Securing a Production Network Infrastructure using Snort

Mohamed Ghretli[1], Remma Aburmila[1], Abdulmunem Almukhtar[1], Sabri Shmeek[1]
[1]Dept. of Computer Networking
College of Computer Technology (CCTT)
Tripoli, Libya
Correspondent email: mghretli@cctt.edu.ly

ABSTRACT

The importance of network security has grown tremendously and intrusion prevention/detection systems (IPS/IDS) have been widely developed to insure the security of the network against suspicious threats. Computer network intrusion detection and prevention system consist of collecting traffic data, analyzing them based on detection rules and generate alerts or dropping them if necessary. In this paper, we used Snort as Intrusion Prevention and configured it in inline mode to protect the production network in real-time. First we present a basic description of the theoretical background and our implementation of Snort in inline mode, and then we provide a discussion of the gained results from many tasks that performed using a penetration testing lab.

Keywords: Information Security; Network security; Snort; Intersion Prevention System.

I. INTRODUCTION

Security is a big issue for all networks in today's enterprise environment. Attackers and intruders have made many successful attempts to bring down high-profile company networks and web services. Many methods have been developed to secure the network infrastructure and communication over the Internet.

The existing network security technologies which are being used nowadays consists of firewalls that are used to block unwanted incoming traffic, data encryption while transmission, vulnerability assessment tools that find and patch security holes present in the network. Intrusion detection and prevention is a relatively new addition to such techniques.

Intrusion detection and prevention methods started appearing in the last decade. Using intrusion detection methods, we can collect and use information from known types of attacks and find out if someone is trying to attack the network or particular hosts. The information collected this way can be used to harden the network security, as well as for legal purposes, which are achieving the goals Of network security Integrity, confidentiality and availability.

Intrusion detection and prevention systems (IDS & IPS) utilized to analyze the network traffic and prohibit any abnormal activities within the network. The IDS monitors the packets on the network and reports back based on applied security rules and signatures, the IDS alone is unable to stop denial of service (DoS) attacks and intrusions particularly from UDP protocols. On the other hand, IPS has the ability to drop malicious packets and close connection session in order to stop further attacks. It analyzes all network traffic passing through and responds with immediate actions to suspicious packets (inline deployment). *Snort* is an open source network intrusion detection and prevention system, which is capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching and matching. It can also detect a variety of attacks and probes, such as buffer overflows, stealth port scans, and Dos attacks. Although all intrusion detection methods are still new, Snort is ranked among the top quality systems available today [1, 2 and 7].

II. SNORT RULES

Snort uses rules, as attack signatures, that allow detecting malicious network activities, such as DoS attacks and reconnaissance network traffic. Snort uses a simple, lightweight rules description language that is flexible and quite powerful. Snort rules are divided into two logical sections, namely the rule header and the rule options. Figure 1 shows an example of a Snort rule. The rule allows displaying an alert message once a TCP packet is detected [3].

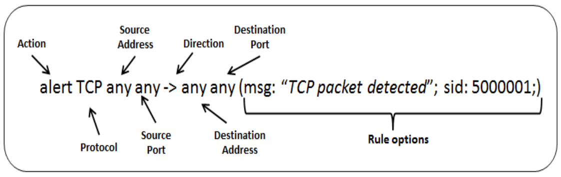


Fig. 1 Snort Rule Format Example [3].

III. METHODOLOGY

A. Production Network

The production network used in this paper consists of several services built and configured on a top of an Esxi server that placed behind a switch. These services are FTP, HTTP, Mail, Active Directory, DNS, and a syslog server, which are configured to receive events from Snort.

B. Building and Configuring Snrot

Snort built on a dedicated server which is positioned between the switch and the outside network, the operating system of this server is Ubuntu 14.04. Snort configured as a router to forward traffic between two different subnets, and it is running in inline mode with NFQ and ip-tables. Before Snort installation from source code, some required packages needs to be installed, these packages are the following [4]:

```
sudo apt-get install -y build-essential
libpcap-dev libpcre3-dev libdumbnet-dev
bison flex zlib1g-dev liblzma-dev libssl-
dev autoconf libtool pkg-config.
```

Also the (NFQ) specific libraries downloaded as the following:

```
sudo apt-get install libnetfilter-queue-dev
```

Then download and install the Data Acquisition library (DAQ) from the Snort website as the following:

```
wget
https://www.snort.org/downloads/snort/daq-
2.0.6.tar.gz
tar -xvzf daq-2.0.6.tar.gz
cd daq-2.0.6
$./configure
make
sudo make install
```

Snort now is ready to be installed from source:

```
sudo wget
https://www.snort.org/snort/snort-
2.9.11.1.tar.gz
tar -xvzf snort-2.9.11.1.tar.gz
cd snort-2.9.11.1
./configure --enable-sourcefire
make
sudo make install
```

With the following command to updates the shared libraries:

```
        sudo ldconfig
```

Since the Snort installation places the Snort binary at /usr/local/bin/snort, it is a common practice to create a symlink to /usr/sbin/snort:

```
sudo     ln     -s     /usr/local/bin/snort
/usr/sbin/snort
```

Some directories needs to be created, as the following:

```
sudo mkdir /etc/snort
sudo mkdir /etc/snort/rules
sudo mkdir /etc/snort/rules/iplists
sudo mkdir /etc/snort/preproc_rules
sudo                                  mkdir
/usr/local/lib/snort_dynamicrules
sudo mkdir /etc/snort/so_rules
```

Create some files that stores rules and IP lists:

```
sudo touch
/etc/snort/rules/iplists/black_list.rules
sudo touch
/etc/snort/rules/iplists/white_list.rules
sudo touch /etc/snort/rules/local.rules
sudo touch /etc/snort/sid-msg.map
```

Create logging directories for Snort logs:

```
sudo mkdir /var/log/snort
sudo mkdir /var/log/snort/archived_logs
```

The following files need to be moved from the extracted Snort and where it installed to the snort configuration directory:

```
cd ~/snort_src/snort-2.9.11.1/etc/
sudo cp *.conf* /etc/snort
sudo cp *.map /etc/snort
sudo cp *.dtd /etc/snort
cd ~/snort_src/snort-2.9.11.1/src/dynamic-
preprocessors/build/usr/local/lib/snort_dyn
amicpreprocessor/
```

```
sudo cp *
/usr/local/lib/snort_dynamicpreprocessor
```

Some parameters need to be configured in /etc/snort/snort.conf file, explained as follows:

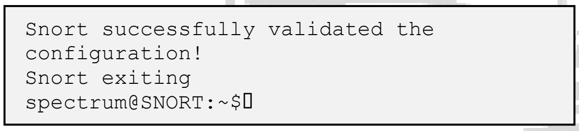
- Set up the network that Snort will protect, in this project, this variable filled with any option so Snort can protect any IP beyond the internal interface, line 45:
  ```
  ipvar HOME_NET any
  ```
- To tell Snort about the locations of all the directories that created before:

```
        var RULE_PATH /etc/snort/rules
        var SO_RULE_PATH
/etc/snort/so_rules
         var PREPROC_RULE_PATH
        /etc/snort/preproc_rules
        var WHITE_LIST_PATH
        /etc/snort/rules/iplists
        var BLACK_LIST_PATH
        /etc/snort/rules/iplists
```

After complete building and configuring Snort, a test is required with the following commend:

```
sudo snort -T -c /etc/snort/snort.conf -Q
```

The result of previous command is shown in figure 2, which validates Snort configuration.



```
Snort successfully validated the
configuration!
Snort exiting
spectrum@SNORT:~$▯
```

Fig. 2 Console Output Validating Snort Configuration

To configure Snort as a router [5], it should forward traffic between two different interfaces. Enabling IPv4 forwarding by uncommenting out a line in the file /etc/sysctl.conf, as follows:

```
        net-ipv4.ip-forwrd=1
```

C. Adding Iptables Rules

After verifying IPv4 forwarding, Network Address Translations (NAT) and forwarding rules configured with iptables. A NAT rule configured to translate any IP address coming from the external interface “eth1” with the IP address of the internal interface “eth0”, the forwarding rule configured to forward all traffic comes from outside to the NFQ after it pass through iptables, then in the configuration of snort in inline mode, there is an option used to let Snort take traffic from the NFQ and analyze it with its components. The ip-tables NAT masquerade and forwarding rules are the following respectively [6]:

```
sudo iptables -t nat -A POSTROUTING -o
eth0 -j MASQUERADE
```

Forwarding rule:

```
sudo  iptables  -A  FORWARD  -j  NFQUEUE  --
queue-num 0
```

D. Running Snort in Inline-mode (NFQ with Iptables)

Snort can run in inline mode using command line with different options, refers to the snort –help to see all snort options. The following command used to run Snort in inline mode:

```
sudo snort -Q --daq nfq --daq-var queue=0 -
c  /etc/snort/snort.conf  -A  Console  -l  -s
/var/log/snort
```

Snort options are shown in table 1, that explains options used in the previous command [5].

Table 1: Snort options with their descriptions

Option	***Description***
-Q	Enable inline mode operation
--daq <type>	Select packet acquisition module (NFQ type chosen here)
--daq-var <name=value>	Specify extra DAQ configuration variable (queue 0 that used by iptables to forward traffic to it.
-c	Use Rules Files that included in the following path, in this case snort.conf
-A	Set alert mode (console chosen here)
-l	Log to directory, in this case the log directory is /var/log/snort
-s	Log alert messages to syslog

After running the previous command, Snort will be loading its preprocessors and reading the rules from the configuration, if there is anything missing or configured incorrectly in snort.conf or in the rules files, Snort will reply with the errors, if there are no errors, the output to the screen should be like figure 3.

```
Commencing packet processing (pid=3288)
Decoding Raw IP4
```

Fig. 3 Output Screen for Snort Operations in Inline Mode

The previous figure shows how Snort is waiting for the traffic to process with the loaded rules and the configuration from the snort.conf.

E. Configuring Snort to Log events to Kiwi Syslog Server

Using syslog-ng demon on Snort sensor and configuring it to receive events from snort demon and send these events to an external syslog server (Kiwi syslog server) for a remote logging. The configuration is as follows:

In snort.conf file, the following line uncommented out to active syslog output module [4]:

```
output alert_syslog: LOG_AUTH  LOG_ALERT
```

And in syslog-ng.conf the following parameters are appended:

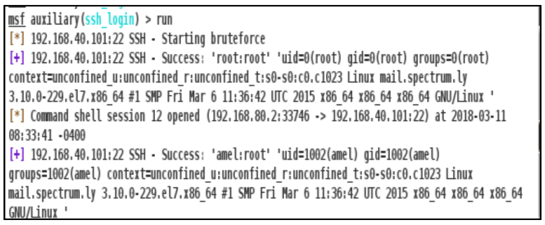
```
destination df_snort_remote{ udp
("192.168.90.101")
Log {
         source(s_src);
        filter(f_auth);
     destination( df_snort_remote);
};
```

IV. Testing and Results

It is not sufficient to build IPS/IDS system on a real production network and running it without proper testing and optimization. Using penetration testing tools, simulated attacks scenarios performed to ensure that Snort can catch malicious traffic and attacks.

A. SSH Brute Force Login

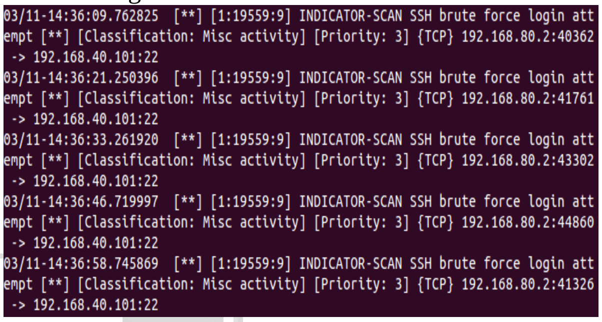
Using ssh_login auxiliary module on metasploit on Kali Linux with the IP address 192.168.80.2 as an attacker, SSH login brute force is performed on a mail server with an IP address of 192.168.40.101 as a possible victim. The result of this attack is catastrophic on the confidentiality and privacy of the users, as the attackers gained access to usernames and passwords of system users as shown in Figure 4.



```
msf auxiliary(ssh_login) > run
[*] 192.168.40.101:22 SSH - Starting bruteforce
[+] 192.168.40.101:22 SSH - Success: 'root:root' 'uid=0(root) gid=0(root) groups=0(root)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 Linux mail.spectrum.ly
3.10.0-229.el7.x86_64 #1 SMP Fri Mar 6 11:36:42 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux '
[*] Command shell session 12 opened (192.168.80.2:33746 -> 192.168.40.101:22) at 2018-03-11
08:33:41 -0400
[+] 192.168.40.101:22 SSH - Success: 'amel:root' 'uid=1002(amel) gid=1002(amel)
groups=1002(amel) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 Linux
mail.spectrum.ly 3.10.0-229.el7.x86_64 #1 SMP Fri Mar 6 11:36:42 UTC 2015 x86_64 x86_64 x86_64
GNU/Linux '
```

Fig. 4 SSH Brute Force Attack

While the previous figure 4 shows the gained usernames, passwords and some information about each user, Figure 5 shows the alert generated messages on Snort console while the attack is commencing.



```
03/11-14:36:09.762825  [**] [1:19559:9] INDICATOR-SCAN SSH brute force login att
empt [**] [Classification: Misc activity] [Priority: 3] {TCP} 192.168.80.2:40362
 -> 192.168.40.101:22
03/11-14:36:21.250396  [**] [1:19559:9] INDICATOR-SCAN SSH brute force login att
empt [**] [Classification: Misc activity] [Priority: 3] {TCP} 192.168.80.2:41761
 -> 192.168.40.101:22
03/11-14:36:33.261920  [**] [1:19559:9] INDICATOR-SCAN SSH brute force login att
empt [**] [Classification: Misc activity] [Priority: 3] {TCP} 192.168.80.2:43302
 -> 192.168.40.101:22
03/11-14:36:46.719997  [**] [1:19559:9] INDICATOR-SCAN SSH brute force login att
empt [**] [Classification: Misc activity] [Priority: 3] {TCP} 192.168.80.2:44860
 -> 192.168.40.101:22
03/11-14:36:58.745869  [**] [1:19559:9] INDICATOR-SCAN SSH brute force login att
empt [**] [Classification: Misc activity] [Priority: 3] {TCP} 192.168.80.2:41326
 -> 192.168.40.101:22
```

Fig. 5 SSH Brute Force Alert

Figure 5 shows the message that refers to SSH brute force attack coming from the IP 192.168.80.2 with port 43302 and going to IP 192.168.40.101 with port 22. The alert message also contains the classification of the attack and its priority. This alert message is generated before adding any custome rules, which means the detection happened based upon a VRT rule.

B. Scan on Mail Service

To protect specific service from attacks and threats, it is essential for the administrator or the network security officer to think as an attacker. First attacker will scan for open ports and services and then performs specific scan for specific service to gain more results. The following scan attack performed using an auxiliary in metasploit framework named smtp_enum. Figure 6 shows the auxiliary tool which is used to scan for mail service and gain information about the techniques used on this mail server, the domain name and a list of usernames in the target server.

```
msf auxiliary(smtp_enum) > exploit
[*]    192.168.40.101:25   Banner:   220   mail.spectrum.ly
ESMTP Postfix
[+] 192.168.40.101:25 Users found: ahmed, aisha, fatima,
mahmod, mailtest, omar, reem, spectrum
[*] Scaned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smtp_enum) >
```

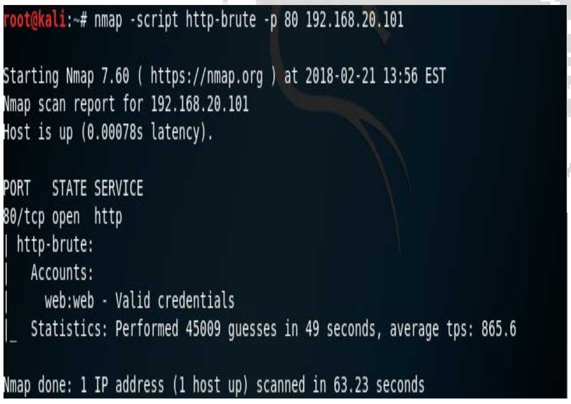
Fig. 6 SMTP_Enum Auxiliary San

Basically we can say that this attack mechanism resembles a port scan, and a port scan rule will detect this attack and generate true positive alarm alert or prevent it based on the rule action. The following rule is added to detect different scans and it is able to detect this auxiliary, the rule as below:

```
drop tcp any any -> any any (msg: "SYN
Attack"; flag:S, 12; sid: 1000003;)
```

C. *Brute Force Attack on HTTP server*

In this task, brute force attack performed using nmap options to gain a username and password for restricted access. Figure 7 shows the result of this attack.



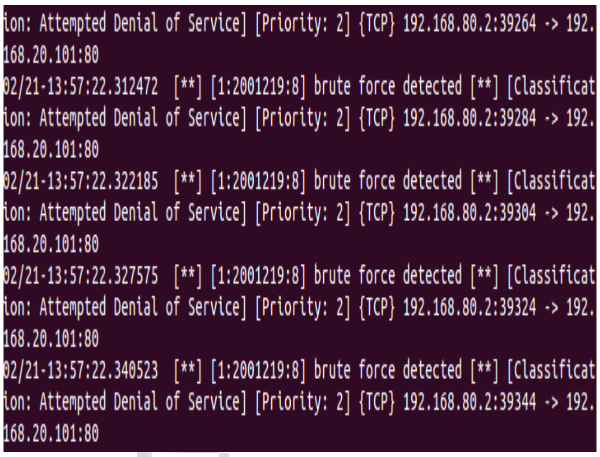
```
root@kali:~# nmap -script http-brute -p 80 192.168.20.101

Starting Nmap 7.60 ( https://nmap.org ) at 2018-02-21 13:56 EST
Nmap scan report for 192.168.20.101
Host is up (0.00078s latency).

PORT   STATE SERVICE
80/tcp open  http
| http-brute:
|   Accounts:
|     web:web - Valid credentials
|_  Statistics: Performed 45009 guesses in 49 seconds, average tps: 865.6

Nmap done: 1 IP address (1 host up) scanned in 63.23 seconds
```

Fig. 7 HTTP Brute Force Attack Result

Figure7 shows that the attacker has gained the username "*web"* and its password "*web"*as valid credentials in less than 50 seconds. In order to prevent such attacks, the following rule is added.

```
drop tcp any any -> 192.168.20.101 80
(msg:"HTTP brute force detected"; flags:S;
threshold:type threshold, track
by_src,count 10, seconds
60;classtype:attempted-dos; sid:2001280;)
```

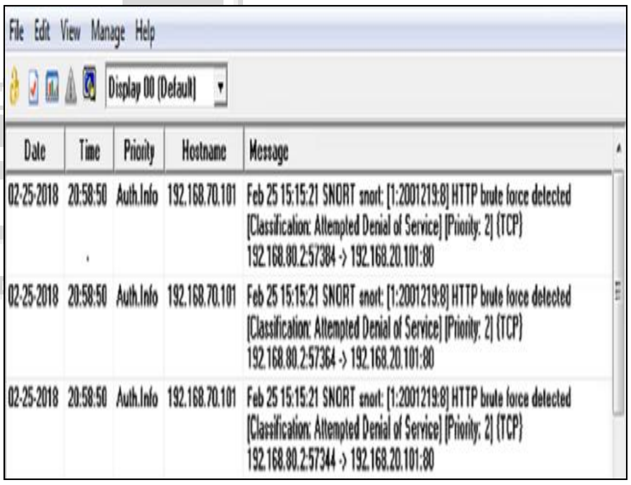
Figure 8 shows how this attack fails after activating the previously shown rule, and the generated message alert by Snort shows the socket number of the attack along with the socket number of the targeted system.



```
ion: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.80.2:39264 -> 192.
168.20.101:80
02/21-13:57:22.312472  [**] [1:2001219:8] brute force detected [**] [Classificat
ion: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.80.2:39284 -> 192.
168.20.101:80
02/21-13:57:22.322185  [**] [1:2001219:8] brute force detected [**] [Classificat
ion: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.80.2:39304 -> 192.
168.20.101:80
02/21-13:57:22.327575  [**] [1:2001219:8] brute force detected [**] [Classificat
ion: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.80.2:39324 -> 192.
168.20.101:80
02/21-13:57:22.340523  [**] [1:2001219:8] brute force detected [**] [Classificat
ion: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.80.2:39344 -> 192.
168.20.101:80
```

Fig. 8 HTTP Brute Force Alert

It is worth showing that while the attack is commencing, the generated event messages are received and logged on the Kiwi syslog server as shown in figure 9.



Date	Time	Priority	Hostname	Message
02-25-2018	20:58:50	Auth.Info	192.168.70.101	Feb 25 15:15:21 SNORT snort: [1:2001219:8] HTTP brute force detected [Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.80.2:57384 -> 192.168.20.101:80
02-25-2018	20:58:50	Auth.Info	192.168.70.101	Feb 25 15:15:21 SNORT snort: [1:2001219:8] HTTP brute force detected [Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.80.2:57364 -> 192.168.20.101:80
02-25-2018	20:58:50	Auth.Info	192.168.70.101	Feb 25 15:15:21 SNORT snort: [1:2001219:8] HTTP brute force detected [Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.80.2:57344 -> 192.168.20.101:80

Fig. 9 Kiwi Syslog Event messages

Figure 9 shows the time-stamped syslog event messages received and the priority configured on the syslog-ng before. Hostname and the IP address that the events received from, and the alert console message which is exactly as the console alert on Snort.

I. CONCLUSION

This paper outlines the implementation of an open source Intrusion Prevention System named Snort. During the complexity of the Intrusion Prevention system installation, some technical issues and difficulties exist. Snort requires a trained operator. In particular, it requires a systems administrator to understand Linux, experience using Snort, and more importantly experience with the TCP/IP protocol suite. The learning curve for writing and applying effective Snort rules is not too steep for IT professionals. As with all security tools, Snort should not be looked at as the final step to defend networks as much as a single brick in the wall of network security. Snort is a tool, and tools are only as good as their users. If users truly want to secure the network, no number of intrusion detection systems and firewalls will keep the attackers out. Common sense and diligence in the day-to-day monitoring are still the rules to live by for effective network security.

ACKNOWLEDGMENT

We are highly indebted to the Department of Computer Networking at College of Computer Technology Tripoli for providing necessary equipment, software regarding this research and access to their production networks and use of facilities.

REFERENCES

[1] Intrusion Detection with SNORT: Advanced IDS Techniques Using SNORT, Apache, MySQL, PHP, and ACID , Rafeeq Ur Rehman, May 18, 2003.

[2] Syngress - Snort 2.1 Intrusion Detection 2nd edition , Jay Beale, Andrew R. Baker,2004.

[3] Z. Trabelsi and S. Zeidan, “IDS Performance Enhancement Technique based on Dynamic Traffic Awareness Histograms”; IEEE ICC Communication and Information Systems Security Symposium, 2014.

[4] Network IDS & IPS Deployment Strategies, SANS Institute Reading Room site; https://www.sans.org/reading-room/whitepapers/intrusion/network-ids-ips-deployment-strategies-2143.

[5] Managing Security with Snort & IDS Tools: Intrusion Detection with Open Source Tools , Kerry J. Cox , Christopher Gerg, Aug 2,2004.

[6] Linux iptables Pocket Reference: Firewalls, NAT & Accounting (Pocket Reference (O'Reilly)) Gregor N. Purdy,2004.

[7] Snort For Dummies ,Charlie Scott, Paul Wolfe, and Bert Hayes,2004.

Mohamed Ghretli was born in Tripoli, Libya, in 1965. He received the B.S.E. (Hon) degree in applied physics from McMaster University, Hamilton, ON, Canada, in 1984, the M.Sc. degree in applied optics from Windsor University, Windsor, ON, in 1992, and the Ph.D. degree in microwaves from the University Putra Malaysia, Selangor, Malaysia in 2005. He received the Gold Medal for best invention in Geneva, Switzerland in 2007 for his valuable research on dual frequency micro-strip sensor published in IEEE SENSORS JOURNAL, VOL. 7, No.12. Currently, he is an associate professor at the College of Computer Technology Tripoli and the head of Cisco Networking Academy. His research Interests includes computer networking, data communication, micro-strip antenna and microwave sensors.

Rima Aburmaylah was born and raised in Tripoli, Libya, in 1994. She received her B.S.E. degree from College of Computer Technologies Tripoli in Networking 2018, besides her passion in the network security field, Rima is interested in Business, strategic planning, and project management, and she works as a solution engineer and as a member of the project management team in IT Company in Libya.

Abdulmunem Almukhtar received the B.S.E degree in Electronic and Computer Engineering from College of Industrial Technology, Misurata, Libya, in 1995, the M.Sc. degree in Computer Engineering from The Libyan Academy in 2010. His recent publications include "Parameter Estimation and Residual Generation for (CARARMA) Model Depend on D-RGELS Algorithm", 4th IEEE Scientific Meeting on Computer Science, in 2018, Istanbul, Turkey. Currently, he is lecturer at the College of Computer Technology Tripoli and the head of Faculty Affairs Office.

Sabri Shmeekh received HND in Computer Hardware from CAHI , Esbia, Libya in 1991 and MS.C in IT from RMIT University, Australia in 1999. He earned many international certificates during his career in Cisco Academy and EMCPA Associate Information Storage and Management. Currently, he is a lecturer at the College of Computer Technology Tripoli and the head of Networking Department.