The project "Capture and analyzing of packets in virtual network".

Georgii Moiseev

Introduction: I chose this topic for my project because I am really interested in working with virtual machine boxes. It's interesting how packets go through the internet and network connection, how they can be captured and used even for compromising the users based on the information that was stolen, how users' data can be obtained surreptitiously and misused. Finally, it is interesting to find out about the systems with which our system interacts. For example, through the IP and MAC address. In most cases, when somebody tries to compromise and steals data, it calls *spoofing attack*. Besides *spoofing attacks,* there exist a bunch of other attacks that can be used using packets like *Teardrop, ICMP Tunnelling, Smurf attack* and etc. Therefore, in the cases when we need to get result in *protection* of our network or even avoiding malicious party, we might use *IPS/IDS*. With such tools, we can make good or sometimes fair security. Thereby, I would like to explain what *IDS* and *IPS* exactly are. So, the *IDS* is an *intrusion detection system* that monitors a network for possible *malicious* activities. When the system has detected such activity, *IDS* informs the user that the threat was detected, but it doesn't take any actions to prevent *malicious* activity. Meanwhile, the IPS after detection *malicious* activity starts to prevent future attacks and possible dangerous activities out of our operating system.

Scenario:

My project is about capturing, observing and analyzing the packets between virtual operating systems, using *Snort/tcpdump*. Moreover, *IDS*(Snort) with set rule "alert" warns about incoming *TCP/IP* by ping command that is mostly used as troubleshooting for connectivity and checking if the connection is approachable. The main function of ping command is sending ICMP ECHO requests to network hosts.

In the beginning of my experiment, from *Windows 10* command prompt I pinged the *Kali Linux*(198.111.251.4) with 32 bytes of data. So, 4 packets were sent and 4 packets were received with 0% loss. It took 0 ms for a trip because in fact, it's the same network area. Besides that, with *Linux* operating system, I pinged *Windows 10*, where I sent and received 6 packets for 79ms with 0% loss. It proves that connection between these 2 operating systems works.

Other tool that helps to check and analyze the packets is *Wireshark*. Though it is not IDS, it helps to capture packets and find out the details about the content of packets. Therefore, I wanted to check what is *packet analyzer* and what it indeed can do. For example, I opened the http://testing-ground.scraping.pro/login and started to capture packets with *Wireshark*.

On that website, I logged in with user "admin" and password "12345". It let me in. My next step was that I stopped capturing packets with *Wireshark*. After that, on *Wireshark* application, I picked the packet that is correlated with HTML and in the line "HTML Form URL Encoded: application/x-www-form-urlencoded" I saw my input. Later, I moved back to my virtual operating systems.

My next step was from *Linux*(198.111.251.4) I pinged my own operating system and captured packets of ICMP with *Wireshark*. I got the list of echo(ping) requests and responses. It made me believe that I can indeed start to work with another important tools and finally try IDS out. As the result, it was interesting testing, nevertheless, it's not my main topic of the project. So, I went further in testing, capturing and analyzing packets.

Later, I pinged the *virtual* machine Windows 10 from Kali Linux and I observed that 7 packets were sent and received back without any loss. It means that a ping command trial was successful. I can say it because it can be seen from one of data packets, captured with *Snort*:

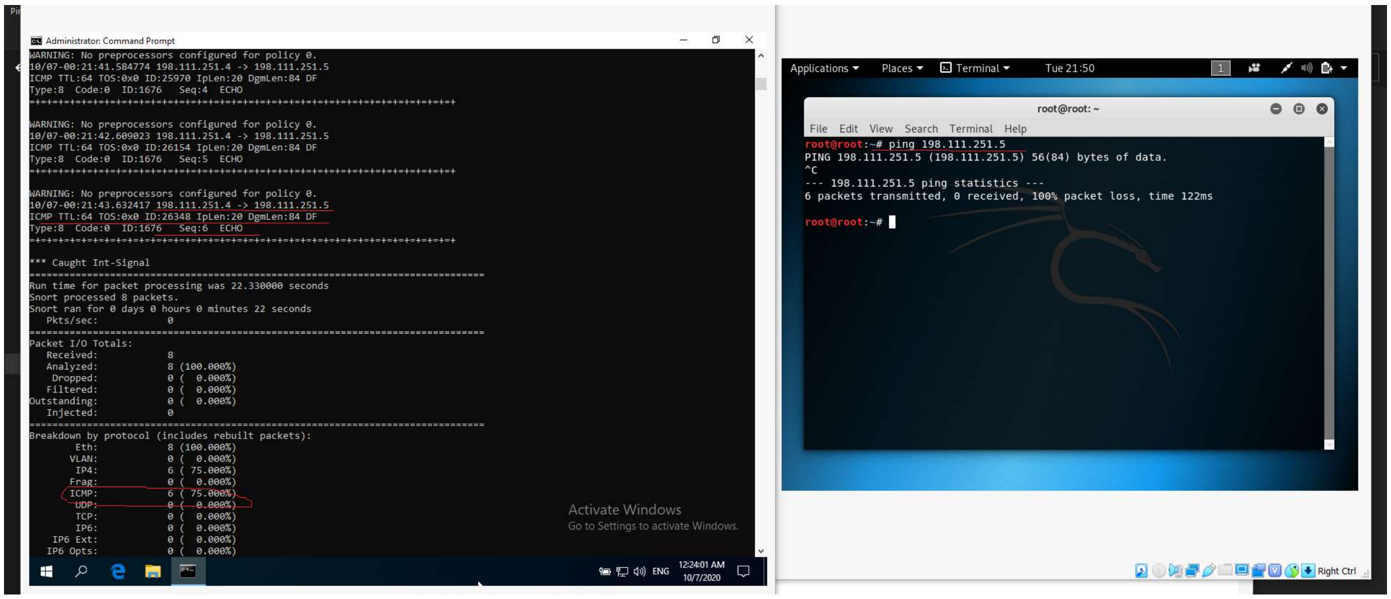
198.111.251.4 -> 198.111.251.5

ICMP TTL:64 TOS:0x0 ID: 26788 IpLen:20 DgmLen: 84 DF

Type:8 Code:0 ID:1676 Seq:6 ECHO

and in the final list I see ICMP: 6(75.000%)

*More details in **Analysis** section.*



```
WARNING: No preprocessors configured for policy 0.
10/07-00:21:41.584774 198.111.251.4 -> 198.111.251.5
ICMP TTL:64 TOS:0x0 ID:25970 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:1676   Seq:4  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
10/07-00:21:42.609023 198.111.251.4 -> 198.111.251.5
ICMP TTL:64 TOS:0x0 ID:26154 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:1676   Seq:5  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
10/07-00:21:43.632417 198.111.251.4 -> 198.111.251.5
ICMP TTL:64 TOS:0x0 ID:26348 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:1676   Seq:6  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

*** Caught Int-Signal
===============================================================================
Run time for packet processing was 22.330000 seconds
Snort processed 8 packets.
Snort ran for 0 days 0 hours 0 minutes 22 seconds
   Pkts/sec:            0
===============================================================================
Packet I/O Totals:
   Received:            8
   Analyzed:            8 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:            8 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:            6 ( 75.000%)
       Frag:            0 (  0.000%)
       ICMP:            6 ( 75.000%)
        UDP:            0 (  0.000%)
        TCP:            0 (  0.000%)
        IP6:            0 (  0.000%)
    IP6 Ext:            0 (  0.000%)
   IP6 Opts:            0 (  0.000%)
```

```
root@root:~# ping 198.111.251.5
PING 198.111.251.5 (198.111.251.5) 56(84) bytes of data.
^C
--- 198.111.251.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 122ms

root@root:~# 
```

Nevertheless, I am going to use the Snort in a more efficient way and firstly, I want to set rules. Set rules will help to identify what kind of packets, protocols and IP addresses we try to connect to our operating system. Therefore, warning messages notify us what exact incoming traffic our operating system receives.

Tools: To cut it short, I just say that my tools that I am going to use are: *Snort*, *Wireshark*, *tcpdump* and Oracle *VM VirtualBox Manager*.

Setup: My first step for setup was preparations of the operating systems. I downloaded and set up 2 operating systems (*Linux 2.6, Windows 10*) on my virtual machine box. Furthermore, in the settings I created the virtual network and called it as "Project Network". Also, I changed the connection of my virtual operating systems and connected them to "Project Network" and now my standard IP address for virtual machines is 198.111.251.4.

Also, I have downloaded my *IDS*(*Snort*) on *Windows 10*. But after the first installation I had to open the file and change few lines for appropriate functionality, because *Snort* wasn't created only for *Windows,* but mostly created for *Linux*. So, that's why I had to make some changes in the text file.

In **snort.conf** I set up the network address which I want to protect. For that goal I wrote "ipvar HOME_NET 198.111.251.0/24". After that I scrolled down to RULE_PATH, and replace

../rules with c:\Snort\rules and replace *../so_rules* with *c:\Snort\so_rules*.

At last, I replaced *../preproc_rules* with *c:\Snort\preproc_rules*

I changed the WHITE_LIST_PATH and BLACK_LIST_PATH from *../rules to c:\Snort\rules*

I set *#config logdir:* to *config logdir:c:\Snort\log*

I changed *usr/local/lib/snort_dynamicpreprocessor* with *C:\Snort\lib\snort_dynamicpreprocessor.*

Similarly, I replaced *usr/local/lib/snort_dynamicengine/libsf_engine.so* with *C:\Snort\lib\snort_dynamicengine\sf_engine.dll.*

I added a comment(#) under inline packet normalization.

Also, I changed *C:\Snort\etc\classification.config* and *C:\Snort\etc\reference.config*.

I found and replaced all ipvar to var. And commented with # lines 501-507

In the next line, I added *output alert_fast: alert.ids* for snort to dump all logs in alert.ids.

Mechanism:

I manipulated the captured data - analyzed packets, but the core of my project is to set alerts and test them. In *local rules* I typed "**alert icmp any any -> any any (msg:"Testing ICMP alert** ";

sid: 1000001;)

If look closer in typed rule:

"**alert**" generates an alert message when the rule is able to be applied for a specific captured packet.

"**icmp**" means that the rule will be applied on all ICMP packets.

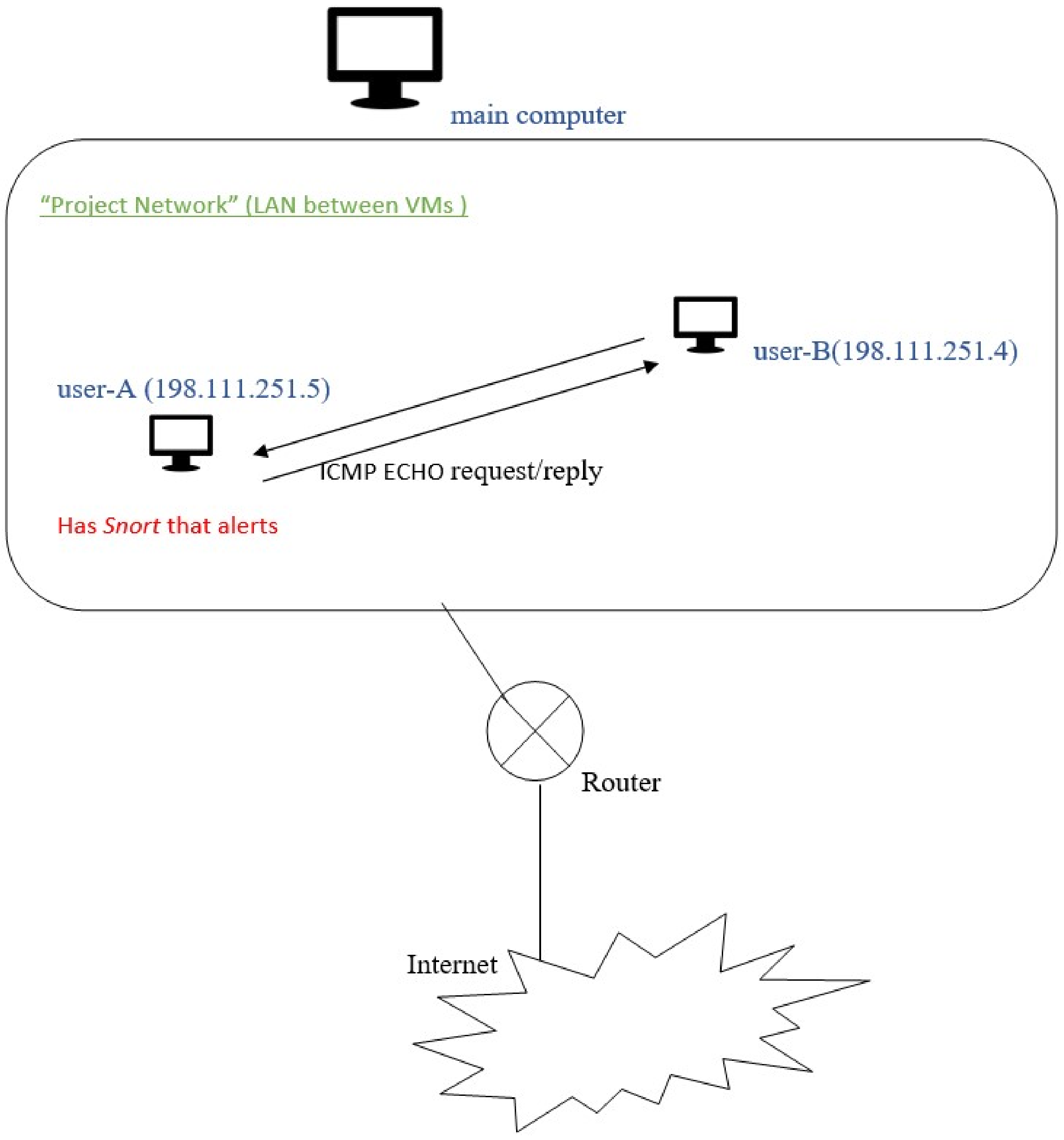
the first "**any**" used for the source IP address and the second for port number.

-> shows the direction of the packet.

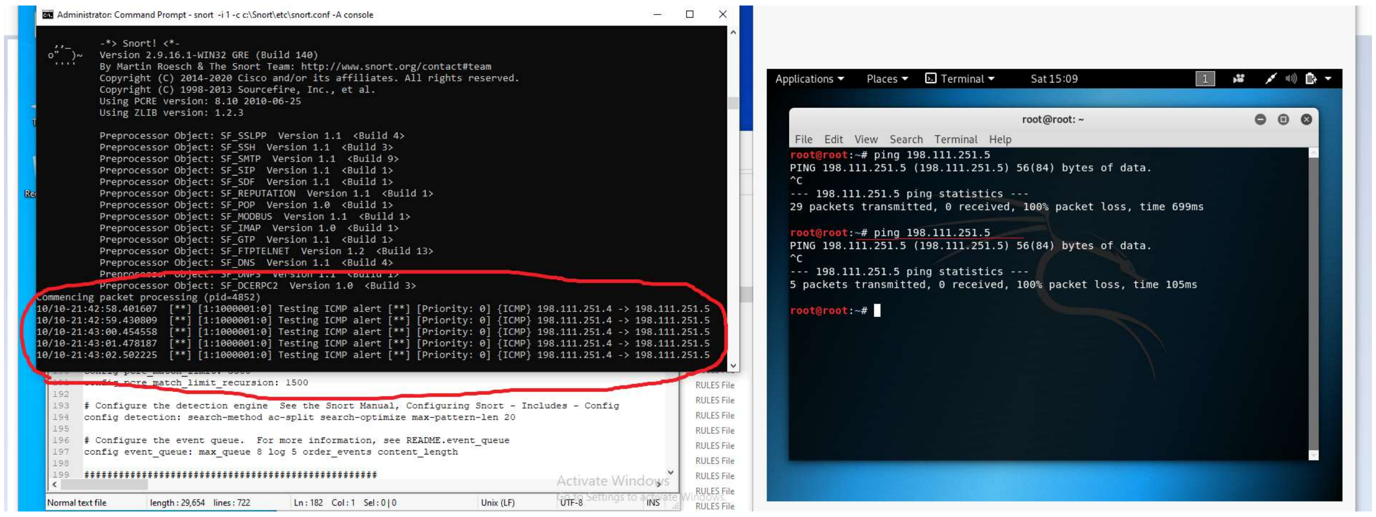
the 3rd "any" used for the destination IP address and 4th for destination port

And in the brackets we have rule options: the message and ***<u>sid</u>*** keyword that allows to identify Snort rules uniquely (it supposes to be greater than 999999).

There is a diagram of hosts (main operating system and 2 virtual operating systems that are sending ICMP ECHO request/reply to each other). <u>Only 2 virtual operating systems participate in the experiment.</u>


main computer
"Project Network" (LAN between VMs)
user-B(198.111.251.4)
user-A (198.111.251.5)
ICMP ECHO request/reply
Has *Snort* that alerts
Router
Internet

Result: This rule allows to identify what kind of packets and alerts the operating system about them when any other virtual machines try to connect with *Windows 10*. So, the *Snort* lets me know when and what machine was pinging, and gives the message "Testing ICMP alert", so I exactly know which machine tried to interact(IP address), in what time it tried to connect because we predict the next step of unknown operating system. I can declare that my *local rule* works, so the experiment is successful:


Administrator: Command Prompt - snort -i 1 -c c:\Snort\etc\snort.conf -A console

```
   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.16.1-WIN32 GRE (Build 140)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2020 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.3

           Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
           Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
           Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
           Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
           Preprocessor Object: SF_SDF  Version 1.1  <Build 1>
           Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
           Preprocessor Object: SF_POP  Version 1.0  <Build 1>
           Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
           Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
           Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
           Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
           Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
           Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
           Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
Commencing packet processing (pid=4852)
10/10-21:42:58.401607  [**] [1:1000001:0] Testing ICMP alert [**] [Priority: 0] {ICMP} 198.111.251.4 -> 198.111.251.5
10/10-21:42:59.430809  [**] [1:1000001:0] Testing ICMP alert [**] [Priority: 0] {ICMP} 198.111.251.4 -> 198.111.251.5
10/10-21:43:00.454558  [**] [1:1000001:0] Testing ICMP alert [**] [Priority: 0] {ICMP} 198.111.251.4 -> 198.111.251.5
10/10-21:43:01.478187  [**] [1:1000001:0] Testing ICMP alert [**] [Priority: 0] {ICMP} 198.111.251.4 -> 198.111.251.5
10/10-21:43:02.502225  [**] [1:1000001:0] Testing ICMP alert [**] [Priority: 0] {ICMP} 198.111.251.4 -> 198.111.251.5
```

```
config pcre_match_limit_recursion: 1500

# Configure the detection engine  See the Snort Manual, Configuring Snort - Includes - Config
config detection: search-method ac-split search-optimize max-pattern-len 20

# Configure the event queue.  For more information, see README.event_queue
config event_queue: max_queue 8 log 5 order_events content_length

###################################################
```

Normal text file
length : 29,654   lines : 722
Ln : 182   Col : 1   Sel : 0 | 0
Unix (LF)
UTF-8
INS
RULES File
Activate Windows
Go to Settings to activate Windows.
Applications
Places
Terminal
Sat 15:09
root@root: ~
File Edit View Search Terminal Help

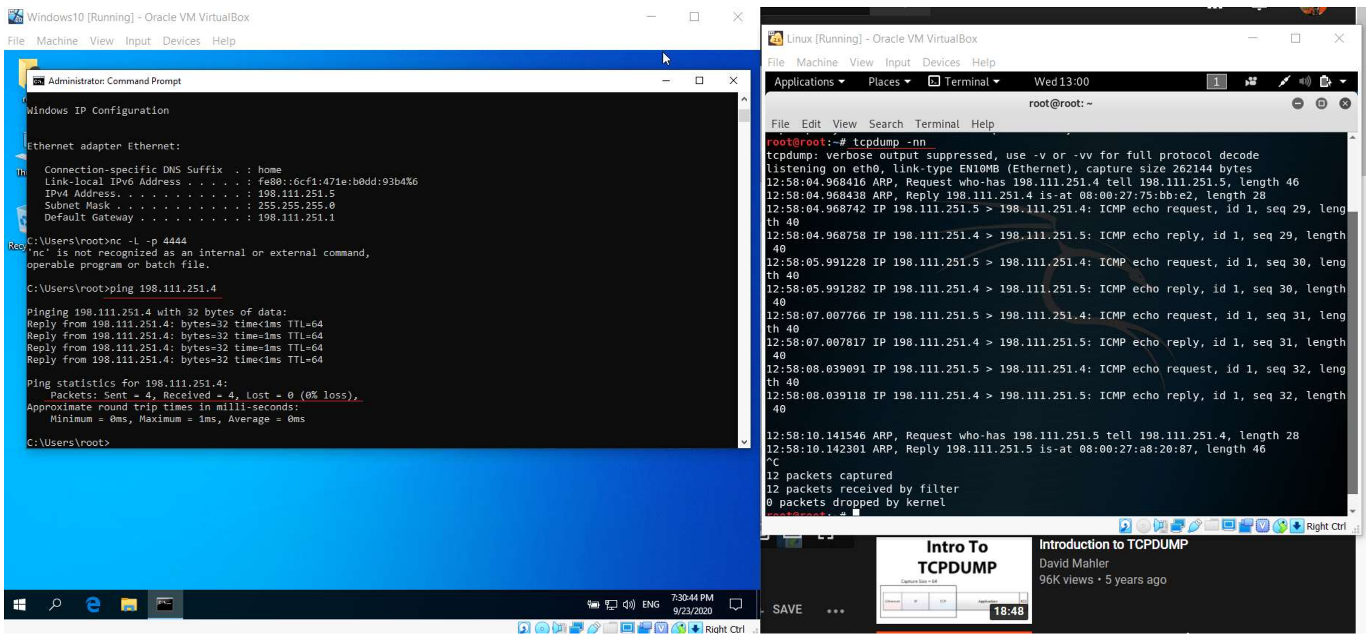
```
root@root:~# ping 198.111.251.5
PING 198.111.251.5 (198.111.251.5) 56(84) bytes of data.
^C
--- 198.111.251.5 ping statistics ---
29 packets transmitted, 0 received, 100% packet loss, time 699ms

root@root:~# ping 198.111.251.5
PING 198.111.251.5 (198.111.251.5) 56(84) bytes of data.
^C
--- 198.111.251.5 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 105ms

root@root:~# 
```

As a result, we can observe the information that we got. And in fact, the packets that we captured can say a lot about the system, which try to intrude. Simply speaking, we have created the rule that warns the user/administrator about any outer intrusion. My goal was to find the way to identify any outgoing intrusions, and *Snort* helped us with that. In fact, it turns out that it is useful tool for testing network traffic detection. Moreover, the most important is that it gives us the ability to identify not the only ICMP protocols, but also other protocols that can be caused to dangerous outcomes for our operating system. By setting the rules *Snort,* for example — we can identify and check the TCP and UDP traffic detection.

Analysis: So I pinged *Kali Lunix* virtual operating system with *Windows 10* and sent 4 packets that were sent there and received back without loss. Meanwhile, on *Kali Lunix*, I used the tcpdump so I could see the traffic that was sent, the length of it and IP of the sender (*Windows 10*) because I typed "tcpdump -nn". It means to display the *numbers* for machine and *ports*. Well, it gave me specific information: link-type EN10MB (*Ethernet*), capture size 262144 bytes. However, it is known that Ethernet is a computer *networking technology* that is used in *local* and *wide* area networks. So, as we can see, Ethernet is used for our local area network (LAN). As for the size of 262144 bytes, it is default value of snapshot size that helps to set the smallest number that will capture the protocol information you're interested in. So, we sent 4 packets and got them back. I will try to explain the process in steps by **referring details**:



Windows10 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Administrator: Command Prompt
Linux [Running] - Oracle VM VirtualBox
Applications Places Terminal Wed 13:00
root@root: ~
File Edit View Search Terminal Help
Intro To TCPDUMP
Introduction to TCPDUMP
David Mahler
96K views • 5 years ago
SAVE
18:48

```
Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : home
   Link-local IPv6 Address . . . . . : fe80::6cf1:471e:b0dd:93b4%6
   IPv4 Address. . . . . . . . . . . : 198.111.251.5
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 198.111.251.1

C:\Users\root>nc -L -p 4444
'nc' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\root>ping 198.111.251.4

Pinging 198.111.251.4 with 32 bytes of data:
Reply from 198.111.251.4: bytes=32 time<1ms TTL=64
Reply from 198.111.251.4: bytes=32 time=1ms TTL=64
Reply from 198.111.251.4: bytes=32 time=1ms TTL=64
Reply from 198.111.251.4: bytes=32 time<1ms TTL=64

Ping statistics for 198.111.251.4:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\Users\root>
```

```
root@root:~# tcpdump -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:58:04.968416 ARP, Request who-has 198.111.251.4 tell 198.111.251.5, length 46
12:58:04.968438 ARP, Reply 198.111.251.4 is-at 08:00:27:75:bb:e2, length 28
12:58:04.968742 IP 198.111.251.5 > 198.111.251.4: ICMP echo request, id 1, seq 29, leng
th 40
12:58:04.968758 IP 198.111.251.4 > 198.111.251.5: ICMP echo reply, id 1, seq 29, length
 40
12:58:05.991228 IP 198.111.251.5 > 198.111.251.4: ICMP echo request, id 1, seq 30, leng
th 40
12:58:05.991282 IP 198.111.251.4 > 198.111.251.5: ICMP echo reply, id 1, seq 30, length
 40
12:58:07.007766 IP 198.111.251.5 > 198.111.251.4: ICMP echo request, id 1, seq 31, leng
th 40
12:58:07.007817 IP 198.111.251.4 > 198.111.251.5: ICMP echo reply, id 1, seq 31, length
 40
12:58:08.039091 IP 198.111.251.5 > 198.111.251.4: ICMP echo request, id 1, seq 32, leng
th 40
12:58:08.039118 IP 198.111.251.4 > 198.111.251.5: ICMP echo reply, id 1, seq 32, length
 40

12:58:10.141546 ARP, Request who-has 198.111.251.5 tell 198.111.251.4, length 28
12:58:10.142301 ARP, Reply 198.111.251.5 is-at 08:00:27:a8:20:87, length 46
^C
12 packets captured
12 packets received by filter
0 packets dropped by kernel
root@root:~#
```

1) The user-A (**198.111.251.5**) initiates *the connection* to the user-B (**198.111.251.4**) at 12:58:04.968416 with *the Address Resolution Protocol*(ARP) that requests by saying "Request

who-has **198.111.251.4** tell **198.111.251.5**, length 46". So, in other words we can say that the *host* user-B(**198.111.251.4**) wants to send the packets to user-A (**198.111.251.5**), but user-B doesn't have the information about the MAC-address, so it sends APR *request* with the length 46 bytes. Moreover, 46 bytes is less than 64 bytes (Ethernet frame minimum size), therefore we can say that the request was sent to the same machine.

2) After the client is getting the reply, saying "Reply **198.111.251.4** is-at 08:00:27:75:bb:e2, length 28" and APR found the MAC-address(Ethernet address) and gives it as reply with the size 28 bytes of packet. So, now we have a chance to connect and start "request/reply" process.

3) "Request/reply" process started. For example, for request we have "IP **198.111.251.5 > 198.111.251.4**: ICMP echo reply, id 1, seq 29, length 40". It says that user-A(**198.111.251.5**) sends ICMP echo request to user-B (**198.111.251.4**) with id 1 that used for identifying the parts of a fragmented datagram, various "seq" sequence that means the byte number of the first byte of data which was sent in the packet and fixed length 40 that identifies the IP packet length in bytes.

4) Then, the user-B (**198.111.251.5**) closes connection and makes similar process that was made with an initiation of connection. At 12:58:10.141546 user-B with ARP requests the data that user-A (**198.111.251.5**) should provide. With "Request who-has **198.111.251.5** tell **198.111.251.4**, length 28" we know that user-A(198.111.251.5) wants to get the packets from user-B(198.111.251.4), but user-B doesn't know the MAC-address of user-A, so in that case ARP was used and user-B(**198.111.251.4**) is getting the MAC-address "08:00:27:a8:20:87" from user-A with *packet* length 46. It is known form "Reply 198.111.251.5 is-at 08:00:27:a8:20:87, length 46". Also, we can notice that we got the same length of *packets* (28 and 46 bytes) that were sent in the first time. We got them back but backwards.

5) Finally, the connection is closed.

It's what we can observe with the *tcpdump* (how connection works). However, now we need to observe the table with packets captured with *Snort*. Previously, if you recall we discussed the basic sniffing with *Snort* and records of captured packets that we got. Those records are:

198.111.251.4 -> 198.111.251.5

ICMP TTL:64 TOS:0x0 ID: 26788 IpLen:20 DgmLen: 84 DF

Type:8 Code:0 ID:1676 Seq:6 ECHO

and in the final list I see ICMP: 6(75.000%)

So, we can see the pinging machine *Linux* (198.111.251.4) sends ping requests to *Windows 10* (198.111.251.5). In the second line, we specifically have "ICMP TTL" that means ICMP Time To Live - the amount of "hops" that a ICMP packet is set to "live" inside of a network before the router can get rid of it. The number 64 means how many "hops" the packet can travel before dropping. TOS 0x0 means Type of Service that parameters for an IP Profile with the size 0x0(NULL). As for "ID: 26788", it is *id* of the packet. IpLen is the IP header length. DgmLen is the total datagram or packet length. Moreover, both IpLen and DgmLen are contained in the IP header of all packets. The line "Type:8 Code:0 ID:1676 Seq:6 ECHO " is parameters of ICMP that mean Type 8 - Echo, Code 0 means Echo Reply, ID: 1676 is id of the message, and finally the seq:6 is the first bytes that were sent. And in the end, we have the table of summarized information about all packets that we captured. And it gives us the chart in percentile where ICMP has 75% of all packets that were received. Thus, *Snort* can give more statistically interesting information than *tcpdump.*

Recommendation: I want to say that I would use Linux operating system for most of the experience rather than *Windows 10* because *Windows 10* has only one advantage — more friendly interface, but it's not convenient for *IDS/IPS* settings. It's obvious because Windows was created for usual users, meanwhile, Linux was created for people who are more aware of coding and know how to interact with the operating system. Moreover, most of the hacking tools that exist are created for UNIX *Linux* for working with their terminals. The last thing that I would like to mention about *Windows* is that on *Windows 10* it is harder to save our anonymity, which is a very important feature during the hacking process.

Also, I would like to suggest setting rules for identification — only protocols that are important mostly for the working employee do their job rather than try to predict all possible unknown IPs and servers that are not included in the list of safe IPs. Simply speaking, set rules to warn users about any unknown IPs and protocols. So instead of avoiding all malicious connectivity, we need to set rules for warning any malicious connectivity and stay with the safe connectivity. It goes without saying that it is better to set rules that identify safe and known IPs and protocols for overall security and prevention of a bad outcome.

Besides that, I would recommend to add "rev" that is used to uniquely identify revisions of Snort rules. Revisions allow to improve or replace signatures and descriptions with updated information. And add "priority" tag that gives the knowledge how risky can be outgoing traffic. For example, it can notify "Executable code was detected" that has a high priority and in most cases when the user doesn't know where specific code came from, so it possibly can be a virus.

Conclusion: In fact, I can say that we can find out the password on the site, which doesn't have encryption (HTTP site). We are able to check the connection between the virtual machines or any other kinds of operating systems that try to set an unpredictable connection with our machine. We can set rules to check what packets go in and go out, so we will be aware of what exactly goes with the traffic, which protocol and IP address. Also, with setting rules we are able to do useful detection and possible protection — for example, we can set a rule to warn that if someone tries spoofing our machine. Finally, by analyzing the packets, we are able to find where and when the loss of packets happened, because some lost packets can be very important and contain company data or even network signatures. Therefore, capturing and analyzing of packets guarantee the security, reliability and high performance of a company network for which you are working.

Sources:

• Jelena Mirkovic, Sven Dietrich, David Dittrich, Peter Reiher - Internet Denial of Service Attack and Defense Mechanisms   (2005, Prentice Hall)

• Stallings, William_ Brown, Lawrie - Computer Security Principles and Practice (2018, Pearson Education Limited)

• https://www.youtube.com/watch?v=naLbhKW62nY&t=120s with title "How to install and run Snort on Windows"

• https://www.youtube.com/watch?v=Fv0g1Fgjykc&t=133s with title "EXPLAINED: TCPDUMP and How to Sniff and Analyse tcp packet (Step-by-step Guide)".

• https://www.dnsstuff.com/ids-vs-ips, en.wikipedia.org

• https://www.informit.com/articles/article.aspx?p=101171&seqNum=2

• http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node31.html