a legacy switch

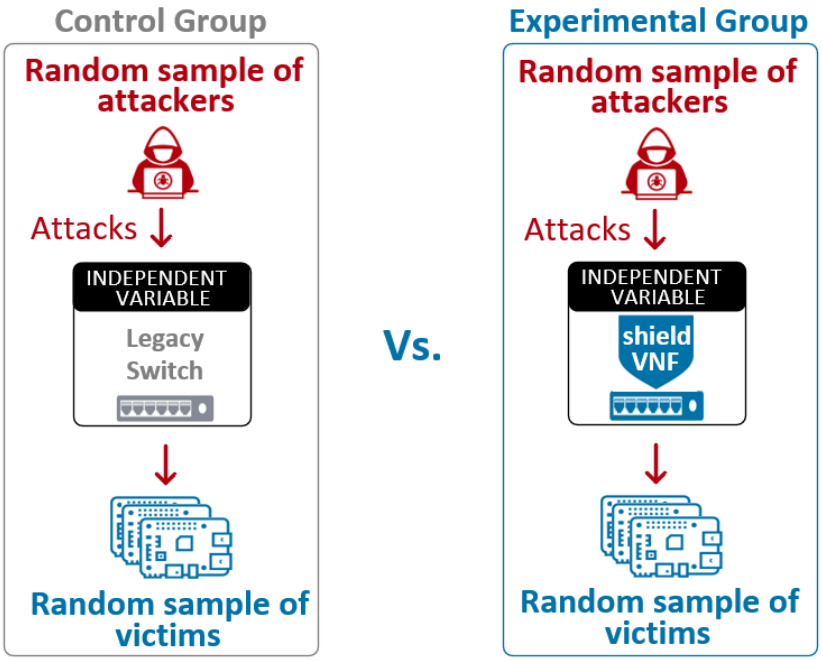


Fig. 4.3 The only difference between control vs. experimental group is the use of ShieldVNF as the independent variable.

or ShieldVNF. In essence, this process achieves a double-blind procedure where we did not have prior knowledge as to which IP addresses would be assigned to which group.

4.3.2 Independent and Dependent Variables

The independent variable in these experiments was the ShieldVNF and the dependent variables varied between experiment. To compare and contrast the outcome (dependent variables), we will either enable or disable ShieldVNF as the independent variable. The dependent variables are specific to each experiment and Table. 6.2 lists the dependent variable for each attack.

The dependent variables listed in Table 6.2 were selected based on what the attacker would expect to see at each stage of the attack. First, the attacker would perform scanning for potential targets that offer SIP services. From this activity the attackers would expect to see the number of SIP servers that they could find. Next, the attackers would enumerate the server trying to find the number of valid accounts that exists on the server. Next, the attacker would attempt a brute force attack to gain entry and expect to see how many accounts were cracked during the attack. The next step would be for the new botnet to establish a command-and-control (CNC) channel with a centralised command-and-control server. At

Table 4.2 Independent and Dependent Variables by Attack type

Experiment	Independent Variable	Dependent Variable
1. SIP server scanning attack	ShieldVNF	# of SIP servers found
2. SIP enumeration attack	ShieldVNF	# of SIP accounts found
3. SIP brute-force attack	ShieldVNF	# of SIP passwords cracked
4. CNC registration process	ShieldVNF	# of bots registered
5. SIP DoS attack	ShieldVNF	# of attack packets received # of packets sent upstream # of successful SIP calls
6. SIP Distributed DoS attack	ShieldVNF	# of attack packets received # of packets sent upstream # of successful SIP calls

this stage, the attacker would be expecting to see the number of bots that can register after the infection. After registration, the new botnet is ready for operation and waiting for the botmaster to issue a command to launch SIP DoS attacks against a victim.

During SIP DoS attack experiments, there are three dependent variables that are important to monitor. The first dependent variable is the number of attack packets received by ShieldVNF. The second dependent variable is the number of packets that ShieldVNF forwarded to the upstream switch. The third dependent variable is the number of successful SIP calls during the attack. Figure 4.4 shows the dependent variables involved during a SIP DDoS Flooding attack. While the first and third dependent variables are self-explanatory, the second variable (number of packets sent upstream) needs more details. When ShieldVNF considers a packet as legitimate, ShieldVNF will perform a normal IP routing function and hand off the packet to the next hop, which is the upstream switch. However, when ShieldVNF determines

that a packet is malicious, ShieldVNF will drop the packet and will not forward the attack packet to the upstream switch. When we see a low number of packets sent upstream during a DoS attack, it means that ShieldVNF has successfully detected and dropped the attack packets.

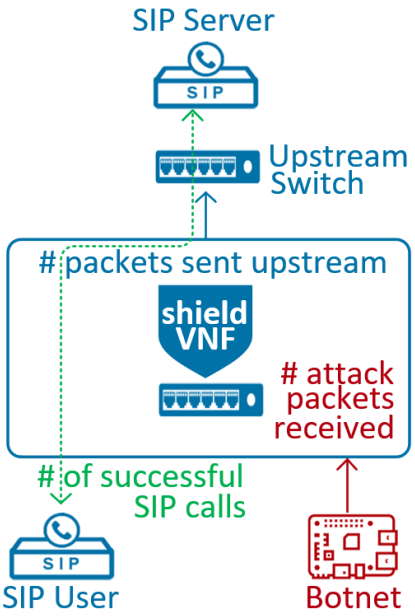


Fig. 4.4 Three dependent variables involved during SIP DoS flooding attack experiment: # of successful SIP calls made, # of attack packets received, and # of packets sent upstream. Table 4.3 shows the relationships between these variables in a failed vs. successful experiment.

The criteria for successful experiments is determined by the number of successful SIP calls. From the end-users' perspective, a successful solution is demonstrated by having successful calls even during a DoS flooding attack from a botnet. Inspecting the three dependent variables, this condition is demonstrated by having a high number of attack packets received from botnet, a low number of packets sent upstream (because the attack packets have been successfully detected and dropped by ShieldVNF), and a high number of successful SIP calls. Conversely, the end-users will perceive the solution as failed when they are not able to make SIP calls. This outcome is caused by ShieldVNF failing to detect the SIP DDoS Flooding attack from the botnet, and continuing to forward the attack packets upstream. Table 4.3 shows the relationship between dependent variables for a failed vs. successful experiment.

Table 4.3 Failed vs. Successful experiments criteria

Failed Experiment			Successful Experiment		
Number of attack packets received	Number of packets sent upstream	Number of successful SIP calls	Number of attack packets received	Number of packets sent upstream	Number of successful SIP calls
High	**High**	**Low**	**High**	**Low**	**High**

4.3.3 Data Collection and Measurements

The tools used in the experiment produced the data, and the data were collected before and after the experiment. Four types of tools were used for the experiment: attack tool, SIP call generator tool, botnet simulation, and networking tool. Table 4.4 lists the tools involved during experiments. The next few paragraphs describe how to use these tools and show the data that can be generated from these tools.

Table 4.4 List of tools used in experiments

Tool	Type	Data Generated
sipvicious_svmap (Figure 4.5)	Attack tool	Number of servers found on the network.
sipvicious_svwar (Figure 4.6)	Attack tool	Number of valid accounts on the server
sipvicious_svcrack (Figure 4.7)	Attack tool	Number of accounts that can be cracked
SIPp (Figure 4.10)	Call generator	Number of successful SIP calls
netstat (Figure 4.11)	Network tool	Number of packets sent and received by each host
miraisim.py (Figure 4.9)	botnet simulation	Number of bots that can register with centralised CNC server

Measuring the number of SIP servers discovered

The number of SIP servers discovered is useful to measure the number of potential targets. When an attacker finds a SIP server, the attacker could do further probing to find vulnerabilities to exploit. In this case, the fewer number of SIP servers discovered, the safer the environment. To obtain the data about the number of SIP servers available, we will use the svmap tool and supply the target range as a parameter. For example,

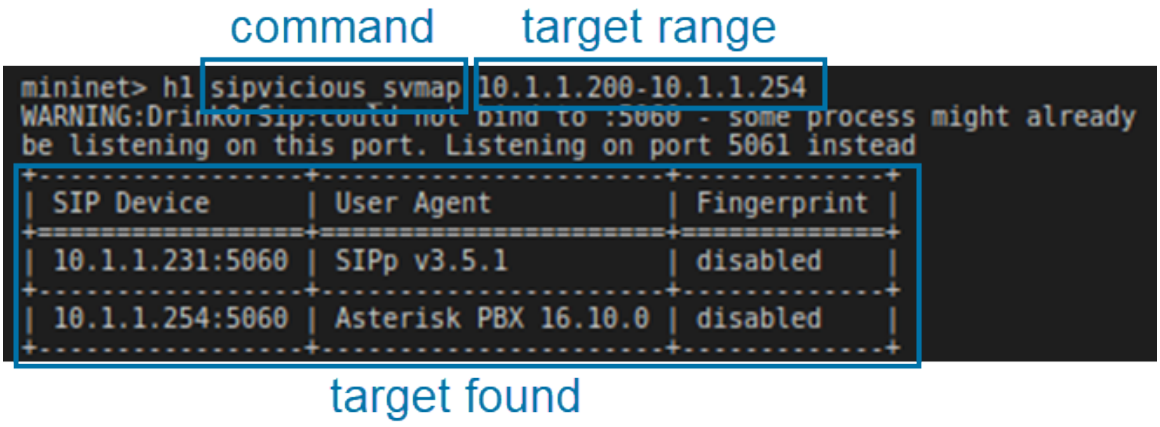


Fig. 4.5 sipvicious_svmap tool discovered 2 SIP servers in the target range.

`"sipvicious_svmap 10.1.1.200 - 10.1.1.254"` command will launch svmap to scan 54 addresses from 10.1.1.200 to 10.1.1.254 for SIP servers. In figure 4.5 it shows that svmap found 2 servers within that range (10.1.1.231 and 10.1.1.254). From the signature, svmap can see the software that the server is using, e.g., 10.1.1.231 is using SIPp version 3.5.1, whereas 10.1.1.254 is using Asterisk PBX version 16.10.0

Measuring the number of valid accounts on a SIP server

The number of extensions discovered on a given server is helpful to measure the attack surface on a server. The more extensions that an attacker can discover, the larger is the surface and, therefore, more prone to exploits. To measure this data, we will look at a tool called svwar. For example, `"sipvicious_svwar 10.0.0.1"` command will launch svwar and probe the server for available extensions. As shown in figure 4.6, svwar found 12 extensions at the target (101, 102, 103, etc.). Out of 12 extensions, 10 require authentication while the other two do not.

Measuring the number of accounts that can be brute forced

In order to measure the number of extensions that are vulnerable to brute force attacks, we would look at an exploit tool called svcrack, which would perform brute force attacks on these extensions. The fewer passwords that are vulnerable, the better. For example, `"sipvicious_svcrack -u 101 -d dictionary.txt 10.0.0.1"` command will launch svcrack tool to crack the password of extension 101 on the SIP Proxy server with IP address of 10.0.0.1, using passwords from a file (dictionary.txt). In figure 4.7, svcrack was able to crack the password for extension 101.

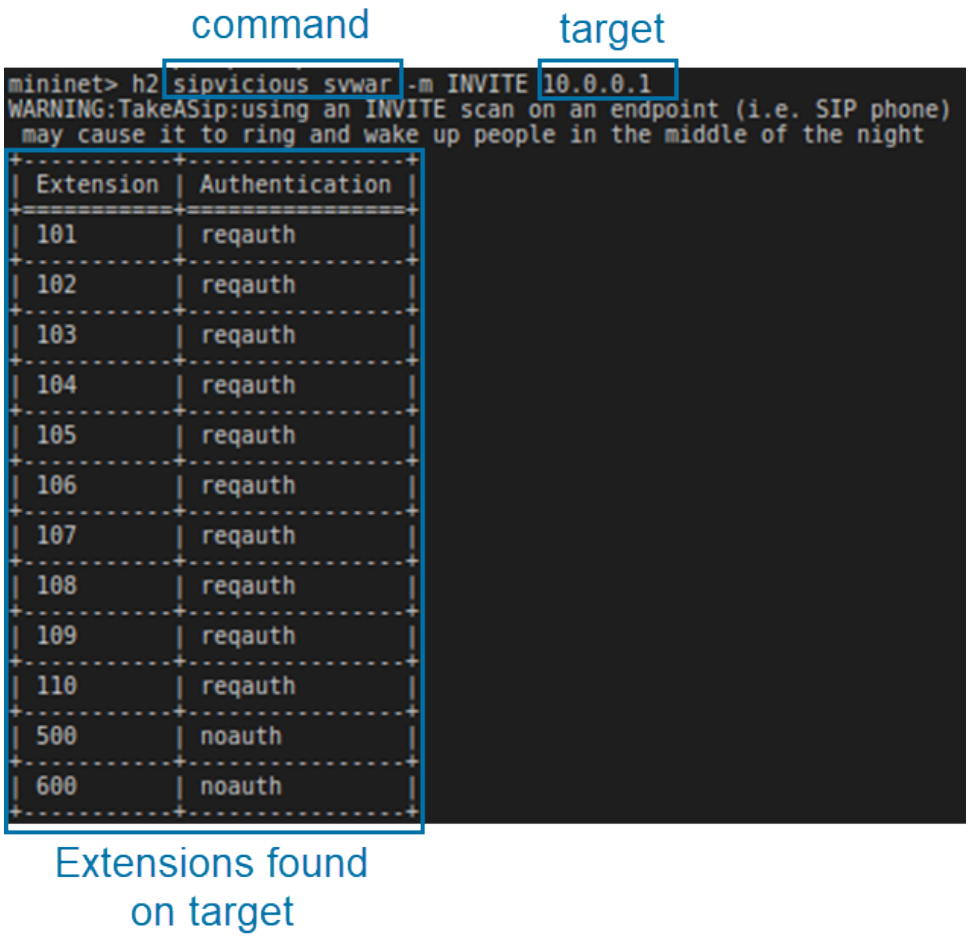


Fig. 4.6 sipvicious_svwar tool discovered 12 valid extensions on a SIP server. 10 extensions require authentication and 2 extensions do not require authentication.

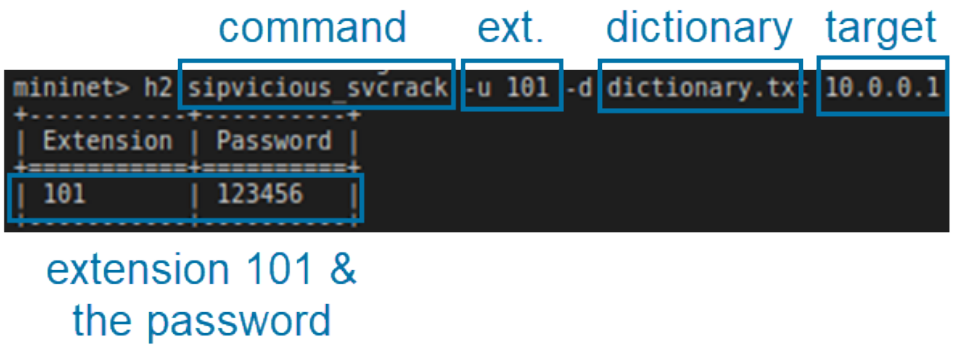


Fig. 4.7 sipvicious_svcrack tool cracked 1 password for extension 101 on SIP server 10.0.0.1.

Measuring the number of botnets that successfully registered with the CNC server

When the bot first starts, it will register with the centralised Command and Control (CNC) server. If these registrations are blocked, the bot will not be able to join the botnet and operate

as intended. In a secured environment, none of the bots would be able to register with the CNC, which then renders them ineffective. In order to be able to measure the number of bots that can register with a command and control (CNC) server, we downloaded a dataset that contains a packet capture from a real Mirai botnet set up in the lab. In this packet capture, as shown in Figure 4.8, we see a Mirai botnet registered with the Command and Control (CNC) server.

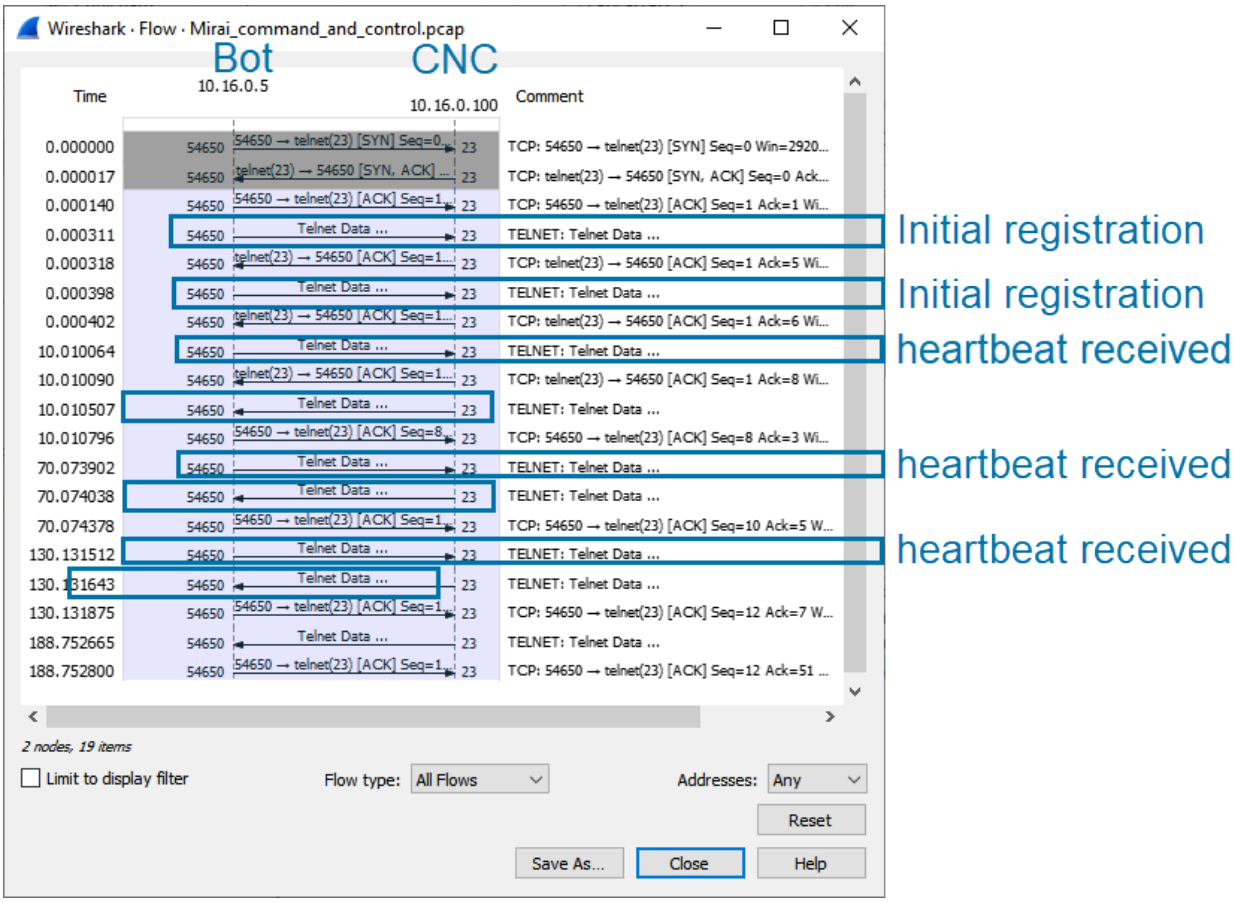


Fig. 4.8 Mirai bot registration (taken from a real Mirai session dataset)

A python script simulated the event for a bot and CNC communication. This simulator matched the behaviour of Mirai botnet, exactly to the packet level. Just like the real session above, the simulated Mirai bot registered itself with a CNC with a specially crafted payload (4 bytes: `\0x 00 00 00 01`). It followed with the second portion of registration that consists of one-byte payload (`\0x 00`). After this handshake, the new bot was registered with CNC and starts sending and receiving 2 bytes payload (`\0x 00 00`). In figure 4.9, a bot has successfully registered with CNC, and the heartbeat between the bot and CNC server was sent and received successfully.

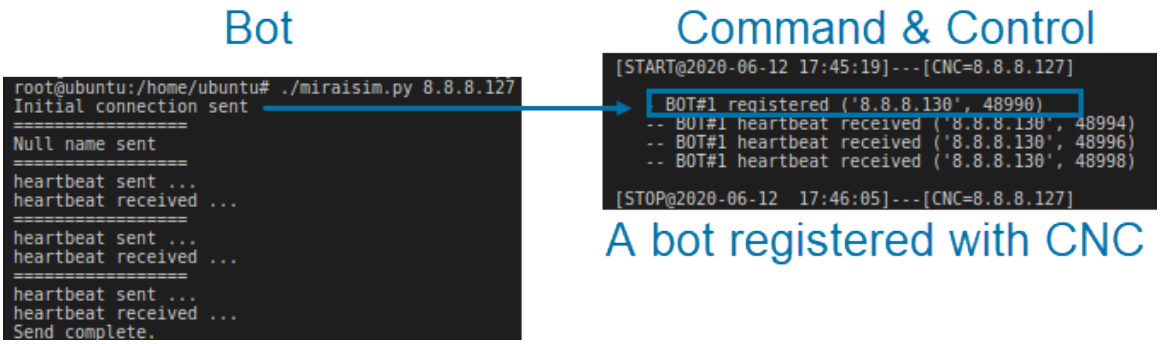


Fig. 4.9 One Bot has successfully registered with the simulated CNC server.

Measuring the number of successful SIP calls

SIPp tool created SIP calls so that we could measure the number of successful SIP calls. This data point is important to measure because we would like to measure the number of successful calls when the DoS attack was running in parallel with legitimate client calls. To the end-user, a successful call is a metric that matters since it directly affects the users. In a secured environment, the number of successful calls should be high. For this purpose, we turn to a tool called SIPp. SIPp is an open-source and widely used SIP traffic generator tool that was created by Hewlett Packard for the SIP community. At the launch time, SIPp can either be a client or a server. At the end of every SIP test session, SIPp produces a statistic screen that allows the user to observe how the test went.

In this example, a SIP session was running between a client and a server. The client was making one call per second for 60 seconds. When the test session ended, SIPp produced a screen, as shown in 4.10. The figure shows the start time (1), the duration of the test (2), the call rate (3), and the number of successful calls (4). In this case, all calls were successful.

Measuring packets sent and received by each host

Another important data point to observe is the number of packets that each host sends and receives. This is important because we can get an indication whether a host is an attacker or legitimate user by the number of packets they sent. For example, with a DoS Flooding attack, the botnet will send a burst of packets in a short-time period. Information about packets sent and received can be obtained from the virtual switch as it has counters that keep track of this for every port on the switch. A network command line called `netstat` is able to show all the ports on a switch along with its statistics. As an example, Figure 4.11 shows a virtual switch that is connected to 10 virtual hosts. Host1 is connected to s1-eth1 interface and Host10 is connected to s1-eth10 interface. The third column from the left shows that Host1 and Host10 have received 119 packets. The sixth column shows that Host1 has sent 209 packets and 208

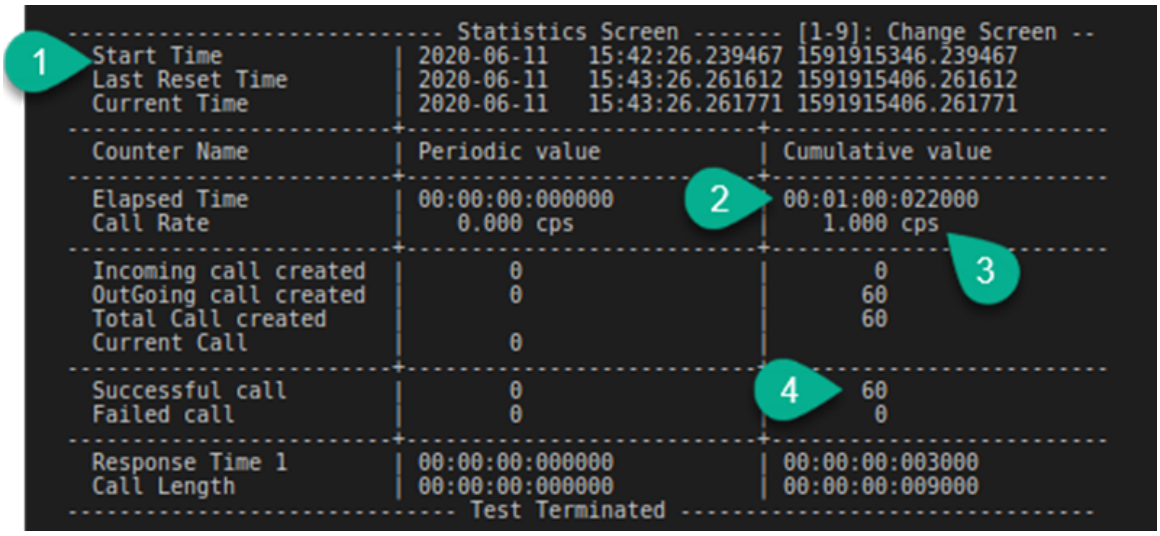


Fig. 4.10 SIPp shows 60 successful calls (4) and other details like start time (1), elapsed time (2), and call rate, which is 1 call per second (3)

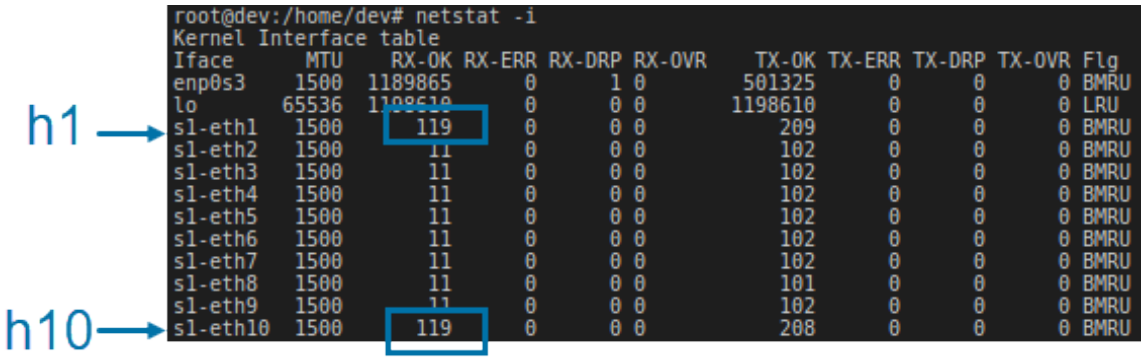


Fig. 4.11 Netstat tool shows the number of packets sent and received by each host. In this example, h1 and h10 received 119 packets.

packets. Packet counts also indicate whether or not the switch has successfully mitigated an attack. By comparing the number of packets received from the botnet with the number of packets sent to the upstream switch, we can get the data about whether ShieldVNF is operating as designed. Figure 4.12 shows how the packet counters can indicate whether or not ShieldVNF was able to mitigate the attack.

4.4 Instrumentation and Implementation

This section describes in detail the way that the experiments were implemented in the Amazon Web Services (AWS) environment. It starts with AWS virtual private cloud (VPC) and EC2 virtual machines that hosts the virtual network, virtual hosts and virtual switch. We will

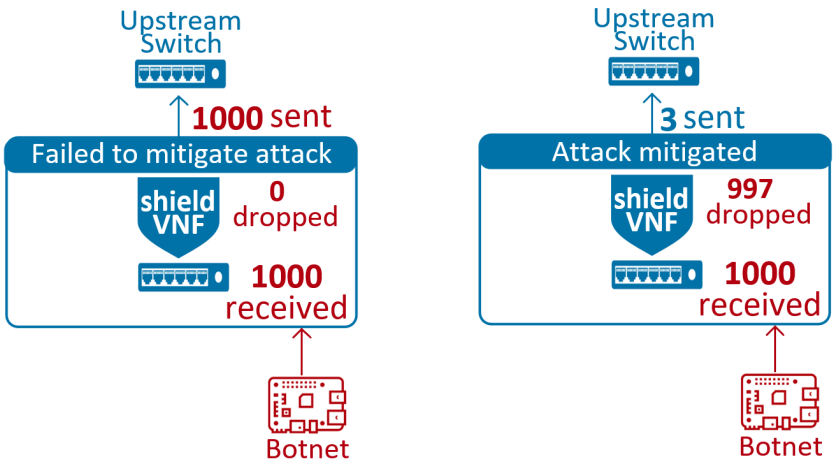


Fig. 4.12 By looking at the number of packets sent/received, we can see whether ShieldVNF has successfully mitigated an attack. In this example, ShieldVNF received 1000 attack packets from botnet. The ShieldVNF on the left failed to mitigate the attack and forwarded 1000 malicious packets upstream, whereas the ShieldVNF on the right has successfully mitigated the attack and forward only legitimate packets to the upstream switch.

look at how the VNF is being implemented on the virtual switch's programmable data plane, followed by a walk through to describe the mechanics of how the VNF works.

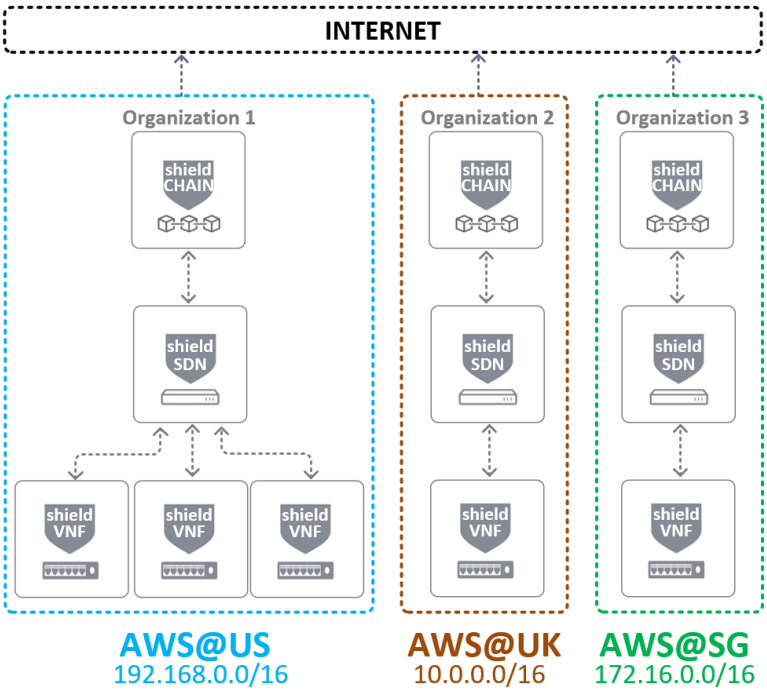


Fig. 4.13 Three AWS Virtual Private Cloud (US, UK, and SG) and the EC2 virtual machine instances within each Virtual Private Cloud.

4.4.1 Cloud Computing Environment

Amazon Web Service (AWS) cloud computing environment was used in these experiments to simulate a global environment with sites in North America, the United Kingdom, and Singapore. Having an environment in multiple parts of the world reflects the scale and extent of the problem. We could have used a local server to set up a similar environment. However, there would be too many localised variables involved and it would be hard to replicate the environment. AWS was chosen because it is publicly accessible and offer a standardised environment.

AWS services used for implementation were the Elastic Compute Cloud (EC2) virtual machines and the virtual private cloud (VPC). The experiments used the EC2 t2.micro instances and VPC in three different countries (US, UK, and Singapore). Figure 4.13 shows a number of instances within each private cloud. In this environment, each EC2 instance could only connect to other instances within the same VPC. This is to simulate a real network where a host could only connect to other hosts within the same organisation. Communication between organisations would need to connect via the Internet.

4.4.2 Virtual Network, Virtual Switch, & Virtual Hosts

Each AWS EC2 instance launches Mininet software to create a virtual network environment that consists of a virtual switch and virtual hosts. Mininet created several virtual hosts that are connected to a virtual switch as depicted in Figure 4.14. This level of programmability allows for a flexible and powerful ways to create a different networking environment for a different experiment. For example, to emulate an attacker with ten potential targets, we could write a python script that instantiates 11 virtual hosts, 1 for the attacker, and 10 for the potential target hosts.

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

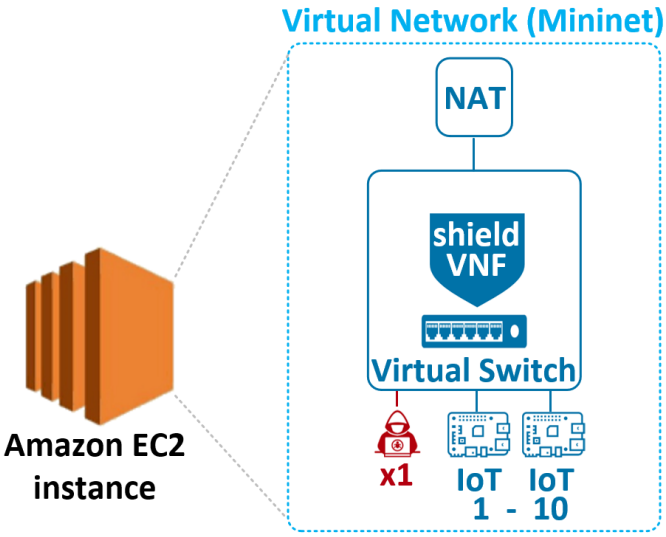


Fig. 4.14 A virtual network environment running on an AWS EC2 instance that consists of a virtual switch and 11 virtual hosts (1 for attacker and 10 for IoT devices).

4.4.3 VNF

ShieldVNF is a VNF that is written in P4 programming language which allows the users to create a custom code to process each packet. In this case the code is to prevent, detect, and mitigate a SIP DDoS flooding attack from IoT botnet. This code is running on the switch's data plane and follows this workflow: parse, inspect, verify, and count, as depicted in Figure 4.15.

Parse

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

Inspect

Once ShieldVNF has parsed these fields, the packet goes through the **inspect** phase, where ShieldVNF performs deep packet inspection. There are two main tasks for this phase: first is to identify whether this packet is part of the known attack, and second, whether this is part of the SIP attack. ShieldVNF accomplishes the first task by comparing the following

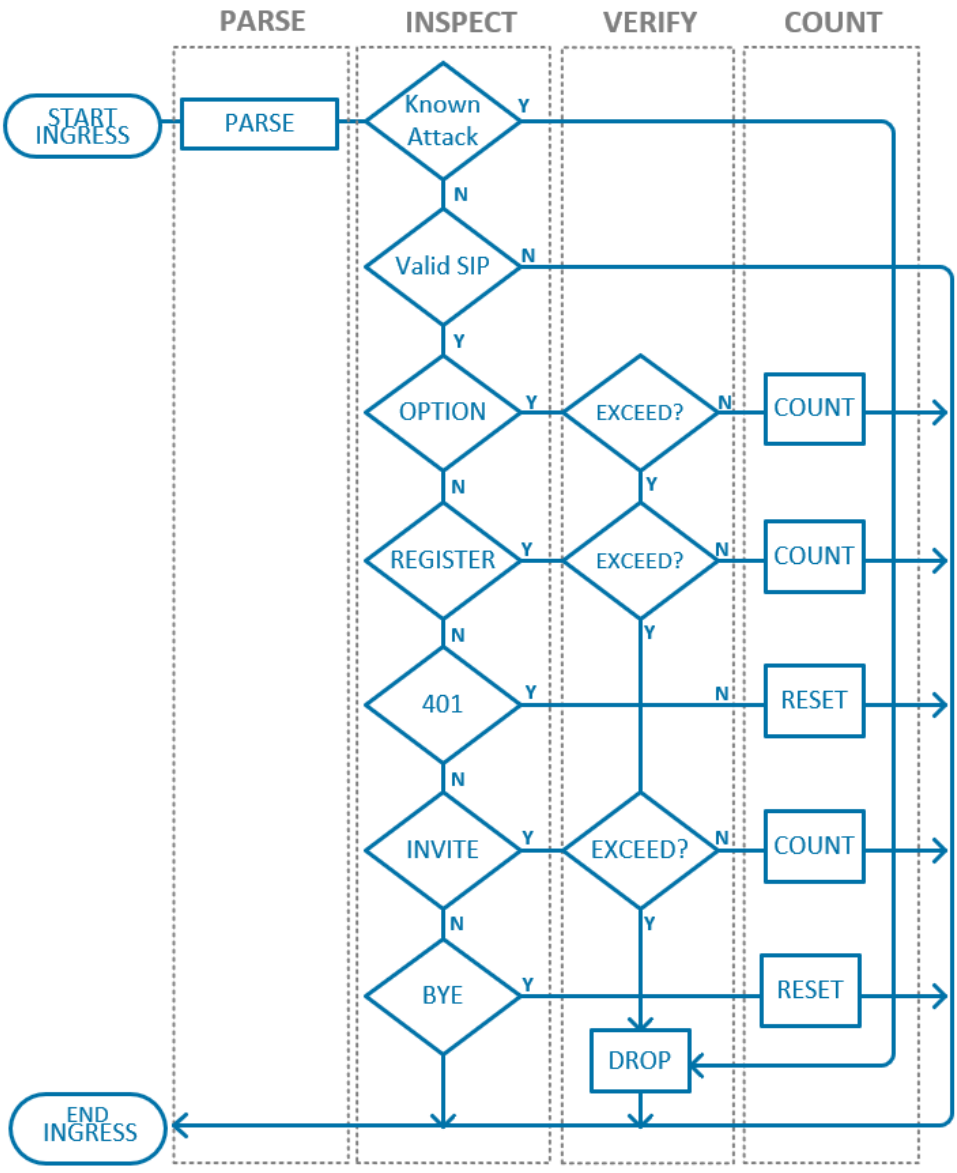


Fig. 4.15 ShieldVNF workflow has 4 phases: Parse, Inspect, Verify, and Count.

key pieces of information and dropping the packets if that information matches previously known record:

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

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

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

Verify

After learning about the intent for the packet by examining the payload and how it affects the session state, ShieldVNF performs verification in the **verify** phase. In this phase, ShieldVNF validates whether or not this packet has exceeded the threshold that was previously set for the session. If it has, ShieldVNF will drop the packet at the end of the process. If not, ShieldVNF will move it to the next phase. For example, a normal SIP client-server session starts with an `INVITE` packet and ends with a `BYE` packet. When there are hundreds of `INVITE` packets without the corresponding `BYE` packet, it indicates a DDoS attack, and ShieldVNF needs to drop those packets. With ShieldVNF, the threshold for an open `INVITE` sessions is three, which means that every client can only initiate three concurrent SIP calls at any given time. This threshold is configurable, but reasonable for a normal SIP client-server interaction. When the counter for each sender is still below the threshold, ShieldVNF will simply count this packet against the sender. At the end of a normal session (and when the expected `BYE` packet is encountered), the counter for that particular sender is reset to zero.

Count

In the **count** phase, even when the packet has not exceeded the threshold, ShieldVNF will count the number of packets associated with each session. The counter is increased and decreased as necessary, depending on whether or not it follows the expected operation as per

SIP protocol. For example, when a SIP-client initiates a call using the SIP `INVITE` packet, ShieldVNF sets the counter for the first INVITE packet for this session to one. When the call ends, and the SIP-client receives a SIP `BYE` packet, ShieldVNF will reset this counter to zero. In this instance, the SIP session followed proper call establishment protocol, and therefore the value of this counter will be zero. This counter is the one that gets verified in the previous phase. Considering that the edge Ethernet switch has a limited amount of memory, it is necessary to use a data structure that is efficient for tracking each session that is generated by the IoT devices. For this operation, bloom filter (Bloom, 1970) provides a space-efficient data structure which will be described in detail next.

Data Structure

ShieldVNF uses Bloom filter to keep track of packets and how it affects its session state. Keeping track of a session requires inspecting the packets that are associated with that particular session, and this operation could take up much space in the memory. With a limited amount of memory available in the edge switch, it is important to select a data structure that is appropriate for this context. ShieldVNF selected Bloom filter due to its space-efficient property for insert and lookup operations. It does not store the actual information about the object itself, but rather, it stores whether or not it has seen this object previously. Since it does not store the details about each packet and session, it is efficient in using memory space. Bloom Filter is originally used to check whether or not an element is a member of a given set of elements. In this context, Bloom Filter is adapted as a Counting Bloom Filter (CBF) so that it can count the frequency of the packet that we are tracking. When the frequency has exceeded a threshold, this packet is considered part of a malicious session, and therefore it will get dropped. In order to uniquely identify and track each session while preserving memory space, ShieldVNF is using a hash function for this purpose.

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

- Destination IP: 49.21.4.2
- Destination Port: 5060
- Physical port on the edge switch: 12

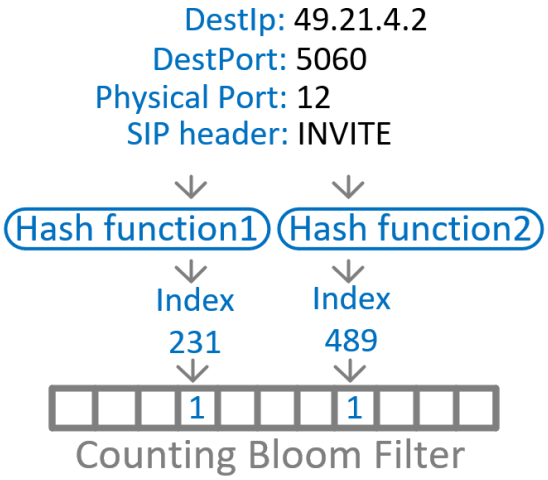


Fig. 4.16 Using counting bloom filter (CBF) data structure to keep track of the number of SIP packets. This packet is hashed to index 231 & 489. Since this is the first time the CBF encountered this packet, the value for index 231 & 489 is set to 1.

- SIP header: INVITE

This information is then fed to hash1 and hash2 and produced 231 and 489, respectively. Since this is the first packet that ShieldVNF sees, ShieldVNF sets the value of element number 231 and 489 to 1 as depicted in Figure 4.16. The next time ShieldVNF sees the same packet and the same hash, ShieldVNF will set the value to 2. In essence, ShieldVNF uses each array element as a packet counter. In order to put it into a proper context, these packets need to be tracked together as a session.

ShieldVNF tracks session state by correlating SIP packets payload that indicates whether this is a SIP request or response. When inspecting a SIP payload, ShieldVNF is checking for a SIP request and response in order to deduce the state of a SIP transaction. When a SIP client initiates a connection to the server, it goes through the following states: Calling, Proceeding, Completed and Terminated as described in the RFC3261 (Rosenberg et al., 2002).

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

4.4.4 A Walkthrough of Packet and Session Tracking

In order to describe how these components work together, let us consider a walkthrough that demonstrates how packet counter and session tracking work together to detect and mitigate SIP DDoS attacks. In Figure 4.17, an IP camera is connected to physical port number 3 on the edge switch. It sends a SIP INVITE packet to a remote SIP server and the packet is received by the switch that has ShieldVNF enabled. ShieldVNF parses the headers and performs an inspection. From the inspection process, ShieldVNF realises that this packet contains a SIP INVITE request. ShieldVNF needs to track this packet due to potential DDoS attack associated with it. The two hash functions generate two index numbers (3) and (7) for this particular packet based on four parameters: destination IP, port IP, 3 (the port number on the edge switch), and INVITE (the payload of SIP header). ShieldVNF then sets the value of index 3 and 7 location to 1 to indicate that this is the first packet of this session. If the IP camera sends another packet that shares the same details (destination IP, destination UDP, port 3, and INVITE), the counter will be increased by one. Any variance on the four input elements will generate different index numbers. For example, a different destination IP address will generate different index numbers even though the packet is using the same destination UDP port. This process describes the northbound process (from an IP camera to the SIP server). In the next section, we will look at the southbound process (from the SIP server to the IP camera).

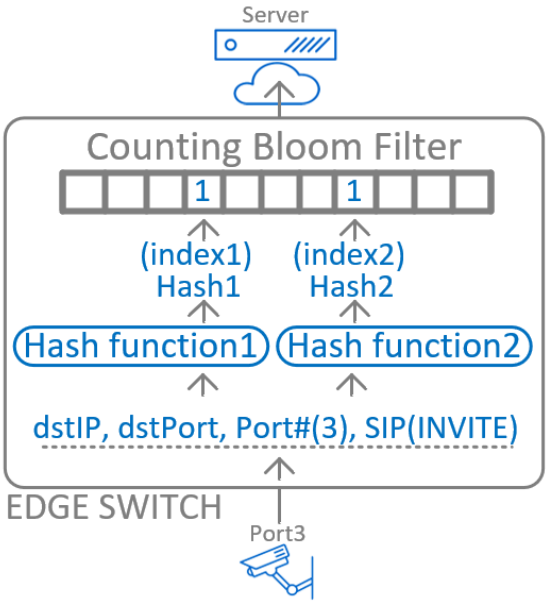


Fig. 4.17 Northbound: From camera To server: Counting Bloom Filter sets the number of active SIP session to 1. A botnet infection and DoS attack is suspected when this number keeps increasing.

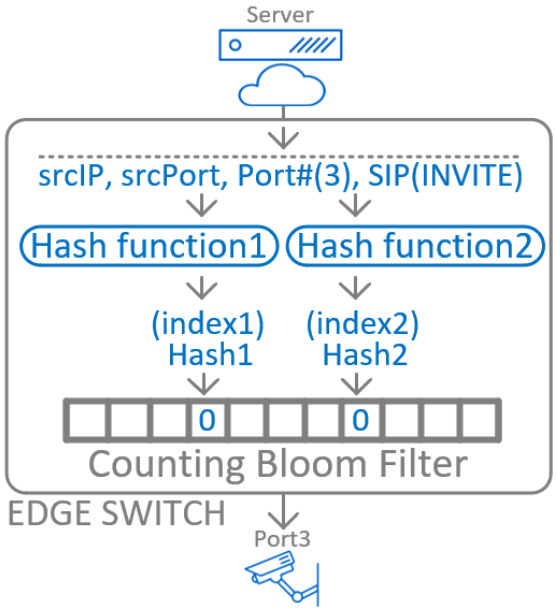


Fig. 4.18 Southbound: From server To camera: Counting Bloom Filter resets the number of active SIP session to 0.

When the SIP server sends a session termination request by sending a `BYE` packet to the IP camera, it will reset the counter at index 3 and 7 location, back to zero (Figure 4.18). When the server sends a response to the original request, ShieldVNF will use the same hash algorithms to generate the same index numbers. In order to generate the same index numbers as the original request, ShieldVNF has to modify the input for the hash function so that it matches with the original request. In this process, ShieldVNF substitutes the source IP with the destination IP. ShieldVNF also substitutes the destination UDP with the source UDP. This step is important because we need to ensure that the hash generates the same index so that it can reset the correct counters in the CBF array.

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

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

4.5 Experiments

The following experiments validate the hypothesis that ShieldVNF is able to prevent, detect, and mitigate SIP DDoS Flooding attacks on a single switch. There are six experiments in total that follow the attacker's journey from the initial scanning attack to the SIP Distributed DoS attack. These experiments will provide data points for data analysis and drawing conclusions. Table 1.2 list the experiments.

Table 4.5 Experiments for ShieldVNF following the attacker's journey

Experiments (The attacker's journey)	**Sample population** (random selection & random assignment)	**Independent Variable**	**Dependent Variable**
1. SIP Scanning attack (finding servers)	3 attackers & 10 targets	ShieldVNF	# of SIP server found
2. Enumeration attack (finding accounts)	3 attackers & 10 accounts	ShieldVNF	# of SIP accounts discovered
3. Brute force attack (cracking passwords)	3 attackers & 10 passwords	ShieldVNF	# of passwords cracked
4. CNC channel (establishing command & control with botmaster)	10 bots	ShieldVNF	# of bots registered
5. SIP DoS attack	1 attacker & 1 caller	ShieldVNF	# of successful call & # of packets sent upstream
6. SIP Distributed DoS attack	100 attackers & 1 caller	ShieldVNF	# of successful call & # of packets sent upstream

4.5.1 Experiment 1: SIP scanning attack

The first experiment is to answer the first research question, i.e., How can an adversary be detected and mitigated from discovering SIP servers as their potential victims? If ShieldVNF is working as designed, then the adversaries will only find some potential victims.

Procedures

The first experiment performed two tests: with a legacy switch and with ShieldVNF running on a programmable switch as shown in Figure 4.19.

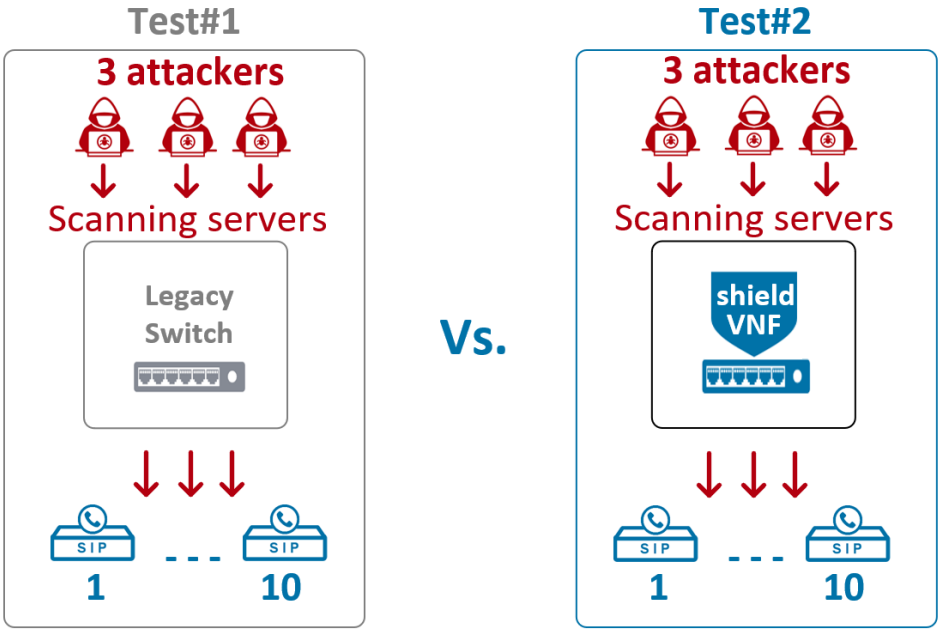


Fig. 4.19 **Experiment 1:** Attackers scanning for SIP servers (Test1 with legacy switch vs. Test2 with ShieldVNF)

As per RFC3261 for SIP, a server is supposed to respond when a client sends an `OPTIONS` request. This response is how the SIP client learns that the other host supports SIP. The attacker uses this method to scan a wide range of IP addresses to discover SIP servers by probing each IP address with an `OPTIONS` request. A legitimate SIP client is typically already assigned with a primary and secondary SIP Proxy server. Therefore, it does not have to scan a wide range of IP addresses to find a SIP Proxy.

In this test, there were three attackers and ten SIP servers, all attached directly to a single switch. The test script dynamically generated an IP address for each host and their role (attacker or victim) at run time from a range (10.1.1.1-10.1.1.250) to remove bias. This

method achieved a double-blind effect, where the participant and researcher did not know the IP address and roles of each host before the experiment.

A python script launched the mininet environment, the attackers, and the SIP servers. The process started with attacker#1 using the `SIPVicious_svmap` tool to scan a range of IP addresses (10.1.1.1 - 10.1.1.250). When attacker#1 completed the attack, attacker#2 started scanning and attacker#3 came at the end. The rationale behind using three attackers was to make sure that the results showed a consistent pattern and did not display wide variability.

ShieldVNF limited the number of unacknowledged `OPTIONS` packets that each port could send to three. When the `OPTIONS` packet is acknowledged with `200 OK` packet, the counter is reset to zero. However, when there are three or more unacknowledged packets pending, ShieldVNF automatically drops the remaining `OPTIONS` packets.

Result

Table 4.6 **Experiment 1: Result:** ShieldVNF blocks SIP scanning attack (3 discovered vs. 10)

Experiment 1: SIP Scanning Attack	Legacy Switch		**ShieldVNF**	
	Number of servers available	Number of servers discovered	Number of servers available	Number of servers discovered
Attacker #1	10	10	10	3
Attacker #2	10	10	10	3
Attacker #3	10	10	10	3

Below is a screenshot result of the two tests, Figure 4.20 shows the outcome with the legacy switch, while Figure 4.22 depicts the result with ShieldVNF.

Discussion

With a legacy switch, the attackers were able to launch a SIP scan attack where it sent a `SIP OPTION` request to each IP address that it could find. The screenshot (Figure 4.20) shows that all three attackers were able to discover all ten SIP servers.

With ShieldVNF, on the other hand, there was a rule imposed on each port where it limited the number of `SIP OPTION` packets to three. As a result, each attacker could only discover three SIP servers because ShieldVNF dropped the fourth to the tenth `SIP OPTION` packets, as shown in Figure 4.22. This outcome presents evidence that ShieldVNF was able to detect and mitigate the SIP scan attack.

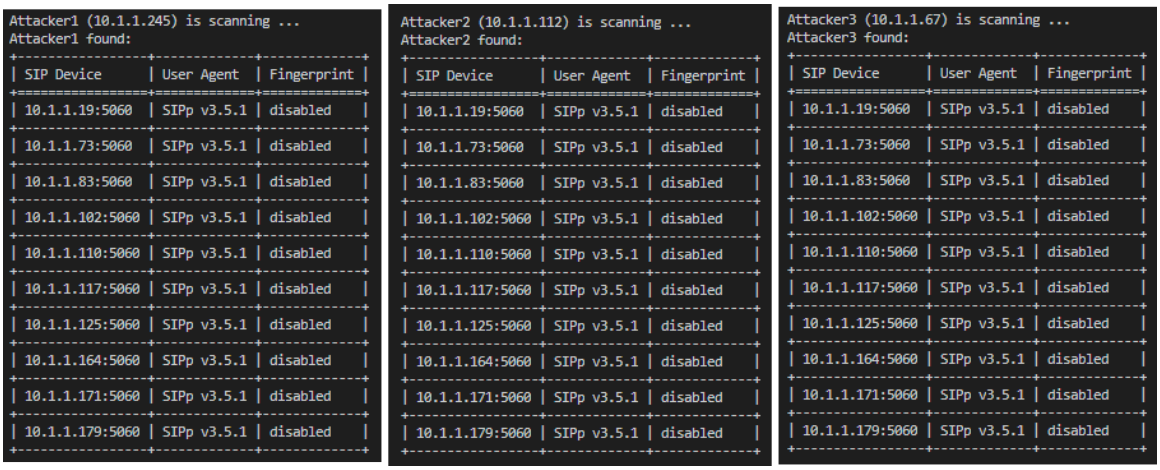


Fig. 4.20 **Experiment 1: Test1 Result:** SIP Scan attack with legacy switch. Attacker1 (left), Attacker2(middle), Attacker3(right) found 3 servers.

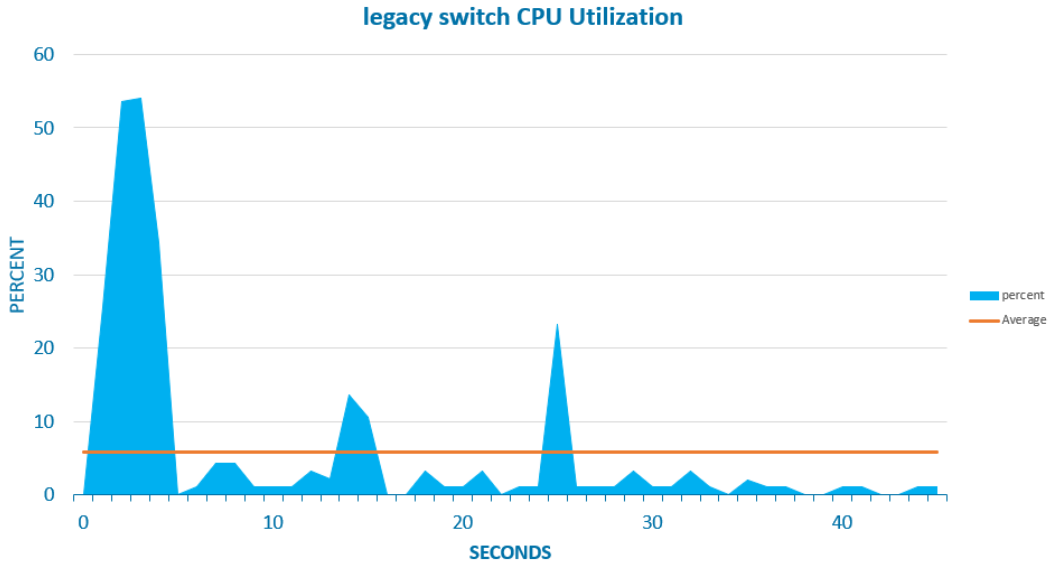


Fig. 4.21 **Experiment 1: Test1:** CPU utilisation of legacy switch.

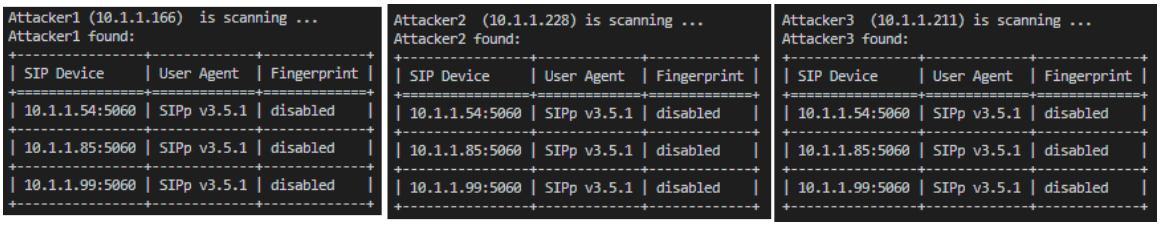


Fig. 4.22 **Experiment 1: Test2 Result:** SIP Scan attack with ShieldVNF. Attacker1 (left), Attacker2(middle), Attacker3(right) found 10 servers.

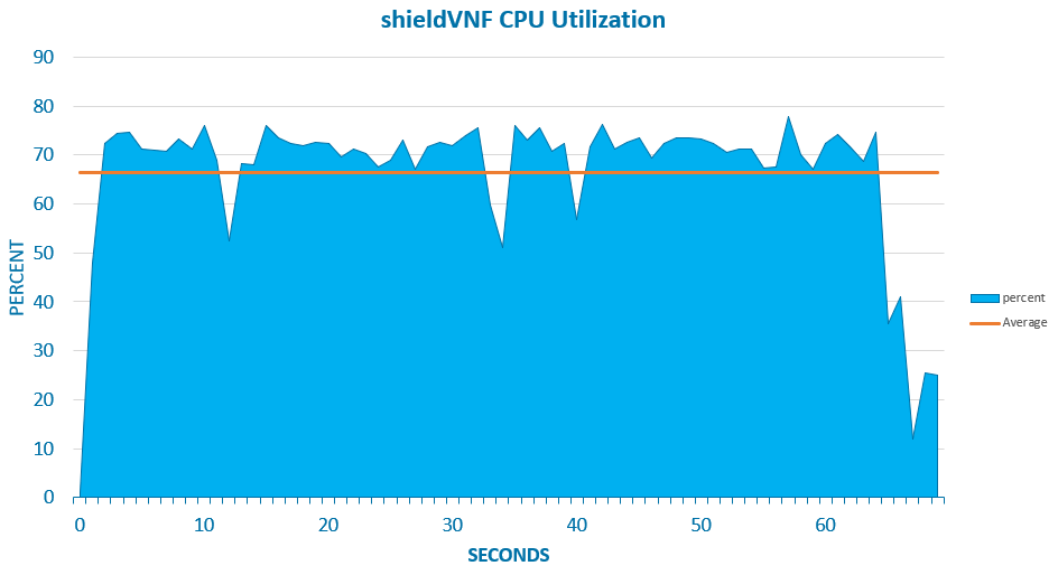


Fig. 4.23 **Experiment 1: Test2:** CPU utilisation of ShieldVNF.

4.5.2 Experiment 2: SIP enumeration attack

The second experiment is to answer the second research question, i.e., How can an adversary be detected and mitigated from enumerating valid SIP accounts on a SIP server? If ShieldVNF is working as designed, then the adversaries will only find some potential accounts.

Procedures

This experiment performed two tests: one with a legacy switch and the other with ShieldVNF running on a programmable switch. The objective of this test is for the attacker to discover valid SIP accounts that exist on the SIP server. The attacker achieves this by enumerating possible SIP accounts and looking for responses from a valid account, as depicted in Figure 4.24.

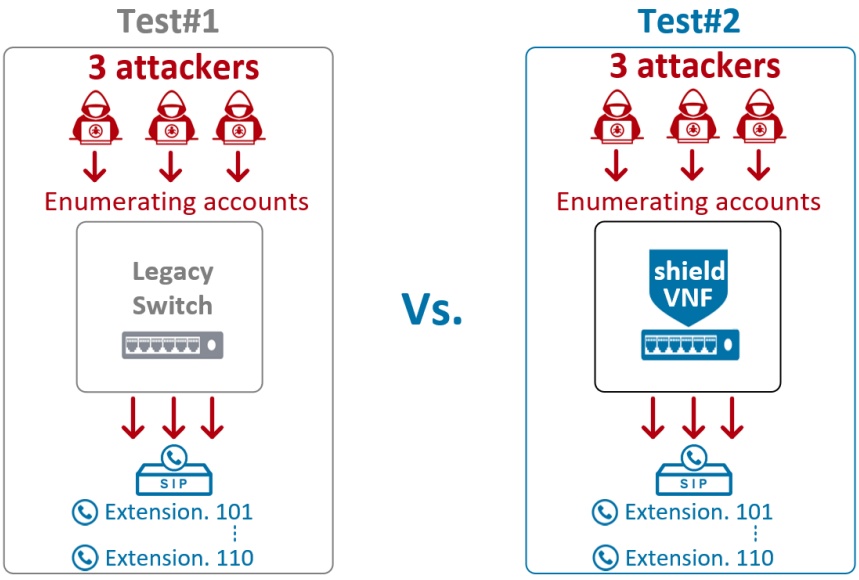


Fig. 4.24 **Experiment 2:** Attackers scanning for valid accounts/extension on a SIP Proxy server (Test1 with legacy switch vs. Test2 with ShieldVNF)

According to RFC3261, when a SIP Proxy receives a SIP `INVITE` request for an extension that does not exist, the SIP Proxy is supposed to respond with a `"404 Not Found"` packet. On the other hand, when the client sends the invite to a valid extension, it is supposed to respond with a `"401 Unauthorized"` packet. A hacker sends invite packets to a range of extensions, e.g., 101, 102, 103, and takes note of the response (404 vs. 401).

For each test, there were three attackers and one SIP server with ten valid extensions (101-110). The test script randomly generated IP addresses for the attackers and SIP server at run time in order to remove bias and to ensure that the solution did not depend on specific IP addresses.

A python script launched a mininet environment, the attackers, and a SIP server. The process started with attacker#1 using the `SIPVicious_svwar` tool to scan the SIP server. The command used to launch this attack was `"sipvicious_svwar -m INVITE x.x.x.x"`, where x.x.x.x was the randomly generated IP address for the SIP Proxy server. When attacker#1 completed this process, attacker#2 started the scanning process. Finally, attacker#3 performed the same scan against the SIP proxy server.

ShieldVNF, restricted any given port to a maximum of three INVITE packets.

Result

Table 4.7 **Experiment 2: Result:** ShieldVNF blocks SIP enumeration attack (3 discovered vs. 10)

Experiment 2: SIP Enumeration Attack	Legacy Switch		**ShieldVNF**	
	Number of accounts available	Number of accounts discovered	Number of accounts available	Number of accounts discovered
Attacker #1	10	10	10	3
Attacker #2	10	10	10	3
Attacker #3	10	10	10	3

Below is a screenshot from the result of the two tests. Figure 4.25 shows the outcome with legacy switch, while Figure 4.27 shows the outcome with ShieldVNF.

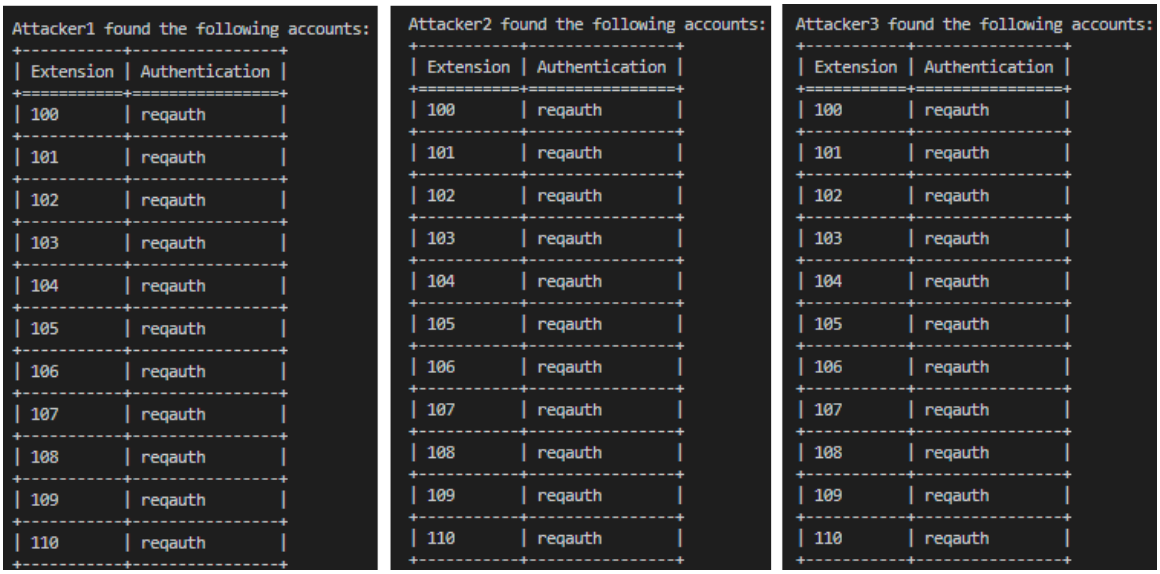


Fig. 4.25 **Experiment 2: Test1 Result:** SIP Enumeration attack with legacy switch. Attacker1 (left), Attacker2(middle), Attacker3(right) found 11 accounts.

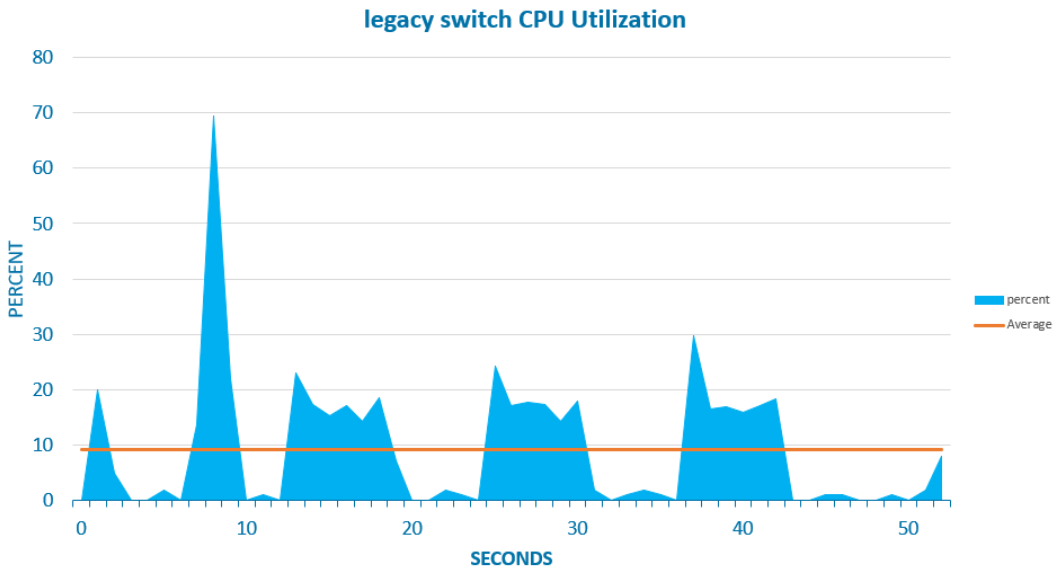


Fig. 4.26 **Experiment 2: Test1:** CPU utilisation of legacy switch.

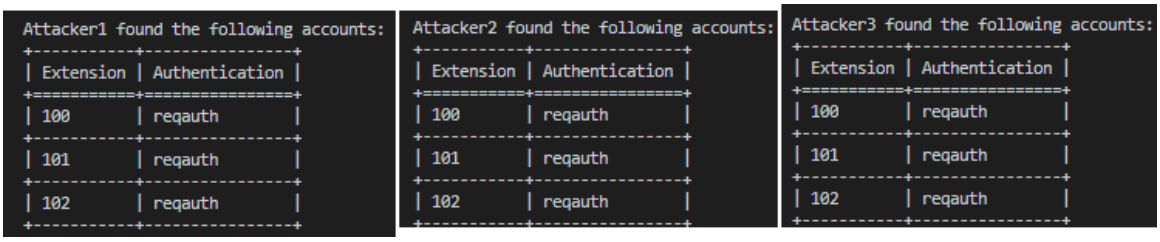


Fig. 4.27 **Experiment 2: Test 2 Result:** SIP Enumeration attack with ShieldVNF. Attacker1 (left), Attacker2(middle), Attacker3(right) found 3 accounts.

Discussion

With legacy switch, the attackers were able to find extension 100-110 that were on the SIP Proxy server (Figure 4.25), whereas the attacker only found three extensions (100-102) with ShieldVNF (Figure 4.24). In this instance, ShieldVNF detected that `INVITE` packets were received repeatedly from 1 port. This condition hit a threshold limit of three unacknowledged packets and therefore dropped the remaining packets. As a result, the attacker only found the first three accounts.

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

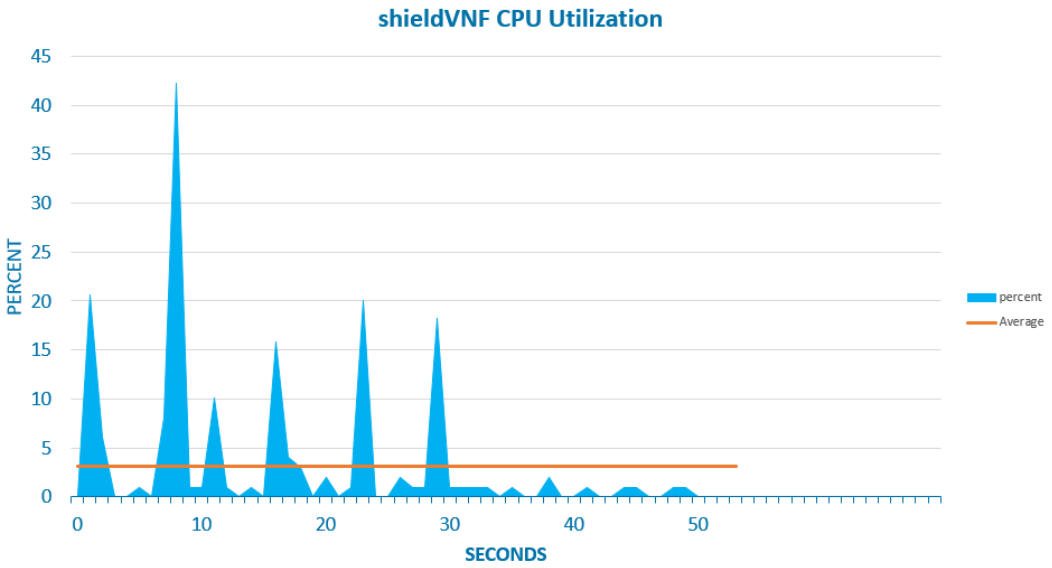


Fig. 4.28 **Experiment 2: Test2:** CPU utilisation of ShieldVNF.

4.5.3 Experiment 3: SIP brute force attack

The third experiment is to answer the third research question, i.e., How can an adversary be detected and mitigated from brute-forcing the passwords of SIP accounts? If ShieldVNF is working as designed, then the adversaries will not crack any passwords.

Procedures

Once the attacker has discovered an account, the next step is to brute force the account to gain entry. Similar to the previous experiment, this experiment performed two tests: legacy switch and ShieldVNF. The objective of this test is to brute force and discover the password for this account. The attacker achieved this by using `SIPVicious_svcrack` tool with a dictionary file that contains a list of popular passwords, as depicted in Figure 4.29.

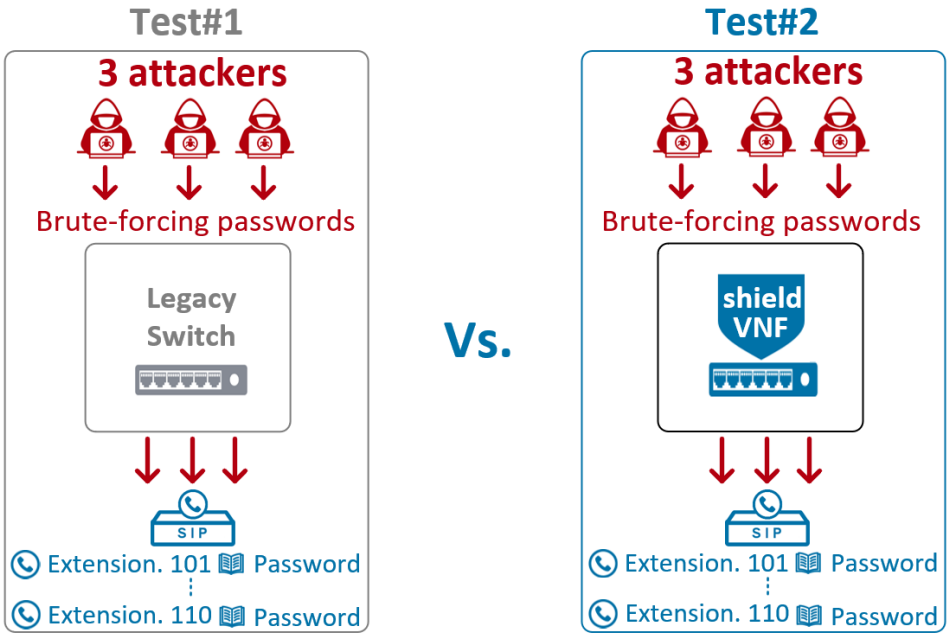


Fig. 4.29 **Experiment 3:** Attackers brute forcing the extensions on a SIP server for valid password (Test1 with legacy switch vs. Test2 with ShieldVNF)

In order for the attacker to brute force a SIP extension, the attacker would repeatedly send `REGISTER` packets with the extension and password. When the password is not correct, the SIP Proxy server would respond by sending a `401 Unauthorized` packet. The attacker would repeat this process by trying to register again with a new password from the dictionary file until it is exhausted. When the password is correct, the SIP Proxy server would respond by sending a `200 OK` packet.

For each test, there were three attackers and one SIP server with ten valid extensions (101-110). Following the same pattern, IP addresses for attackers and the SIP server were randomly generated and assigned at run time.

A python script launched a mininet environment, the attackers, and a SIP server. The process started with attacker#1 using `SIPVicious_svcrack` tool to brute force the extensions on the SIP server. The command used to launch this attack was `"sipvicious_svcrack -u x -d dictionary.txt y.y.y.y"`, where `x` was the extension (101-110) and `y.y.y.y` was the IP address of the SIP server. When attacker#1 has completed this process, attacker#2 started the brute force process. Finally, attacker#3 performed the same attack against the SIP proxy server.

The dictionary file used in this experiment was the same dictionary file that is commonly used by the attackers. The dictionary file is the same as the one used by a real password cracker tool called John the Ripper (JtR) that contains 3107 commonly used passwords.

Result

Table 4.8 **Experiment 3: Result:** ShieldVNF blocks SIP brute force attack (0 cracked vs. 10)

Experiment 3: SIP Brute force Attack	Legacy Switch		**ShieldVNF**	
	Number of accounts with password	Number of password cracked	Number of accounts with password	Number of password cracked
Attacker #1	10	10	10	0
Attacker #2	10	10	10	0
Attacker #3	10	10	10	0

Below is a screenshot from the result of the two tests. Figure 4.30 shows the outcome with legacy switch, while Figure 4.32 shows the outcome with ShieldVNF.

Discussion

With a legacy switch, the attackers were able to crack the password for extension 101-110 on the SIP proxy server (Figure 4.30), whereas the attacker did not crack any password with ShieldVNF (Figure 4.29). In this test, ShieldVNF detected repeated registration attempts coming from a port.

ShieldVNF restricts multiple SIP registration attempts to three. When this threshold is exceeded, ShieldVNF drops the subsequent packets. Since the attacker was not successful in

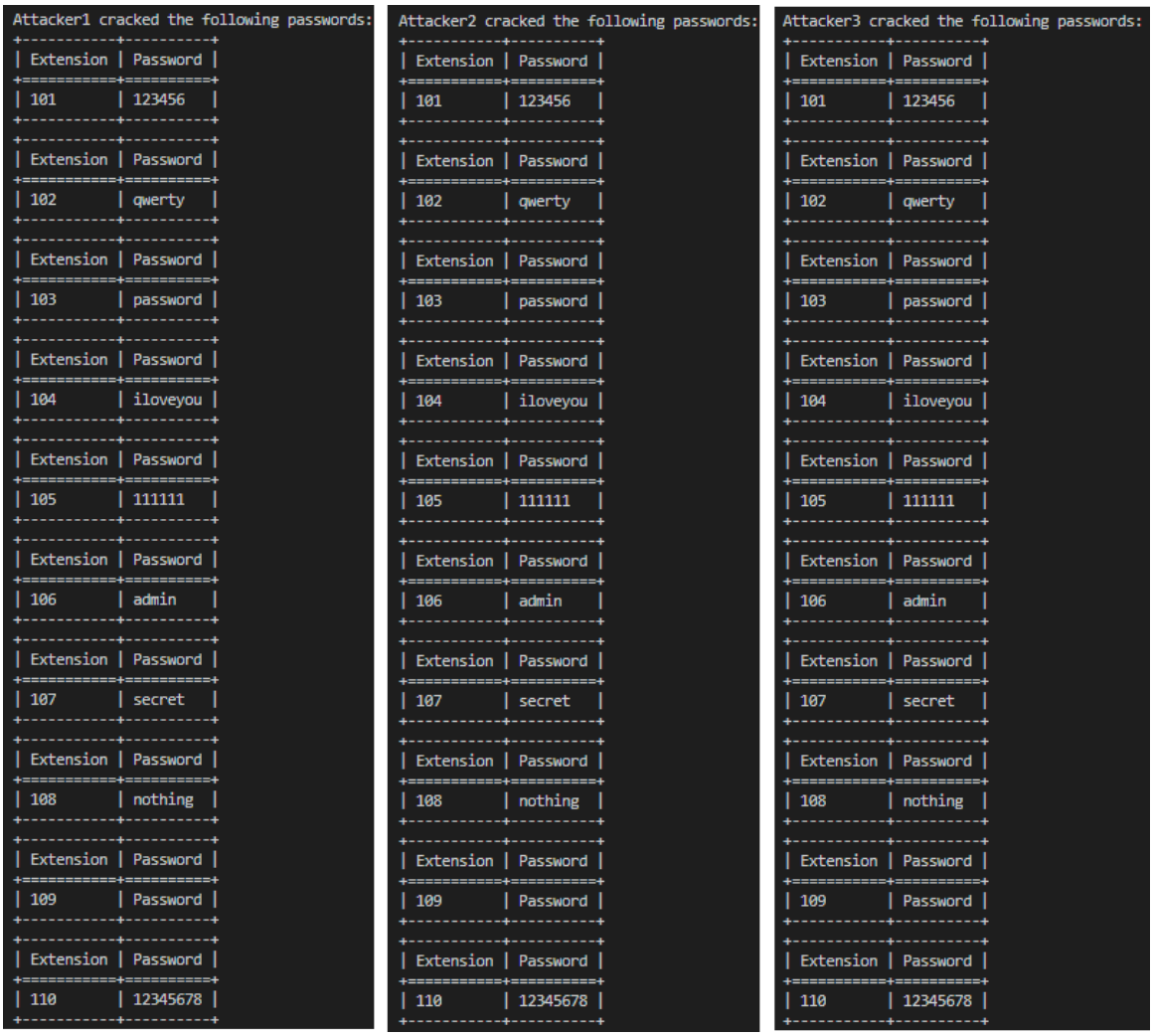


Fig. 4.30 **Experiment 3: Test1 Result:** SIP brute force attack with legacy switch. Attacker1 (left), Attacker2(middle), Attacker3(right) cracked 10 passwords.

their first three attempts in guessing the password, they did not find anything. This outcome presents evidence that ShieldVNF was able to detect and mitigate SIP brute force attacks.

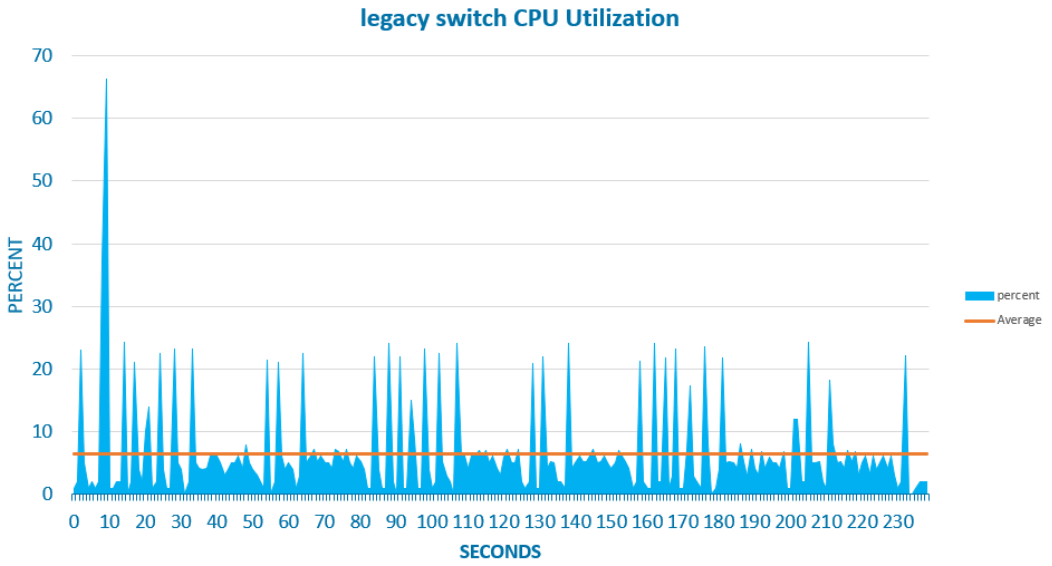


Fig. 4.31 **Experiment 3: Test1:** CPU utilisation of legacy switch.

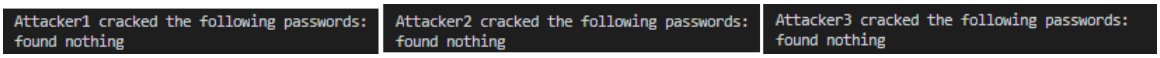


Fig. 4.32 **Experiment 3: Test2 Result:** SIP brute force attack with ShieldVNF. Attacker1 (left), Attacker2(middle), Attacker3(right) found nothing.

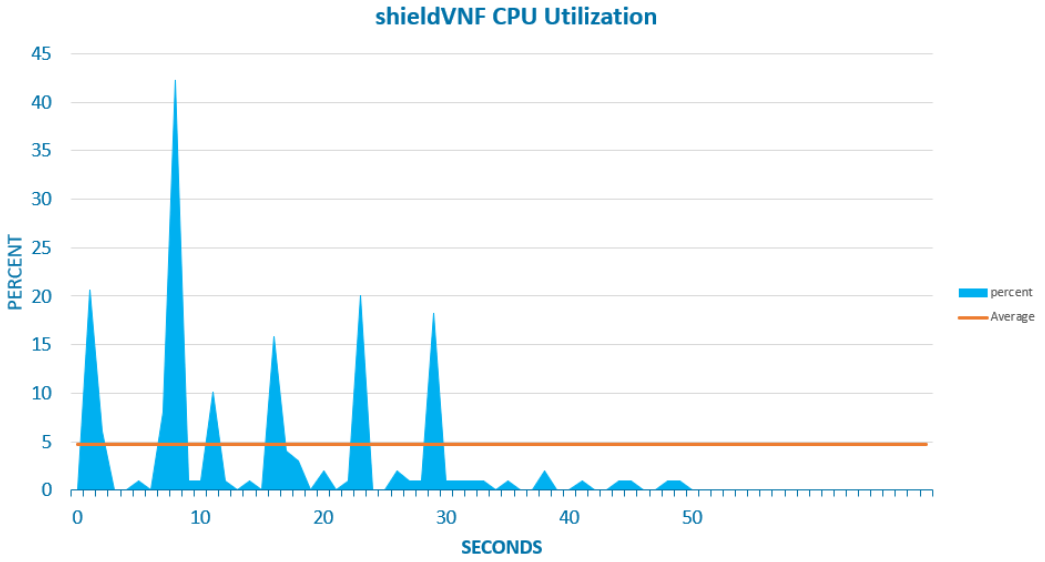


Fig. 4.33 **Experiment 3: Test2:** CPU utilisation of ShieldVNF.

4.5.4 Experiment 4: Command-and-Control Channel Establishment

The fourth experiment is to answer the fourth research question, i.e., How can a botnet be prevented from registering with its CNC server? If ShieldVNF is working as designed, then the botnet will not be able to register to its CNC server.

Procedures

Similar to previous experiments, this experiment performed two tests to contrast the different outcomes when using a legacy switch vs. ShieldVNF. The objective of this test is to block bot communication with its central command and control (CNC) server, as depicted in Figure 4.34. This step is important to prevent the botmaster from being able to control the bot and, therefore, render the bot to be non-functional for the intended purpose of launching an attack. Since the bot is not able to connect to the CNC server and receive commands, it is effectively neutralised and does not pose an immediate threat.

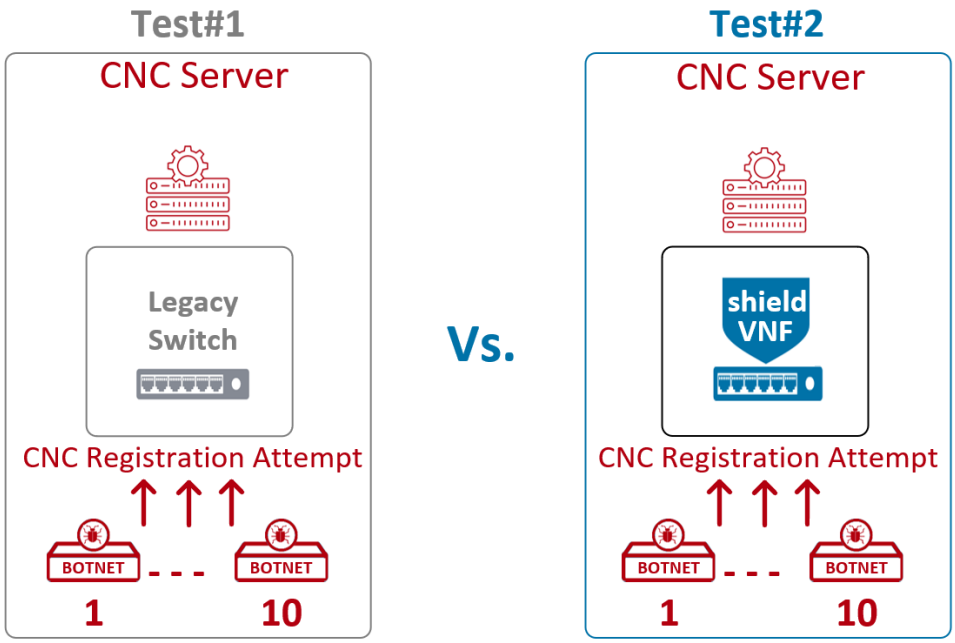


Fig. 4.34 **Experiment 4:** IoT-turned-to-botnet attempting to register to the centralised Command and Control (CNC) server (Test1 with legacy switch vs. Test2 with ShieldVNF)

In this experiment, we will be looking at the way Mirai botnet and its CNC operates so that we test the hypothesis that ShieldVNF can detect and mitigate CNC channel establishment activities. When Mirai has successfully infected an IoT device, the new bot will attempt to connect to the CNC server using Telnet (TCP/23) with a specific payload. The first packet will contain `\x00000001` and the second packet will contain `\x00`. At this stage, the bot

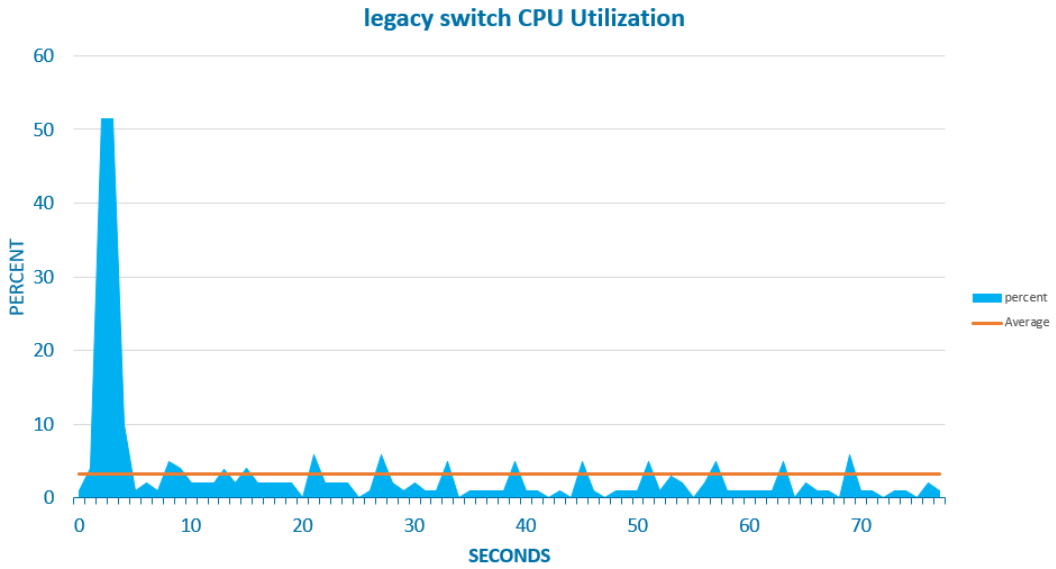


Fig. 4.35 **Experiment 4: Test1:** CPU utilisation of legacy switch.

has registered with the CNC. From this point on, the bot and CNC will maintain a regular heartbeat every 60 seconds with the payload content of `\x0000`.

A python script will launch a mininet environment that consists of 1 host as the CNC server and ten hosts as the Mirai botnet. Similar to previous experiments, IP addresses and role assignments were generated at run time to remove bias and ensure that the solution works for any IP addresses. Instead of using a real Mirai botnet binary, CNC and Mirai simulators were used to simulate the communication between botnet and CNC. The simulation matched the real pattern at the packet level, i.e., if we were to look at the packets exchanged between these two entities, it would be hard to differentiate between packets generated by simulator vs. the real Mirai binary.

Result

Table 4.9 **Experiment 4: Result:** ShieldVNF blocks CNC Establishment attempt (0 registered vs. 10)

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

Below is a screenshot from the result of the two tests. Figure 4.36 shows the outcome with a legacy switch, while Figure 4.38 shows the outcome with ShieldVNF.

Discussion

With a legacy switch, all bots were able to connect to the CNC server, as well as the ongoing heartbeat. Figure 4.36 shows the perspective from the CNC-side that counts how many bots were successfully registered, and the number of heartbeat packets received from each bot. With ShieldVNF, the bots were not able to connect to CNC, and therefore Figure 4.38 shows that none of the bots were registered.

With the bots not able to register, the CNC would not be able to send a command for the bots to launch a DoS attack. This outcome presents evidence that ShieldVNF was able to detect and prevent bots registration to the central CNC server.

```
CNC received report from the following bots:

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

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

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

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

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

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

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

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

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

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

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

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

Fig. 4.36 **Experiment 4: Test1 Result:** Bot registration attempt with legacy switch. 10 Bots registered at the Command-and-Control (CNC) server.

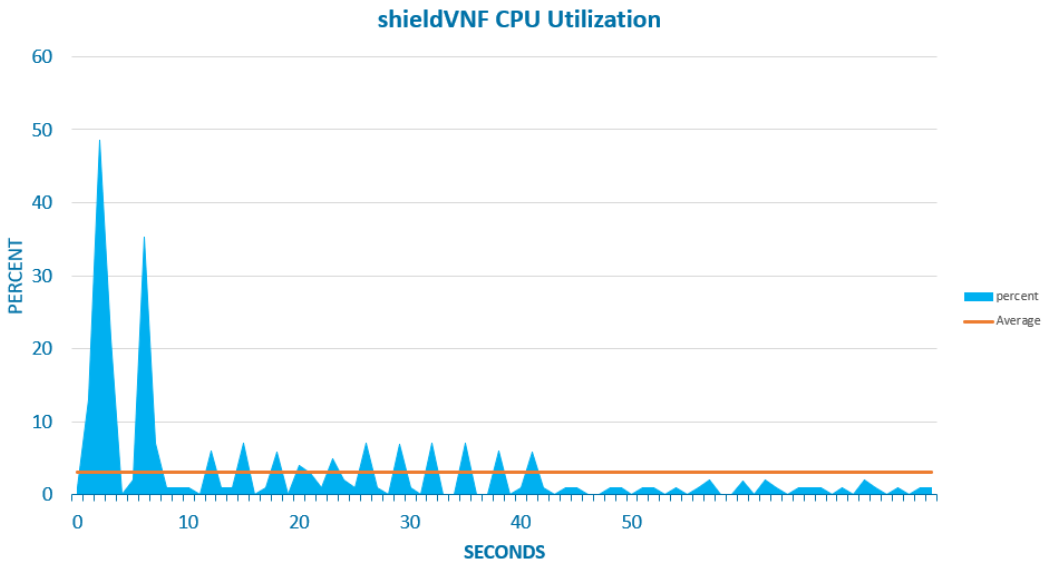


Fig. 4.37 **Experiment 4: Test2:** CPU utilisation of ShieldVNF.

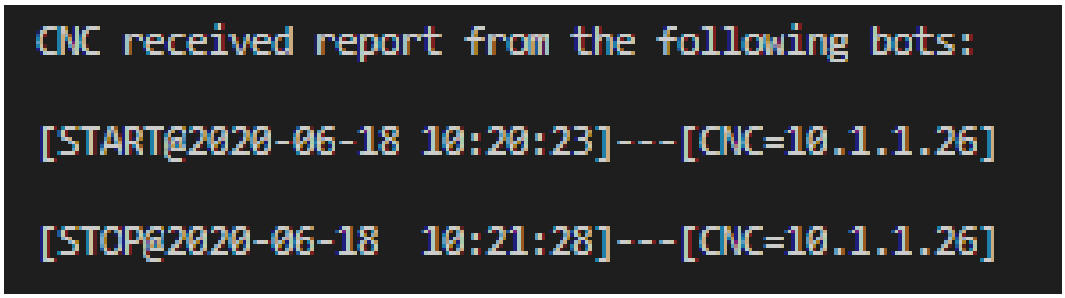


Fig. 4.38 **Experiment 4: Test2 Result:** Bot registration attempt with ShieldVNF. 0 Bot registered at the Command-and-Control (CNC) server.

4.5.5 Experiment 5: SIP DoS attack

The fifth experiment is to answer the fifth research question, i.e., How can SIP DoS attacks from IoT botnets be detected and mitigated? If ShieldVNF is working as designed, then a legitimate user is still able to make SIP calls and malicious packets are dropped.

Procedures

Assuming that the attacker was able to get through the previous experiments, at this stage, the attacker has successfully recruited the IoT device to be part of the botnet and ready to launch a SIP DoS attack. The hypothesis for this experiment is that, ShieldVNF can detect and mitigate SIP DoS attack from a single attacker, as shown in Figure 4.39.

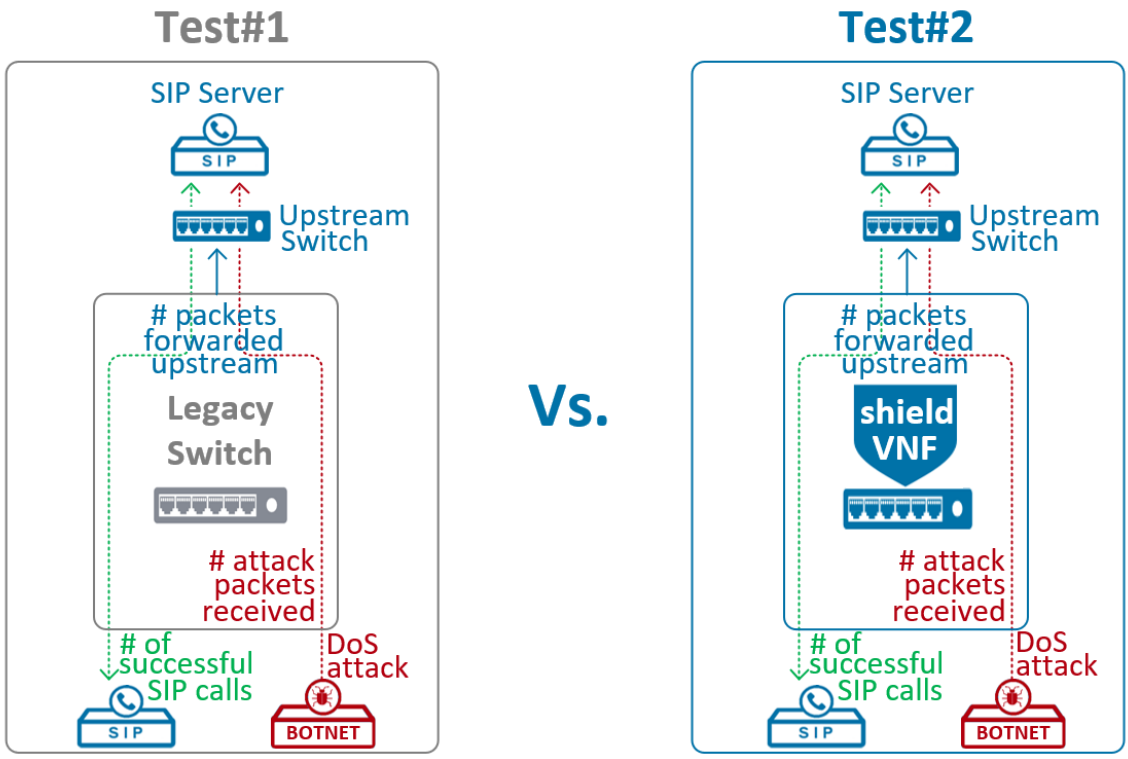


Fig. 4.39 **Experiment 5:** SIP DoS attack from a single attacker (Test1 with legacy switch vs. Test2 with ShieldVNF)

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

The test environment had four hosts: SIP-client1, SIP-client2, PBX, and the Bot. SIP-client1 represents a legitimate user that is calling SIP-client2 via the corporate phone system (PBX). A Bot is on the same virtual-switch as SIP-client1. The process started with SIP-client1 making a call at a rate of 1 call per second to SIP-client2. At the 20th second after SIP-client1 started making calls, the bot started a SIP DoS attack against the PBX.

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

Result

Table 4.10 **Experiment 5: Result:** ShieldVNF blocks SIP DoS attack (60 successful calls vs. 20)

Experiment 5: SIP DoS Attack	**Legacy Switch**			**ShieldVNF**		
	Number of attack packets received	Number of packets forwarded upstream	Number of successful calls	Number of attack packets received	Number of packets forwarded upstream	Number of successful calls
Caller	NA	NA	**20**	NA	NA	**60**
Attacker #1	4,000,182	**4,000,120**	NA	4,000,073	**228**	NA

Below is a screenshot from the result of the two tests. Figure 4.40 and 4.41 shows the outcome with the legacy switch, while Figure 4.43 and 4.45 shows the outcome with ShieldVNF.

Discussion

With a legacy switch, there were two important observations to make: the number of successful calls from the perspective of SIP-client1 and the number of packets that were forwarded upstream. As depicted in Figure 4.40, there were only 20 successful calls, which means that SIP-client1 was able to make calls for the first 20 seconds of the test. Once the attack started at the 20th second, it disrupted legitimate calls, and therefore the results show only 20 calls completed.

With regards to the number of packets forwarded upstream (via NAT interface), Figure 4.41 shows that the legacy switch forwarded all the attack packets upstream since it was not

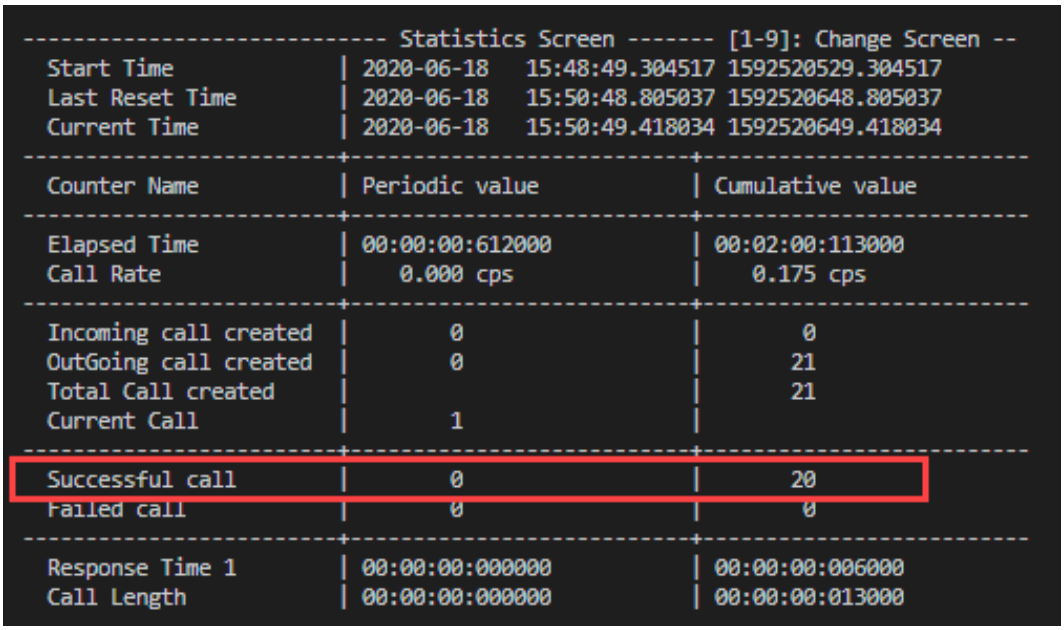


Fig. 4.40 **Experiment 5: Test1 Result:** SIP DoS attack to a SIP Server from 1 bot with legacy switch. Calls were disrupted when the attack started. 20 successful calls completion.

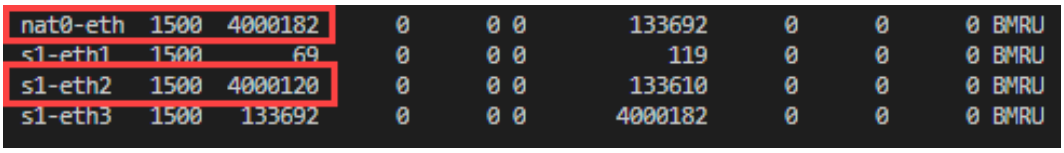


Fig. 4.41 **Experiment 5: Test1 Result:** SIP DoS attack to a SIP Server from 1 bot with legacy switch. Attack packets received from `s1-eth2` interface were forwarded upstream to `nat0-eth` interface.

aware that these packets were malicious. The attacker was connected to interface s1-eth2 and sent 4000120 packets, and the legacy switch forwarded around the same number upstream (4000182).

In contrast with ShieldVNF, Figure 4.43 shows 60 successful calls, which means that the test calls were not interrupted. The test calls were at one call/second rate. With 60 successful calls, it shows that the SIP DoS attack that started at the 20th second did not disrupt the SIP service because the attack was detected and mitigated by ShieldVNF.

Other evidence and data points are from the number of packets that got forwarded upstream. With ShieldVNF, the attacker sent 4000073 packets, but yet only 228 packets were forwarded upstream, which means that ShieldVNF successfully dropped most of those malicious packets.

This outcome presents evidence that ShieldVNF was able to keep legitimate calls up and running while successfully detect and mitigate the SIP DoS attack from 1 attacker.

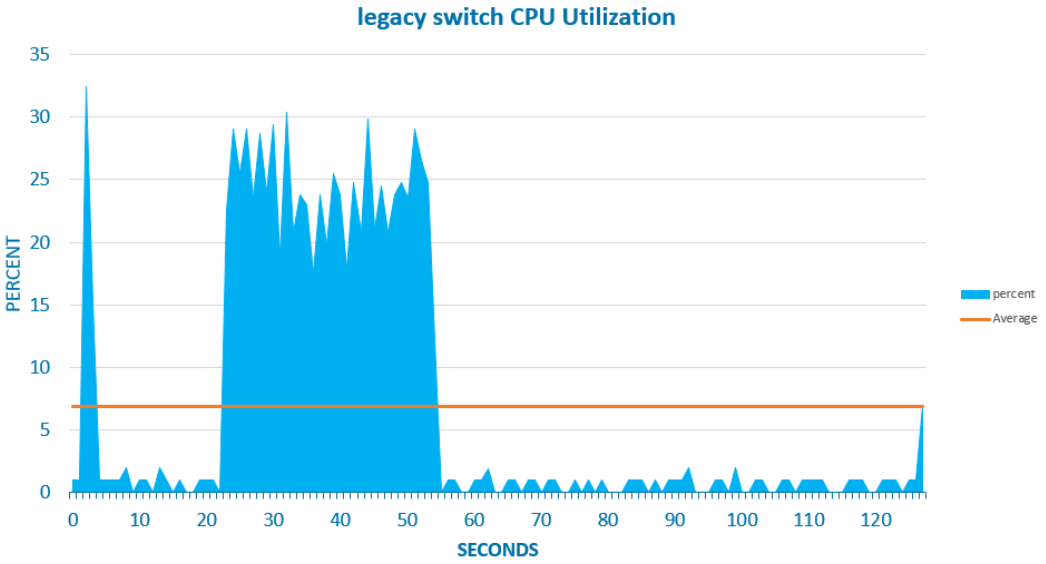


Fig. 4.42 **Experiment 5: Test1:** CPU utilisation of legacy switch.

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

Fig. 4.43 **Experiment 5: Test2 Result:** SIP DoS attack to a SIP Server from 1 bot with ShieldVNF. All calls were successful. 60 successful call completion.

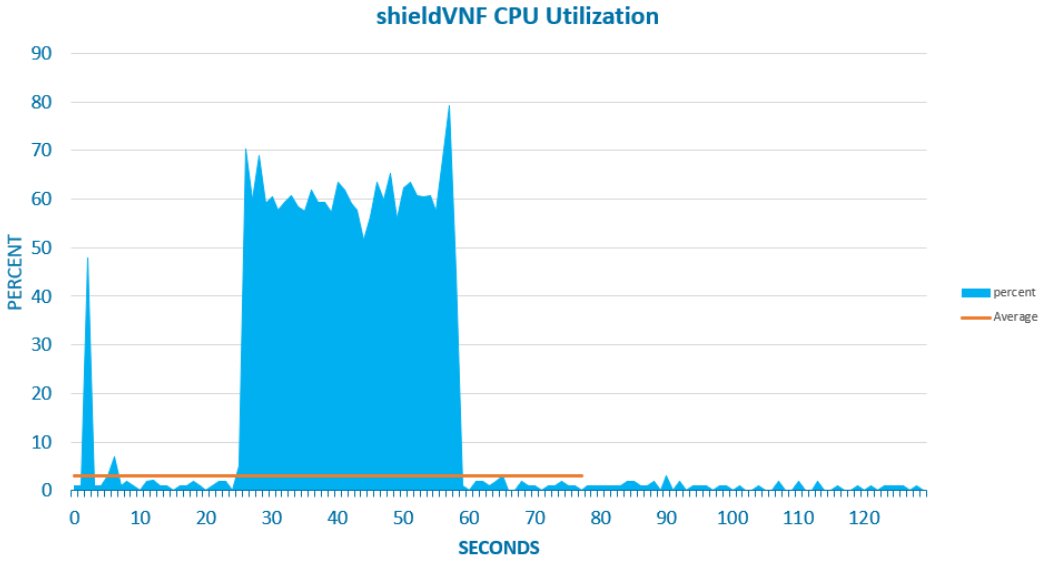


Fig. 4.44 **Experiment 5: Test2:** CPU utilisation of ShieldVNF.

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

Fig. 4.45 **Experiment 5: Test2 Result:** SIP DoS attack to a SIP Server from 1 bot with ShieldVNF. Most of attack packets were dropped and not forwarded upstream.

4.5.6 Experiment 6: SIP Distributed DoS attack

The sixth experiment was to answer the sixth research question, i.e., How can SIP Distributed DoS attacks from IoT botnets be detected and mitigated? If ShieldVNF is working as designed, then a legitimate user is still able to make SIP calls and malicious packets are dropped.

Procedures

This experiment presents a case where the majority of the IoT devices were infected and recruited by a botnet. The hypothesis for this experiment is that ShieldVNF can detect and mitigate a SIP Distributed DoS attack from one hundred attackers, as shown in Figure 4.46.

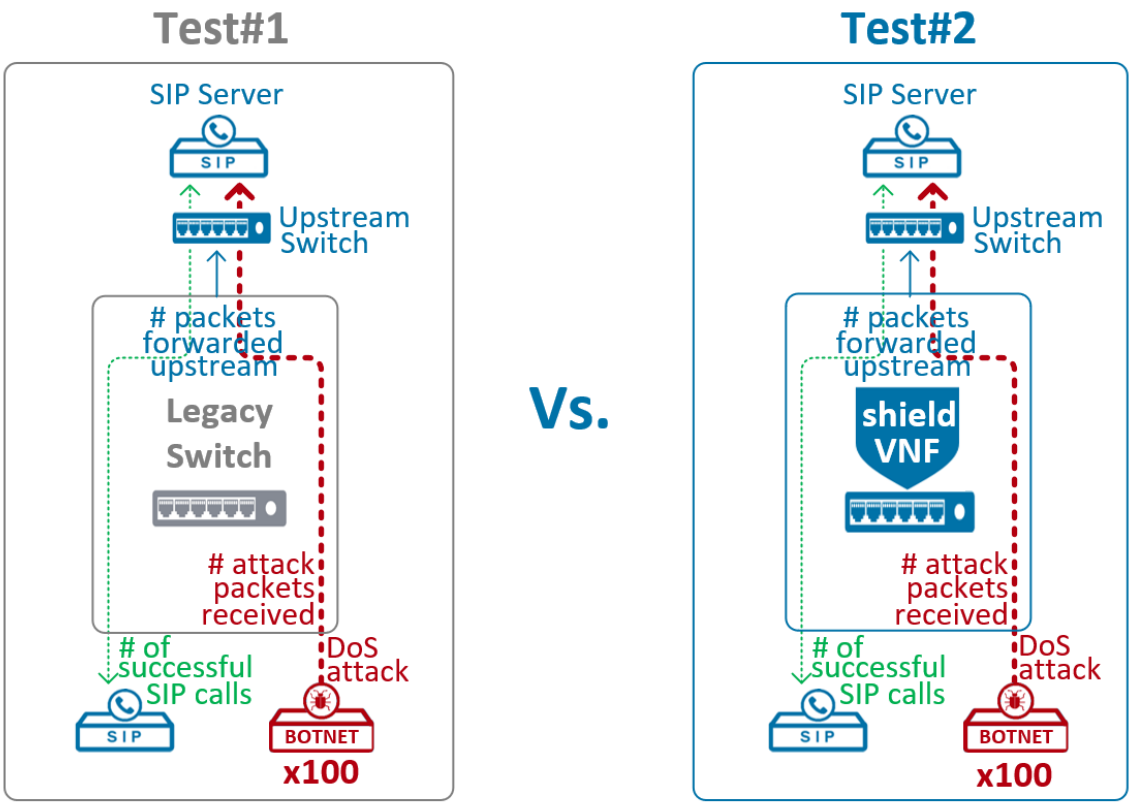


Fig. 4.46 **Experiment 6:** SIP Distributed DoS attack from 100 attackers (Test1 with legacy switch vs. Test2 with ShieldVNF)

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

The test environment consisted of 103 hosts: SIP-client1, SIP-client2, PBX, and 100 bots. This experiment performed two tests, one with legacy and the other with ShieldVNF. The test script randomly generated the IP address of SIP-Client1 and 100 bots at run time. The same attack tool, inviteflood, was used to send `INVITE` packets.

Result

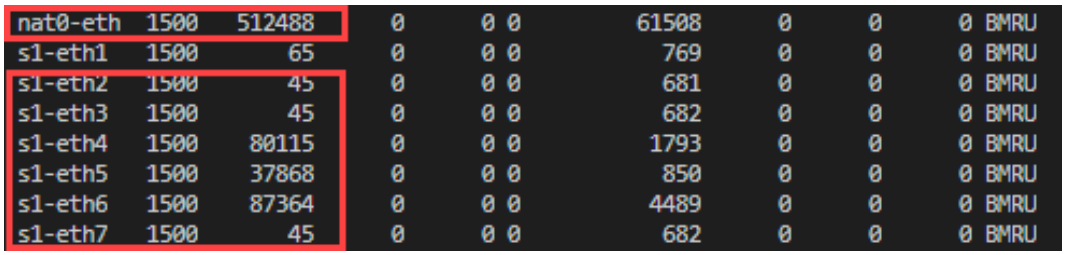
Table 4.11 **Experiment 6: Result:** ShieldVNF blocks SIP Distributed DoS attack (59 successful calls vs. 20)

Experiment 6: SIP Distributed DoS Attack	**Legacy Switch**			**ShieldVNF**		
	Number of attack packets received	Number of packets forwarded upstream	Number of successful calls	Number of attack packets received	Number of packets forwarded upstream	Number of successful calls
Caller	NA	NA	**20**	NA	NA	**59**
100 Attackers		**512,488**	NA	4,000,073	**3061**	NA

Below is a screenshot from the result of the two tests. Figure 4.47 and Figure 4.48 shows the outcome with legacy switch. Figure 4.49 and Figure 4.51 shows the result with ShieldVNF.

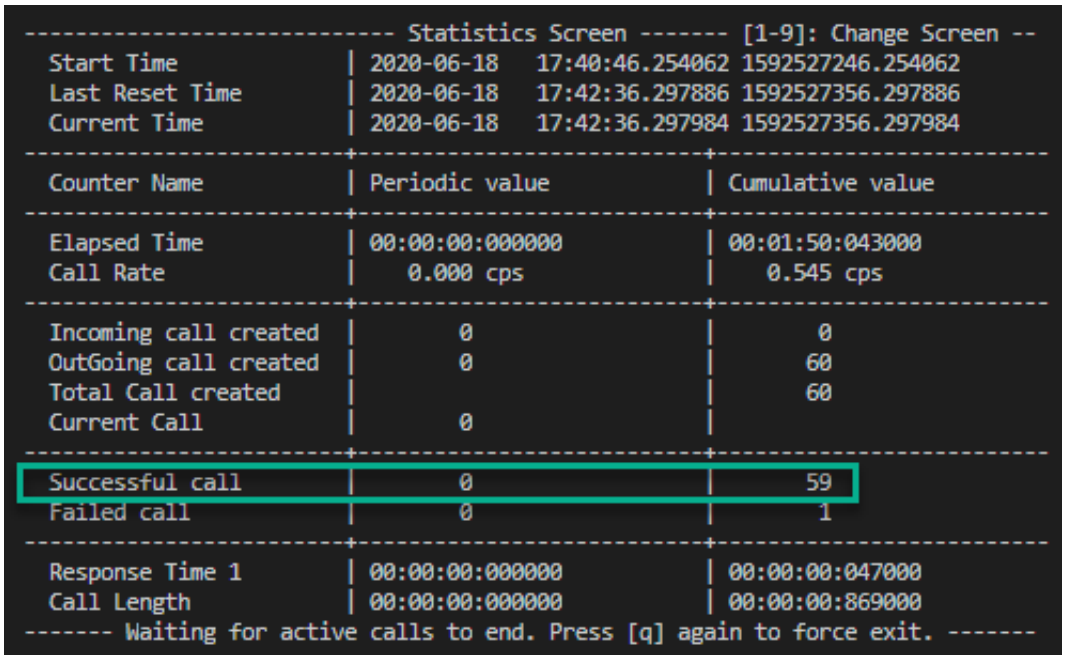
```
------------------------- Statistics Screen ------- [1-9]: Change Screen --
  Start Time              | 2020-06-18   17:15:46.052212 1592525746.052212
  Last Reset Time         | 2020-06-18   17:17:46.669703 1592525866.669703
  Current Time            | 2020-06-18   17:17:46.734138 1592525866.734138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

Fig. 4.47 **Experiment 6: Test1 Result:** SIP DoS attack to a SIP Server from 100 bots with legacy switch. Calls were disrupted when the attack started. 20 successful calls completion.



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

Fig. 4.48 **Experiment 6: Test1 Result:** SIP DoS attack to a SIP Server from 100 bots with legacy switch. Most attack packets from botnets were forwarded upstream to `nat0-eth` interface.



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

Fig. 4.49 [**Experiment 6: Test2 Result:** SIP DoS attack to a SIP Server from 100 bots with ShieldVNF. 59 calls were successful.

Discussion

Similar to the previous experiment with a single bot attacker, in this experiment we were also interested in two dependent variables: the number of successful calls and the number of packets that were forwarded upstream. As depicted in Figure 4.47, SIP-client1 was only able to make 20 successful calls. The timing coincided with the time when the 100 bots started attacking the PBX. Figure 4.51 shows that the legacy switch forwarded the malicious packets upstream.

With ShieldVNF, we had 59 successful calls out of 60 attempted calls (Figure 4.49), and the number of the forwarded packet was 3061 (Figure 4.51) which means that ShieldVNF

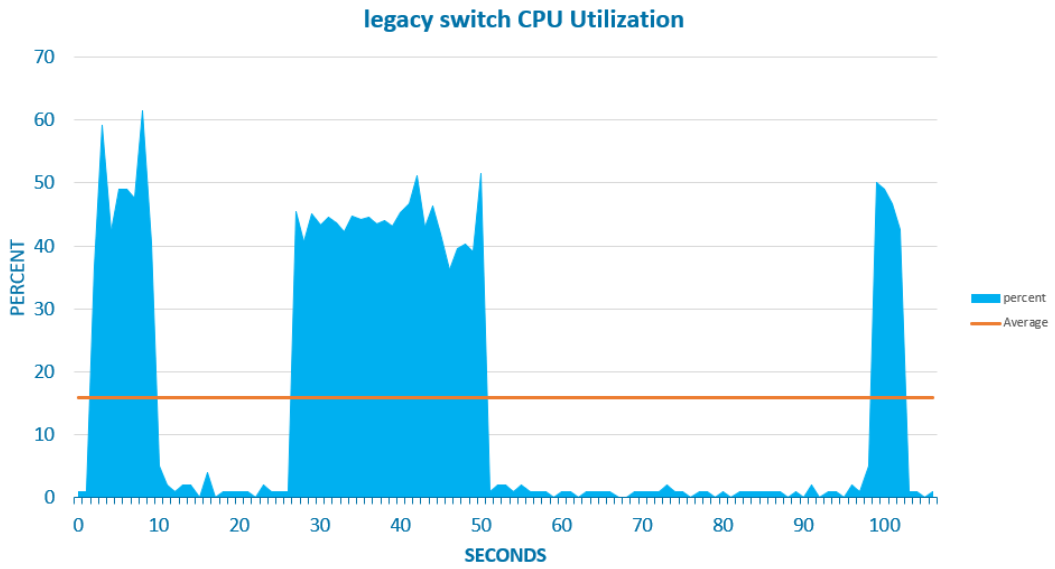


Fig. 4.50 **Experiment 6: Test1:** CPU utilisation of legacy switch.

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

Fig. 4.51 **Experiment 6: Test2 Result:** SIP DoS attack to a SIP Server from 100 bots with ShieldVNF. Most of the attack packets from botnets were dropped and not forwarded upstream.

detected and dropped the malicious packets. This data point supports the hypothesis that ShieldVNF is able to detect and mitigate SIP Distributed DoS attacks.

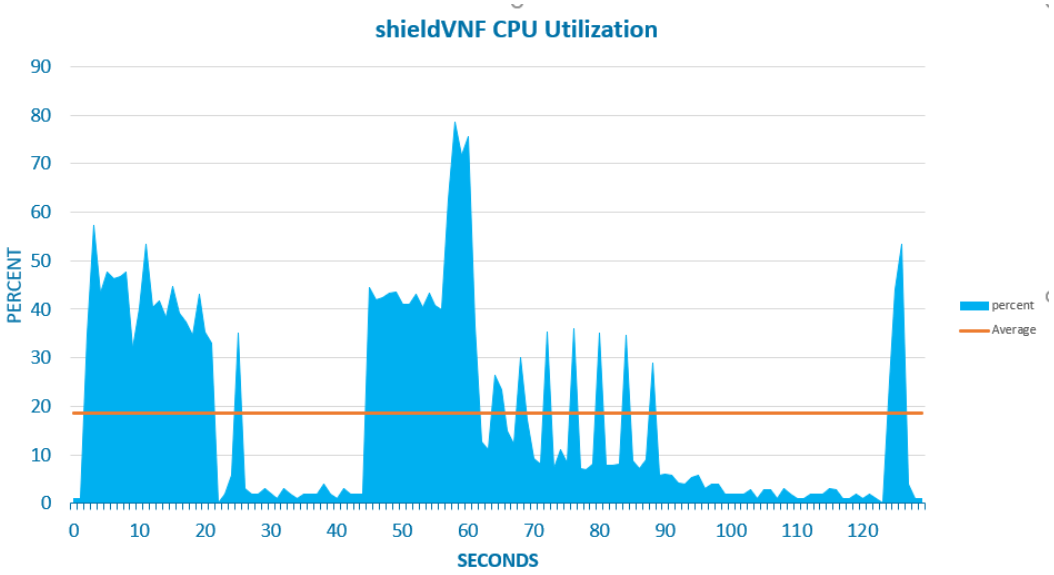


Fig. 4.52 **Experiment 6: Test2:** CPU utilisation of ShieldVNF.

4.6 Chapter Summary

From these experiments, ShieldVNF demonstrated that it is effective in providing the first line of defence for proactive and reactive measures. The proactive measures cover cases that prevent the IoT devices from being recruited by a botnet, such as blocking a SIP scanning attack, account enumeration attack, and brute force attack. The reactive measures provide coverage for cases where the attacker has recruited the IoT devices and turned them into botnets. In this chapter, we looked at mitigating CNC communication, SIP DoS attack from a single bot, and Distributed SIP DoS attacks from 100 bots.

The data from the experiment supports the hypothesis that ShieldVNF can detect and mitigate a Distributed SIP DoS attack from a botnet. ShieldVNF followed defence-in-depth principles that presented a new challenge every time the attacker proceeded to the next step in his/her attack journey. Even in the worst-case scenario where the attacker succeeded in recruiting and launching an attack from 100 bots, ShieldVNF was able to neutralise this attack, and the data showed that the Distributed SIP DoS attack did not disrupt the SIP calls that were running on the same switch.

Besides providing defence on one switch, ShieldVNF also provided attack intelligence so that other switches could prepare against the same attack. Considering the nature of botnet, where it cast a wide net for the attack, there is a good chance that other IoT devices in the corporate network would also experience the same attack. From these experiments, the follow on question is how to replicate this success to other switches in the enterprise effectively. Relying on a manual process for the network security personnel to intervene would take considerable time, especially when dealing with hundreds of switches installed in one organisation. We address that question in the next chapter.

Chapter 5

ShieldSDN: Scaling SIP DDoS Defence to Multiple Switches

5.1 Overview

In the previous chapter, it was shown that ShieldVNF mitigated the attack and produced an IOC about the attack, which will play a crucial role in this chapter. The objective of this chapter is to develop mechanisms to scale the defence from one switch to multiple switches in the organisation. To achieve this objective, we will investigate three sub-questions which will be validated by three experiments:

- What approaches can be used to collect the IOCs from one switch and install it as packet filtering rules on other switches?
- What strategies could be implemented to manage a limited amount of memory on ShieldVNF?
- What methods can be used to efficiently manage ShieldVNF memory when dealing with persistent threats?

In the SIPshield framework, it is the role of ShieldSDN to scale the defence beyond a single switch. The idea is that after a switch has successfully mitigated an attack, ShieldSDN collects the IOCs from the source switch and installs it on other switches as packet filtering rules. This approach enables other switches to drop malicious packets and therefore replicate the success that the source switch had. Figure 5.1 depicts the before and after of this process.

This chapter starts with ShieldSDN as the proposed solution, followed by the design of experiments, then the implementations of the experiment. The last section of this chap-

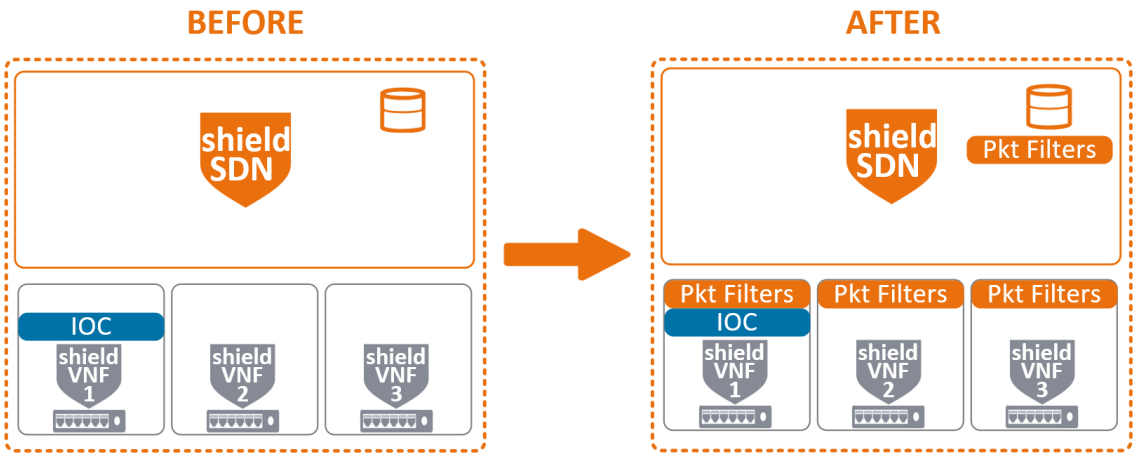


Fig. 5.1 ShieldSDN collects IOC from the source switch (ShieldVNF1) and installs it as packet filtering rules on other switches (ShieldVNF1, ShieldVNF2 & ShieldVNF3).

ter describes three experiments that include procedures, result, and discussion for each experiment.

5.2 The Proposed Solution: ShieldSDN

5.2.1 A Synchronised Defence Posture Approach

In the event of a botnet outbreak, the attack would typically leverage the same exploits on multiple devices in multiple locations. Therefore, it is important to have an automated and synchronised mechanisms where the organisation can contain the outbreak quickly before it spreads to other locations. An organisation would typically purchase IoT devices in bulk and install these devices in multiple locations. This deployment pattern implies that those devices would use the same firmware, and therefore, share the same vulnerabilities that can be exploited by a botnet. During an outbreak, the organisation needs to quickly address the vulnerability which can easily overwhelm the staff. Instead of doing this manually, automation is proposed so that the organisation can efficiently synchronise the defence posture for all switches in the organisation.

In addition to a synchronised defence approach, efficient memory management is required, considering the finite amount of memory that each switch has. Compared with a server, an Ethernet switch comes with limited amount of memory space. This limitation requires efficient use of memory so that the proposed solution can operate continuously. One way to conserve memory is by deleting packet filters that have expired so that new filter can be installed. However, when the same attackers reappear after the packet filter has been deleted,

ShieldSDN needs to recreate the filter. These repeat offenders are considered persistent threats and need to be dealt with differently.

ShieldSDN treats Advanced Persistent Threats (APT) with a different strategy that takes into account the frequency of the attacks. APT is often associated with well-organised attackers where they are being careful and strategic in the way they carry out their attacks. It is also persistent, and may lie dormant for a long time before it re-emerges and launches attacks. APT poses an interesting challenge considering that each switch has a limited amount of memory. Instead of using the default expiry value, ShieldSDN would assign an expiry time that is exponentially longer, based on the attack frequency.

5.2.2 ShieldSDN: SDN Controller for IOC Management

ShieldSDN is an SDN controller that is proposed as a solution to the research questions set in this chapter. ShieldSDN automates the task of synchronising defence posture among multiple switches in the organisation. It does this by automating the collection and distribution of IOC, deleting expired IOC, and managing the rules for persistent threats. Being an SDN controller, ShieldSDN is able to communicate with these switches to query, insert, and remove the rules on these switches. In this instance, ShieldSDN queries ShieldVNF for IOCs about the attacker (or the victim) that has been mitigated previously. Then, ShieldSDN will install these IOCs as packet filtering rules on other switches. In addition, ShieldSDN stores them in a centralized database for long-term storage and analytics. When all the switches have the same information, they have reached a synchronised state where each switch can then take action based on the same set of information.

There are three types of information that ShieldSDN synchronises among all switches: indicators about the known attacker, the known Command-and-Control (CNC) server, and the known victim that ShieldVNF has produced. With the known attacker information, 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 switch will drop the packet. With the known victim information, each switch will inspect the destination IP address. If the destination IP matches, this means that this packet is going to a DDoS victim. In this case, the switch will drop the packet so that it does not participate in the attack. Similarly, when the destination IP address matches with known CNC information, the switch will drop the packet to prevent the botnet from registering to its CNC server.

ShieldSDN conserves the switch's memory by expiring old rules from the switch and storing it in a centralized database. Considering the limited amount of memory that each switch has, ShieldSDN initially assigns a random expiry time between 5 to 10 minutes, so that the switch keeps only the rules for active attacks. The use of randomised expiry times is

meant to make it less predictable for the attacker. The initial 5 to 10 minutes expiry time was chosen for two reasons. First, the assumption is that most attacks are short-lived. Second, in case that it affected legitimate uses, they do not have to wait for too long before they can resume their legitimate activities. If the first assumption is wrong and the attacker keeps attacking, ShieldSDN will create an exponentially longer expiry time based on the attack frequency. At current implementation, ShieldVNF can keep 1024 records of each indicator (attacker, CNC, and victim). Since ShieldVNF 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 bogus entries in ShieldVNF which will drop legitimate packets.

Persistent threats that reappear after being deleted will get a longer expiry time based on the frequency of the attack. There might 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 recognises 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 power of 2 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.

5.3 Design of Experiments

5.3.1 Sequence and Workflow

The design for experiments in this section is around IOC management as described in the previous section. There are three experiments that we will look at in this section: collection and distribution of IOC, expiration of IOC, and handling IOC for persistent threats.

The first experiment is for ShieldSDN to collect IOC from the ShieldVNF1 and distribute it to ShieldVNF2 and ShieldVNF3. In this use case, ShieldSDN probes ShieldVNF1 to see if ShieldVNF1 has produced IOC based on the incidents it had. When ShieldVNF1 has produced IOCs, ShieldSDN retrieves it, distributes it to ShieldVNF2 and ShieldVNF3, then stores it in the SQL database. In this experiment, we can quantitatively measure the number

of IOCs from the source switch (ShieldVNF1) with the destination switch (ShieldVNF2 and ShieldVNF3) to validate that ShieldSDN has correctly performed its function. A positive result is achieved when we have a synchronised state where ShieldVNF1, ShieldVNF2, and ShieldVNF3 have the same number of packet filtering rules.

The second experiment is about managing and deleting expired IOCs. In this experiment, ShieldSDN queries the database for expired IOCs and deletes them from all the switches. When ShieldSDN first inserted these IOCs into the database, each IOC record had a random expiry time between 5 to 10 minutes. When ShieldSDN queries the database, it retrieves the IOCs that have already expired. When it finds these IOCs, ShieldSDN connects to all ShieldVNFs and deletes them. For validation, we will quantify the number of expired IOCs that exist on ShieldVNF. A positive result is achieved when ShieldVNF1, ShieldVNF2, and ShieldVNF3 no longer have the expired rules.

The last experiment in this chapter is dealing with persistent threats. In this experiment, we will pay close attention to two variables: frequency of the offence and the IOC's expiry time for these offence. The idea is to assign an expiry time that is exponentially longer, based on the frequency of the offence. A positive result is achieved when the repeat offenders are assigned with an expiry time that is exponentially longer than the previous offence.

5.3.2 Independent and Dependent Variables

Each experiment has a set of independent and dependent variables that are captured in Table 6.2. These variables provide data points for analysis and establish causality.

Table 5.1 Independent and Dependent Variables by Experiments

Experiment	Independent Variable	Dependent Variables
1. IOC collection, distribution, & storage	ShieldSDN	# of entries in KnownAttacker table # of entries in KnownVictim table # of entries in database
2. IOC expiration	ShieldSDN	# of IOC expired in database # of IOC deleted in ShieldVNF
3. Persistent Threats	attack simulation	# of attack frequency Expiry time

5.3.3 Data Collection and Measurements

In these experiments, ShieldSDN is the independent variable and the dependent variables are the observable outcomes. As part of the process, the script generates random IOCs at run

time in order to remove bias. These IOCs serve as the quantifiable data for the experiments outlined in the previous section.

ShieldSDN collects the data by running queries against the register, table, and database that stores the data. For example, in the first experiment, ShieldSDN queries attackerIp, cncIp, and victimIp registers, which hold the data that ShieldVNF has dealt with previously. Another data structure type that is relevant to ShieldSDN is the Match-Action table. In contrast to register, the table controls the packet flow. The table holds rules that specify what criteria to match and what action to take when it found a match. ShieldSDN also uses a relational database as the long-term storage for IOC, considering that ShieldVNF only has a limited amount of memory available. The tools used to retrieve the data are the command line tools for the switch and SQL database.

Data measurement is performed by counting the number of records that are relevant for each particular experiment. After ShieldSDN retrieved the data, we can count the number of entries that ShieldSDN returned. The only non-nominal data involved in these experiments is the expiry time.

5.4 Instrumentation and Implementation

The experiments involved the ShieldSDN 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 ShieldVNF. The database is used for long-term storage of IOC and analytics. Figure 5.2 shows the internal components of ShieldSDN.

5.4.1 SDN Controller

ShieldSDN is an SDN controller application that is written in node.js that interfaces with ShieldVNFs 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 (Microsoft, 2020b) (Amazon, 2020b) (Google, 2020a). This portability ensures that ShieldSDN implementation is scalable from lab-scale to cloud production.

In these experiments, ShieldSDN script was running on an AWS EC2 instance. The same instance hosts the scripts and the SQLite database (SQLite, 2020). The instance has direct access to all ShieldVNF switches in the organisation and communicates via IP. ShieldSDN runs every minute and is launched by a task scheduler (cron) to communicate with ShieldVNF

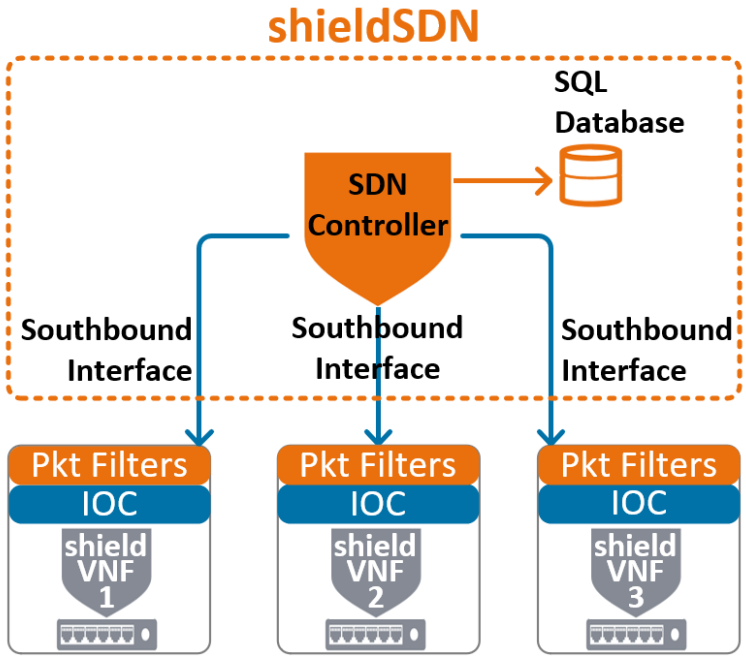


Fig. 5.2 ShieldSDN components: SDN controller, southbound interface to ShieldVNF, and SQL database for long-term storage of IOC records.

switches. ShieldSDN uses a command-line interface tool that is provided by the bmv2 switch and the prescribed parameters to get the desired results.

5.4.2 Southbound Interface

ShieldSDN uses a command line tool (`simple_switch _cli`) to interact with ShieldVNF for reading and writing operations. In particular, the four important commands are `register_read`, `register_write`, `table_clear` and `table_add`.

IOC retrieval and deletion from registers

The `register_read` and `register_write` commands are for ShieldSDN to retrieve and insert IOCs from the registers. For example, to retrieve IOC about the attacker's IP address, ShieldSDN would connect to ShieldVNF and issue "`register_read victimrIp_register` command." ShieldVNF would then return all the content of attackerIp_register as depicted in Figure 5.3

From this figure, we can see that most elements contains zero, whereas some elements contain an integer which is a decimal representation of an IP address. For example, the entry "`1226008369`" when converted to binary would become "`0100 1001 0001 0011 0110`

```
RuntimeCmd: register_read victimIp_register
register index omitted, reading entire array
victimIp_register= 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1226008369, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

Fig. 5.3 ShieldSDN retrieves IP addresses (decimal representation) that are under attack

0111 0011 0001", which is the "73.19.103.49" IP address. Besides collecting a victim's IP address, ShieldSDN also retrieves the victim's port number that is under attack with the `register_read victimPort_register` command.

Packet filtering rules insertion and deletion from tables

The `table_clear` and `table_add` commands are for ShieldSDN to delete and insert packet filtering rules that exist on ShieldVNF's tables. These tables are the KnownAttacker_table, KnownCNC_table, and KnownVictim_table. ShieldVNF will inspect every packet that it receives against these tables and drop the packet when a match is found.

ShieldSDN installs or removes packet filtering rules using the `table_add` command parameter. For example, the command `table_add MyIngress.KnownVictim_table drop 73.19.103.49/32 5060` will create an entry in `MyIngress.KnownVictim_table`, which will `drop` packets with `73.19.103.49` as the destination IP and `5060` as the destination port.

ShieldSDN that creates a packet filtering rule is also responsible for deleting these entries. When a ShieldSDN interacts with multiple ShieldVNFs, it is important for the ShieldSDN

to keep track of the ShieldVNF that it is interacting with. For example, suppose that ShieldSDN1 is responsible for installing a packet filtering rule on ShieldVNF1, ShieldVNF2, and ShieldVNF3. In this case, ShieldSDN1 is also responsible for deleting the expired IOC from ShieldVNF1, ShieldVNF2, and ShieldVNF3. ShieldSDN1 will also store these data in the SQLite database, which we will review next.

5.4.3 IOC Database

ShieldSDN stores the data in a SQL database using three tables: KnownAttacker, KnownCNC, and KnownTarget. Similar to the tables in ShieldVNF, these tables are for storing information about attackers, CNCs and targets that ShieldVNF have dealt with previously. The schema for these tables are shown below. Table 5.2 shows the schema to hold information about the known attacker, Table 5.3 shows the schema for the known CNC, whereas Table 5.4 shows the schema for the known targets.

Table 5.2 Database schema for KnownAttacker table

Column	Description
attacker_id	Primary key for a known attacker
time	Time stamp when the IOC was entered
switchNo	The switch that originates the IOC
expiryTime	Expiry time for this IOC
frequency	The number of occurrences
attacker	IP address of the attacker
tablehandle	Identifier for Match-Action Table
registerIndex	Identifier for a register entry
vnfdeleted	Flag (yes/no) for deletion at ShieldVNF
bcndeleted	Flag (yes/no) for deletion at Blockchain

5.5 Experiments

These experiments validate the hypothesis that ShieldSDN can scale the defence from one switch to multiple switches within an organisation. ShieldSDN does this by converting IOCs from the source ShieldVNF to packet filtering rules in other ShieldVNFs. In essence, ShieldSDN replicates the success that one ShieldVNF had to other ShieldVNFs within the organisation which geographically may span across multiple sites and countries. With limited memory on ShieldVNF, ShieldSDN needs to manage these rules while also dealing with persistent threats.

Table 5.3 Database schema for KnownCnc table

Column	Description
cnc_id	Primary key for a known CNC
time	Time stamp when the IOC was entered
switchNo	The switch that originates the IOC
expiryTime	Expiry time for this IOC
frequency	The number of occurrences
cnc	IP address of the cnc
tablehandle	Identifier for Match-Action Table
registerIndex	Identifier for a register entry
vnfdeleted	Flag (yes/no) for deletion at ShieldVNF
bcndeleted	Flag (yes/no) for deletion at Blockchain

Table 5.4 Database schema for KnownVictim table

Column	Description
target_id	Primary key for a known victim
time	Time stamp when the IOC was entered
switchNo	The switch that originates the IOC
expiryTime	Expiry time for this IOC
frequency	The number of occurrences
target	IP address of the victim
tablehandle	Identifier for Match-Action Table
registerIndex	Identifier for a register entry
vnfdeleted	Flag (yes/no) for deletion at ShieldVNF
bcndeleted	Flag (yes/no) for deletion at Blockchain

There are three experiments in total that follow the sequence and workflow, as described in Table 5.5. Each experiment will start with a description of the procedure, followed by the result, and close with a discussion.

5.5.1 Environment Setup

Procedures

A python script prepares the environment for the subsequent experiments. This script simulates a situation where ShieldVNF1 has 100 records of attackers, CNC, and victims in its registers. Figure 5.4 depicts the before and after in this setup. This step is necessary to set the stage so that ShieldSDN can collect these IOCs and install them as packet filtering rules on other switches.

Table 5.5 Experiments for ShieldSDN: scaling the defence from one switch to multiple switches in the organisation.

Experiments (IOC management)	**Sample population** (random selection and random assignment)	**Independent Variable**	**Dependent Variable**
Environment Setup	100 Known Attackers 100 Known CNC 100 Known Victims	ShieldSDN	# of register entries # of table entries
1. IOC collection and Packet filter installation	100 Known Attackers 100 Known CNC 100 Known Victims	ShieldSDN	# of pkt filter rules~ # database records
2. IOC expiration	100 Known Attackers 100 Known CNC 100 Known Victims	ShieldSDN	# of pkt filter rules # database update
3. Persistent Threat	100 Known Attackers 100 Known CNC 100 Known Victims	ShieldSDN	# of pkt filter rules # database update

Result

The result of environment setup procedure is captured in Table 5.6 below. There are four nodes involved (ShieldVNF1, ShieldVNF2, ShieldVNF3, and SQL database) and the table describes the before-and-after in executing the setup script for the IOC and packet filter-ingrules. Figure 5.4 shows how the setup script prepared the environment by loading IOCs to ShieldVNF1. The IOCs are found in the registers:

1. `attackerIp_register` storing the IP address of the attacker
2. `attackerPort_register` storing the Port number of the attacker
3. `cncIp_register` storing the IP address of the targeted CNC
4. `cncPort_register` storing the Port number of the targeted CNC
5. `victimIp_register` storing the IP address of the targeted DDoS victim
6. `victimPort_register` storing the Port number of the targeted DDoS victim

Figure 5.5 shows the content of `registerIp_register` on ShieldVNF after the experiment. It shows that ShieldVNF stored the IP address in binary, and these are its decimal

representations. For example `2080993444` is `7c0970a4` in hexadecimal, which translates to `124.9.112.164`. These IOC will be collected by ShieldSDN and eventually installed as packet filtering rules on ShieldVNF1, ShieldVNF2 and ShieldVNF3.

Table 5.6 **Environment Setup:** 100 IOCs were created in preparation for experiment 1, 2, and 3 (B = Before setup; A = After setup)

Environment Setup	shield VNF1		shield VNF2		shield VNF3		SQL	
	B	A	B	A	B	A	B	A
# of IOC in attackerIp_**register**	0	100						
# of IOC in attackerPort_**register**	0	100						
# of IOC in victimIp_**register**	0	100						
# of IOC in victimPort_**register**	0	100						
# of IOC in cncIp_**register**	0	100						
# of IOC in cncPort_**register**	0	100						
# of Packet Filtering Rule in KnownAttacker_**table**	0	0						
# of Packet Filtering Rule in KnownCNC_**table**	0	0						
# of Packet Filtering Rule in KnownVictim_**table**	0	0						

Discussion

The registers of ShieldVNF1 were the working memory of attack detection, which contained the IP address and port information of the attacker, the CNC, and the targeted victim. These entries need to be converted as packet filtering rules and installed in the KnownAttacker_table, KnownCNC_table, and KnownVictim_table so that the ShieldVNF can drop the packet immediately without having to perform attack detection tasks again.

At this stage, the script has prepared ShieldVNF1 with 100 attackers, CNC, and victims. The script has generated these IOC entries randomly at run time and also assigned these IOCs to a random group (attacker vs. CNC vs. victim) to ensure the validity that ShieldSDN will work with any IPv4 addresses. This method also removes bias and establishes causality.

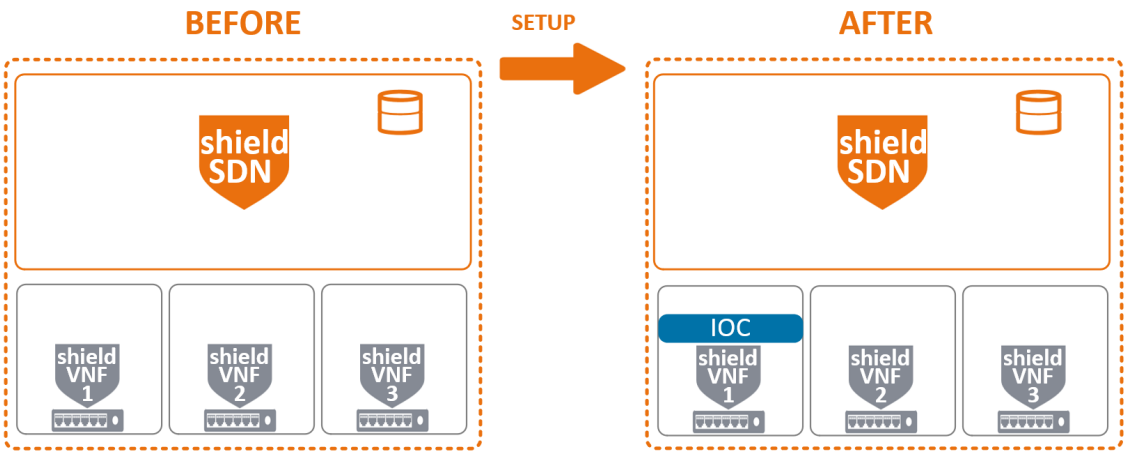


Fig. 5.4 **Setup:** The setup script registers 100 IOCs on ShieldVNF1, getting ready for ShieldSDN to distribute, convert, and install it on ShieldVNF1, ShieldVNF2 and ShieldVNF3 as packet filtering rules.

The number of IOCs is arbitrary, but for this experiment 100 is chosen for ease of statistical analysis.

The registers at ShieldVNF2 and ShieldVNF3 are empty because they have not encountered the attack yet. The registers only store first-hand information, and since ShieldVNF2 and ShieldVNF3 did not produce the IOC themselves, these registers were empty.

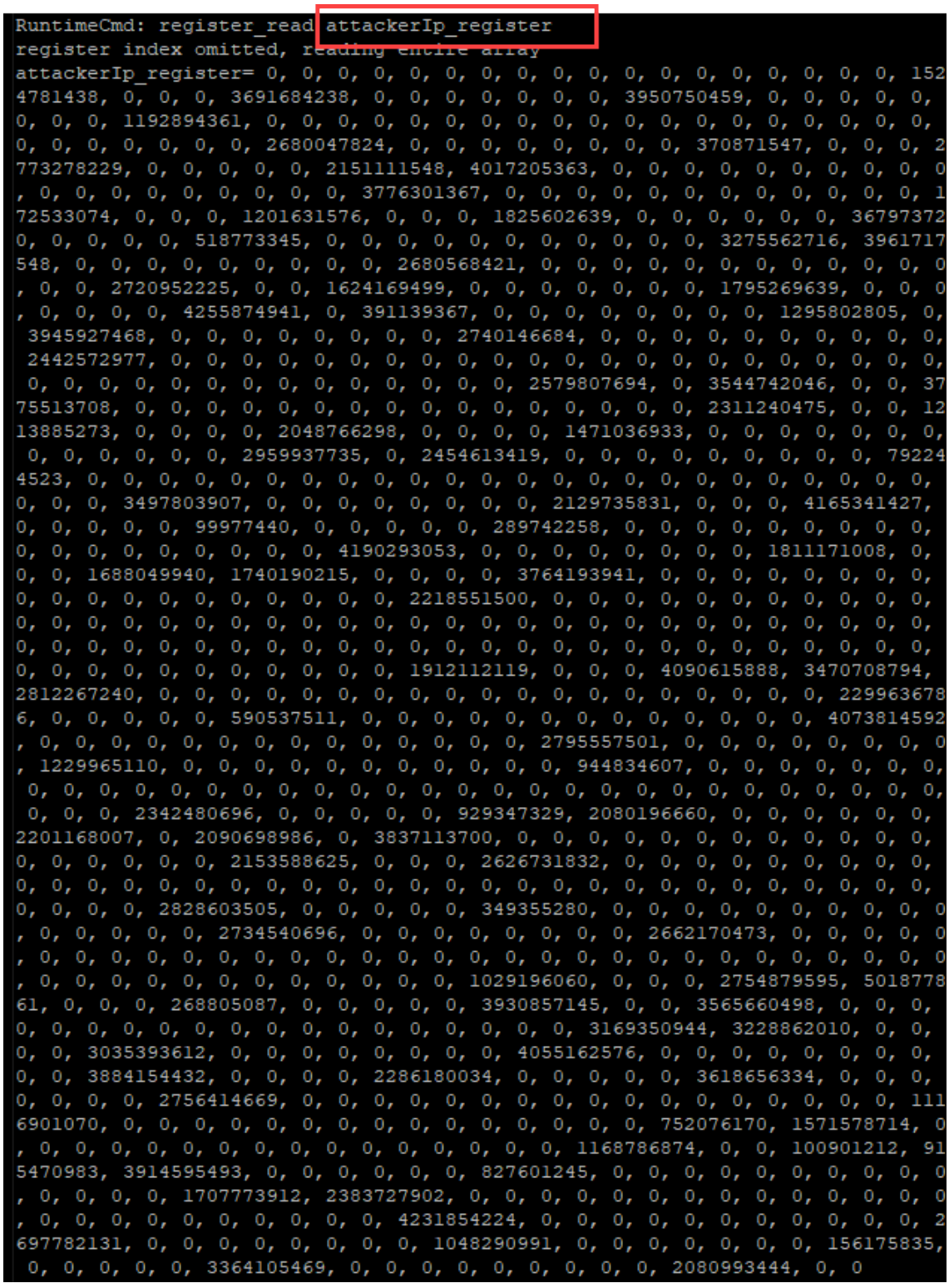


Fig. 5.5 **Setup:** The content of attackerIp_register on ShieldVNF (100 IOC records).

5.5.2 Experiment 1: Automated IOC Collection and Packet Filter Installation On Multiple Switches

The first experiment is to answer the first research question, i.e., What approaches can be used to collect the IOCs 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 ShieldVNF1, ShieldVNF2, and ShieldVNF3) and new records in a SQL Database.

Procedures

The process started with ShieldSDN retrieved the contents of ShieldVNF1 registers for known attackers, known CNC, and known victims. Upon receiving this information, ShieldSDN was supposed to install packet filtering rules in KnownVictim_table, KnownCNC_table and KnownAttacker_table on ShieldVNF1, ShieldVNF2, and ShieldVNF3. In addition, ShieldSDN was also supposed to create entries in the database for long-term storage. Figure 5.6 shows the sequence of this experiment.

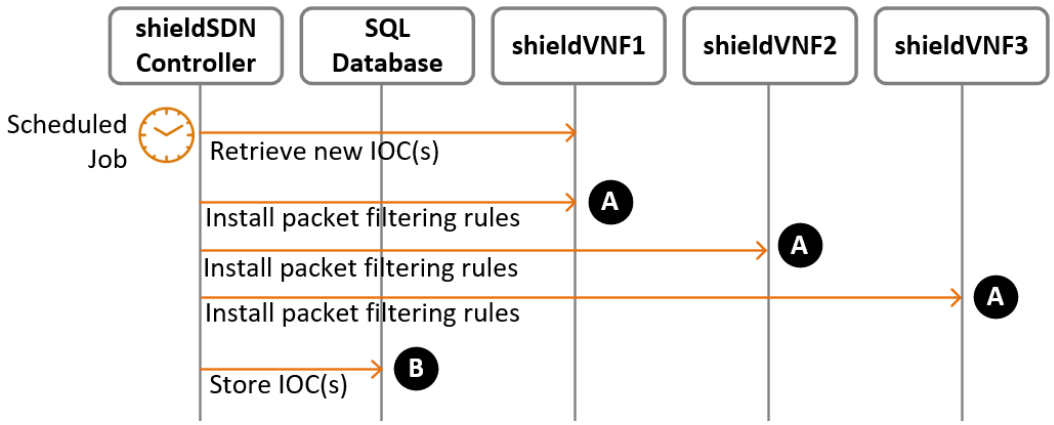


Fig. 5.6 **Experiment 1:** Sequence diagram for experiment 1. Snapshot of result at point "A" is depicted in Figure 5.13. Snapshot of result at point "B" is depicted in Figure 5.14.

Positive outcome is demonstrated when packet filtering rules are successfully created in the KnownVictim_table, KnownCNC_table and KnownAttacker_table at ShieldVNF1, ShieldVNF2, and ShieldVNF3. Negative outcome is demonstrated when these tables are empty.

Result

The result of this experiment is captured in the Table 5.7 below. After the experiment, the tables in ShieldVNF1, ShieldVNF2, and ShieldVNF3 are populated with packet filtering

rules. Out of 100 IOCs, 100 packet filtering rules were created. Figure 5.12 shows the before and after for this experiment.

The experiment took 64 seconds to run. Figure 5.7 shows the CPU utilisation and Figure 5.8 shows the memory utilisation of ShieldSDN during the experiment. On average, CPU utilisation on ShieldSDN is 59.58% during this experiment. Figure 5.9 shows the CPU utilisation of ShieldVNF1, Figure 5.10 shows the CPU utilisation of ShieldVNF2, and Figure 5.11 shows the CPU utilisation of ShieldVNF3. On average, CPU utilisation on ShieldVNF is 0.9% during this experiment.

Table 5.7 **Experiment 1:** ShieldSDN collects IOCs from ShieldVNF1, installs these IOCs as packet filtering rules at ShieldVNF1, ShieldVNF2 and ShieldVNF3, and stores these in the SQL database (B=Before experiment; A=After experiment).

Experiment 1: IOC Distribution	shield VNF1		shield VNF2		shield 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 Packet Filtering Rule in KnownAttacker_**table**		100		100		100		100
# of Packet Filtering Rule in KnownCNC_**table**		100		100		100		100
# of Packet Filtering Rule in KnownVictim_**table**		100		100		100		100

Discussion

This experiment demonstrates 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 was using the command line tool `simple_switch_cli` to connect to ShieldVNF1. For installing packet filtering rules on ShieldVNF1, ShieldVNF2 and ShieldVNF3,

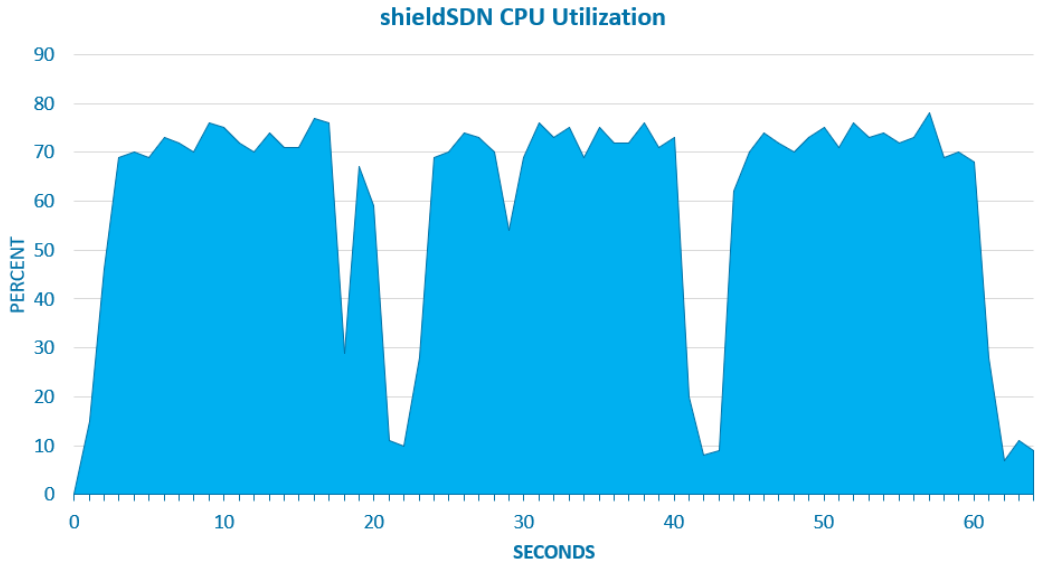


Fig. 5.7 **Experiment 1:** CPU utilisation by ShieldSDN.

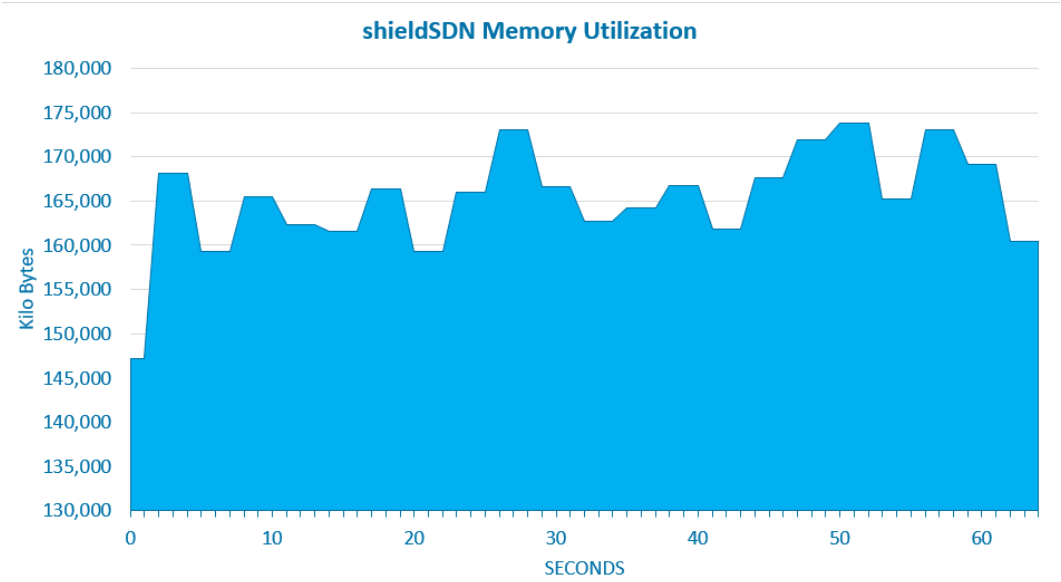


Fig. 5.8 **Experiment 1:** Memory utilisation by ShieldSDN.

ShieldSDN used the same tool, but using different command and parameters. Figure 5.13 shows the number of entries in KnownAttacker_table and KnownVictim_table on ShieldVNF1, ShieldVNF2, and ShieldVNF3.

A different CPU utilisation between ShieldSDN (59.58%) and ShieldVNF (0.9%) is expected as ShieldSDN performed the active role of extracting, converting, and installing IOC(s). From a ShieldVNF perspective, those activities did not impose a heavy workload that and did not negatively impact the performance.

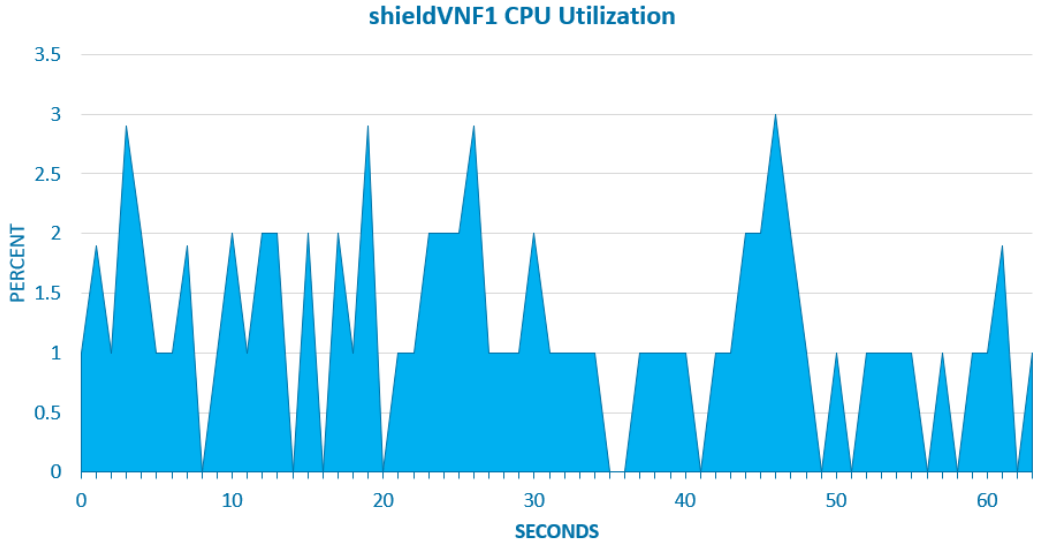


Fig. 5.9 **Experiment 1:** CPU utilisation of ShieldVNF1.

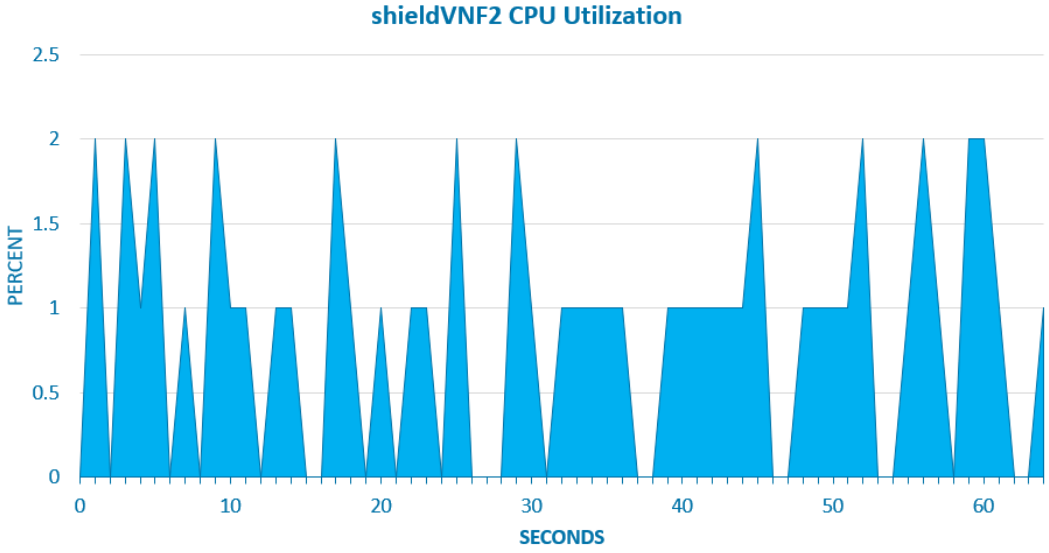


Fig. 5.10 **Experiment 1:** CPU utilisation of ShieldVNF2.

In addition, ShieldSDN also successfully created records for the IOCs in the SQL database tables. Table 5.8 shows one entry in the database table. In this particular record, it shows that this is a new attacker (167.118.98.95:5060), and 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 threat, which we will look at shortly.

These screenshots show that the IOCs that were originated from ShieldVNF1 have been successfully installed as packet filtering rules at ShieldVNF1, ShieldVNF2, and ShieldVNF3. With these rules installed, these switches will be able to drop malicious packets without

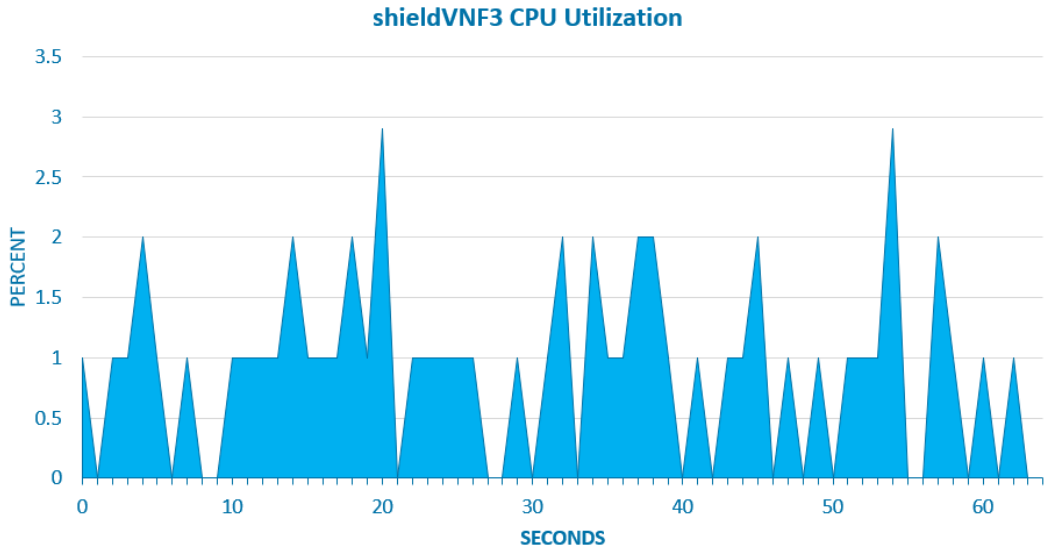


Fig. 5.11 **Experiment 1:** CPU utilisation of ShieldVNF3.

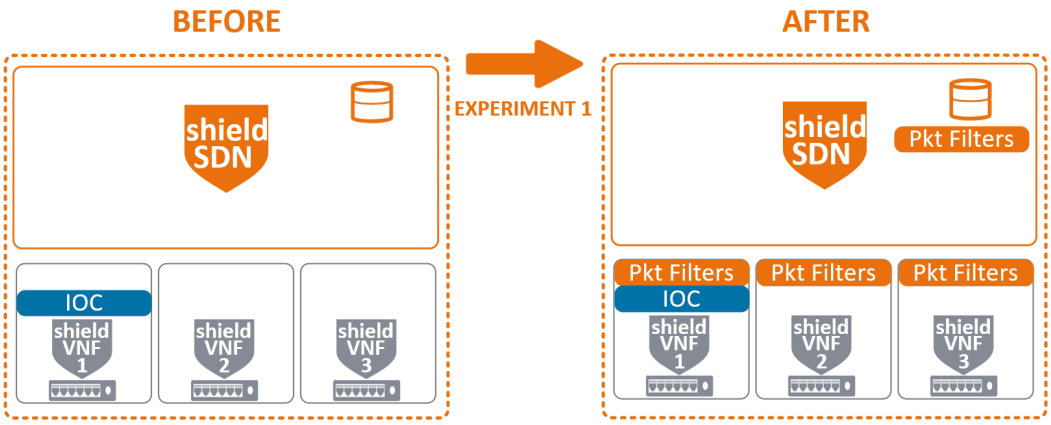


Fig. 5.12 **Experiment 1:** ShieldSDN collected IOCs from ShieldVNF1, converted them, and installed them as Packet Filtering Rules on ShieldVNF1, ShieldVNF2, and ShieldVNF3. ShieldVNF drops the packets that match with these filters.

having to do attack detection tasks again. Having packet filtering rules on ShieldVNF2 and ShieldVNF3 is significant because it proves that ShieldSDN was able to replicate the success from one switch (ShieldVNF1) to multiple switches in the organisation (ShieldVNF2 and ShieldVNF3). 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.

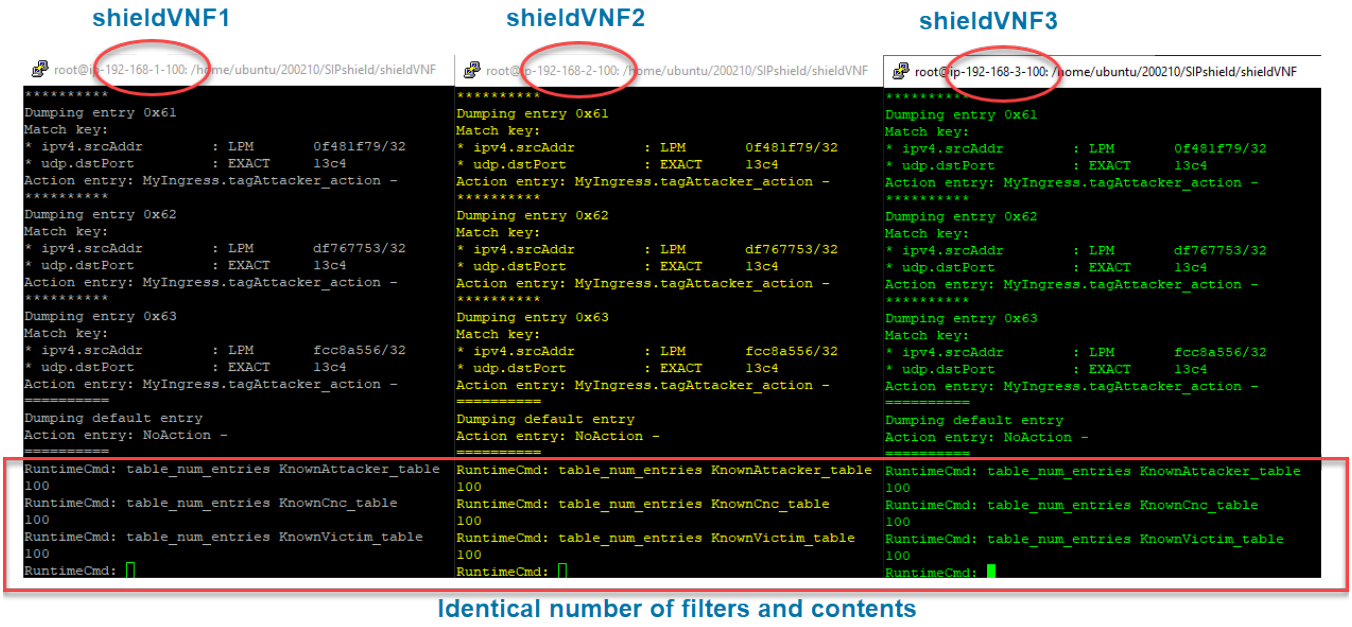


Fig. 5.13 **Experiment 1:** At the end of experiment 1, ShieldVNF1, ShieldVNF2, and ShieldVNF3 have identical filters installed.

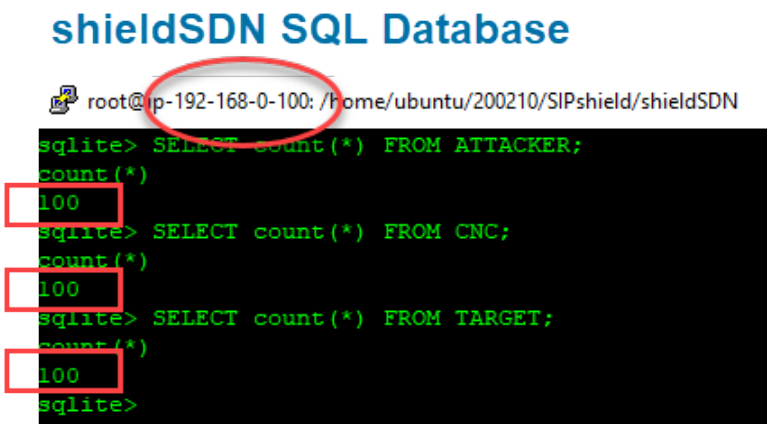


Fig. 5.14 **Experiment 1:** At the end of experiment 1, the IOC(s) are stored in the database.

Table 5.8 **Experiment 1:** Attacker record in SQL database

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

5.5.3 Experiment 2: Automated IOC Expiration Management

The second experiment is to answer the second research question, i.e., What strategies could be implemented to manage a limited amount of memory on ShieldVNF? If ShieldSDN is working as designed, then it will be able to remove the expired packet filtering rules (on ShieldVNF1, ShieldVNF2, and ShieldVNF3) and update the records in SQL Database.

There are two risks that makes this IOC management process necessary. The first is the risk of running out of memory and the second is the risk of a self-inflicted DoS attack. The first is caused by the limitation on the switch's resources, while the second is caused by the attacker that manipulates packet filtering rules by inserting malicious rules that drop packets from legitimate users. The proposed approach is to remove the expired IOCs and packet filtering rules from ShieldVNF1, ShieldVNF2, and ShieldVNF3 to conserve memory space.

Procedures

The process is started with ShieldSDN making a query to the database to retrieve IOCs that are older than the current timestamp. Once ShieldSDN has identified these IOCs, ShieldSDN connects to ShieldVNF1, ShieldVNF2, and ShieldVNF3 and deletes those rules. ShieldSDN also removes IOC entries in the originating switch (in this case, the registers in ShieldVNF1). For the records in SQL database tables, ShieldSDN sets the "VNFdeleted" field to 1 to indicate that this record has been deleted from the switch. Figure 5.15 shows the sequence of this experiment.

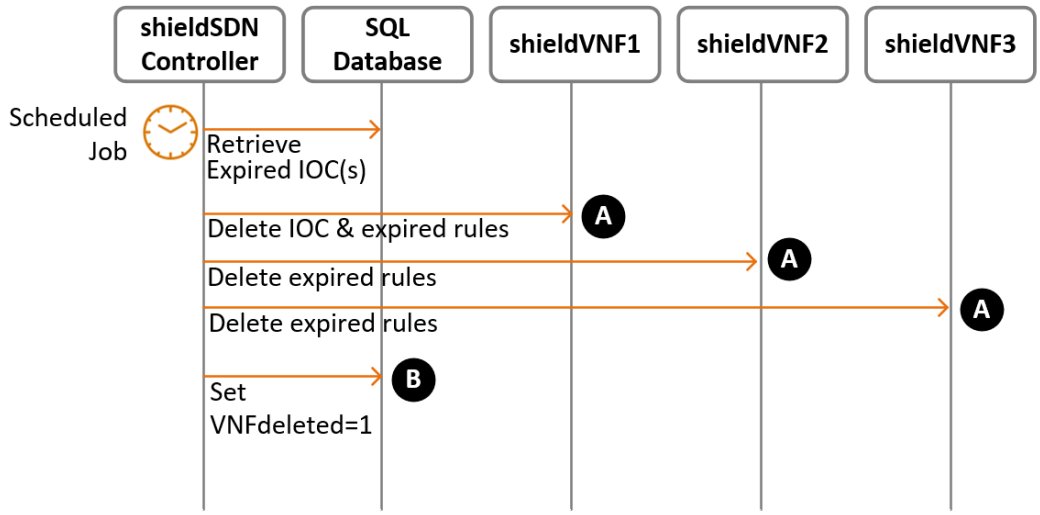


Fig. 5.15 **Experiment 2:** Sequence diagram for experiment 2. Snapshot of result at point "A" is depicted in Figure 5.20. Snapshot of result at point "B" is depicted in Figure 5.21.

Result

The result of this experiment is captured in Table 5.9 below. After the experiment, the expired IOC records and packet filtering rules have been deleted from the registers and tables at ShieldVNF1, ShieldVNF2, and ShieldVNF3. The records in the SQL database have also been marked with `1` which indicates that this record has been deleted from the switch. Figure 5.16 shows the progression as ShieldSDN expired the IOC(s) at various time.

The experiment took 54 seconds to run. Figure 5.17 shows the CPU utilisation and Figure 5.18 shows the memory utilisation of ShieldSDN during the experiment.

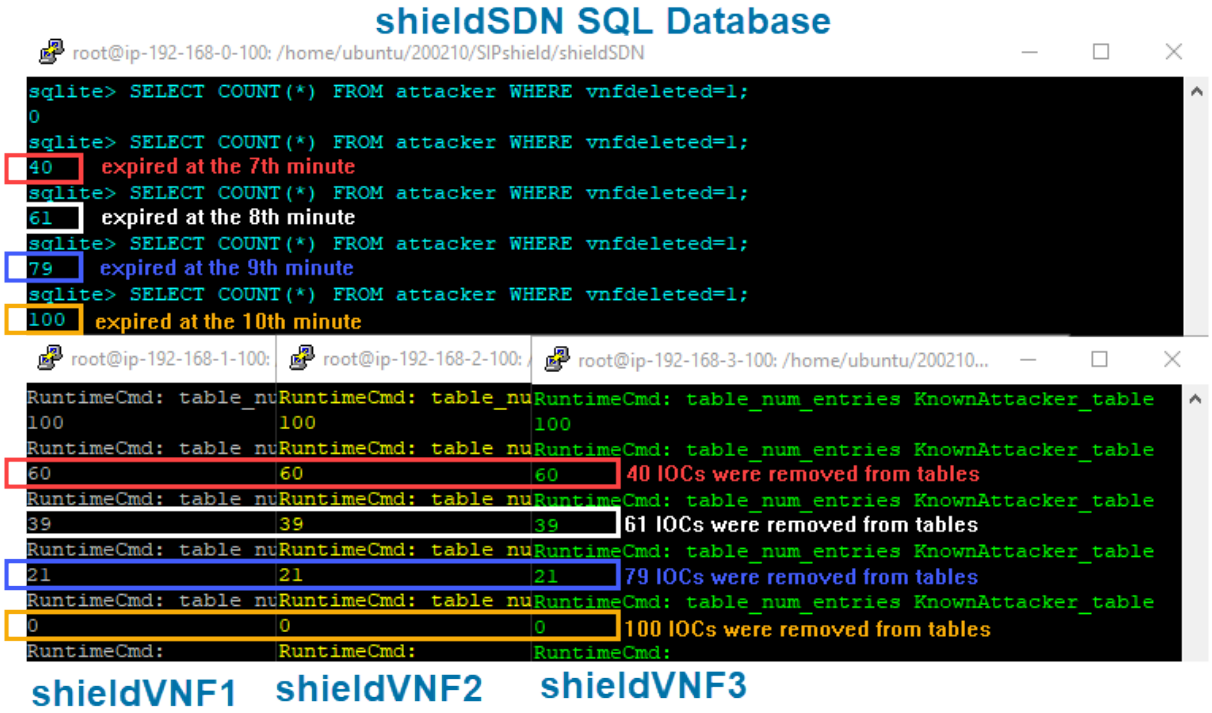


Fig. 5.16 **Experiment 2:** IOCs were assigned random expiry time between 5-10 minutes. As the IOCs expired, they are removed from the VNFs and the record in the database is set with `vnfdeleted=1`.

Figure 5.19 and Table 5.9 depicts the before and after of the experiment after the 10th minute. Out of 100 expired IOCs and packet filtering rules, 100 IOCs and packet filtering rules were deleted.

Discussion

In this experiment, ShieldSDN was able to retrieve expired IOC from the database and removed expired IOC records and packet filtering rules from ShieldVNF1, ShieldVNF2, and

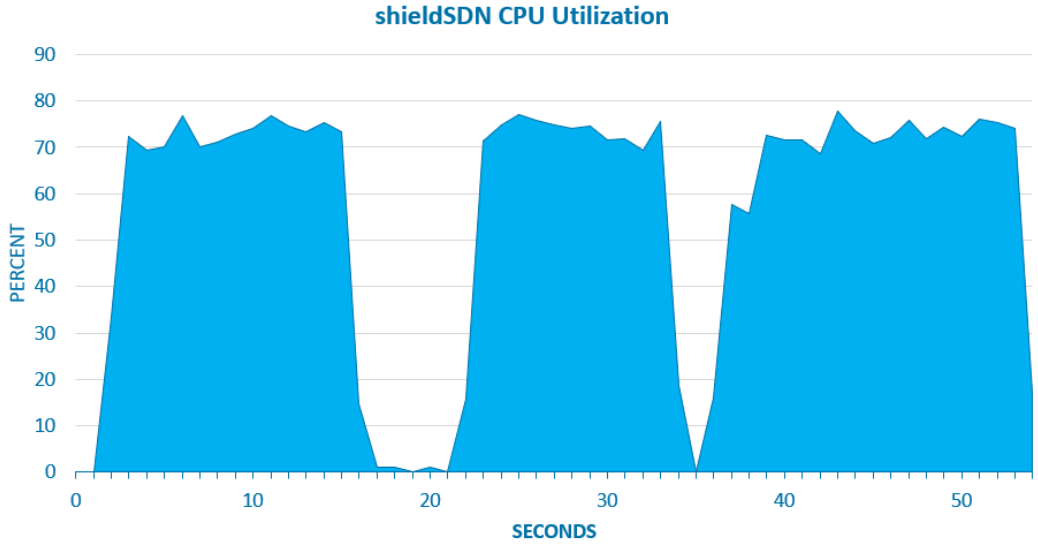


Fig. 5.17 **Experiment 2:** CPU utilisation by ShieldSDN.

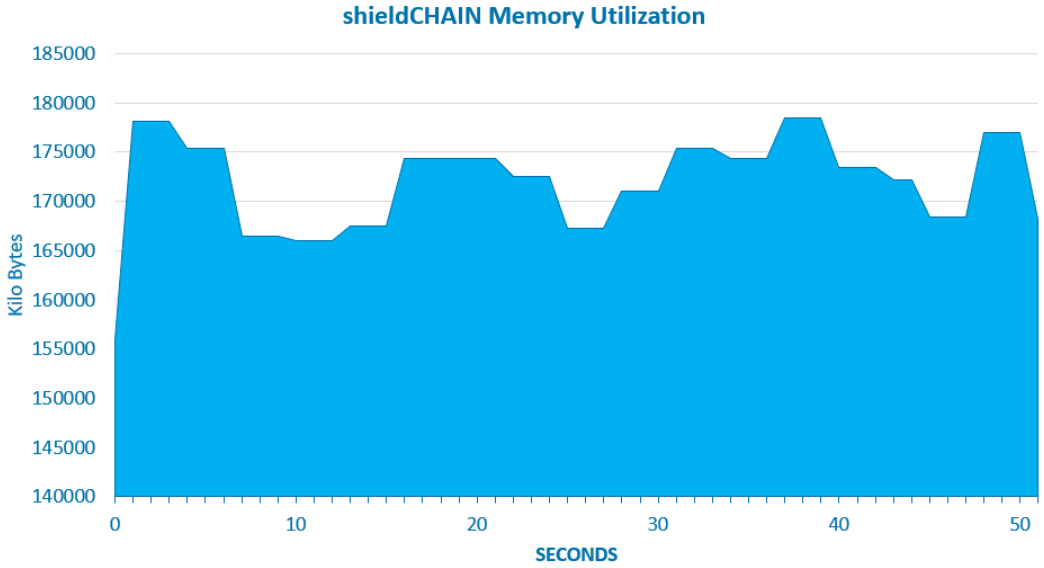


Fig. 5.18 **Experiment 2:** Memory utilisation by ShieldSDN.

ShieldVNF3. In the database, ShieldSDN also marked each of those records as deleted, i.e., ShieldSDN sets the field `VNFdeleted` to 1. This flag helps ShieldSDN to keep track of the IOCs that are still active on the switches. With this field marked as deleted, ShieldSDN would not retrieve this IOC again the next time it queries the database.

From the ShieldVNF perspective, Figure 5.20 shows that the packet filtering rules has been deleted from the KnownAttacker, KnownCnc, and KnownVictim table. This is shown in the command line prompt where it returned 0 entries in the table.

Table 5.9 **Experiment 2:** IOC Expiration (Before and After the 10th Minute). (B = Before; A = After experiment).

Experiment 2: IOC Expiration	shield VNF1		shield VNF2		shield VNF3		SQL	
	B	A	B	A	B	A	Before	After
# of IOC in attackerIp_**register**	100	0						
# of IOC in attackerPort_**register**	100	0						
# of IOC in victimIp_**register**	100	0						
# of IOC in victimPort_**register**	100	0						
# of IOC in cncIp_**register**	100	0						
# of IOC in cncPort_**register**	100	0						
# of Packet Filtering Rule in KnownAttacker_**table**	100	0	100	0	100	0	100	deleted=1
# of Packet Filtering Rule in KnownCNC_**table**	100	0	100	0	100	0	100	deleted=1
# of Packet Filtering Rule in KnownVictim_**table**	100	0	100	0	100	0	100	deleted=1

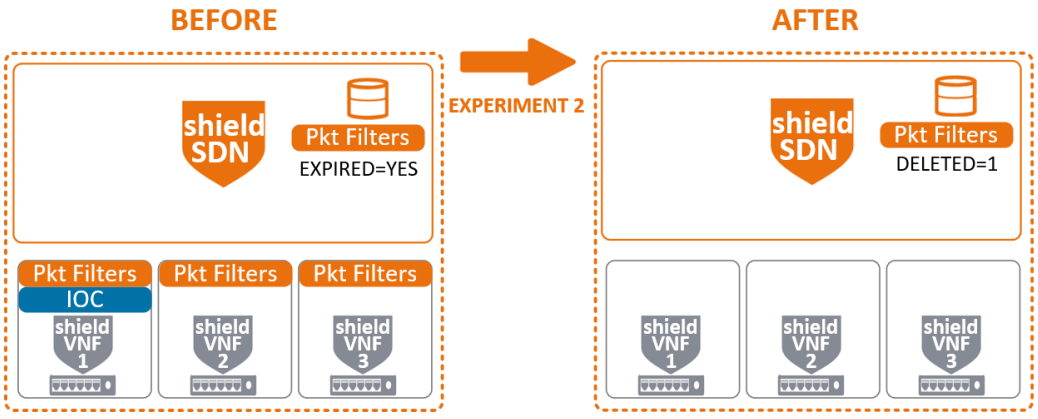


Fig. 5.19 **Experiment 2:** ShieldSDN retrieved expired filters from SQL database and removed it from ShieldVNF1, ShieldVNF2, and ShieldVNF3. In SQL database, the deleted field is set to 1.

From the database perspective, Figure 5.21 shows the number of records that have been deleted by ShieldSDN. This is shown in the SQL client command line where it counted 100 records that have `vnfdeleted` field set to 1.

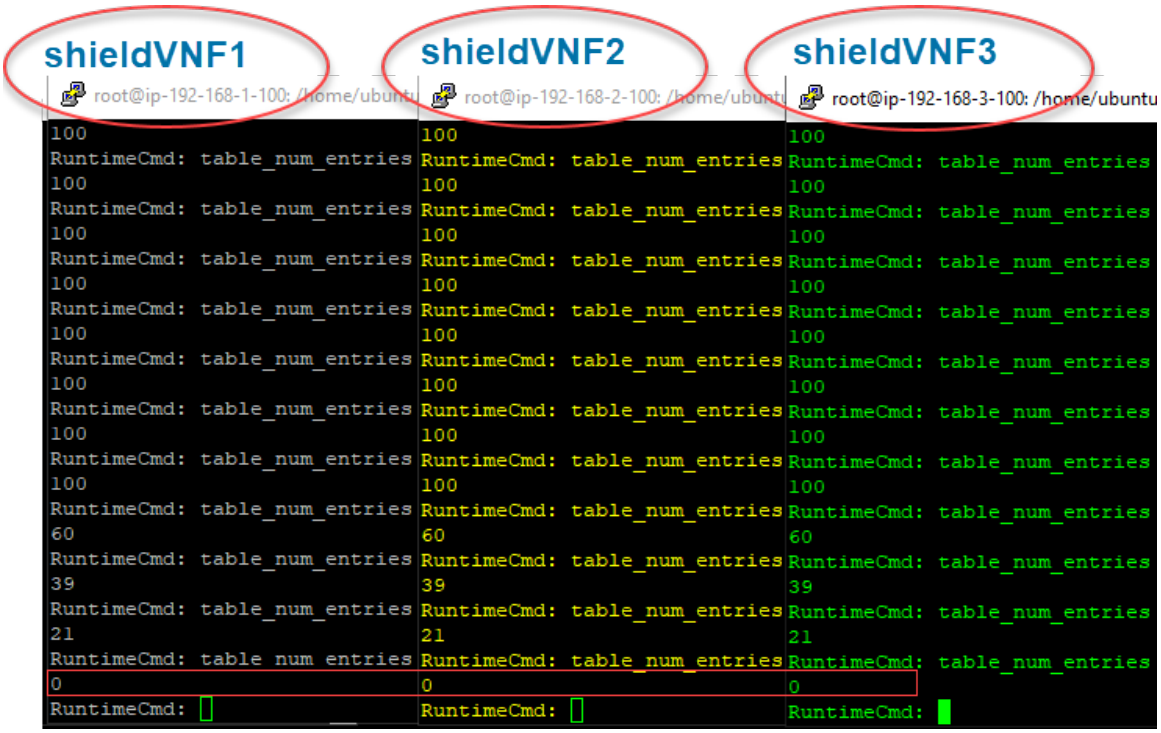


Fig. 5.20 **Experiment 2:** Expired rules were deleted from the KnownAttacker, KnownCnc, and KnownVictim table. The number of entries is 0.

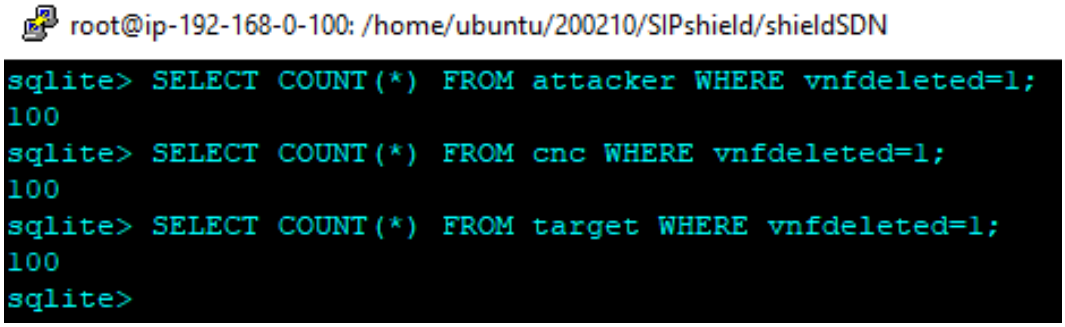


Fig. 5.21 **Experiment 2:** In SQL tables, there are 100 entries that were marked as deleted (i.e., `vnfdeleted=1`).

These screenshots demonstrate that ShieldSDN successfully removed the IOC and packet filtering rules from ShieldVNF1, ShieldVNF2, and ShieldVNF3. With these removals, the switches free up the limited memory space so that it can use its resources to mitigate attacks that are currently active. This automated IOC expiration management function from ShieldSDN is significant because it helps ShieldVNF to conserve memory resources and continue to function as designed. Out of 100 IOCs in the registers and 100 packet filtering rules in the tables, 100 IOCs and 100 rules were deleted. This is statistically significant and demonstrates a positive outcome for this experiment.

5.5.4 Experiment 3: Automated Persistent Threat Management

The third experiment is to answer the third research question, i.e., What methods can be used to efficiently manage ShieldVNF memory when dealing with persistent threats? If ShieldSDN is working as designed, then it will be able to install packet filtering rules (on ShieldVNF1, ShieldVNF2, and ShieldVNF3) with an expiry time that is exponentially longer. The idea is to force the attackers to spend more time and effort as they launch more attacks. The assumption is that the attackers will find it not worth their time and then they will go after easier targets.

From the previous experiment that represents the first round of attack, the packet filters have expired and been deleted from ShieldVNF. In this case, ShieldVNF has no recollection of previous records which implies that it has to re-learn about the attacks. To identify repeat offenders, ShieldSDN turns to the records in the database and searches for an attacker that has the same characteristics. When ShieldSDN finds the record for a repeat offender, it will check the `frequency` field to determine the number of offences that this attacker has previously committed. The database is meant for long-term data storage and the records are kept as long as the storage capacity on the server allows. The data is useful for multiple use cases like generating audit reports, analytics, or to train predictive models.

The challenge when dealing with repeat offenders is to minimise the number of rules that ShieldVNF needs to keep in memory. Instead of having multiple rules with short life-spans, the proposed approach is to create a packet filtering rule with exponentially longer expiry times based on the frequency of attack. This approach allows ShieldVNF to keep the rules for a longer period of time that, in effect, will reduce the overall number of entries required.

Procedures

Experiment 1 was the first round of attack. This experiment performs the second and third round of attack to validate the functionality of ShieldSDN in dealing with repeat offenders. The second and third round of attack follows the same five steps:

1. Setup script prepares 100 IOCs in the registers of ShieldVNF1
2. ShieldSDN retrieves these values from ShieldVNF1
3. ShieldSDN queries SQL database and found the records of these repeat offenders
4. ShieldSDN recreates the packet filtering rules in the KnownAttacker_table, the KnownCNC_table, and the KnownVictim_table at ShieldVNF1, ShieldVNF2, and ShieldVNF3
5. ShieldSDN updates the record to reflect the second round (or third) of attack

Figure 5.22 shows the sequence of this experiment.

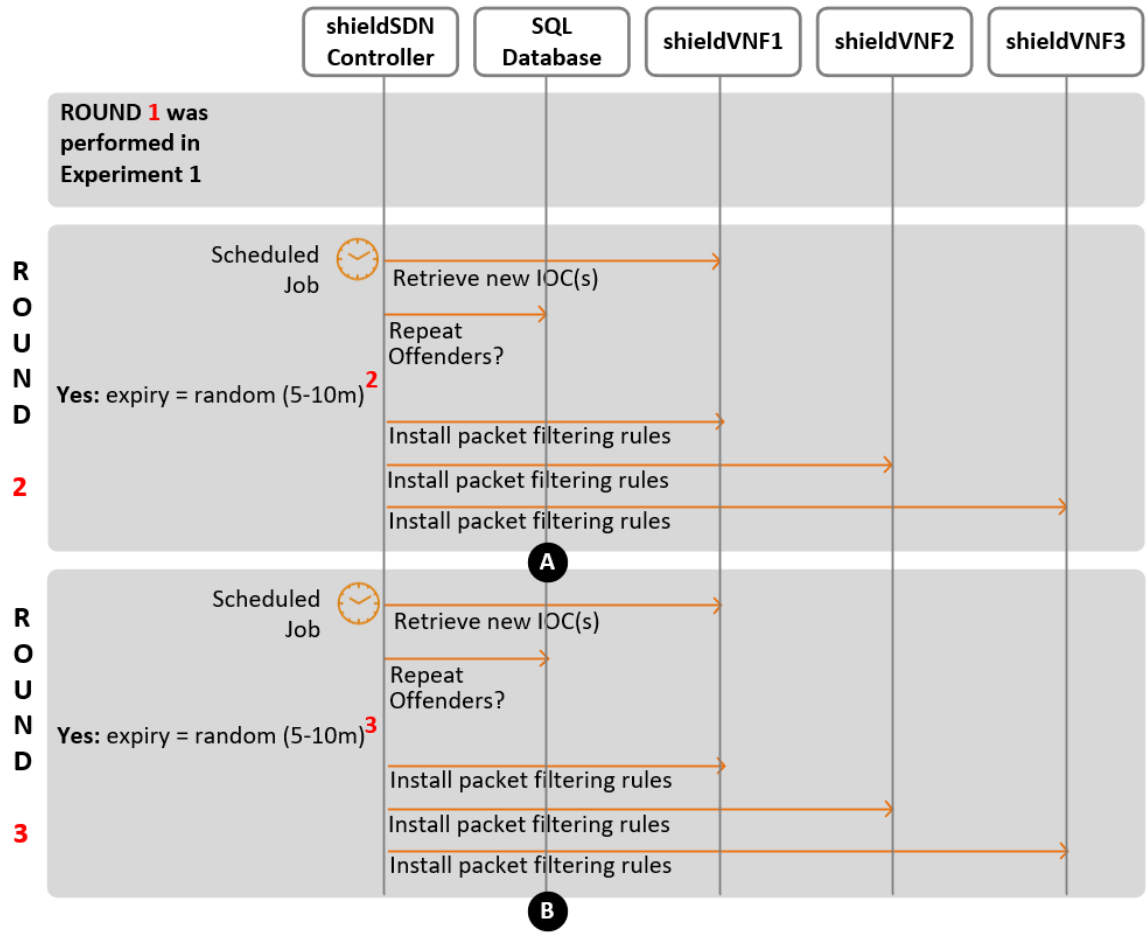


Fig. 5.22 **Experiment 3:** Sequence diagram for experiment 3.Snapshot of result at point "A" is depicted in Figure 5.23. Snapshot of result at point "B" is depicted in Figure 5.26.

With persistent threats, ShieldSDN sets the `expirytime` field to be exponentially longer based on the value in the frequency field. To set this value, ShieldSDN uses a function from Javascript built-in objects, `Math.pow(base, exponent)`, and follows the following steps:

- Step 1: Select a random integer between 5 and 10
- Step 2: Use this randomly selected integer as the base, and use the frequency value as the exponent

As an example, supposed that ShieldSDN selected 5 as the random integer. For the second offence, the timeout is 25 minutes.

$$5^2 = 25$$

For the third offence, the timeout is 125 minutes.

$$5^3 = 125$$

Result (Attack Round 2)

The result of this experiment for the second round is captured in a screenshot in Figure 5.23. It shows the records in the database with frequency is set to two and expiry time which is 25 minutes.

Fig. 5.23 **Experiment 3: Round 2:** Repeat offender has Frequency column greater than 1 and expiry time is set to 25 minutes.

The experiment took 70 seconds to run. Figure 5.24 shows the CPU utilisation and Figure 5.25 shows the memory utilisation of ShieldSDN during the experiment.

Result (Attack Round 3)

The result of the third round is captured in is captured in a screenshot in Figure 5.26. After the experiment, the packet filtering rules remains active but the records in SQL database has

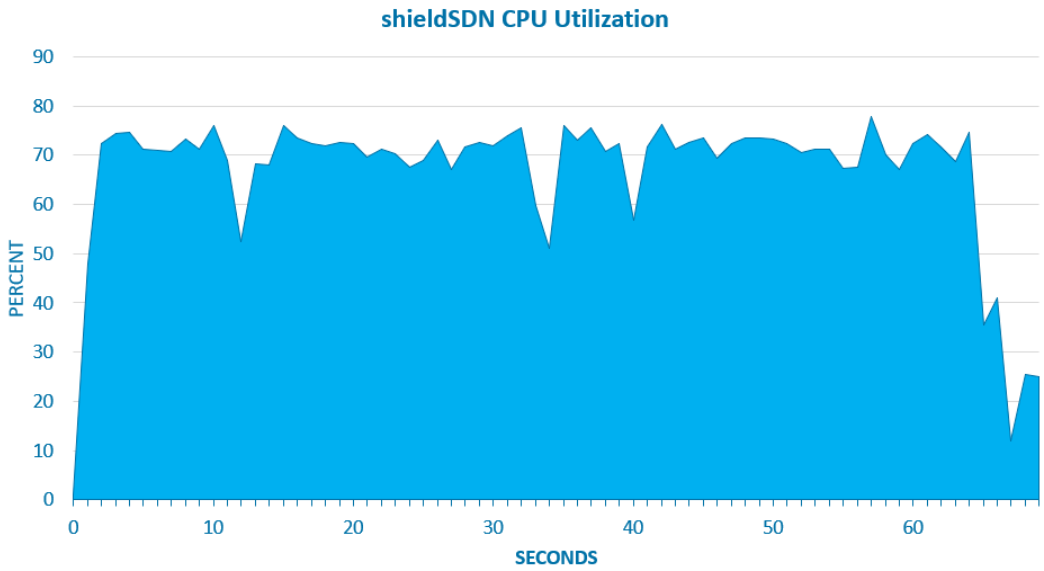


Fig. 5.24 **Experiment 3:** CPU utilisation by ShieldSDN.

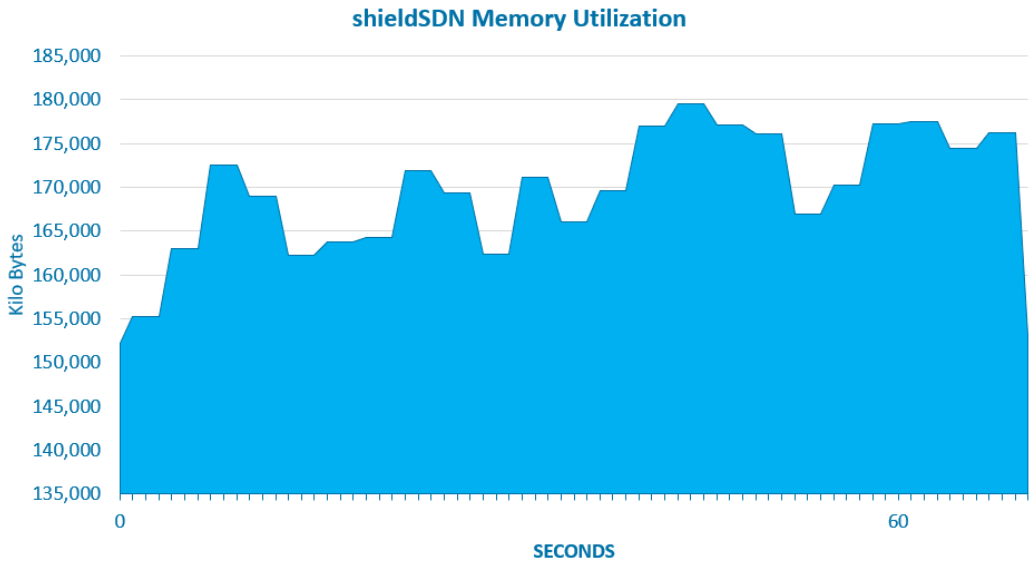


Fig. 5.25 **Experiment 3:** Memory utilisation by ShieldSDN.

been updated. The `freq` field is set to 3 to reflect the number of offence associated with this particular attacker, and the `expirytime` field is set to 125 minutes.

Discussion

Persistent threat presents a challenging case for ShieldVNF considering the finite amount of memory that it has available. In this particular experiment, ShieldSDN demonstrated that it was able to efficiently manage memory resources when dealing with persistent threats.

Fig. 5.26 **Experiment 3: Round 3:** ShieldSDN encountered repeat offenders (APT), reactivated the packet filtering rule (deleted=0) and sets longer expiry time (49 minutes for 2nd repeat offence)

ShieldSDN achieved this by assigning exponentially longer expiry time for repeat offenders so that the number of packet filtering rules in the memory can be kept to minimum.

The result from attack round two demonstrates that ShieldSDN was able to increment the `frequency` field to reflect the number of incident that ShieldSDN has encountered in the past. For this second round of attack, ShieldSDN set this field to two. The `expirytime` for the second offence also updated accordingly to 25 minutes.

The result from attack round three demonstrates that ShieldSDN was able to increment the `frequency` field to three and the `expirytime` to exponentially longer than the second round. When the same attacker re-offend for the third time, the `expirytime` is set to 125 minutes, as depicted in Figure 5.26.

The ability for ShieldSDN to efficiently manage limited memory resources when dealing with persistent threats is significant for two reasons. The first reason is because it helps ShieldVNF to conserve memory resources so that it can continue to function. The second reason is to frustrate the attacker so that they can stop the attack and move on to easier targets. The idea is to increase the time and effort required on the attacker's side to a point where they realise that they are not getting any return on their effort. Out of 100 packet filtering

rules, 100 rules have been assigned with an expiry time that is random and exponentially longer. This demonstrates a positive outcome for this experiment.

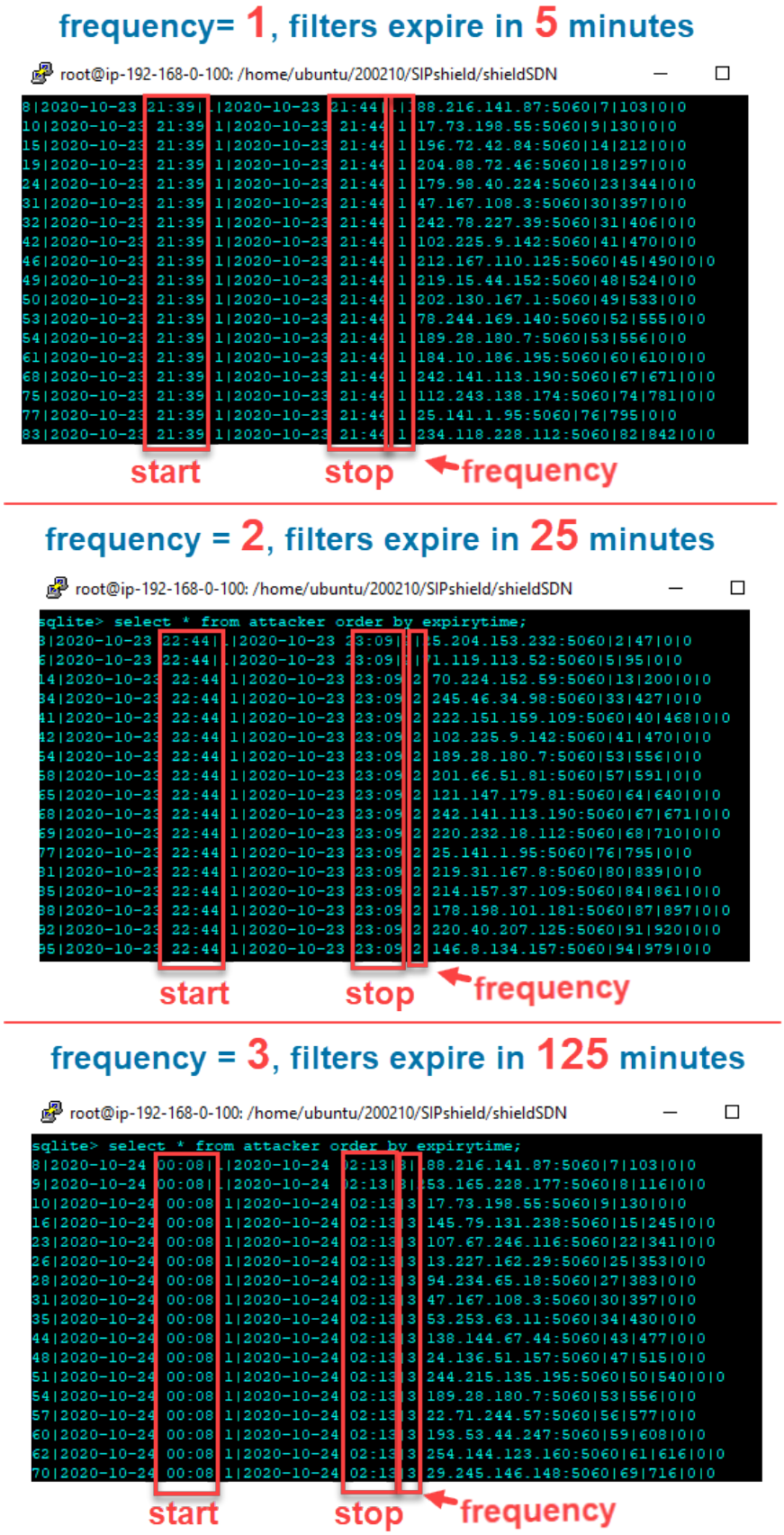


Fig. 5.27 **Experiment 3: Round 1-3:** ShieldSDN assigns repeat offenders with longer expiry time as the attack frequency increases.

5.6 Chapter Summary

ShieldSDN can scale the defence from one switch to multiple switches in the organisation. Through the experiments in this chapter, we validated that ShieldSDN was able to collect IOCs from one switch and install them as packet filtering rules on multiple switches. In order to conserve memory and to keep ShieldVNF running as intended, efficient management of memory resources becomes important. To this end, we validated that ShieldSDN was able to remove expired packet filtering rules from the memory of ShieldVNF1, ShieldVNF2, and ShieldVNF3. Persistent threats present a challenge where the repeat offenders will reappear after the rules have been deleted. To this end, ShieldSDN was able to create packet filtering rules with random and exponentially longer expiry times. This approach allows ShieldVNF to be making efficient use of memory space when dealing with repeat offenders. At the same time, random and unpredictable expiry time would force the attacker to spend more time and effort. The expected result would be for the attacker to realise that they are not getting any return on their effort and therefore stop the attack and move on to other target.

Lessons learned from one organisation about mitigating a botnet attack would also be beneficial for other organisations in the larger community. Botnet attacks that exploits popular IoT devices would affect many organisations with varying cybersecurity maturity. Some organisations are more equipped to deal with outbreaks than other organisations. Since the community is interconnected one way or the other, vulnerability with one organisation still imposes risks on other organisations. Therefore, it is important for organisations to band together to build a synchronised defence posture so they can fight together to defeat the botnet. This task presents a new challenge on how to synchronise a defence posture across autonomous organisations, and that is the topic that we will explore in the next chapter.

Chapter 6

ShieldCHAIN: Scaling SIP DDoS Defence to Multiple Organisations

6.1 Overview

In the previous chapter, ShieldSDN has scaled the defence from one switch to multiple switches. The objective of this chapter is to scale the defence from one organisation to multiple organisations. To achieve this objective, we will investigate three sub-questions which will be validated by three experiments:

- What strategies could be adopted to share threat intelligence with the community?
- What strategies could be adopted to retrieve threat intelligence from the community?
- In what ways could a switch leverage the community-sourced intelligence as packet filtering rules?

From the SIPshield framework, this chapter investigates the ShieldCHAIN element that allows scaling the defence beyond a single organisation. The idea is to replicate the success that one organisation had to multiple organisations so that they can mitigate the same attacks. In the previous chapter, ShieldSDN in organisation1 stored the IOC in the SQL database. ShieldCHAIN retrieves these IOCs and shares it to Blockchain. This approach enables ShieldCHAIN in other organisations (organisation2 & organisation3) to retrieve these IOCs from Blockchain and install them as packet filters to drop malicious packets that matches the IOCs. Figure 6.1 depicts the process where it started as packet filters from the source-organisation (organisation1) and finished as packet filters on the switches of the receiving-organisations (organisation2 & organisation3). In essence, ShieldCHAIN

provides a collaborative defence framework where detection efforts by one organisation can be leveraged as attack prevention efforts by other organisations in the community.

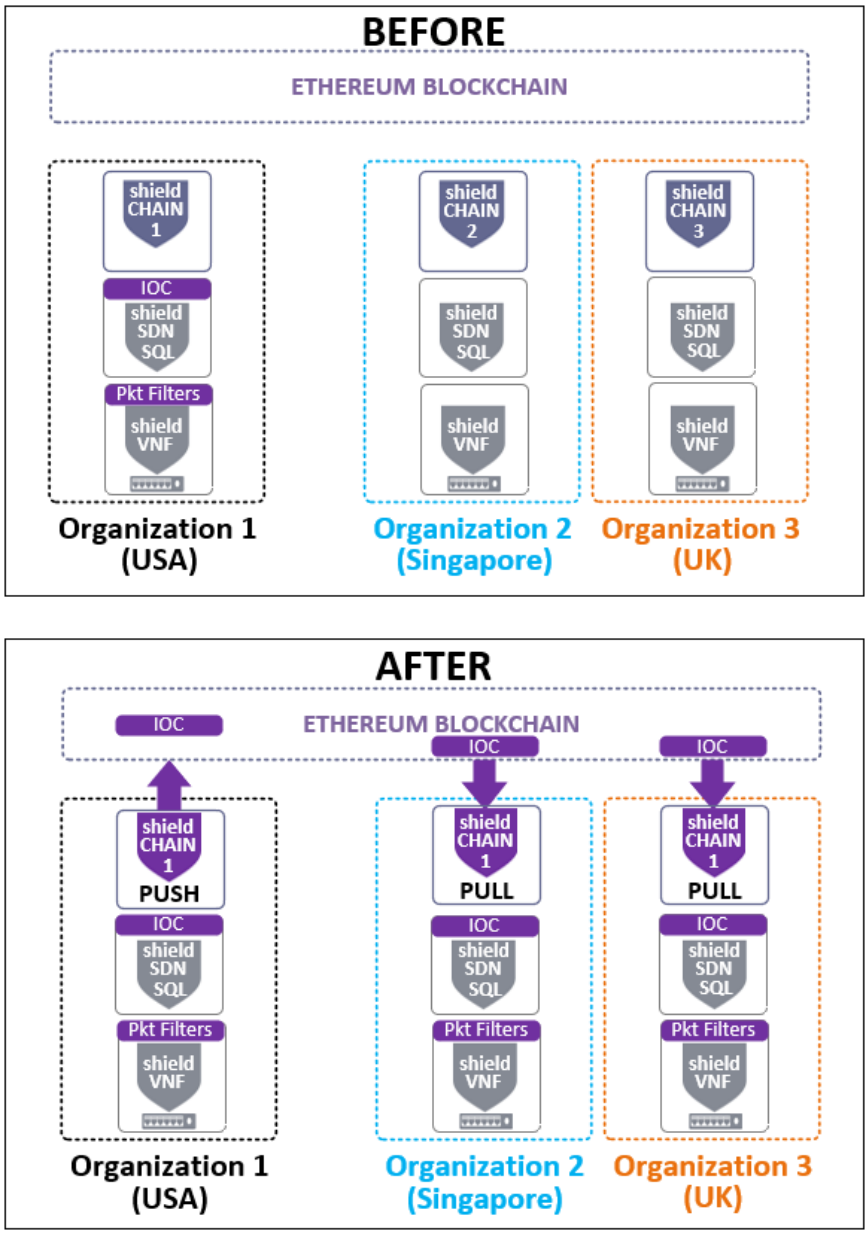


Fig. 6.1 ShieldCHAIN in organisation1 sends IOC to Ethereum Blockchain. ShieldCHAIN in organisation2 and organisation3 retrieves and installs it as packet filtering rules on their switches.

This chapter starts with ShieldCHAIN as the proposed solution, followed by the design of experiments and then the implementation for the experiments. The last section describes three experiments which include the procedure, result, and discussion of each experiment.

6.2 The Proposed Solution: ShieldCHAIN

6.2.1 Collaborative Defence Framework Approach & Its Challenges

Collaborating and sharing threat intelligence within a trusted community is essential to fight against botnet-originated Distributed DoS attacks. A botnet-originated attack is carried out by a group of distributed devices that are orchestrated to achieve malicious purposes. One way to keep up with distributed attacks is to form a distributed defence. This type of defence is formed by a like-minded community that collaborates to defend against distributed attacks. One method to achieve this is by sharing threat intelligence within the community so that each member of the community has situational awareness and an early warning. This approach enables the community to be proactively prepared should the same attack occur in the member's network. However, there are multiple challenges to implement this approach such as lack of personnel, lack of common framework for collaboration, and lack of practical means to implement the framework.

Personnel constraints are preventing organisations from participating in sharing threat intelligence. Organisations are already facing personnel pressures to maintain a good security posture. Additional initiatives need to be assessed based on the potential benefit versus the investment required. The proposed solution would need to offer compelling benefits without requiring considerable investment on the member's part. Even when they agree to participate, lack of an established framework presents a challenge.

A common framework is required to govern threat intelligence sharing within the community. Threat intelligence sharing is a relatively new initiative and, as such, there is no standardised framework that is used across industries, government, and academia. A common framework helps to define the roles and responsibilities among members. Even when the community agrees on a common framework, the next challenge will be to agree on the mechanism to deploy the intelligence and mitigate the attacks.

Organisations are lacking practical means to implement threat sharing and collaborative defence framework. Considering that different organisations have different equipment from different manufacturers, there is a big gap to convert the threat intelligence into an attack mitigation mechanism that is actionable. Typically, the process would require the security administrator to manually review the intelligence, then manually configure the firewall, intrusion detection/prevention system, or switches. As such, this workflow is not scalable to put it into practice.

While the benefits of a collaborative defence framework is appealing, these constraints are real and impede the implementation of such a framework. In the next section we will

look at ShieldCHAIN, which is the proposed solution that enables a collaborative defence approach and addresses these concerns.

6.2.2 ShieldCHAIN: Blockchain Application for a Collaborative Defence Framework

ShieldCHAIN consist of two components: a dApp and a smart contract that is deployed on Ethereum Blockchain. ShieldCHAIN facilitates threat intelligence sharing within a community by automating the sharing, retrieval, and managing the IOC among members in a community. In order to participate, each member organisation would run their own instance of dApp in their organisation, but each dApp will access the same smart contract that is deployed on Ethereum Blockchain. Figure 6.2 depicts ShieldCHAIN with its components and how these components interact.

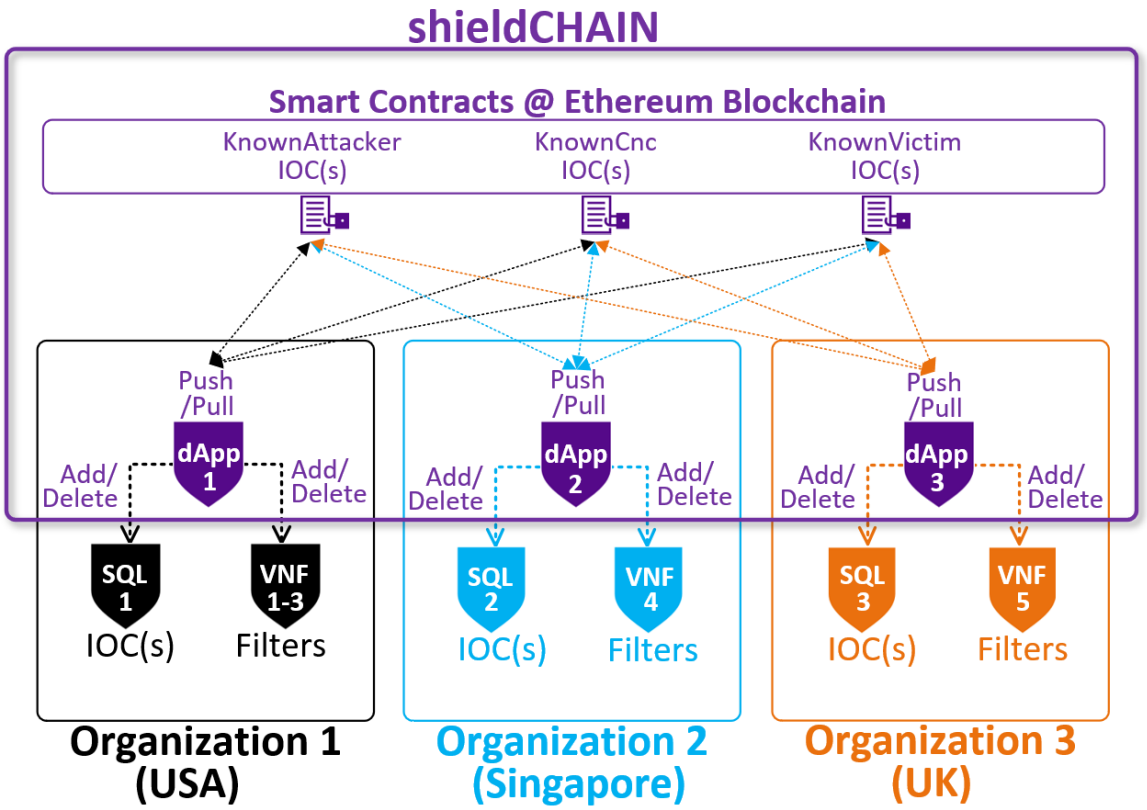


Fig. 6.2 ShieldCHAIN consists of smart contracts deployed at the Ethereum Blockchain and dApp deployed by each participating organisation.

ShieldCHAIN uses Blockchain technology due to its intrinsic features that facilitate data sharing between independent organisations. 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 peer-to-peer, and there is no central authoritative 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.

The decentralised Applications (dApps) serve 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 organisation. Each member will use their dApp as the front-end interface to access three services: smart contract, SQL database, and ShieldVNF. The dApp interfaces with a smart contract to create, delete, retrieve threat intelligence stored on the Ethereum Blockchain. The dApp interfaces with a SQL database to access the IOC records that were created by ShieldSDN. Lastly, the dApp interfaces with ShieldVNF to install packet filtering rules.

The smart contract serves as the back-end component of ShieldCHAIN. Smart contract is a code that gets deployed on the Ethereum Blockchain 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 the Blockchain.

The dApp is scheduled to connect to the smart contract at a regular interval. When an organisation has IOC to share with the community, the dApp will push the IOC by writing to the smart contract. For the reverse operation, when an organisation would like to read the current state of threat intelligence, the dApp will pull it from the smart contract. These read and write tasks are scheduled to occur at regular intervals via job scheduler. This approach provides automation and alleviates the need for manual intervention from the security administrator. With the dApp triggered at certain times, there is an inherent time delay for sharing and receiving IOC. Considering that time is of the essence when dealing with botnet outbreaks, this approach has a risk implication where the outbreak may have occurred before the community can retrieve the IOC.

The IOCs could be posted to other platforms such as a central server; however, unique features from Blockchain would need to be substituted with other technologies. Blockchain presents features conducive to publicly maintaining a distributed, tamper-resistant, and shared state of data. The acceptance of an IOC is mainly derived from the publisher's reputation,

Table 6.1 Considerations for back-end storage options

	Public Blockchain	Permissioned Blockchain	Centralized Server
Degree of trust required	Low	High	High
Initial investment required	Low	Medium	High
Anticipated adoption rate	High	Medium	Low
Ongoing administrative effort	Low	Medium	High
Risk of malicious submissions	High	Low	Low

as opposed to the individual IOC data's reputation. Blockchain provides publicly verifiable evidence and strong attribution to the publishers. Based on the publisher's reputation, influence, and relevancy to their work, each member of the community would build a web of trust relevant to their work and environment.

In addition to maintaining a shared state, Blockchain also provides logic and functions through smart contracts. In the SIPshield framework, Blockchain does not defend against attacks directly. Rather, it serves as the back-end features that facilitate sharing of IOCs produced by the front-end components (shieldVNF). The smart contract implements these back-end features by defining the data storage, logic, and functions. Dynamic arrays are the data storage that is used to store IOC information. The smart contract also defines the logic to operate on this data, such as addition, deletion, and update. Finally, the smart contract provides functions that can be called remotely by the distributed applications to add, delete, or update IOC.

Besides the intrinsic features of Blockchain described previously, there are broader considerations concerning the choice of back-end technologies and their implications. There are three possible models where IOC sharing within a community can take place: public Blockchain, permissioned Blockchain, and a centralized server. Several factors to consider are the degree of trust required to make the collaboration work, the upfront investment required, the projected adoption rate, the ongoing administrative requirements, and the risk of malicious submissions. The table below compares the difference between these models:

Each model is presenting a different level of trust that is required for this collaboration to work. With a public blockchain, it is a trustless environment where anybody can participate. The trust in the community needs to be earned over time by making positive contributions. With a permissioned blockchain and a centralized server, the trust is implied where the com-

munity trusts the organizer's credibility and adheres to the rules imposed by this authoritative entity. One rule might be related to how and when the member can create and read IOC feed and how to verify the records' integrity. Another rule might impose fees expected from each member to offset the operation cost in exchange for the rights to submit and receive IOC feed. Complying with the prescribed rules present an investment that the members need to consider.

The investment required and rate of adoption are inversely interlinked. The upfront investments are those costs associated with implementing the new system so that they can participate. For example, the new members may need to purchase new hardware, software, administrative burden, and staff training. With a public blockchain, the investment requirement is low since there are no proprietary components involved and little administrative overhead required to start participating. With a permissioned blockchain and centralized server, the organizer might prescribe specific hardware, software, skill requirements, and procedures before participating. For permissioned Blockchain that utilizes publicly available Blockchain-as-a-Service infrastructure from Amazon, Azure, and IBM, it costs less than standing up a new permissioned blockchain network and makes it more affordable. For a centralized server, the organizer would need to build the infrastructure and customized application and network connectivity, which will incur a higher total cost than the other two models. For example, the smart contract feature that is available on the Blockchain provides the logic, storage, and functions that can be called remotely by the distributed applications over the Internet. In contrast, in a centralized server model, the organizer will need to build a custom application, database, and secure network connection for the remote database clients to connect to the server. The potential members would need to perform a cost-benefit analysis of these investments before joining. High overhead serves as a gatekeeper and negatively affects the overall adoption rate, while low overhead will promote the adoption among the potential members.

The ongoing administrative effort for each model is different and requires careful considerations. With the public blockchain model, the administrative tasks are assumed by the node operators. Similarly, for the permissioned blockchain-as-a-service model, the cloud providers (Amazon, Azure, IBM) perform the daily administrative tasks. However, with a centralized server, the responsibilities lie with the organizer to keep the system running, which requires strong business justification to keep this model sustainable for the long run.

The risks of malicious submissions are less with the permissioned Blockchain and centralized server model. With checks and balances performed before joining the community, the members would have been carefully screened, which significantly reduces the likelihood of intentional and malicious submissions from members. Public Blockchain presents a

greater risk in this aspect since the same security controls are not available. On the other hand, public Blockchain could create a community guideline and processes where many members are empowered to flag and remove malicious content, essentially crowdsourcing the effort to maintain IOC records' integrity and community asset. With more individuals involved in the auditing process and safeguarding the community asset, it is arguable that they could achieve the same security level as one or two staff employed by the centralized server model.

In the context of promoting early adoption in building a collaborative community, the public blockchain model offers more favorable characteristics. Based on its native features, public Blockchain and smart contracts lend more naturally towards a decentralized trust model (web of trust). The decentralized model also tends to have has less administrative overhead than the centralized model, which is attractive to potential members. Public Blockchain also offers more robust privacy protection than permissioned Blockchain or centralized server because the users are identifiable only by their blockchain address. This level of anonymity allows the users to share information about cybersecurity incidents without the fear of getting bad publicity. This privacy feature address one of the fundamental problems in sharing threat intelligence within a community.

Depending on the dynamics and the needs at a later stage, the community might want to evaluate other models and making the necessary adjustment. Whichever model best serves the community, it is essential to empower the defenders by disseminating IOC information. This approach would make it harder for the adversaries to exploit the silos among the defenders, perpetuate their attacks, and inflict damage to the network.

ShieldCHAIN provides a collaborative defence framework that is affordable, automated, and practical. The challenges identified earlier to collaborate against a botnet attack were lack of personnel, no common framework, and the heavy workload required to implement such a collaboration. From financial perspective, ShieldCHAIN is designed to use commodity Ethernet switches that are cost effective for member organisations to own. From a personnel involvement perspective, the mechanism where ShieldCHAIN shares and retrieves IOCs on a schedule provides automation that relieves the personnel from performing manual tasks. From a practical stand point, ShieldCHAIN allows the organisation to start small and increment deployment. With low-cost hardware, an automated framework, and flexible deployment, ShieldCHAIN presents a solution for the community to collaborate against a botnet attack.

In the next section we will look at the experiments that validate whether or not Shield-CHAIN can scale the defence from one organisation to multiple organisations in a community.

But first we will look at the design of experiments, followed by the implementation, then the tree experiments.

6.3 Design of Experiments

6.3.1 Sequence and Workflow

The design for experiments in this section is around inter-organisation IOC exchange facilitated by ShieldCHAIN so that different organisations can collaborate, i.e., share and retrieve IOCs. There are three experiments that we will look at in this section: creating a new IOC entry in Blockchain, retrieving IOC records from Blockchain, and installing packet filtering rules on ShieldVNF.

In the first experiment, organisation1 has mitigated the attacks in their network and would like to share IOCs with the community (organisation2 and organisation3). In this experiment, organisation1 is the contributor and will create new IOC transactions in the Blockchain. A positive outcome of this experiment is demonstrated by new records being created in the Blockchain.

In the second experiment, organisation2 and organisation3 would like to read the current state of threat intelligence. In this experiment, ShieldCHAIN queries the Blockchain to retrieve the current state of IOCs and store it in the SQL database. A positive outcome of this experiment is demonstrated by having the IOC stored in the database tables.

In the third experiment, organisation2 and organisation3 would like to use these IOCs as packet filtering rules at ShieldVNF. In this experiment, ShieldCHAIN will retrieve the IOCs from the database and install them as packet filtering rules at ShieldVNF. A positive outcome of this experiment is demonstrated by having packet filtering rules installed at ShieldVNF. Reaching this stage is significant because it demonstrates that ShieldCHAIN has scaled the defence from organisation1 to organisation2 and organisation3.

6.3.2 Independent and Dependent Variables

Each experiment has a set of independent and dependent variables that are captured in Table 6.2. These variables provide data points for analysis, and establish causality.

6.3.3 Data Collection and Measurements

In these experiments, ShieldCHAIN is the independent variable and the dependent variables are the number of IOCs that get distributed by ShieldCHAIN. The setup script generates 100

Table 6.2 Independent and Dependent Variables by Experiments

Experiment	Independent Variable	Dependent Variables
1. Automated IOC Distribution to Blockchain	ShieldCHAIN	# of new entries in Blockchain
2. Automated IOC Collection from Blockchain	ShieldCHAIN	# of new entries in Database
3. Automated Packet filter installation at ShieldVNF	ShieldCHAIN	# of new packet filter rules created at ShieldVNF

random IOC records at run time in order to remove bias. These IOCs serve as the measurable data for the experiments outlined in the previous section. A positive outcome is demonstrated by having successfully distributed 100 IOC records from the source organisation, and create 100 packet filtering rules on the switches of the receiving organisations.

ShieldCHAIN collects the data from three places: the SQL database, the smart contract, and the tables on ShieldVNF. From the previous chapter, ShieldSDN has created data (IOC records) in the database. From the perspective of the IOC producer (organisation1), its ShieldCHAIN retrieves these IOCs from the database and then pushes them to the smart contract. From the perspective of the IOC consumers (organisation2 and organisation3), their ShieldCHAIN pulls the IOCs from the smart contract, stores them in SQL database, and installs them as packet filtering rules in their ShieldVNF's table.

Data measurement is performed by counting the number of IOCs that are generated by ShieldCHAIN. For the first experiment, we measure the number of IOC records that get created in Blockchain. We could use the online Blockchain explorer tool called Etherscan (Etherscan, 2020) for verification. For the second experiment, we measure the number of IOC records that get created in the SQL Database by using the SQL client. For the third experiment, we measure the number of packet filtering rules that are created on ShieldVNF.

The data is measurable end-to-end from the database in the source organisation to the packet filtering table in the receiving organisations. These chain of experiments produce measurable data at each step and designed to build on top of each other, so that any failure during the ShieldCHAIN process can be observed and identified clearly. Positive or negative outcomes of the experiment are validated by the number of IOC records that need to be observable during each stage of the experiment. Having described the data collection and measurement, the implementation of ShieldCHAIN will be discussed next.

6.4 Instrumentation and Implementation

There are three components involved in the implementation of ShieldCHAIN: smart contract, dApp, and API to access Ethereum Blockchain. The smart contract contains the logic, data structure, and functions for IOC management on the Ethereum Blockchain, whereas the dApp provides the front-end or client interface for the smart contract, database, and ShieldVNF. The API makes it practical for users to interface with the Ethereum Blockchain so that the users do not have to build and maintain an Ethereum Blockchain node of their own.

6.4.1 Smart Contract

A smart contract is a code that gets deployed to Blockchain that processes how the data are received, stored, and shared. The data are 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 that have been 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.

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. 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 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 for ShieldSDN to retrieve all IOC in the smart contract, ShieldSDN will call the `getAll()` function.

Table 6.3 Smart contract and its address at Kovan network

Smart Contract	Address at Kovan Network
Attacker	`0x9FACE6372e53B5Cc61848340AA194E709773CEa4`
CNC	`0x53D721ebA7Db1c8e8e54b878D79340C74B4900A7`
Victim	`0x31C39540518B1a337D9E43aE492AE5AC7e84c305`

ShieldCHAIN deploys three smart contracts to hold different information: KnownAttacker, KnownVictim, and KnownCNC. Table 6.3 lists the addresses of 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 smart contracts at the same time. As the name implies, KnownAttacker smart contract holds information about the attacker's source IP address and port number 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 about 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 organisation from participating in the SIP DoS attack (assuming 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 so that it is ready for accepting calls from the dApp, which is the topic of the next discussion.

6.4.2 DApp

A dApp is a node.js script that interfaces with the smart contract, SQL database, and ShieldVNF. Every organisation that wants to participate in this collaborative defence framework needs to run a copy of dApp. This dApp allows the organisation 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 organisations. Each organisation 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 is using 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. The functions are listed in Table 6.4. The data structure is a dynamic array of strings that store one IOC in each element.

Table 6.4 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

The dApp interacts with the SQL database to retrieve and insert IOC records that it received 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 requires interaction between dApp and 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 is interfacing with ShieldVNF to install packet filtering rules so that the switch can drop malicious packets. The other critical function that the dApp performs is interfacing with the ShieldVNF 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 ShieldVNF. The dApp will install packet filtering rules by using command line tools `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. Similarly, the dApp also uses the same command line tool to remove expired packet filtering rules from the switch.

6.4.3 Infura: REST API for Ethereum

Instead of calling the smart contract on Ethereum Blockchain directly, the dApp interacts with the smart contract through Infura (Infura Inc, 2020). Infura is an online service that provides REST API access to the smart contract that is 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 implementation 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 method is to lower down the technical and logistical barriers for accessing the Blockchain so that we can maximise participation from many organisations that would like to get involved.

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 contracts. The URLs that are accessible by the dApp are listed in Table 6.5. 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.

Table 6.5 End-points for accessing ShieldCHAIN smart contracts

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

6.5 Experiments

The experiments in this section validate the hypothesis that ShieldCHAIN can scale the defence from one organisation to multiple organisations in the community. ShieldCHAIN does this by sharing threat intelligence from one organisation with a broader community so that the members are able to drop malicious packets based on the intelligence learned from the community. In essence, ShieldCHAIN replicates the success of organisation1 to organisation2 and organisation3. In this collaborative framework, attack detection and mitigation efforts performed by one organisation can be leveraged as an attack prevention method by other organisations in the community.

There are three experiments in total that follow the sequence and worklow as described in Table 6.6. The setup script will prepare the environment with 100 IOC records in organisation1's SQL Database. Experiment 1 will validate that ShieldCHAIN is able to retrieve these IOC records and create the corresponding records in Blockchain. In experiment 2, we

will validate that the ShieldCHAIN in organisation2 and organisation3 are able to retrieve these IOC records from the Blockchain and store them in their SQL database. In experiment 3, ShieldCHAIN will use these IOCs and insert them as packet filtering rules on their ShieldVNF.

Table 6.6 Experiments for ShieldCHAIN: Sharing IOC to multiple external organisations

Experiments (IOC distribution management)	**Sample population** (random selection random assignment)	**Independent Variable**	**Dependent Variable**
Environment Setup	100 Known Attackers 100 Known Victims 100 Known CNCs	Setup Script in Org1	# of register entries # of table entries
1. IOC sharing to Blockchain	100 Known Attackers 100 Known Victims 100 Known CNCs	ShieldCHAIN in Org1	# of Blockchain entries
2. IOC retrieval from Blockchain	100 Known Attackers 100 Known Victims 100 Known CNCs	ShieldCHAIN in Org2 and Org3	# of database entries
3. Packet filter installation to ShieldVNF	100 Known Attackers 100 Known Victims 100 Known CNCs	ShieldCHAIN in Org2 and Org3	# of register entries # of table entries

6.5.1 Environment Setup

Procedures

A python script prepares the environment in organisation1 with IOC records that will be used for subsequent experiments. This script simulates a situation where ShieldVNF1 has produced 100 IOC records for attackers, victims, and CNC. In addition, this script also creates 100 records in ShieldSDN's SQL Database tables (victim table, attacker table, and CNC table). Figure 6.3 depicts the before and after of the setup script. The ShieldSDN icon is highlighted to indicate that it is loaded with IOC records.

Result

The result of environment setup procedure is captured in the Table 6.7 below. Figure 6.4 shows that there are 100 packet filtering rules in KnownAttacker_table, KnownVictim_table, and KnownCnc_table. Figure 6.5 shows that there are 100 records in the SQL Database tables (attacker, target, and cnc).

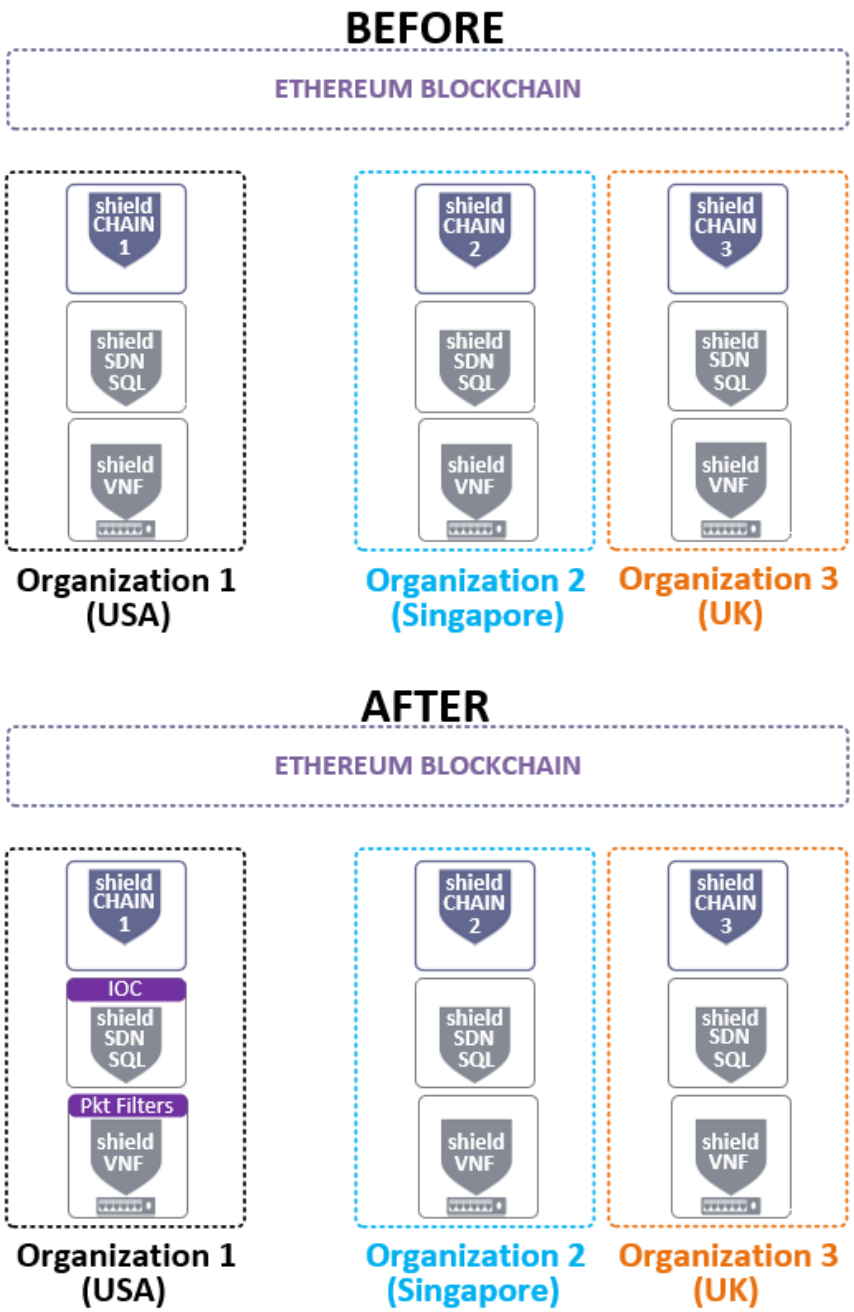


Fig. 6.3 **Setup:** The setup script created IOC entries in ShieldSDN database tables. These IOC entries set the stage for ShieldCHAIN to retrieve them from SQL database and share them with the community via the smart contract on the Ethereum Blockchain.

Discussion

The setup script has prepared the necessary environment for the experiments that will follow shortly. The script has inserted 100 IOCs records for each type (attackers, CNC, and target)

Table 6.7 **Environment Setup:** Preparing the environment for IOC distribution to external organisations. (B=Before; A=After)

Dependent Variables	VNF1 @ Org1		SDN1 @ Org1		Blockchain		VNF4 @ Org2		SDN4 @ Org2		VNF5 @ Org3		SDN5 @ Org3	
	B~	A	B	A	B	A	B~	A	B	A	B	A	B	A
#IOC in **register**	0	100												
#Filters in **table**	0	100												
#IOC in **SQL**	0	100												
#IOC in **Blockchain**														

```
RuntimeCmd: table_num_entries KnownAttacker_table
100
RuntimeCmd: table_num_entries KnownVictim_table
100
RuntimeCmd: table_num_entries KnownCnc_table
100
RuntimeCmd:
```

Fig. 6.4 **Setup:** Packet filtering rules are loaded to the tables on ShieldVNF.

```
sqlite> SELECT COUNT(*) FROM attacker;
100
sqlite> SELECT COUNT(*) FROM target;
100
sqlite> SELECT COUNT(*) FROM cnc;
100
sqlite>
```

Fig. 6.5 **Setup:** SQL database tables are loaded with IOCs, ready to be distribution by ShieldCHAIN.

at ShieldVNF. In addition, the script also inserted 100 IOC entries at the ShieldSDN database tables (attackers, CNC, and target). At the end of the experiment 3, we will compare the number of entries at ShieldVNF1 in organisation1, versus ShieldVNF4 in organisation2, and ShieldVNF5 in organisation3. A positive outcome of the experiment is demonstrated when the number of entries at the originating organisation (ShieldVNF1) is the same as the number of entries at the receiving organisations (ShieldVNF4 & ShieldVNF5).

6.5.2 Experiment 1: Automated IOC Distribution to Blockchain

This first experiment is to answer the first research question, i.e., What strategies could be adopted to share threat intelligence with the community? If the ShieldCHAIN in organisation1 is working as designed, then it will create IOC records on the Blockchain that are accessible by the community.

Procedures

The process started with ShieldCHAIN in organisation1 making the SQL query to the database for the IOCs. Once the IOCs were retrieved, the ShieldCHAIN created 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 target) that the member organisations can retrieve.

Result

The result of this experiment is captured in the Table 6.8 below. Before the experiment, there are 0 records in the Blockchain. After the experiment, there are 100 records present. Figure 6.6 shows the before and after diagram.

The experiment took 1552 seconds (25.86 minutes) to run. Figure 6.7 shows the CPU utilisation and Figure 6.8 shows the memory utilisation of ShieldCHAIN during the experiment.

Table 6.8 **Experiment 1:** ShieldCHAIN queries IOCs from the Database and adds it to Ethereum Blockchain.
(B=Before; A=After)

Dependent Variables	VNF1 @ Org1		SDN1 @ Org1		Blockchain		VNF4 @ Org2		SDN4 @ Org2		VNF5 @ Org3		SDN5 @ Org3	
	B~	A~	B	A	B	A	B~	A	B	A	B	A	B	A
#IOC in **register**														
#Filters in **table**														
#IOC in **SQL**														
#IOC in **Blockchain**					0	100								

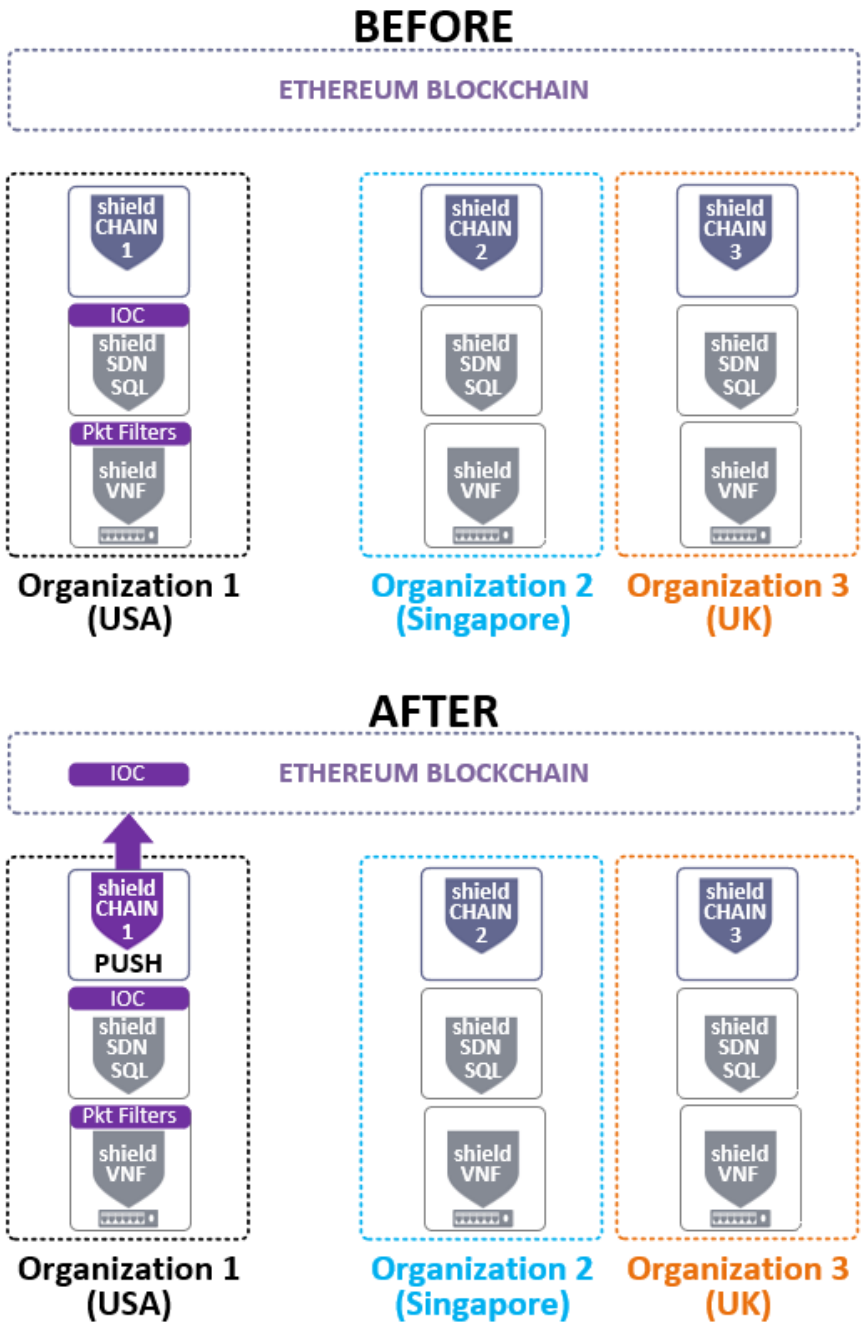


Fig. 6.6 **Experiment 1:** ShieldCHAIN queries IOCs from the Database and adds it to Ethereum Blockchain.

Discussion

ShieldCHAIN in organisation1 retrieved IOC records from the database and created 100 IOCs for each smart contract in the Blockchain. Figure 6.9 shows the IOC records that were created on Victim, CNC, and Attacker smart contracts. The public can use an online

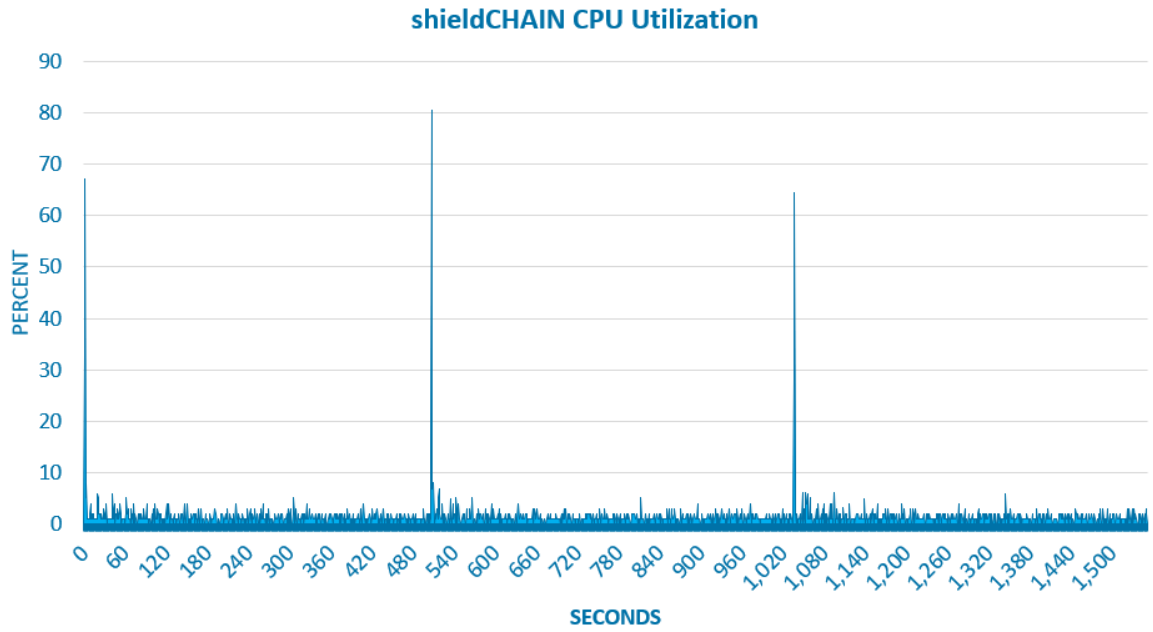


Fig. 6.7 **Experiment 1:** CPU utilisation of ShieldCHAIN for the experiment 1

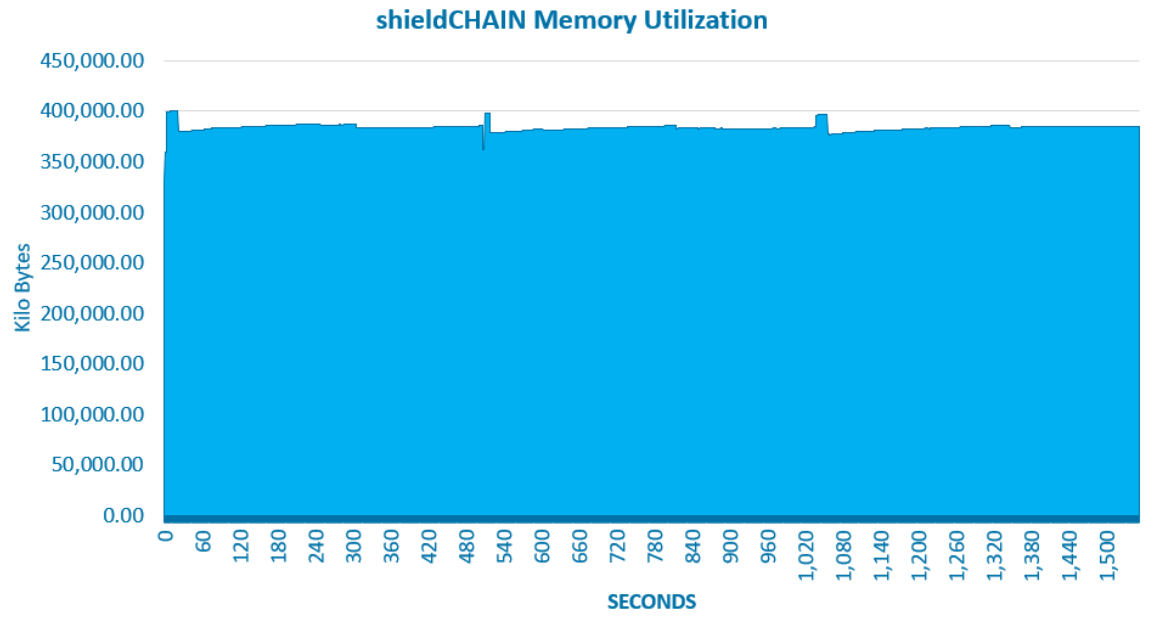


Fig. 6.8 **Experiment 1:** Memory utilisation of ShieldCHAIN on for the experiment 1

blockchain explorer (Etherscan.io) to visually verify that these IOC records were created. Figure 6.10 shows a screenshot of Etherscan.io that list the transactions created to store the Attacker records. In Figure 6.11 we see a similar screenshot that shows the IOC records for CNC, whereas in Figure 6.12 we have the IOC records for Target. As an example, the content of this transaction is shown in Figure 6.13 where it shows the IOC in the input data field.

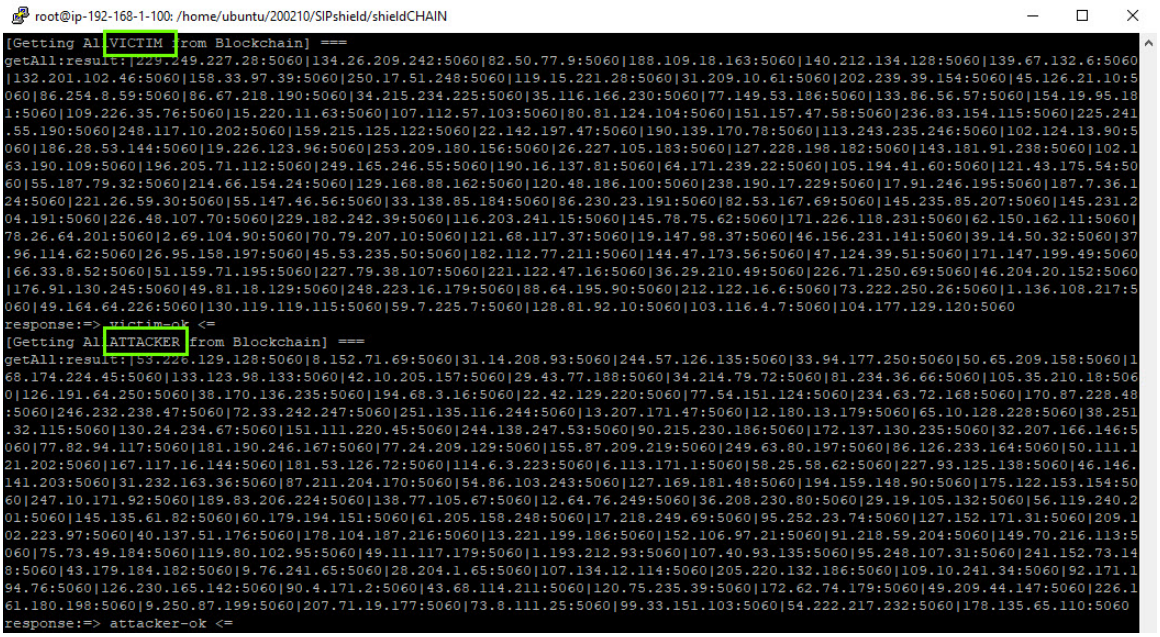


Fig. 6.9 **Experiment 1:** ShieldCHAIN has created 100 IOC records for Attacker and Victim at Blockchain.

Blockchain took about 5 seconds on average to create a transaction. 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). This is attributed to the tasks involved for the miners to create a new transaction and append it to a block. This transaction 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 the use case of 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 organisation1 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 organisation to other organisations in the community.

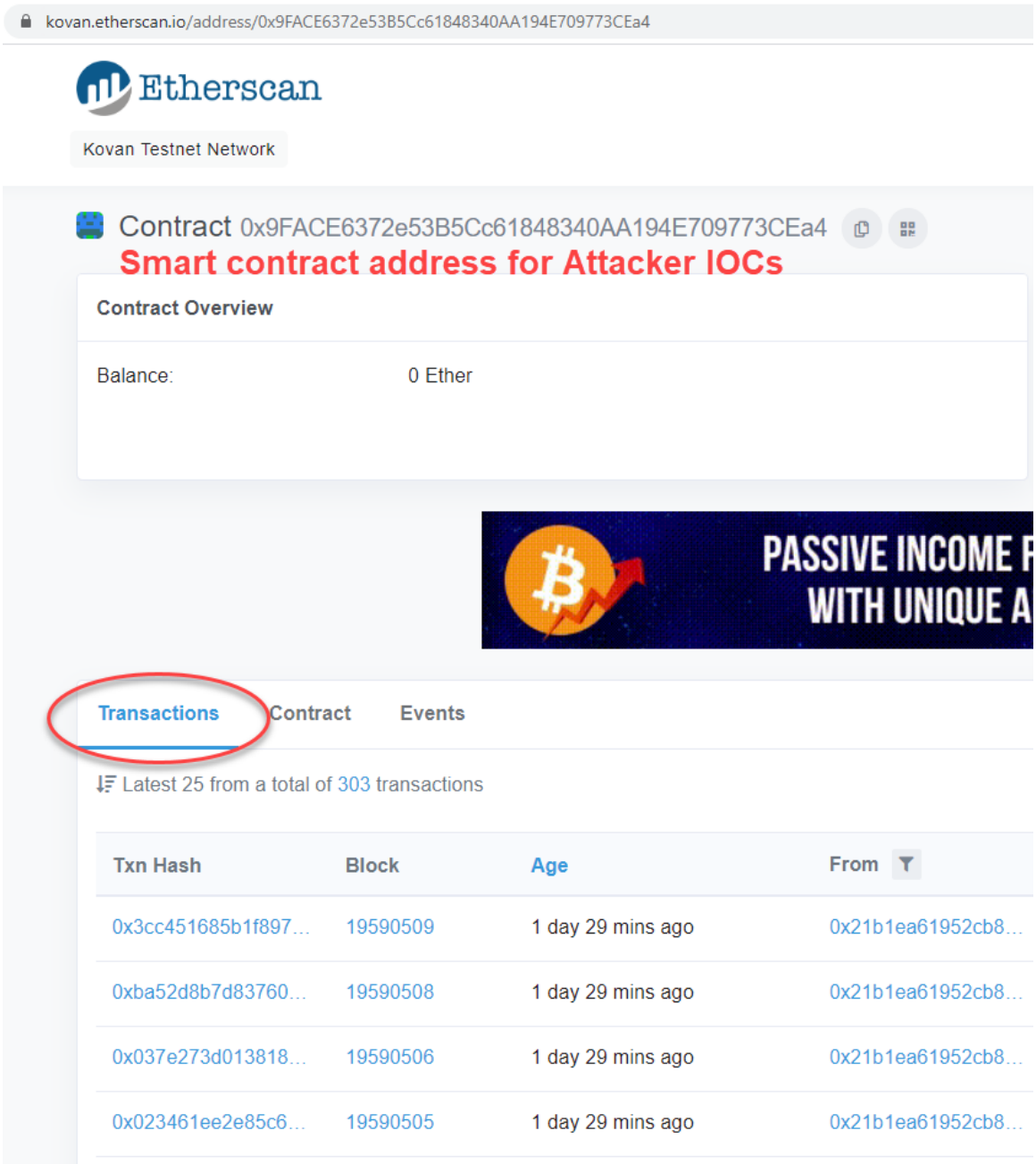


Fig. 6.10 **Experiment 1:** Attacker smart contract shows the transactions that were created for IOC records.

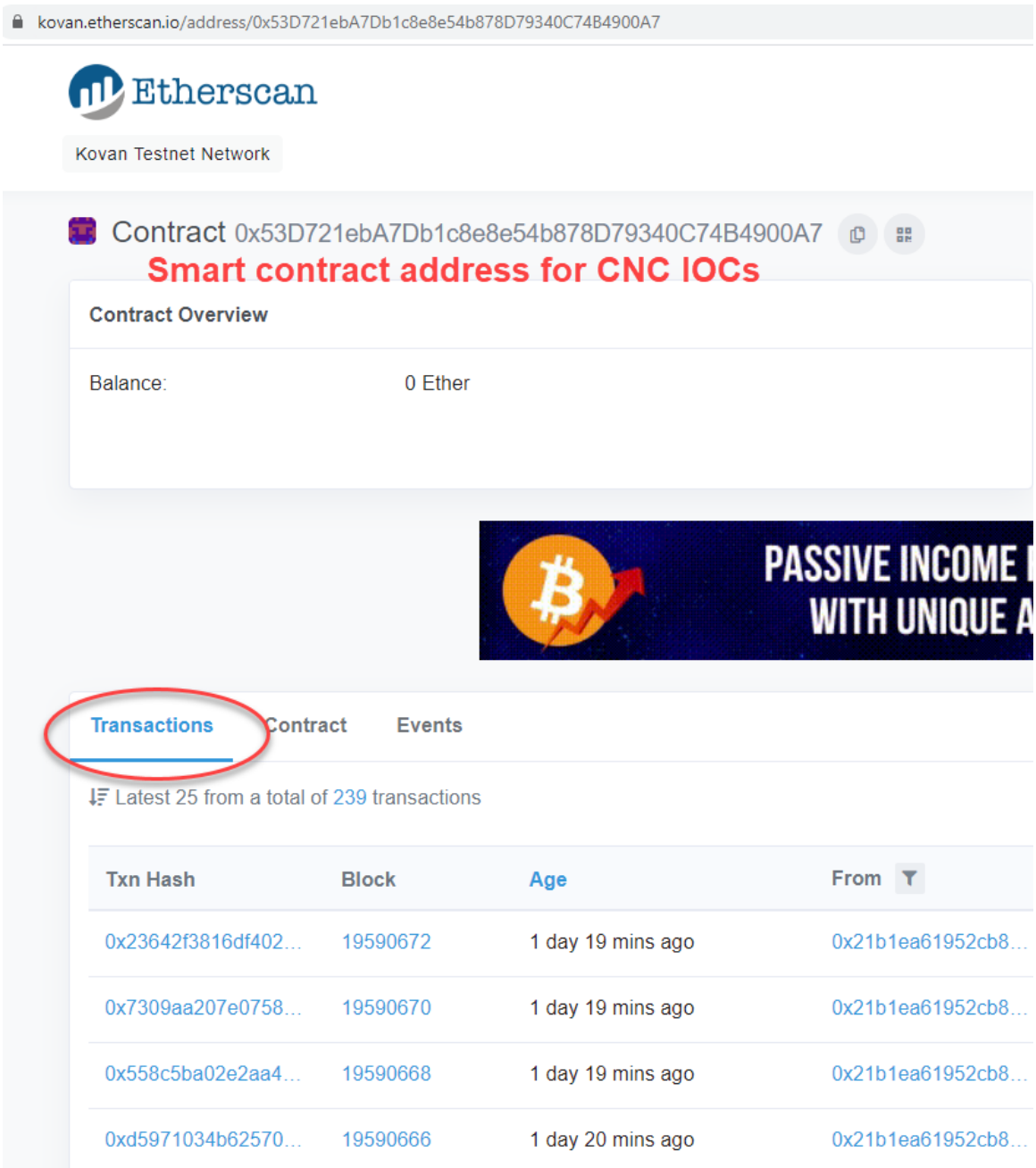


Fig. 6.11 **Experiment 1:** CNC smart contract shows the transactions that were created for IOC records.

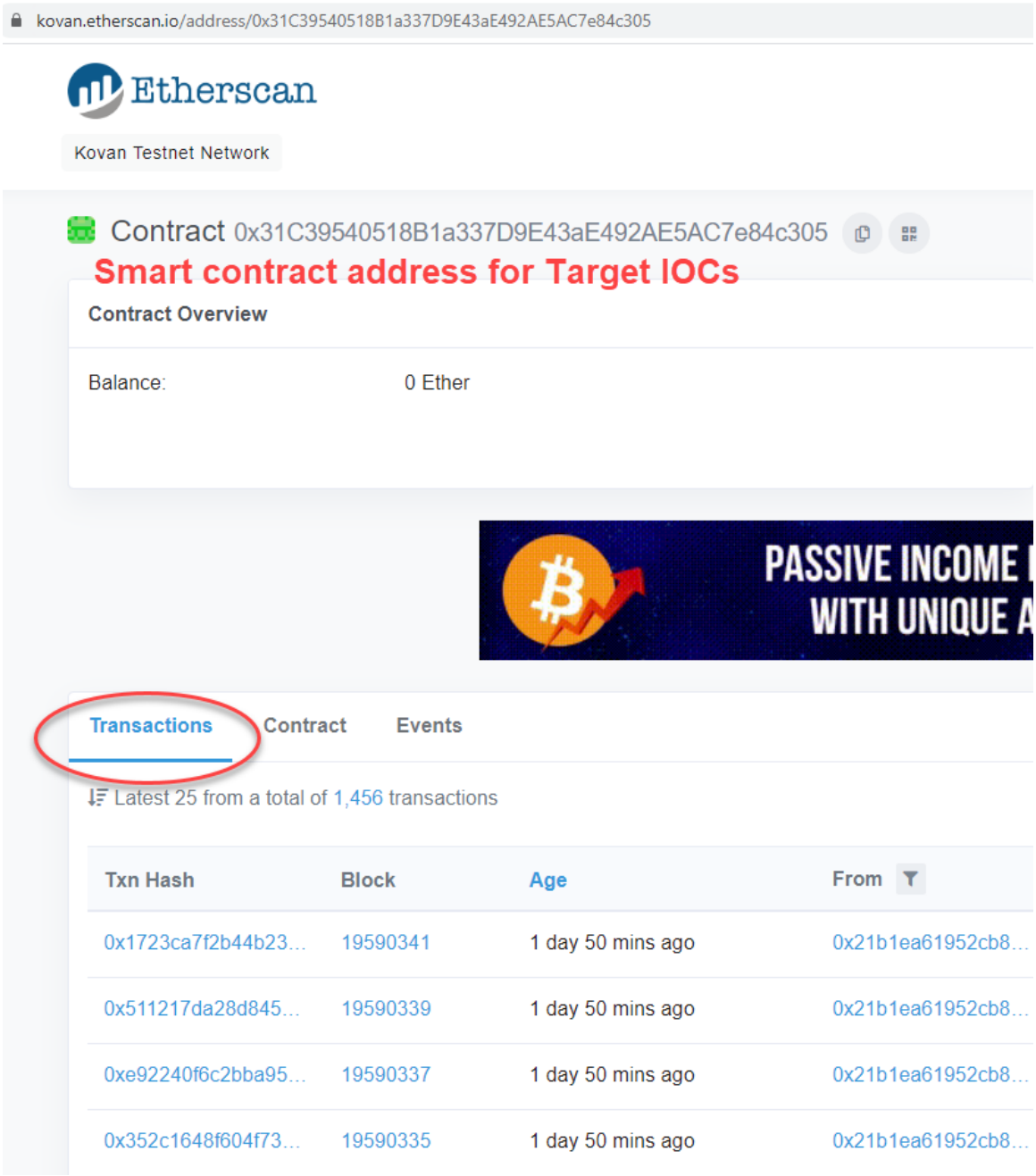


Fig. 6.12 **Experiment 1:** Target smart contract shows the transactions that were created for IOC records.

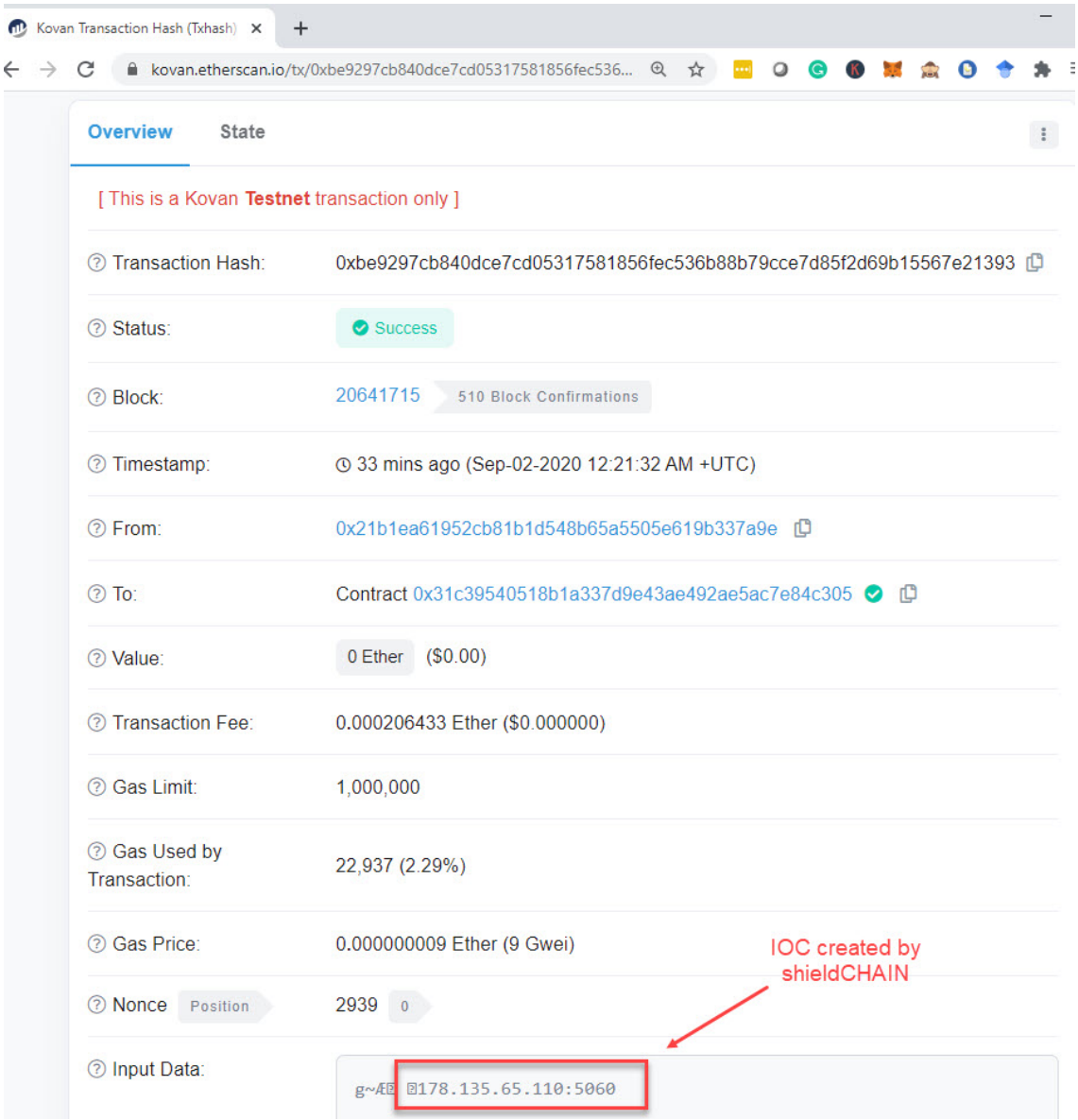


Fig. 6.13 Verifying new record that just created by ShieldCHAIN. This record shows the IOC (IP address: Port) that was created by ShieldCHAIN.

6.5.3 Experiment 2: Automated IOC Collection From Blockchain

The second experiment is to answer the second research question, i.e., What strategies could be adopted to retrieve threat intelligence from the community? If shielCHAIN in organisation2 and organisation3 is working as designed, then it 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.

Procedures

The process started with ShieldCHAIN in organisation2 and organisation3 calling a function in a smart contract to retrieve all IOCs. Once the IOCs were retrieved, ShieldCHAIN inserted the IOC into the appropriate table in their ShieldSDN's database tables.

Result

The result of this experiment is captured in Table 6.9 below. The experiment took 18 seconds to run. Figure 6.15 shows the CPU utilisation and Figure 6.16 shows the memory utilisation of ShieldCHAIN during the experiment.

Table 6.9 **Experiment 2:** ShieldCHAIN2 and ShieldCHAIN3 retrieves IOCs from Blockchain and inserts it to their ShieldSDN database.
(B=Before; A=After)

Dependent Variables	VNF1 @ Org1		SDN1 @ Org1		Blockchain		VNF4 @ Org2		SDN4 @ Org2		VNF5 @ Org3		SDN5 @ Org3	
	B	A~	B	A	B	A	B	A	B	A	B	A	B	A
#IOC in **register**														
#Filters in **table**														
#IOC in **SQL**									0	100			0	100
#IOC in **Blockchain**														

Discussion

ShieldCHAIN that was running in organisation2 and organisation3 was able to retrieve all 100 IOC records that were produced by organisation1, a member of this community. After

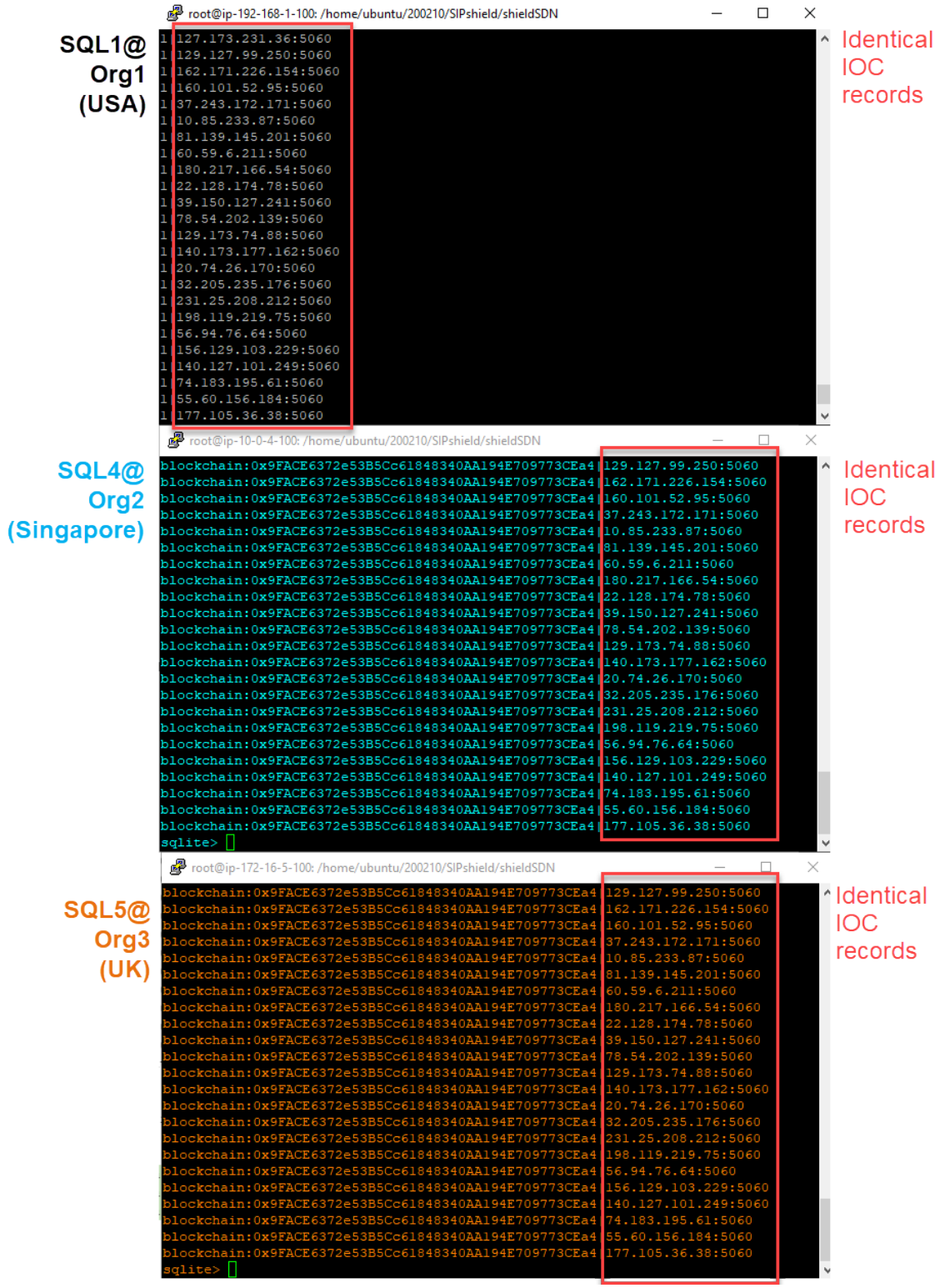


Fig. 6.14 **Experiment 2:** Identical IOC records in SQL database in organisation1(USA), organisation2(Singapore), and organisation3(UK).

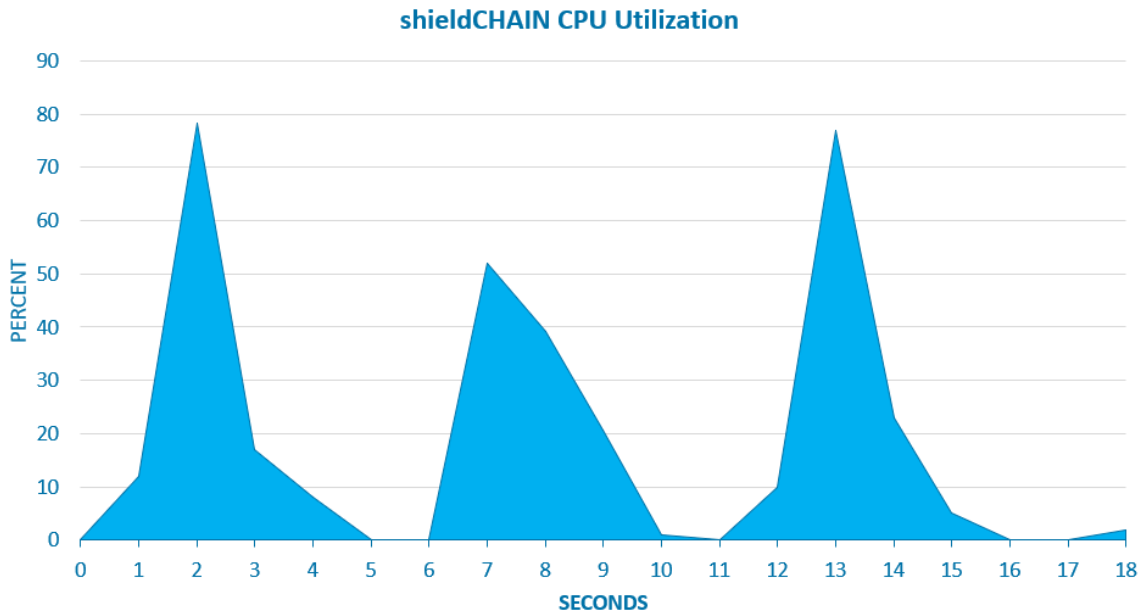


Fig. 6.15 **Experiment 2:** CPU utilisation of ShieldCHAIN for the experiment 2

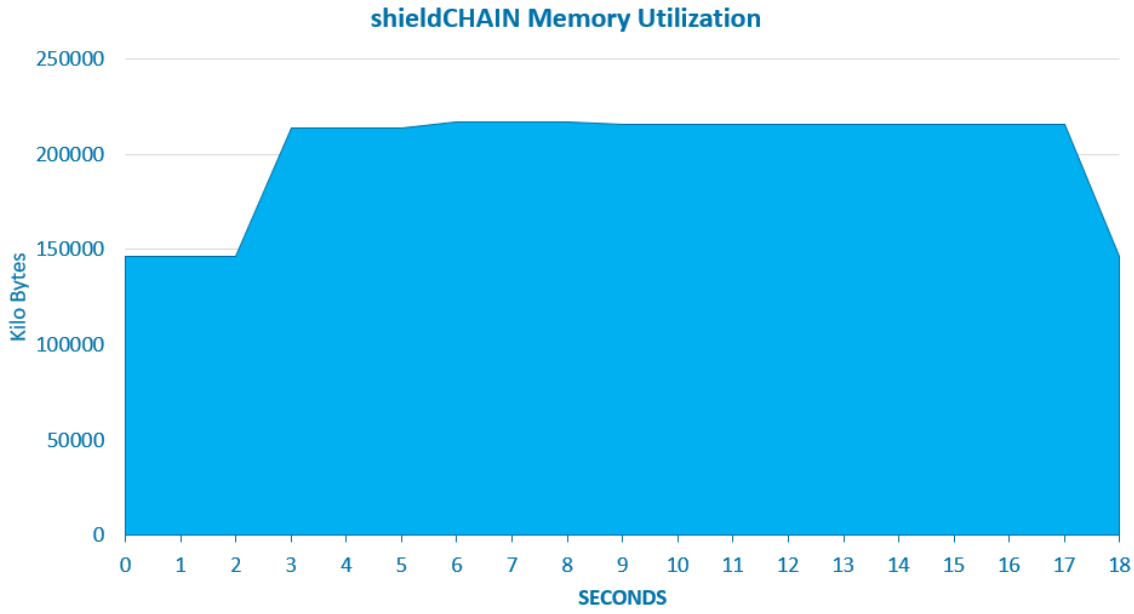


Fig. 6.16 **Experiment 2:** Memory utilisation of ShieldCHAIN for the experiment 2

retrieving IOC records from the Blockchain, ShieldCHAIN then inserted these IOCs in the ShieldSDN database tables (attacker, CNC, and target). Figure 6.21 shows the before and after of the experiment from the SQL perspective in organisation2 and organisation3. Before the experiment, there were 0 records in these three tables. After the experiment, there were 100 records created.

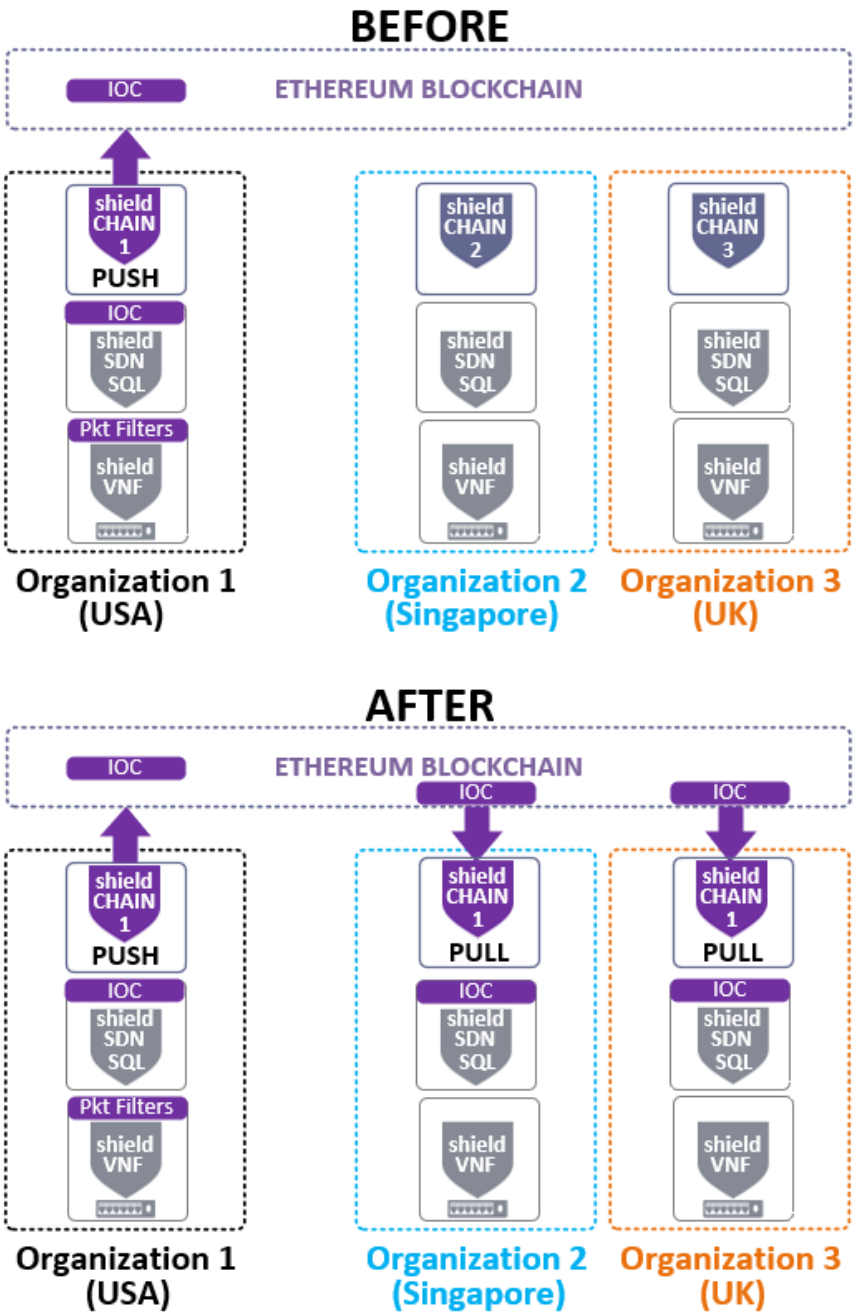


Fig. 6.17 **Experiment 2:** ShieldCHAIN2 and ShieldCHAIN3 retrieves IOCs from Blockchain and inserts it to their ShieldSDN database.

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. As such the experiment ran much faster than the previous one. Even though the speed for the read operation is still slower than the regular

SQL database, the Blockchain offers unique features that are appropriate for exchanging IOCs between organisations.

This screenshot from the SQL perspective shows that the IOC records that were originated by the community have been successfully retrieved and inserted into the database. 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 organisation2 and organisation3 are able to get these IOCs is significant because they are now prepared to mitigate attacks that match with the IOCs. In essence, organisation2 and organisation3 is able to leverage the detection effort performed by organisation1 for attack prevention in their respective organisations. This approach alleviates the need for organisation2 and organisation3 to repeat detection efforts again for the same attacks. This is how the community collaborates to defend against IoT botnet attacks.

6.5.4 Experiment 3: Automated Packet Filter Installation at ShieldVNF

The third experiment is to answer the third research question, i.e., In what ways could a switch leverage the community-sourced intelligence as packet filtering rules? If ShieldCHAIN in organisation2 and organisation3 is working as designed, then it will be able to retrieve IOC records from the SQL Database then install these as packet filtering rules on ShieldVNF.

Procedures

The process started with ShieldCHAIN queries to the ShieldSDN database for IOCs records with query parameter set as "`switchNo` = blockchain". This search criterion will retrieve only those IOCs that originated from the Blockchain. Next, ShieldCHAIN converts these IOC records and installs these as packet filtering rules on ShieldVNF tables.

Result

The result of this experiment is captured in Table 6.10 below. On the ShieldVNF in organisation2 and organisation3, there were 0 packet filtering rules before the experiment and 100 rules after the experiment.

The experiment took 80 seconds to run. Figure 6.19 shows the CPU utilisation and Figure 6.20 shows the memory utilisation of ShieldCHAIN during the experiment.

Figure 6.18 shows that before the experiment, the IOCs were stored in the SQL database. After the experiment, these IOCs were installed as packet filtering rules on ShieldVNF. The same result was observed in organisation2 and organisation3.

Figure 6.21 shows the content of packet filtering rules on the ShieldVNF in organisation1 (USA), organisation2 (Singapore), and organisation3 (UK). The packet filtering rules installed on all switches are identical.

Discussion

The ShieldCHAIN that was running in organisation2 and organisation3 was able to retrieve all 100 IOC records from the SQL database and installed them as packet filtering rules on ShieldVNF. The experiment showed that ShieldCHAIN was able to utilise threat intelligence that was generated by the community and created packet filtering rules from this intelligence. ShieldCHAIN then installed these packet filtering rules on ShieldVNF tables. Out of 100 records of threat intelligence in the database (which were sourced from the community),

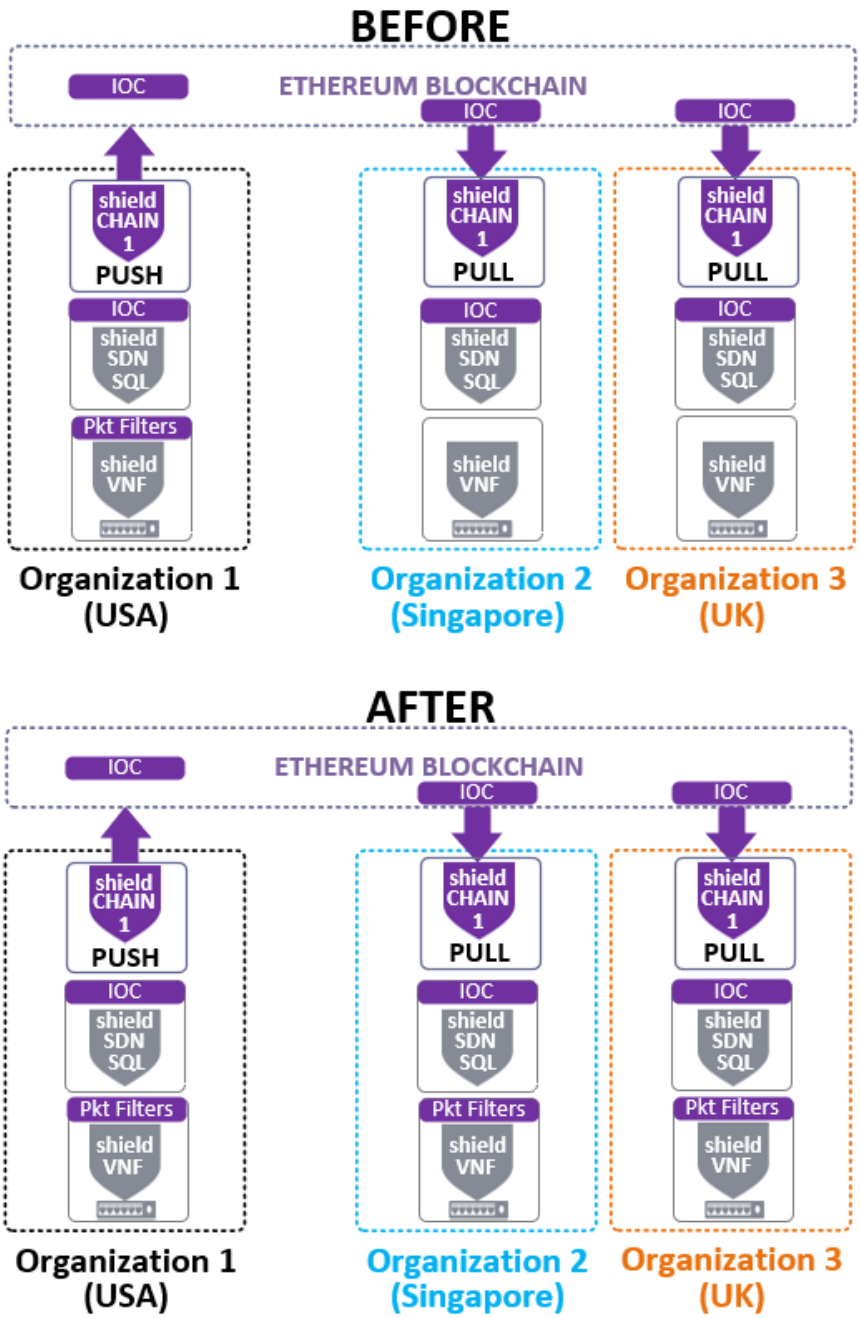


Fig. 6.18 **Experiment 3:** ShieldCHAIN queries ShieldSDN database to retrieve IOCs that originated from Blockchain, converts, and inserts it to ShieldVNF as packet filtering rules.

ShieldCHAIN was able to install 100 packet filtering rules on ShieldVNF. This is statistically significant and demonstrates a positive outcome for this experiment.

ShieldVNF from organisation1 (in the USA), ShieldVNF from organisation2 (in Singapore), and ShieldVNF from organisation3 (in UK) have identical packet filtering rules. As shown in Figure 6.21, identical packet filtering rules means that these three switches

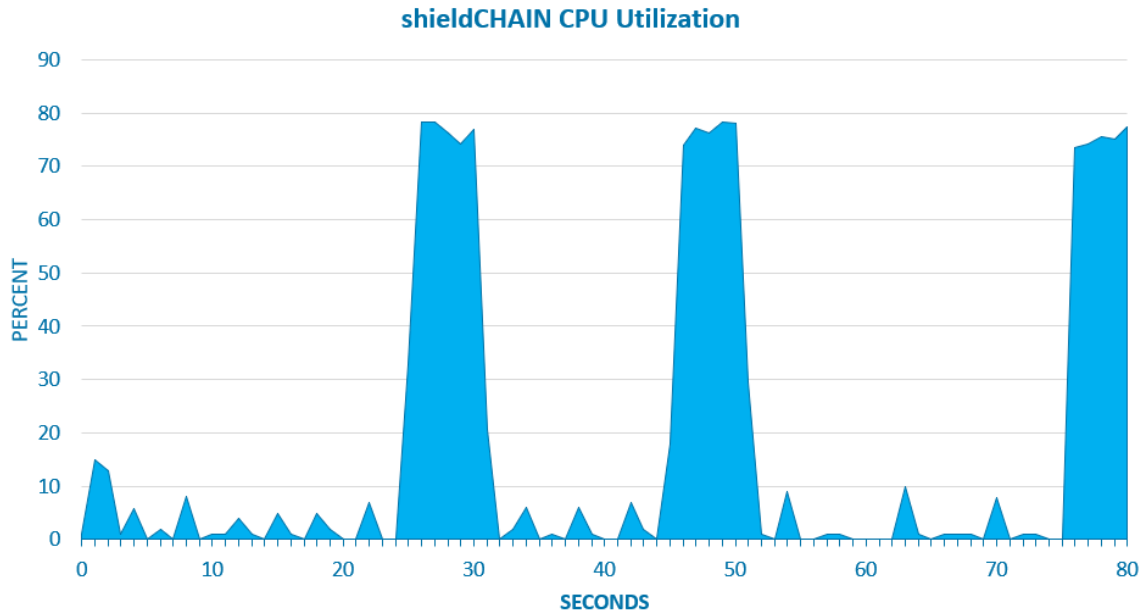


Fig. 6.19 **Experiment 3:** CPU utilisation of ShieldCHAIN on for the experiment 3

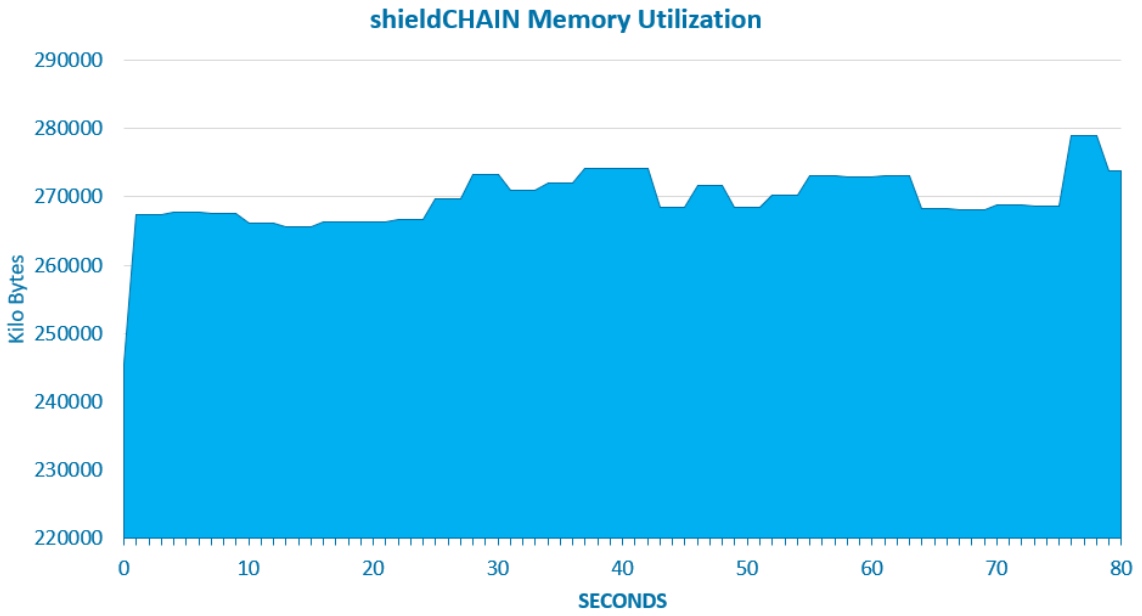


Fig. 6.20 **Experiment 3:** Memory utilisation of ShieldCHAIN on for the experiment 3

are synchronised in their defence posture even though they are owned by three different organisations. This also means that they will be able to mitigate against the same attack should the same botnet outbreak occur in both organisations.

Table 6.10 **Experiment 3:** ShieldCHAIN queries ShieldSDN database to retrieve IOCs that originates from Blockchain, converts, and inserts it to ShieldVNF as packet filtering rules. (B=Before; A=After)

Dependent Variables	VNF1 @ Org1		SDN1 @ Org1		Blockchain		VNF4 @ Org2		SDN4 @ Org2		VNF5 @ Org3		SDN5 @ Org3	
	B~	A	B	A	B	A	B~	A	B	A	B	A	B	A
#IOC in **register**														
#Filters in **table**							0	100			0	100		
#IOC in **SQL**														
#IOC in **Blockchain**														

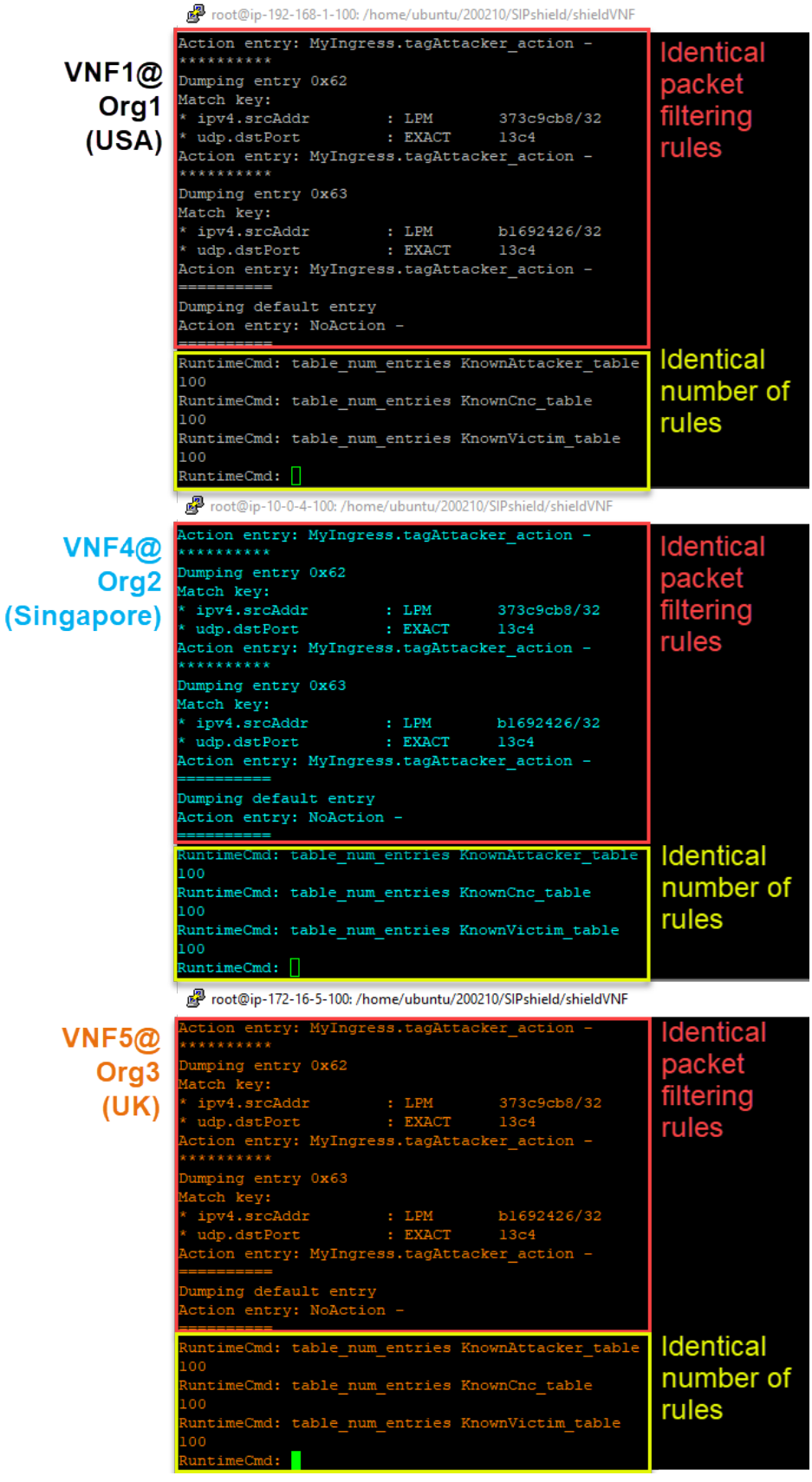


Fig. 6.21 **Experiment 3:** ShieldCHAIN created identical packet filtering rules on VNF in organisation2 (Singapore) and organisation3 (UK).

6.6 Chapter Summary

The collaboration of ShieldCHAIN in multiple organisations has met the objective of this chapter to scale the defence from one organisation (organisation1) to multiple organisations (organisation2 and organisation3). What started as packet filtering rules on ShieldVNF in organisation1, allowed ShieldCHAIN in organisation1 to be able to share these IOC records to the Blockchain so that they were available for the community. This finding answers the first research question.

Then the ShieldCHAIN in organisation2 and organisation3 was able to retrieve threat intelligence from the community and store it in their SQL database tables. In this case, ShieldCHAIN in both organisations provided an answer to the second research question. Further, ShieldCHAIN in organisation2 and organisation3 was able to install community-sourced intelligence as packet filtering rules on ShieldVNF in their organisation. This finding answers the third research question. At the end of this process, end-to-end, packet filtering rules that were originated from ShieldVNF in organisation1 were installed in ShieldVNF in organisation2 and organisation3 which means that they are now synchronised and will be able to reduce the risk of attacks from the same attacker.

While Blockchain technology is showing real potential, the components that make up a solution (e.g., the programming language, compiler, tools, libraries) are still under active development. The teams that built these components work independently from each other. We need to pay close attention to the dependencies of each component that we use in our Blockchain applications because just one of these components may introduce a change that may compromise a previously developed solution to the problem. For example, when a team changed a function name in a library, this change could cause unresolved dependency, and therefore, a compilation error.

Solidity, as a smart contract programming language, is still maturing and has its unique characteristics when compared to other modern languages (ethereum, 2020b; Wohrer and Zdun, 2018). It requires a manual workaround to perform typical array operations such as update or delete. It does not have high-level string operations such as concatenation, extraction. This limitation requires the developer to do it manually using the lower-level constructs of the language. Besides, the web3 library (ethereum, 2020a) and truffle framework (Truffle Suite, 2020) also have some restrictions on their own. For example, while Solidity supports returning strings as part of the dynamic array, web3 does not. This intricacy between the language and the library imposes additional challenges.

It is necessary to balance the requirements against Blockchain characteristics in order to gain the optimum use of Blockchain technology. For example, different operations have a different associated cost, forcing the developer to be mindful of things like data structure,

storage, and the algorithm that contributes to the overall cost. The Ethereum Yellow paper Appendix G published the fee schedule(Wood, 2019).

Speed is another essential factor for consideration. Due to its nature, the miners need to mine each Blockchain transaction before it gets appended to a block. This mining operation requires more time than the traditional database insert operation. As such, the Ethereum clients (or the dApp) need to accommodate this type of asynchronous operation to ensure proper timing and interaction between the client and the Ethereum network. In our experiment, it took about 7 seconds to create a new insert transaction in the Blockchain (from 21:27:41 to 21:27:48). It took about 3 seconds to retrieve the data. The performance difference between writing and reading is expected because of the level of operation involved. With the writing operation, the Blockchain network has to perform computational operations to create and append a new block.

Chapter 7

Conclusion

7.1 Overview

The aim of this research was to investigate a scalable defence framework for the SIP DDoS Flooding attack from IoT botnets. This chapter concludes this dissertation by reviewing the findings, contributions to knowledge, current limitations, and future work.

7.2 Research Findings

The research question in this dissertation is how to defend against SIP DDoS Flooding attack from IoT botnets. The thesis is that a SIP DDoS Flooding attack from an IoT botnet can be prevented, detected, and mitigated by securing the edge of the IoT networks using a scalable framework that uses VNF, SDN, and Blockchain technology.

Analysis from the literature exposes the gaps and identifies the requirements for the proposed solution. Being inherently weak in their security posture, IoT devices are vulnerable to being hijacked and converted into botnets. Distributed DoS attacks launched from botnets expose the gaps in legacy defence models which are easily overwhelmed and, therefore, not sustainable models moving forward. These gaps suggest that distributed attacks from botnets require distributed defences. To facilitate a distributed defence model, a common framework is required that governs how the members collaborate. The framework needs to be modular, practical, and accessible in terms of financial and personnel requirements to maximise members' participation. These insights and requirements have led to the proposed solution, SIPshield: a scalable SIP DDoS defence framework built upon VNF, SDN, and Blockchain technology.

Discussion on our findings are organised around the three research objectives that support the research aim. The first objective is to develop mechanisms for prevention, detection, and mitigation of SIP DDoS Flooding attacks on a single switch. The idea is, first, to secure the network edge where IoT devices are connected, then scale the success to multiple switches in the organisation, and eventually, to include switches in external organisations. Securing a single switch is achieved by ShieldVNF in chapter 4. ShieldSDN explained in chapter 5 achieved the second objective, i.e., to scale the defence from one switch to multiple switches. The third objective is to develop mechanisms to scale the defence from one organisation to multiple organisations. This objective is achieved by ShieldCHAIN as discussed in chapter 6. Experiments were performed to validate each hypothesis and below are the findings from each component.

7.2.1 ShieldVNF: SIP DDoS Defence At The Network Edge

The first objective was to develop mechanisms for prevention, detection, and mitigation mechanisms for SIP attacks on a single switch. The findings support the hypothesis that ShieldVNF is able to secure a switch by performing preventive and reactive measures against IoT botnet attacks. The preventive measures prevent IoT devices from being recruited to become part of a botnet in the first place. Should the preventive measure failed, the reactive measures prevent the botnet from being able to connect to its command-and-control server, and therefore, prevent the botnet to launch SIP DoS and Distributed DoS attacks. Blocking command-and-control connection is significant for two reasons. First, a CNC connection attempt informs the network operator about the presence of a botnet in the network. Second, without connection to the CNC server, the botnet is not going to be able to register and receive commands for launching attacks. Should the botnet still able to evade detection, ShieldVNF is able to detect and mitigate the SIP DDoS flooding attack against a remote victim.

ShieldVNF also successfully dealt with multiple attack vectors and advanced persistent threats. Instead of offering protection for only a single attack vector, ShieldVNF successfully detected and mitigated multiple attack vectors: SIP scanning attack, SIP brute-force attack, CNC connection attempt, SIP DoS Flooding attack, and SIP Distributed DoS flooding attack. With advanced persistent threats, the adversaries launch different attacks depending on which stage of an attack they are currently in. Referring to the cyber kill chain spectrum (Hutchins et al., 2011), ShieldVNF is able to secure the switch in the following stage: Reconnaissance (SIP scan attack), Exploitation (SIP brute-force attack), Command and Control (CNC connection attempt), and Actions on Objective (SIP DoS and SIP DDoS attack).

Besides mitigating the attack, ShieldVNF produced threat intelligence which is critical for scaling the defence to other switches. The experimental data show that ShieldVNF was able to produce three types of threat intelligence: KnownAttacker, KnownCNC, and KnownVictim. This information provides evidence that is actionable by other switches. The experiment demonstrated that other switches were able to consume this threat intelligence and use it to formulate attack prevention rules.

7.2.2 ShieldSDN: SDN Controller for IOC Management

The second objective was to develop mechanisms for threat intelligence collection and distribution among all switches within an organisation. Our findings support the hypothesis that ShieldSDN is able to collect threat intelligence from the switch on which the attack originates, and install this intelligence as packet filtering rules on multiple switches within the same organisation.

ShieldSDN is able to conserve memory usage on ShieldVNF by removing expired packet filters. With limited memory resources on ShieldVNF, it is important for these filters to be managed so that ShieldVNF's capacity is not overloaded or manipulated by adversaries which could inflict self denial-of-service. ShieldSDN achieved this by keeping track of the expiry time for each rule and automatically removing the rules from ShieldVNF when they expired. This feature frees up memory capacity on ShieldVNF so that ShieldVNF is able to keep running.

ShieldSDN is able to deal with persistent threats by installing packet filters with expiry times that correspond to the frequency of attack. The expiry time is randomized and repeat offenders will get banned for an exponentially longer period of time. This feature strikes a balance between two opposing constraints: the need to preserve the available memory on the ShieldVNF, and the need to keep mitigation rules (i.e., packet filtering rules) active in memory for repeat offenders.

7.2.3 ShieldCHAIN: Blockchain Application for Collaborative Defence Framework

The third objective was to develop mechanisms for threat intelligence sharing and retrieval among organisations. The findings support the hypothesis that ShieldCHAIN is able to leverage the distributed app and the smart contract to facilitate dissemination of threat intelligence from one organisation to multiple organisations in the community. With external organisations, there is no direct network connectivity among them and the proposed solution is for ShieldCHAIN to collaborate through the Blockchain. The smart contracts that are

deployed on the Ethereum Blockchain serve two purposes: to provide functions that can be called by the distributed apps and to maintain the state of current threat intelligence.

The ShieldCHAIN in the source-organisation pushes threat intelligence to the Blockchain, whereas the ShieldCHAIN in a receiving-organisation pulls from the Blockchain. In this case, the Ethereum Blockchain serves as a common repository and a centralised location to store threat intelligence about KnownAttacker, KnownCNC, and KnownVictim. The timing for this sharing and retrieval of threat intelligence is coordinated to occur at regular intervals via scheduled jobs. At this stage, the objective of sharing threat intelligence between organisations has been achieved. The last step to make this threat intelligence effective is to install it as packet filtering rules on ShieldVNF.

ShieldCHAIN installs community-sourced intelligence as packet filtering rules on ShieldVNF. Our findings show that ShieldCHAIN was able to install threat intelligence as packet filtering rules on the ShieldVNF. At this stage, the ShieldVNF in multiple organisations have identical packet filtering rules on their tables. This demonstrates that these switches, even though they belong to different organisations, are synchronised on their packet filters and will be able to defend against the same attacks.

7.3 Contribution to Knowledge

My significant and original contributions to knowledge include the following:

1. **SIPshield framework**. A scalable and collaborative SIP DDoS defence framework that secures the edges of IoT networks with VNF, SDN, and Blockchain technology. Figure 7.1 depicts SIPshield and it consists of the following elements:

   (a) **ShieldVNF** is a VNF running on a programmable data plane. It secures the edge of an IoT network by providing attack prevention, detection, and mitigation against SIP DDoS Flooding attacks from botnet. In addition, it produces an IOC that is useful as threat intelligence for other switches and organisations.

   (b) **ShieldSDN** is an SDN controller and application that collects threat intelligence from a switch where the attack occurred, and installs it as packet filtering rules in multiple switches in the same organisation.

   (c) **ShieldCHAIN** is a Blockchain distributed Application (dApp) and smart contract that facilitates multiple organisations to share and retrieve the current state of threat intelligence so that they can install it as packet filtering rules on their switches.

2. **An integrated study.** The knowledge produced from this study shows that it is feasible to integrate novel concepts from networking research, IP Telephony research, and threat intelligence research, and apply them towards building a novel solution addressing an active research area in cyber security.

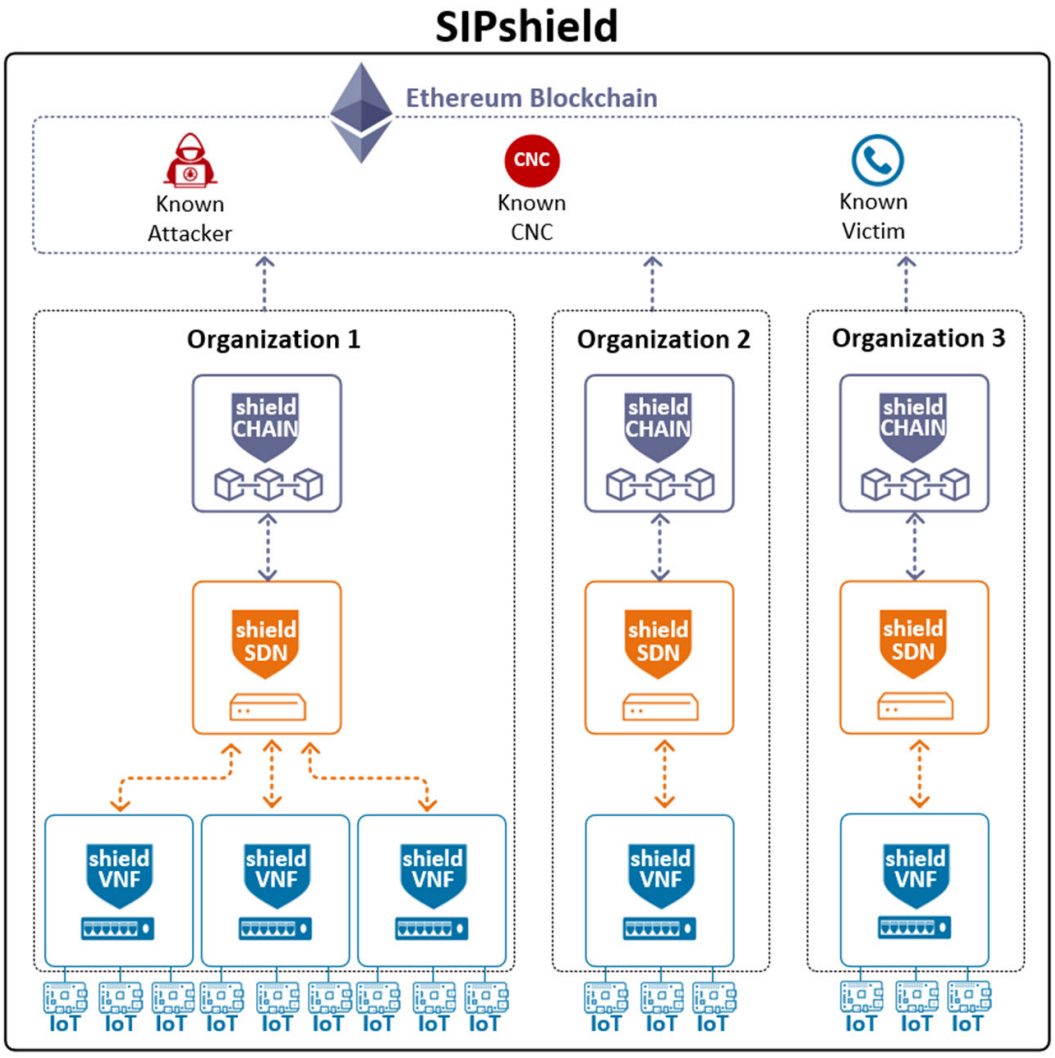


Fig. 7.1 SIPshield: A Scalable SIP DDoS Defence Framework With VNF, SDN, and Blockchain

7.3.1 Research Significance and Impact

The significance of this research is that it provides a way to secure SIP-based systems against SIP DDoS Flooding attacks from IoT botnets so that the telecommunication services remain available for legitimate users as and when they need them.

This research demonstrates the power and capabilities available when VNF is pushed to the edge of the network, and this is significant because the Ethernet switch's data plane is currently underutilised. The network switches at the edge of the networks are commonly perceived solely as tools to provide network connectivity for the endpoints. With a programmable data plane, it is now possible for the network switch to provide much more function than just raw connectivity. For example, the Virtualized Network Function on data planes can provide functions such as security inspections, performance enhancements, application-layer translations, etc. that were previously performed by the middleboxes. It is anticipated that the concept of using VNF over data planes will catch on in the near future, especially in the age of edge computing.

This research shows that the edge of the network can produce valuable evidence that can be centralized and correlated to provide actionable insights. As edge computing technology gains momentum in the years to come, more and more functions will be pushed out to the edge. The network edge devices with their programmable data plane can be utilized as widely-distributed sensors that provide evidence. With the ability to inspect application-layer payload, they can sense and produce evidence for all kinds of indicators for many different use cases. For example, they can inspect the payload in MQTT packets to read status or commands from IoT devices. This evidence can then be centralised to provide a consolidated view of the entire IoT network.

The output of this research has both social and economic impact potential. From the economic impact perspective, this defence framework helps the industry to secure their telecommunication systems so that they can continue to serve the public that uses telecommunication service as a means to reach the business. For example, call centres can remain operational to serve customers instead of giving in to ransomware with DDoS as a threat. From the social impact perspective, this framework helps the government to make sure that government services remain accessible by the public, including access to police services, firefighter services, and medical services.

7.3.2 Potential Contributions to Other Research

DDoS research

In the area of DDoS research, the source-end defence approach was historically underutilised but it is now worth another consideration. There are two trends that suggest it is time to reconsider source-end defence approach: the growing trend of botnet attacks and the rise of programmable networks. Botnet attacks expose the inadequacy and sustainability of destination-end defence approach. Now with the network itself being more software-centric

and programmable, a source-defence approach can reduce the cost of implementation while increasing capability to mitigate attacks near or at the source. Source-end defence allows the defender to mitigate the attack early and minimize collateral damage. This is especially true in the age of IoT, where the attacks are widely distributed.

SIP security

Deep packet inspection for SIP is now feasible for every SIP endpoint. In a similar vein, historically it was not economically feasible to provide deep packet inspection for each SIP endpoint. But with VNF that is running directly on the switch data plane, it is now affordable to protect each SIP endpoint through deep packet inspection at the SIP level. This level of inspection allows the network operator to provide a layer of protection for SIP endpoints that currently have weak security postures.

Below are three potential contributions where SIPshield could benefit and influence existing research. The first proposal by Tas et al. (2020) presented a novel defence mechanism that consists of statistics, inspection, and action modules. The authors could adopt the ShieldVNF approach (i.e., using a programmable data plane) to produce data points for the statistics module and drop packets for the action module. This approach would allow this proposal to be implemented on network switches, which would produce richer data points and faster action due to its close physical proximity to the source of the attack.

The second proposal by Semerci et al. (2018) periodically counts SIP requests and responses sent or received by the server. The authors could use a Bloom Filter to count these messages and implement it on the server's NIC card using a programmable data plane using cards like (Netronome, 2020a) or (Pensando, 2020). With this approach, the server could off-load the counting tasks to the NIC card, so that the server's computing resources could focus on a change-point detection method, for example using adaptive tracking of Mahalanobis distances.

The third proposal by Tsiatsikas et al. (2015) introduced Machine Learning techniques to detect the DDoS and low-and-slow attacks where it uses Machine Learning models for training and testing scenarios. The author could adopt the same approach with ShieldVNF and ShieldSDN working together in synergy. In this use case, ShieldVNF could focus on producing the raw data points, where ShieldSDN could collect those data points from the VNF and use them to train the Machine Learning model on the ShieldSDN server.

Network Security

Packet filtering for Network Access Control (NAC) is typically implemented based on the criteria at layer 2-4, but with VNF, the criteria for granting access can be composed by tracking application layer properties. For example, we could now set a rule that drops all SIP packets until the endpoint has successfully authorised itself by sending a SIP REGISTER message. Application-layer access control ensures that in appropriate packets are prohibited from even entering the network. For example, this would be useful in the case of inter-satellite links (ISL) in low orbit earth satellite constellations like Starlink (Handley, 2019) where bandwidth is at a premium and carrying packets related to a DoS attack would be a waste of scarce resources.

Defence-in-depth can be applied at a virtual switch to the core network switch level. With more and more switches supporting a programmable data plane, the network operator can build VNF that is appropriate for each level. For example, in a data centre environment, a VNF can be deployed at virtual switches (Shahbaz et al., 2016) running on a hypervisor server that hosts multiple virtual machines, while another VNF can be deployed at a top-of-rack switch or network core switch that uses the Tofino chip. In an Internet Service Provider environment, VNF can be deployed at a broadband network gateway level or virtual residential gateway (Proença et al., 2019). This level of flexibility should be leveraged to create a secure networking environment.

7.3.3 Current Limitations

At its current state, SIPshield does have limitation around the area of DDoS protection for its components, inspection of encrypted packets, and detection of DDoS attacks caused by non-flooding scenarios.

- ShieldVNF, ShieldSDN, and ShieldCHAIN are vulnerable to DDoS. When the attacker knows about the mechanisms of SIPshield, it will become an attractive target for the attacker to attack SIPshield, using the DDoS method, so that SIPshield is unable to function properly and not able to detect attacks. When this has happened, they could further the attacks against other targets without being detected. A potential solution would be to implement packet filtering rules on ShieldVNF where connections to any of SIPshield components can only be initiated from trusted entities.

- ShieldVNF is currently not able to inspect encrypted packets. This is due to the fact that there is no abstraction or resources for the VNF to terminate encrypted sessions. The attackers could use the TLS session to launch an attack and therefore evade inspection

by ShieldVNF. Further research might want to focus on this area for encrypted traffic analysis. For example, by feeding observable and non-encrypted information about the packets, such as packet size, layer 2-4 headers, packet inter arrival rate, etc. to machine learning models so that it can run predictive analytics to differentiate between legitimate vs. malicious packets.

- The attack detection module is currently not able to detect a SIP DDoS attack that is caused by flow or payload tampering. Besides the DDoS attack by flooding, a SIP is also vulnerable to flow tampering and payload tampering (Ehlert et al., 2010). Since this type of attack presents a different way of causing DoS, it would require a different method of tracking the session state. Although this requirement goes beyond the scope of the present investigation, it could be addressed by doing an inspection of the entire application-layer payload instead of a few initial bytes. In this case, ShieldVNF will have a richer context and situational awareness that is critical for identification whether or not the flow and payload have been tampered with.

- ShieldSDN and ShieldCHAIN are currently operate based on a scheduled task which includes an inherent time delay between when the IOC was discovered and when it was disseminated to the community. One approach which might address this limitation is by adopting a publish-subscribe messaging pattern. With this pattern, the subscribers will receive the message when it fits within the prescribed criteria. Another approach that will reduce the time delay is by using the event-driven messaging pattern where the originating organization will trigger an event that causes the receiving organizations to be notified of the occurrence of such an event.

7.4 Future Work

SIPshield provides an early investigation into edge computing cybersecurity research, which could be further developed in a number of different directions depending on the needs and opportunities that emerge in the near future. These discussions for future work are organised around each component. The enhancement for each component could be carried out independently, which might spark other innovations and improvements along the way. Another area that could be investigated is in the realms of scaling the solution with hardware-based solutions along with field-trials to study the feasibility of field deployment with real workloads.

7.4.1 ShieldVNF

Enhancement for ShieldVNF could take place in many different directions, for example, in adding a new detection engine, update mechanism, or introducing the concept of an app store for VNF.

- ShieldVNF could be further improved by adding new detection modules to enrich and increase detection accuracy. At the moment, attack detection is based on anomalies from the perspective of SIP protocol session state compliance. The detection can certainly be improved by adding other detection methods such as artificial intelligence, machine learning, or statistical models. In essence, the rest of the framework stays the same, but we enrich the detection mechanisms.

- Another improvement would be to add encrypted traffic analysis. Currently, the detection is done by doing deep packet inspection on non-encrypted packets. With encrypted packets, the detection would not be able to keep track of the SIP session state. Enhancements could be made so that SIP state machine detection could still be performed even though the packets are encrypted.

- Management and upkeep of ShieldVNF can be enhanced by having a centralised update mechanism for the VNF. Considering that the threats are constantly evolving, there is a need to be able to conveniently modify the VNF code so that it is relevant to the latest strain of botnet that is currently active. This centralised management system could also serve as a version control system so that the network operator can keep track of the current version of VNF that is deployed on the switches throughout the organisation. This would also facilitate roll-back should the newer version introduce unexpected results.

- As the use and popularity of VNF over data planes grow, a centralised application store that showcases peer-reviewed VNF applications would be helpful. In this framework, the VNF can be treated like a distributed middlebox application. Over time, there could be multiple applications written by multiple authors to address different kinds of needs. For example, the VNF developers could write VNF for security functions, speed improvement functions, translation functions, etc. Future researchers could enhance this further by building a mechanism that serves as an authorised application store, where the community could review and recommend the apps as they built experience with them. In this model, it serves as an informal peer-review that gives an indication of the quality and usefulness of that VNF.

7.4.2 ShieldSDN

Future research for ShieldSDN could be in the form of adding more sophisticated routing, deep analytics, or mitigation engine generators that take considerations of the probability and impact factors.

- Advance routing rules to optimise local mitigation resources. With centralised visibility, ShieldSDN could install advanced routing rules that take into consideration the availability of mitigation resources. For example, a network operator might have several mitigation resources at the fog computing level. ShieldSDN could interrogate the current capacity of these resources and install routing rules accordingly so that the workload can be distributed more fairly and does not overload any one resource.

- Deep analytics to generate prediction rules that provide evidence-based decision-making. Since ShieldSDN holds the records of previously detected incidents, it could be mined for the purpose of generating predictive rules. For example, some of the properties of the incident could be fed into a machine learning model where it could provide prediction rules. These prediction rules are not meant to mitigate or drop packets, but rather, to serve as honeypots which allow the network operators to confirm or deny that a specific strain of botnet is detected in the network.

- ShieldSDN can be further enhanced by generating expiry time based on more complex criteria. Currently, the expiry time for mitigation rules is based on the frequency. The expiry-time could be enriched by taking a more rigorous risk-based approach where it is weighted based on the probability and impact factors. The expiry-time calculation could consider similarities to what is being reported versus current assets. For example, if it is known that the attack is aimed against a certain IoT device with a certain firmware level, ShieldSDN should assign a longer expiry time when similar device and firmware is present in the network since the probability of getting an attack is higher than IoT devices with different device type and firmware version. Similarly, longer expiry time could be assigned for more sensitive targets since it would generate a higher impact for the organisation.

- Within the context of the ISP-and-subscribers network, the ShieldSDN at the ISP network could install a packet filtering rule that causes the subscriber to take action. When a malware is present in one subscriber's network, the ISP could install a packet filtering rule which will get the attention of the subscriber. For example, when a malware is detected coming from a subscriber's network, the ISP could install a packet filtering rule that blocks the subscriber's DNS traffic. The subscriber will notice this

disruption and contact the ISP. At this time, the ISP could advise the subscriber to take responsibility for cleaning the malware in their network before the service can be resumed.

7.4.3 ShieldCHAIN

The work for ShieldCHAIN could be further developed by integrating ShieldCHAIN to existing communities and adding richer social elements to encourage positive contribution and discourage negative influence in the community.

- ShieldCHAIN could be enhanced by providing integration with existing threat intelligence platforms for higher awareness and a more informed decision-making process. For example, ShieldCHAIN data can be further enriched by having integration with authoritative sources like National Vulnerability Database, Common Vulnerability Scoring Systems, National Cyber Security Center, MITRE, etc. The threat intelligence gathered from these sources would allow ShieldCHAIN to convert these credible sources to mitigation rules that can be installed on the VNF. The integration would also allow ShieldCHAIN to contribute actionable intelligence to these communities.

- Reward mechanisms that are designed to encourage and acknowledge the positive contribution from the community. Considering that ShieldCHAIN is a community, future work could incorporate reward mechanisms in the form of participation metrics that give visibility and credibility to users for their positive contribution. For example, the number of reports contributed, the number of citations or recommendations from their peers, the number of subscribers for intelligence sourced from this member, the number of downloads for their reports, etc. Besides acknowledging the positive contribution, these metrics also served as informal quality assessment for the community. Like other open-source contributions, the top contributors would typically rise up to the top of the list.

- Punishment mechanisms that are designed to discourage negative influence inside and outside of the community. Vulnerabilities in IoT devices exist partly due to the economic factors where these devices are manufactured with the lowest cost as possible to maximise financial return. One way to discourage this trend is to publish negative metrics to give visibility to the public and encourage accountability on the manufacturers' side. For example, the negative metrics could include things like the number of devices hijacked by botnet, the number of days since the last firmware update, the number of vulnerabilities discovered on the device, etc. To enable these

metrics, the community can use a standard reporting template that includes the MAC address (traceable to manufacturers), device type, firmware version, etc. The idea is to introduce financial consequences to these manufacturers that produce IoT devices with a weak security posture. The financial consequences are achieved by empowering the community with evidence-based information that discourages them from doing business with these manufacturers in the future. This information should influence the decision-makers in rewarding government contracts or industry bids to a manufacturer. Since financial factors contribute to the problem, it should also be leveraged as part of the solution.

7.5 Closing

Securing the edge of IoT networks with the approach presented in this thesis provides an opportunity to protect our progress from attack by adversaries, and restores the IoT to its original purpose. The edge of the IoT network presents research opportunities that are wide open, not just for cybersecurity but also for other use cases. The network has a great opportunity to contribute and is strategically positioned to defend against botnets so that our innovation with IoT is working for us instead of being hijacked and used against us.

When these measures have been implemented and it is our turn to call 999 for an emergency, we can have a justified confidence that our call will be answered and we will hear from the operator that says "Help is on the way!"

References

Abdelsayed, S., Glimsholt, D., Leckie, C., Ryan, S., and Shami, S. (2003). An efficient filter for denial-of-service bandwidth attacks. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, volume 3, pages 1353–1357. IEEE.

Afek, Y., Bremler-Barr, A., Hay, D., Shafir, L., and Zhaika, I. (2020). Demo: Nfv-based iot security at the isp level. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–2.

Agrawal, N. and Tapaswi, S. (2019). Defense Mechanisms against DDoS Attacks in a Cloud Computing Environment: State-of-the-Art and Research Challenges. *IEEE Communications Surveys & Tutorials*, pages 1–1.

Al-Haidari, F., Salah, K., Sqalli, M., and Buhari, S. M. (2017). Performance modeling and analysis of the edos-shield mitigation. *Arabian Journal for Science and Engineering*, 42(2):793–804.

Al Shorman, A., Faris, H., and Aljarah, I. (2019). 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*, 11(7):2809–2825.

Amazon (2020a). Amazon virtual private cloud. https://aws.amazon.com/vpc/. Web. Accessed 29 Aug. 2020.

Amazon (2020b). Amazon web services (aws) - cloud computing services. https://aws.amazon.com/. Web. Accessed 29 Aug. 2020.

Amazon (2020a). Aws shield managed ddos protection. https://aws.amazon.com/shield/. Web. Accessed 29 Aug. 2020.

Amazon (2020b). Azure functions serverless compute | microsoft azure. https://aws.amazon.com/lambda/. Web. Accessed 29 Aug. 2020.

Ang, K. L. M. and Seng, J. K. P. (2019). Application Specific Internet of Things (ASIoTs): Taxonomy, Applications, Use Case and Future Directions. *IEEE Access*, 7:56577–56590.

AWS Shield (2020). Threat landscape report – q1 2020. https://aws-shield-tlr.s3.amazonaws.com/2020-Q1_AWS_Shield_TLR.pdf. Web. Accessed 29 Aug. 2020.

Bafoutsou, G., Koukounas, A., and Dekker, M. (2020). *TELECOM SERVICES SECURITY INCIDENTS 2019*. European Union Agency for Cybersecurity (ENISA).

Barefoot Networks (2020). Product brief tofino page | barefoot. https://barefootnetworks.com/products/brief-tofino/. Web. Accessed 29 Aug. 2020.

Barnum, S., Martin, R., Worrell, B., and Kirillov, I. (2012). The cybox language specification. *draft, The MITRE Corporation*.

Behal, S. and Kumar, K. (2017). Detection of ddos attacks and flash events using novel information theory metrics. *Computer Networks*, 116:96–110.

Bellovin, S. M. (1989). Security problems in the tcp/ip protocol suite. *SIGCOMM Comput. Commun. Rev.*, 19(2):32–48.

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

Bloom, B. H. (1970). Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426.

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

Boro, D. and Bhattacharyya, D. K. (2017). Dyprosd: a dynamic protocol specific defense for high-rate ddos flooding attacks. *Microsystem Technologies*, 23(3):593–611.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95.

Broder, A. and Mitzenmacher, M. (2004). Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509.

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

Cabrera, J. B., Lewis, L., Qin, X., Lee, W., Prasanth, R. K., Ravichandran, B., and Mehra, R. K. (2001). Proactive detection of distributed denial of service attacks using mib traffic variables-a feasibility study. In *2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings. Integrated Network Management VII. Integrated Management Strategies for the New Millennium (Cat. No. 01EX470)*, pages 609–622. IEEE.

Carpenter, B. and Brim, S. (2002). Rfc3234: Middleboxes: Taxonomy and issues.

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking control of the enterprise. *ACM SIGCOMM computer communication review*, 37(4):1–12.

Casado, M., McKeown, N., and Shenker, S. (2019). From ethane to sdn and beyond. *ACM SIGCOMM Computer Communication Review*, 49(5):92–95.

Chen, E. (2006). Detecting DoS attacks on SIP systems. *1st IEEE Workshop on VoIP Management and Security, 2006.*, pages 2–7.

Chen, R. and Park, J.-M. (2005). Attack diagnosis: Throttling distributed denial-of-service attacks close to the attack sources. In *Proceedings. 14th International Conference on Computer Communications and Networks, 2005. ICCCN 2005.*, pages 275–280. IEEE.

Chen, R., Park, J.-M., and Marchany, R. (2006). Track: A novel approach for defending against distributed denial-of-service attacks. *Technical P˜ eport TR ECE—O6-02. Dept. of Electrical and Computer Engineering, Virginia Tech*.

Chen, Z., Jiang, F., Cheng, Y., Gu, X., Liu, W., and Peng, J. (2018). Xgboost classifier for ddos attack detection and analysis in sdn-based cloud. In *2018 IEEE international conference on big data and smart computing (bigcomp)*, pages 251–256. IEEE.

Chiosi, M. (2012). Network functions virtualisation. an introduction, benefits, enablers, challenges call for action. https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

Cisco (2020). Cisco silicon one. https://www.cisco.com/c/en/us/solutions/service-provider/innovation/silicon-one.html?dtid=osscdc000283/. Web. Accessed 4 Sep. 2020.

Connolly, J., Davidson, M., and Schmidt, C. (2014). The trusted automated exchange of indicator information (taxii). *The MITRE Corporation*, pages 1–20.

Council to Secure the Digital Economy (CSDE) (2019). The C2 Consensus on IoT Device Security Baseline Capabilities. Technical report.

Council to Secure the Digital Economy (CSDE) (2020). INTERNATIONAL BOTNET AND IOT SECURITY GUIDE 2020. Technical report.

Dang, H. T., Sciascia, D., Canini, M., Pedone, F., and Soulé, R. (2015). Netpaxos: Consensus at network speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pages 1–7.

Denning, P. J. (2005). Is computer science science? *Communications of the ACM*, 48(4):27–31.

Dodson, D., Polk, T., Souppaya, M., Barker, W. C., Grayeli, P., and Symington, S. (2019a). Securing Small-Business and Home Internet of Things (IoT) Devices Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD). (November).

Dodson, D., Souppaya, M., Jones, T., Lear, E., Cohen, D., Dodson, D., Souppaya, M., and Lear, E. (2019b). Securing Small-Business and Home Internet of Things (IoT) Devices Dakota Consulting. (November).

Dodson, D., Souppaya, M., Jones, T., Lear, E., Cohen, D., Dodson, D., Souppaya, M., and Lear, E. (2019c). Securing Small-Business and Home Internet of Things (IoT) Devices Dakota Consulting. (November 2019).

edge-core (2020). Barefoot networks brings the power of p4 programmability to open compute project's sonic. https://www.edge-core.com/news-inquiry.php?cls=3&id=281/. Web. Accessed 4 Sep. 2020.

Ehlert, S. (2009). *Denial-of-service detection and mitigation for SIP communication networks*. PhD thesis, Berlin Institute of Technology.

Ehlert, S., Geneiatakis, D., and Magedanz, T. (2010). Survey of network security systems to counter SIP-based denial-of-service attacks. *Computers and Security*.

Enable Security (2020). Github - enablesecurity/sipvicious: Sipvicious oss is a set of security tools that can be used to audit sip based voip systems. https://github.com/EnableSecurity/sipvicious. Web. Accessed 29 Aug. 2020.

ethereum (2020a). Github - ethereum/web3.js: Ethereum javascript api. https://github.com/ethereum/web3.js/. Web. Accessed 29 Aug. 2020.

ethereum (2020b). Solidity. https://solidity.readthedocs.io/en/v0.7.0/. Web. Accessed 29 Aug. 2020.

Etherscan (2020). Ethereum (eth) blockchain explorer. https://etherscan.io/. Web. Accessed 29 Aug. 2020.

Fagan, M., Megas, K. N., Scarfone, K., and Smith, M. (2020). Foundational cybersecurity activities for IoT device manufacturers. Technical report, National Institute of Standards and Technology, Gaithersburg, MD.

Fagan, M., Yang, M., Tan, A., Randolph, L., and Scarfone, K. (2019). Security Review of Consumer Home IoT Products. *Nist*, page 41.

Fiedler, J., Kupka, T., Ehlert, S., Magedanz, P. D. T., and Sisalem, D. D. (2007). [Survey 1] VoIP Defender : Highly Scalable SIP-based Security Architecture. *Proceedings of the 1st International Conference on Principles, Systems and Applications of IP Telecommunications*, pages 11–17.

Foundation, T. R. P. (2020). Teach, learn, and make with raspberry pi – raspberry pi. https://www.raspberrypi.org/. Web. Accessed 29 Aug. 2020.

Gartner Research (2013). Definition: Threat intelligence. https://www.gartner.com/en/documents/2487216/definition-threat-intelligence. Web. Accessed 29 Aug. 2020.

Geneiatakis, D., Vrakas, N., and Lambrinoudakis, C. (2009). Utilizing bloom filters for detecting flooding attacks against sip based services. *computers & security*, 28(7):578–591.

Geng, X. and Whinston, A. B. (2000). Defeating distributed denial of service attacks. *It Professional*, 2(4):36–42.

Gil, T. M. and Poletto, M. (2001). Multops: A data-structure for bandwidth attack detection. In *USENIX Security Symposium*, pages 23–38.

Gllgor, V. D. (1986). On denial-of-service in computer networks. In *1986 IEEE Second International Conference on Data Engineering*, pages 608–617.

Global Cyber Alliance (2020). Autoamated iot defence ecosystem (aide). https://www.globalcyberalliance.org/aide/. Web. Accessed 7 Aug. 2020.

GlobalPlatform (2019). Introducing IoTOPIA Global Platform. A practical implementation guide to secure IoT devices across all markets and in line with global requirements. (October).

Gonzalez, J. M., Anwar, M., and Joshi, J. B. (2011). A trust-based approach against ip-spoofing attacks. In *2011 Ninth Annual International Conference on Privacy, Security and Trust*, pages 63–70. IEEE.

Google (2020a). Cloud functions | google cloud. https://cloud.google.com/functions. Web. Accessed 29 Aug. 2020.

Google (2020b). Google cloud armor. https://cloud.google.com/armor. Web. Accessed 29 Aug. 2020.

Guri, M., Mirsky, Y., and Elovici, Y. (2017). 9-1-1 DDoS: Attacks, Analysis and Mitigation. *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017*, pages 218–232.

Handley, M. (2019). Using ground relays for low-latency wide-area routing in megaconstellations. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 125–132, New York, NY, USA. Association for Computing Machinery.

Handley, M., Raiciu, C., Agache, A., Voinescu, A., Moore, A. W., Antichi, G., and Wójcik, M. (2017). Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 29–42.

Harrison, A. (2020). Cnn - the denial-of-service aftermath - february 14, 2000.

Hoque, N., Bhattacharyya, D. K., and Kalita, J. K. (2015). Botnet in DDoS Attacks: Trends and Challenges. *IEEE Communications Surveys and Tutorials*, 17(4):2242–2270.

Huang, Y. and Pullen, J. M. (2001). Countering denial-of-service attacks using congestion triggered packet sampling and filtering. In *Proceedings Tenth International Conference on Computer Communications and Networks (Cat. No. 01EX495)*, pages 490–494. IEEE.

Huici, F., Niccolini, S., and D'Heureuse, N. (2009). [Survey 1+2] Protecting SIP against very large flooding DoS attacks. *GLOBECOM - IEEE Global Telecommunications Conference*, pages 1–6.

Hussain, I., Djahel, S., Zhang, Z., and Naït-Abdesselam, F. (2015). A comprehensive study of flooding attack consequences and countermeasures in Session Initiation Protocol (SIP). *Security and Communication Networks*, 8(18):4436–4451.

Hutchins, E. M., Cloppert, M. J., Amin, R. M., et al. (2011). Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80.

Infura Inc (2020). Ethereum api | ipfs api gateway | eth nodes as a service | infura. https://infura.io/. Web. Accessed 29 Aug. 2020.

Jin, X., Li, X., Zhang, H., Soulé, R., Lee, J., Foster, N., Kim, C., and Stoica, I. (2017). Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136.

John, A. and Sivakumar, T. (2009). Ddos: Survey of traceback methods. *International Journal of Recent Trends in Engineering*, 1(2):241.

Kambourakis, G., Moschos, T., Geneiatakis, D., and Gritzalis, S. (2007). Detecting dns amplification attacks. In *International workshop on critical information infrastructures security*, pages 185–196. Springer.

Kappelman, L., Johnson, V., Maurer, C., McLean, E., Torres, R., David, A., and Nguyen, Q. (2018). The 2017 sim it issues and trends study. *MIS Quarterly Executive*, 17(1).

Keromytis, A. D. (2012). [SIP Security] A Comprehensive Survey of Voice over IP Security Research. *Ieee Communications Surveys & Tutorials*, 14(2):514–537.

Keromytis, A. D., Misra, V., and Rubenstein, D. (2002). Sos: Secure overlay services. *ACM SIGCOMM Computer Communication Review*, 32(4):61–72.

Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., and Wobker, L. J. (2015). In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*.

Kim, Y., Lau, W. C., Chuah, M. C., and Chao, H. J. (2006). Packetscore: a statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE transactions on dependable and secure computing*, 3(2):141–155.

Kottler, Sam (2020). February 28th ddos incident report. https://github.blog/2018-03-01-ddos-incident-report/. Web. Accessed 29 Aug. 2020.

Koukounas, A., Vytogianni, E., and Dekker, M. (2019). *ANNUAL REPORT TELECOM SECURITY INCIDENTS 2018*. European Union Agency for Network and Information Security (ENISA).

Lau, F., Rubin, S. H., Smith, M. H., and Trajkovic, L. (2000). Distributed denial of service attacks. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*, volume 3, pages 2275–2280 vol.3.

Lear, E., Barker, W. C., Cohen, D., and Harrington, J. (2019a). NIST SPECIAL PUBLICATION 1800-15C Securing Small-Business and Home Internet of Things (IoT) Devices. (November 2019).

Lear, E., Droms, R., and Romascanu, D. (2019b). RFC 8520: Manufacturer Usage Description Specification. *Internet Engineeging Task Force*.

Linux Foundation (2020). Open vswitch. https://www.openvswitch.org/. Web. Accessed 29 Aug. 2020.

Liu, X., Yang, X., and Lu, Y. (2008). To filter or to authorize: Network-layer dos defense against multimillion-node botnets. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 195–206.

Liu, Z., Yin, X., and Lee, H. J. (2014). [SIP+DoS] An Efficient Defense Scheme Against SIP DoS Attack in SDN Using Cloud SFW.

Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., and Shenker, S. (2002). Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73.

Miao, R., Zeng, H., Kim, C., Lee, J., and Yu, M. (2017). Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28.

Microsoft (2020a). Azure ddos protection. https://azure.microsoft.com/en-us/services/ddos-protection/. Web. Accessed 29 Aug. 2020.

Microsoft (2020b). Azure functions serverless compute | microsoft azure. https://azure.microsoft.com/en-us/services/functions/. Web. Accessed 29 Aug. 2020.

Mirkovic, J., Prier, G., and Reiher, P. (2003a). Source-end ddos defense. In *Second IEEE International Symposium on Network Computing and Applications, 2003. NCA 2003.*, pages 171–178. IEEE.

Mirkovic, J. and Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39.

Mirkovic, J., Reiher, P., and Robinson, M. (2003b). Forming alliance for ddos defense. In *New Security Paradigms Workshop*, pages 18–21.

Mizrak, A. T., Savage, S., and Marzullo, K. (2008). Detecting compromised routers via packet forwarding behavior. *IEEE network*, 22(2):34–39.

Molina Zarca, A., Bernal Bernabe, J., Farris, I., Khettab, Y., Taleb, T., and Skarmeta, A. (2018). Enhancing iot security through network softwarization and virtual security appliances. *International Journal of Network Management*, 28(5):e2038. e2038 nem.2038.

Nakamoto, S. (2019). Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot.

netberg (2020a). Aurora 610. https://netbergtw.com/products/aurora-610/. Web. Accessed 29 Aug. 2020.

netberg (2020b). The first aurora product on barefoot tofino silicon has arrived. https://netbergtw.com/articles/the-first-aurora-product-on-barefoot-tofino-silicon-has-arrived//. Web. Accessed 4 Sep. 2020.

NetFPGA (2020). Github - netfpga/p4-netfpga-public: P4-netfpga wiki. https://github.com/NetFPGA/P4-NetFPGA-public. Web. Accessed 29 Aug. 2020.

Netronome (2020a). Netronome announces availability of industry's first integrated p4 and c programming on production server adapter hardware. Web. Accessed 4 Sep. 2020.

Netronome (2020b). P4 programmability for the netronome agilio smartnic - netronome. https://www.netronome.com/blog/p4-programmability-for-the-netronome-agilio-smartnic/. Web. Accessed 29 Aug. 2020.

Netscout (2020). Netscout arbor confirms 1.7 tbps ddos attack; the terabit attack era is upon us. https://www.netscout.com/blog/asert/netscout-arbor-confirms-17-tbps-ddos-attack-terabit-attack-era. Web. Accessed 29 Aug. 2020.

NZ Herald (2020). Nzx halts trading; website down for a third day in a row. https://www.nzherald.co.nz/business/news/article.cfm?c_id=3&objectid=12360028. Web. Accessed 29 Aug. 2020.

Offensive Security (2020a). inviteflood | penetration testing tools. https://tools.kali.org/sniffingspoofing/inviteflood. Web. Accessed 29 Aug. 2020.

Offensive Security (2020b). Kali linux | penetration testing and ethical hacking linux distribution. https://kali.org/. Web. Accessed 29 Aug. 2020.

OpenJS Foundation (2020). Node.js. https://nodejs.org/en/. Web. Accessed 29 Aug. 2020.

Ormazabal, G., Nagpal, S., Yardeni, E., and Schulzrinne, H. (2008). Secure SIP: A scalable prevention mechanism for DoS attacks on SIP based VoIP systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Ozcelik, M., Chalabianloo, N., and Gur, G. (2017). Software-Defined Edge Defense Against IoT-Based DDoS. In *IEEE CIT 2017 - 17th IEEE International Conference on Computer and Information Technology*, pages 308–313.

P4 Language Consortium (2020a). P4. https://p4.org/. Web. Accessed 29 Aug. 2020.

P4 Language Consortium (2020b). Pensando announces p4-programmable platform and joins p4 community. https://p4.org/p4/pensando-joins-p4.html. Web. Accessed 4 Sep. 2020.

p4lang (2020). Github - p4lang/behavioral-model: The reference p4 software switch. https://github.com/p4lang/behavioral-model. Web. Accessed 29 Aug. 2020.

Paley, T. (2016). Mcso: 18-year-old arrested in cyberattack on 911 system.

Park, K. and Lee, H. (2001). On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets. *ACM SIGCOMM computer communication review*, 31(4):15–26.

Peng, T., Leckie, C., and Ramamohanarao, K. (2003). Protection from distributed denial of service attacks using history-based ip filtering. In *IEEE International Conference on Communications, 2003. ICC'03.*, volume 1, pages 482–486. IEEE.

Pensando (2020). Pensando dsc-25 distributed services card. https://pensando.io/wp-content/uploads/2020/03/Pensando-DSC-25-Product-Brief.pdf. Web. Accessed 29 Aug. 2020.

Praseed, A. and Santhi Thilagam, P. (2019). DDoS attacks at the application layer: Challenges and research perspectives for safeguarding web applications.

Proença, J., Cruz, T., Simões, P., and Monteiro, E. (2019). Virtualization of residential gateways: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 21(2):1462–1482.

Rahul, A., Prashanth, S., Suresh Kumar, B., and Arun, G. (2012). Detection of intruders and flooding in voip using ids, jacobson fast and hellinger distance algorithms. *IOSR Journal of Computer Engineering (IOSRJCE)*, 2(2):30–36.

Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., and Knightly, E. (2008). Ddos-shield: Ddos-resilient scheduling to counter application layer attacks. *IEEE/ACM Transactions on networking*, 17(1):26–39.

Rexford, J. (2019). Never waste a mid-life crisis: Change for the better. *SIGCOMM Comput. Commun. Rev.*, 49(5):12–13.

Rosenberg, J., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. (2002). RFC 3261: Session Initiation Protocol (SIP). *Internet Engineeging Task Force*, 1(11):1829–1841.

Sahoo, K. S., Puthal, D., Tiwary, M., Rodrigues, J. J., Sahoo, B., and Dash, R. (2018). An early detection of low rate ddos attack to sdn based data center networks using information distance metrics. *Future Generation Computer Systems*, 89:685–697.

Sapio, A., Abdelaziz, I., Aldilaijan, A., Canini, M., and Kalnis, P. (2017). In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 150–156.

Satyanarayanan, M. (1989). Integrating security in a large distributed system. *ACM Trans. Comput. Syst.*, 7(3):247–280.

Schlette, D., Böhm, F., Caselli, M., and Pernul, G. (2020). Measuring and visualizing cyber threat intelligence quality. *International Journal of Information Security*, pages 1–18.

Schulzrinne, H. (2018). Networking research — a reflection in the middle years. *Computer Communications*, 131:2 – 7. COMCOM 40 years.

Semerci, M., Cemgil, A. T., and Sankur, B. (2018). An intelligent cyber security system against DDoS attacks in SIP networks. *Computer Networks*, 136:137–154.

Shahbaz, M., Choi, S., Pfaff, B., Kim, C., Feamster, N., McKeown, N., and Rexford, J. (2016). Pisces: A programmable, protocol-independent software switch. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 525–538, New York, NY, USA. Association for Computing Machinery.

SIPp (2014). Sipp. http://sipp.sourceforge.net/. Web. Accessed 29 Aug. 2020.

Sotelo Monge, M., Maestre Vidal, J., and Martínez Pérez, G. (2019). Detection of economic denial of sustainability (edos) threats in self-organizing networks. *Computer Communications*, 145:284–308. cited By 7.

SQLite (2020). Sqlite home page. https://www.sqlite.org/. Web. Accessed 29 Aug. 2020.

Srivastava, A., Gupta, S., Quamara, M., Chaudhary, P., and Aski, V. J. (2020). Future IoT-enabled threats and vulnerabilities: State of the art, challenges, and future prospects. *International Journal of Communication Systems*, 33(12):1–40.

Steinberger, J., Sperotto, A., Baier, H., and Pras, A. (2020). Distributed DDoS Defense:A collaborative Approach at Internet Scale. *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2020: Management in the Age of Softwarization and Artificial Intelligence, NOMS 2020*.

Tas, I. M., Unsalver, B. G., and Baktir, S. (2020). A Novel SIP Based Distributed Reflection Denial-of-Service Attack and an Effective Defense Mechanism. *IEEE Access*, pages 1–1.

Tedre, M. and Moisseinen, N. (2014). Experiments in computing: A survey. *The Scientific World Journal*, 2014.

The Secretary of Commerce and Homeland Security (2018a). 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_13800_botnet_report_-_finalv2.pdf.

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

Tichy, W. F. (1998). Should computer scientists experiment more? *Computer*, 31(5):32–40.

Truffle Suite (2020). Sweet tools for smart contracts | truffle suite. https://www.trufflesuite.com/. Web. Accessed 29 Aug. 2020.

Trump, D. J. (2017). 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/. Web. Accessed 7 Aug. 2020.

Tsiatsikas, Z., Fakis, A., Papamartzivanos, D., Geneiatakis, D., Kambourakis, G., and Kolias, C. (2015). Battling against DDoS in SIP: Is machine learning-based detection an effective weapon? In *SECRYPT 2015 - 12th International Conference on Security and Cryptography, Proceedings; Part of 12th International Joint Conference on e-Business and Telecommunications, ICETE 2015*, pages 301–308.

Wang, H., Jin, C., and Shin, K. G. (2007). Defense against spoofed ip traffic using hop-count filtering. *IEEE/ACM Transactions on networking*, 15(1):40–53.

Wohrer, M. and Zdun, U. (2018). Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 2–8.

Wood, G. (2019). Ethereum: A secure decentralised generalised transaction ledger byzantium version d664f - 2019-06-13. https://ethereum.github.io/yellowpaper/paper.pdf. Web. Accessed 30 Jun. 2019.

Wood, G. et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32.

Wu, Z., Pan, Q., Yue, M., and Liu, L. (2019). Sequence alignment detection of tcp-targeted synchronous low-rate dos attacks. *Computer Networks*, 152:64–77.

Xianjun Geng and Whinston, A. B. (2000). Defeating distributed denial of service attacks. *IT Professional*, 2(4):36–42.

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

Yaar, A., Perrig, A., and Song, D. (2003). Pi: A path identification mechanism to defend against ddos attacks. In *2003 Symposium on Security and Privacy, 2003.*, pages 93–107. IEEE.

Yan, Q., Huang, W., Luo, X., Gong, Q., and Yu, F. R. (2018). A multi-level ddos mitigation framework for the industrial internet of things. *IEEE Communications Magazine*, 56(2):30–36.

Yan, Y., Beldachi, A. F., Nejabati, R., and Simeonidou, D. (2020). P4-enabled smart nic: Enabling sliceable and service-driven optical data centres. *Journal of Lightwave Technology*, 38(9):2688–2694.

Yin, D., Zhang, L., and Yang, K. (2018). A ddos attack detection and mitigation with software-defined internet of things framework. *IEEE Access*, 6:24694–24705.

Zargar, S. T., Joshi, J., and Tipper, D. (2013). [DDoS] A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069.

Zhijun, W., Wenjing, L., Liang, L., and Meng, Y. (2020). Low-Rate DoS Attacks, Detection, Defense, and Challenges: A Survey. *IEEE Access*, 8:43920–43943.

Index

112 (Emergency Number in EU), 26
911 (Emergency Number in US), 26
999 (Emergency Number in UK), 26

ACM, 27
API, 169
APT, 51
Aurora, 43
AWS, 83
AWS Lambda, 12
AWS Shield, 6

Bloom filter, 33
bmv2, 12
botnet, 1

CAM, 32
CNC, 8
Common Vulnerability Scoring Systems, 206
corporate espionage, 52
Counting Bloom Filter, 89
CPU, 23
criminal prosecution, 8
CTI, 21, 46
Current Limitations, 202
Cyber Kill Chain framework, 7
CybOX, 47

dApp, 20
Database schema, 131
DDoS, vii
DDoS Attack, 1.3 Tbps, 6
DDoS Attack, 1.7 Tbps, 6
DDoS Attack, 2.3 Tbps, 6
DDoS-as-a-Service, 7
Department of Homeland Security, 2
distributed Application, 20
Distributed Services Card, 43
Diversion tactic, 24
DMZ, 51
DNS, 29
DoS, 1

EC2, 12
Edge-core, 43
EDoS, 2
EENA, 23
ENISA, 25
Ethereum, 17
Ethereum Yellow paper, 193
Etherscan, 166
ETSI, 41

fee schedule, 193
FPGA, 32
Future Works, 203

Github, 6
Google Cloud Armor, 2

Hacktivism, 24
hash function, 89

HTTP, 29

IDS, 7, 33
IEEE, 27
IMS, 34
Infura, 169
Intellectual Roots, 56
inviteflood, 12
IOC, 11
IOC Collection, 137
IOC Collection From Blockchain, 182
IOC Distribution to Blockchain, 174
IOC Expiration, 144
IoT, 1
IPS, 7
ISO, 27
ITU-T, 52

KnownAttacker smart contract, 168
KnownAttacker table, 131
KnownCNC smart contract, 168
KnownCnc table, 132
KnownVictim smart contract, 168
KnownVictim table, 132
kovan, 170

Machine Learning, 34
Microsoft Azure, 2
middlebox, 41
miners, 167
Minimalist contract design, 167
mininet, 12
Mirai, 71
miraisim.py, 78
MITRE, 206
modular blocks, 50
MUD, 38

NAC, 202
NAT, 12
National Vulnerability Database, 206
NCCoE, 38
NENA, 23
Netberg, 43
NetFPGA, 32
Netronome, 32
Netscout, 6
netstat, 78
Network Access Control, 202
NFV, 39
NG112, 23
NG911, 23
NIC, 32
NIST, 37
Node.js, 12

Open vSwitch, 12
OpenFlow, 29
original device manufacturer, 43

P4, 12
patch management systems, 6
PBX, 79
Persistent Threat Management, 149
practical strategies, 50
President's Order, 2
Privacy concerns, 8
Punishment mechanism, 206

RAM, 23
ransomware, 24
REST, 169
Reward mechanism, 206
RFC, 72
RFP, 5
RTP, 62

SDN, 3
Secretary of Commerce, 2
ShieldCHAIN, 17
ShieldSDN, 17
shieldVNF, 16
SIP, vii
SIP brute force attack, 102
SIP DDoS flooding attack, 1
SIP Distributed DoS attack, 116
SIP DoS attack, 111
SIP enumeration attack, 98
SIP scanning attack, 94
SIPp, 12
SIPshield, 15
SIPVicious, 12
smart contract, 12
SMTP, 29
Solidity programming language, 12
SQL, 14
Staffing, 5
STIX, 47
svcrack, 78
svmap, 78
svwar, 78

TAXII, 47
Taxonomy, DDoS defence, 27
Taxonomy, SIP DDoS Attack , 24
TDoS, 24
Telephony Denial-of-Service, 24
Threat Intelligence Platform, 17
Tofino, 32
true experiment design, 74
truffle framework, 192

VNF, 3
VoIP, 6
VPC, 12

web3, 192
Wedge, 43