Network2 Report

Manpreet Parmar
April 13,2023

Summary

The analysis that was done in this report consists of the raw data of three networks: ec2, log-server, and web. In order to process the data in these networks, a set of scripts and tools was used to allow for the processing of log files and searching of malicious data. The results of the analysis are as follows. During the span of the packets captured by the sniffer, a pattern of unknown IP addresses attempting to connect to the SSH server of the EC2 web server was noticed. Though some of these attempts were successful, the users that were logged into were not ones that had any significance. What did become a matter of significance however is how these ssh connections were made.

The IP addresses would attempt to connect to the SSH server through a random sequence of ports on the client's side, but only attempt to connect to the server using its default SSH port 22. Furthermore, an IP address '183.182.109.2' was flagged by the audit service of the server to have numerous attempts at trying to log into the web server through the 'root' user. Each of these attempts were logged as "failed' but this does show signs that an attacker might be trying to gain control of the server.

Now near the last few capture files made by the sniffer, the server shows signs of abnormal behaviour. The server makes a total of 4 get requests, which are downloading files that have been classified as "malware". The malware downloaded was a type called "XordDos" which when executed, performs a dDos attack on a server, thus impacting it's services and limiting its functionality. The attacker most likely used something called a "botnet" which is a net of bots that enact the attack by first brute forcing the ssh connection with the server, and then infecting it with a malware which would then lead onto a series of events such as the ddos attack.

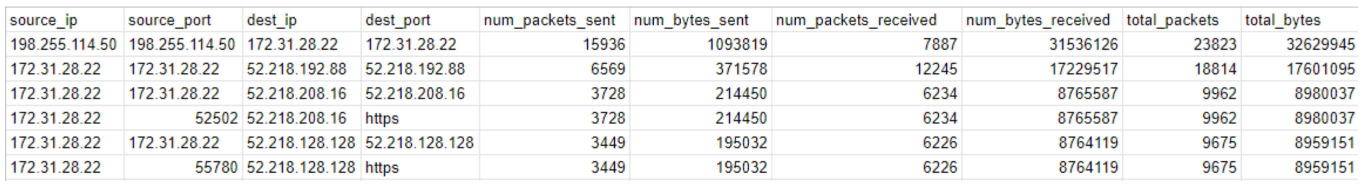
Introduction

In this report, we will be performing in depth analysis on raw data that is based on 3 networks: ec2, log-server, and web. These directories contain files such as tcpdump outputs and log files which each will be analysed together to figure out what is going on in the network. The goal of this report is to explain the main areas of traffic where attention should be placed, in order to figure out where things went wrong. This means looking at items such as system vulnerabilities and malicious traffic that has occurred on the networks. In order to do so, the use of analytical tools and scripts will be needed in order to automate the processing of the data as much as possible. Furthermore, to show the results of the analysis done tables and graphs containing evidence to back the claims made will be shown as well. The traffic types will be separated into three parts: benign, reconnaissance, and malicious. Benign traffic is traffic that performs as normal traffic in an EC2 web server would. Reconnaissance traffic is traffic in which an attacker is attempting to find vulnerabilities within the system. Malicious traffic is traffic that is performing outside of the normal standards set by the network, this type of traffic can be dangerous to the network and must be dealt with as soon as possible.

Analysis

Overview of Network Activity

In order to find out the general statistics of the network, first what needed to be done was to gather information about the conversations taking place in the capture files. In order to do so, a script was made to import each of the files in the /capture folders of the two networks, ec2 and log-server, and the conversations of each file were then outputted into an output.txt file. After the output.txt file was created, to make the information of each conversation easier to sort, the conversations were turned into a .csv file which stored columns of the parameters of each entry including their source IP, destination IP and total bytes. The picture below will show how the excel sheet was set up.

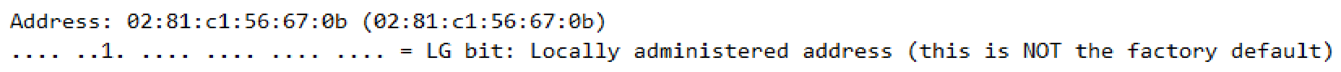


source_ip	source_port	dest_ip	dest_port	num_packets_sent	num_bytes_sent	num_packets_received	num_bytes_received	total_packets	total_bytes
198.255.114.50	198.255.114.50	172.31.28.22	172.31.28.22	15936	1093819	7887	31536126	23823	32629945
172.31.28.22	172.31.28.22	52.218.192.88	52.218.192.88	6569	371578	12245	17229517	18814	17601095
172.31.28.22	172.31.28.22	52.218.208.16	52.218.208.16	3728	214450	6234	8765587	9962	8980037
172.31.28.22	52502	52.218.208.16	https	3728	214450	6234	8765587	9962	8980037
172.31.28.22	172.31.28.22	52.218.128.128	52.218.128.128	3449	195032	6226	8764119	9675	8959151
172.31.28.22	55780	52.218.128.128	https	3449	195032	6226	8764119	9675	8959151

Using this information, the table below will show the source IP addresses that had the most traffic coming and going to them. The cutoff of IP addresses with the most traffic were any conversations which went below 1,000,000 bytes of data in total.

	MAC Address	Hostname/Domain
198.255.114.50	02:81:c1:56:67:0b	198-255-114-50.host-engine.com
172.31.28.22	MS-NLB-PhysServer-32_1b:4e:5c:cf:48	Unknown(Private IP)
183.182.109.2	02:81:c1:56:67:0b	183.182.109.2
198.98.56.172	02:81:c1:56:67:0b	198.98.56.172
198.98.50.201	02:81:c1:56:67:0b	198.98.50.201.ny3.mywebsitehosted.net
209.159.217.61	02:81:c1:56:67:0b	host-61.209-159-217.mybluepeak.net
173.208.179.117	02:81:c1:56:67:0b	trunk3.zaredesign.net

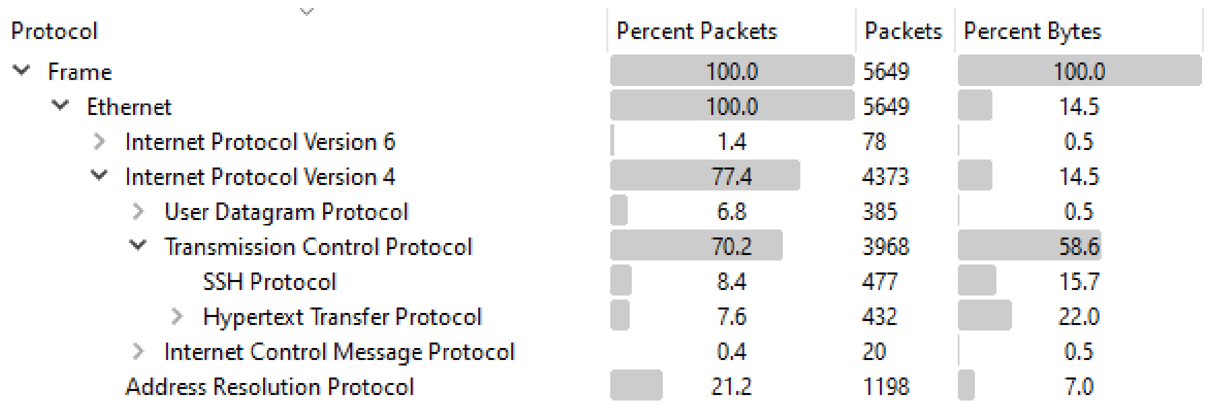
As we can see from the MAC address column of the table above, every IP address, except for 172.31.28.22 is assigned a local MAC address of 02:81:C1:56:67:0b. The reason behind this can be seen by looking at a packet that has one of these IP addresses shown below.



```
Address: 02:81:c1:56:67:0b (02:81:c1:56:67:0b)
.... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
```

The MAC addresses of these IP addresses are “Locally administered”, meaning that instead of using the devices MAC addresses assigned by the manufacturer of the NIC, the address was manually assigned by the network administrator. This information also shows us the significance of the IP address 172.31.28.22. There is a reason for it's MAC address being different from the others and the reason is that it is the IP address of the EC2 web server which the captures are based off of. This is also supported by the IP address being a private IP which can be seen by the second portion of the IP being "31" which is reserved for private IP addresses only.

Now to explain what the main protocols used in these networks are, let's look at the protocol hierarchy graph provided by wireshark in the screenshot below.



Protocol	Percent Packets	Packets	Percent Bytes
Frame	100.0	5649	100.0
Ethernet	100.0	5649	14.5
Internet Protocol Version 6	1.4	78	0.5
Internet Protocol Version 4	77.4	4373	14.5
User Datagram Protocol	6.8	385	0.5
Transmission Control Protocol	70.2	3968	58.6
SSH Protocol	8.4	477	15.7
Hypertext Transfer Protocol	7.6	432	22.0
Internet Control Message Protocol	0.4	20	0.5
Address Resolution Protocol	21.2	1198	7.0

As we can see, the traffic is mainly based off of the IPv4 protocol, which is then split down to the UDP, TCP, and ARP protocols. The traffic that this analysis will be looking at most however, will be the TCP protocol, and to be more specific the SSH and HTTP packets being sent to and from the server.

One thing that should be noted before going onto the next sections is that though the IP addresses above are where a good portion of the analysis being done will be focused, it does not mean that they are the only IP addresses that this analysis will be looking at. In each section, the IP addresses that are being reviewed will be listed in a table similar to the one above to give reference to traffic surrounding them.

Benign Traffic

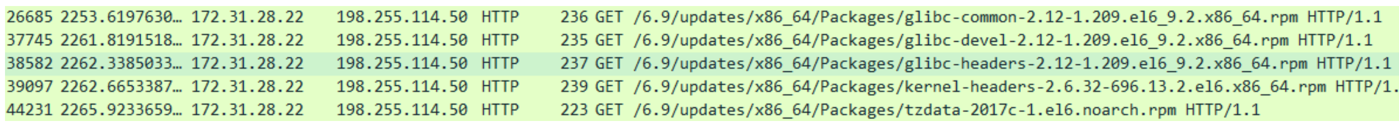
	MAC Address	Hostname/Domain
198.255.114.50	02:81:c1:56:67:0b	198-255-114-50.host-engine.com
198.98.56.172	02:81:c1:56:67:0b	198.98.56.172
198.98.50.201	02:81:c1:56:67:0b	198.98.50.201.ny3.mywebsitehosted.net
209.159.217.61	02:81:c1:56:67:0b	host-61.209-159-217.mybluepeak.net
173.208.179.117	02:81:c1:56:67:0b	trunk3.zaredesign.net

Now to begin on the analysis of the traffic. To explain what normal traffic would look like in this network, we are using the IP addresses above as reference. To validate that these files did not show any signs of malpractice or non normal activity, a shell script called “proc_log.sh” was made that would check in the log entries of the two folders /ec2 and /log-server to grab any log entries that had the IP addresses referenced. Also, some rules that will be explained later were set up to use with the suricata IDS in order to find any signs of abnormalities in the traffic present in the capture folders. But first, let’s explain how the traffic between the EC2 server and its clients works.

Incoming/Outgoing traffic would be in the form of either:

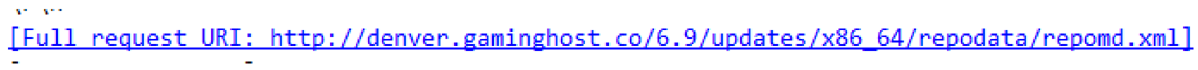
- HTTP Requests
- DNS Requests
- TCP traffic
- SSH

HTTP Requests



```
26685 2253.6197630… 172.31.28.22   198.255.114.50  HTTP     236 GET /6.9/updates/x86_64/Packages/glibc-common-2.12-1.209.el6_9.2.x86_64.rpm HTTP/1.1
37745 2261.8191518… 172.31.28.22   198.255.114.50  HTTP     235 GET /6.9/updates/x86_64/Packages/glibc-devel-2.12-1.209.el6_9.2.x86_64.rpm HTTP/1.1
38582 2262.3385033… 172.31.28.22   198.255.114.50  HTTP     237 GET /6.9/updates/x86_64/Packages/glibc-headers-2.12-1.209.el6_9.2.x86_64.rpm HTTP/1.1
39097 2262.6653387… 172.31.28.22   198.255.114.50  HTTP     239 GET /6.9/updates/x86_64/Packages/kernel-headers-2.6.32-696.13.2.el6.x86_64.rpm HTTP/1.
44231 2265.9233659… 172.31.28.22   198.255.114.50  HTTP     223 GET /6.9/updates/x86_64/Packages/tzdata-2017c-1.el6.noarch.rpm HTTP/1.1
```

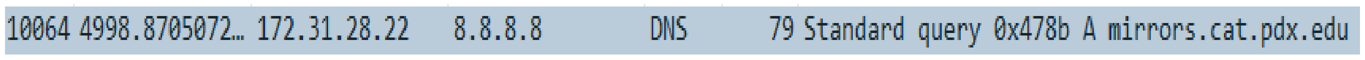
As we can see from the section of GET requests above, the server is normally going to be the one creating the GET request towards another IP address, in this case 198.255.114.50. These requests are usually going to be information that the server requires such as when the server is downloading updates on its current software. To classify a GET request as being something benign, it would be by looking at the type of files/directories being downloaded by the server. If the file is not associated with something that is common for a linux server to have downloaded, then more attention is needed in regards to the IP from where the files came from. Also, another thing to look at is the “Full request URI” of a HTTP GET request packet. In wireshark, when you expand a HTTP packet and go to the HTTP section of the TCP protocol, you can see where the full uri of the request came from, as seen below.



```
[Full request URI: http://denver.gaminghost.co/6.9/updates/x86_64/repodata/repomd.xml]
```

Using this URI, you can start to figure out where exactly this data that the server is downloading came from, and if it is coming from a reputable source.

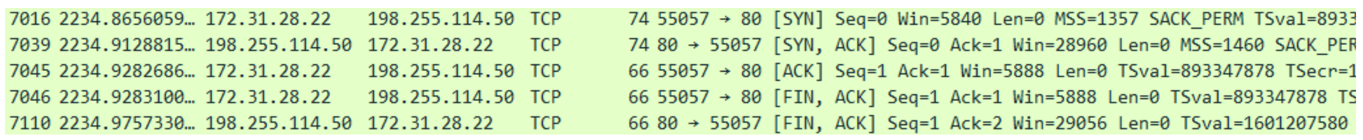
DNS Requests



```
10064 4998.8705072… 172.31.28.22    8.8.8.8         DNS      79 Standard query 0x478b A mirrors.cat.pdx.edu
```

A DNS request is something that the server makes in order to figure out what the IP address associated with a hostname is. It does so by sending this request to the destination IP 8.8.8.8, which is the public DNS resolver provided by google and is one that is normally used to resolve domain names. Typically, the DNS server’s used by a host are ones that are from a limited number of servers available that are trusted and authorised to provide DNS resolution to the network. The thing to mainly pay attention to however, is the query being made. If the DNS server is not able to resolve the IP address of a hostname, then there could be signs of something outside of normal traffic occurring.

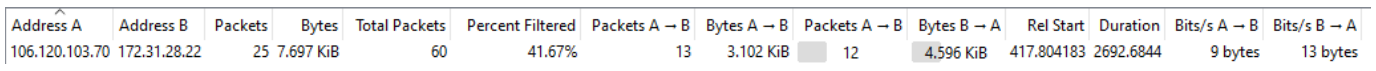
TCP Traffic



```
7016 2234.8656059… 172.31.28.22   198.255.114.50 TCP       74 55057 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1357 SACK_PERM TSval=8933
7039 2234.9128815… 198.255.114.50 172.31.28.22   TCP       74 80 → 55057 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PER
7045 2234.9282686… 172.31.28.22   198.255.114.50 TCP       66 55057 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=893347878 TSecr=1
7046 2234.9283100… 172.31.28.22   198.255.114.50 TCP       66 55057 → 80 [FIN, ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=893347878 TS
7110 2234.9757330… 198.255.114.50 172.31.28.22   TCP       66 80 → 55057 [FIN, ACK] Seq=1 Ack=2 Win=29056 Len=0 TSval=1601207580
```

TCP traffic consists of a series of connections being made, acknowledgements being sent and connections being closed. In a normal state, TCP traffic would see connections being made and after conversations have completed those connections being closed. To establish what is normal and abnormal in TCP traffic a few things can be focused on. In an attack that occurs over the TCP protocol, attackers will create hand crafted packets which function in abnormal ways. For example, if an attacker wanted to attack a server with a TCP SYN flood attack, that would mean they are sending a series of SYN packets to the server at rapid speeds, and when the server receives these packets, it will believe that the attacker is attempting to make a connection to the server and send SYN ACK's back for each SYN packet sent. Now what would happen after is that the attacker would never respond back to the server, meaning that the server would constantly be retransmitting SYN ACK packets, waiting for the reply back from the attacker. These types of attacks' goal is to use up the server's resources by overloading them with as many connections as possible. So in order to validify that the traffic related to TCP is working as normal, the speed at which connections are being opened and closed can be checked.

SSH Traffic



Address A	Address B	Packets	Bytes	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets A → B	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
106.120.103.70	172.31.28.22	25	7.697 KiB	60	41.67%	13	3.102 KiB	12	4.596 KiB	417.804183	2692.6844	9 bytes	13 bytes

Now with SSH traffic things get more difficult. Since communication between a host and client is done through encryption, the actual data being sent cannot be seen by someone outside of the connection. Though this does have the benefit of protecting the data being sent from a client to a server, and the other way around, it does make looking at the patterns of SSH traffic more difficult to figure out. So in order to look at how a conversation's behaviour is, we can look at things like the amount of data being sent. In the screenshot above, a normal SSH connection is shown with the amount of data being sent from the server to the client and from the client to the server being relatively the same. SSH conversations where one side is sending more data to the other should be paid more attention to, as there might be signs that data that isn't supposed to be sent over is being received.

Overall, using these patterns we will see how the malicious traffic in these three networks deviates from the normal, benign data.

Reconnaissance Traffic

Port Scanning

In order to figure out areas where vulnerabilities may be exposed, attackers will look at all avenues in order to find out whatever they can. In terms of ports, port scanning is a commonly used technique for attackers to do this. Port scanning works by having a device try every possible port that is on the target device, and see if any of these ports are “open”. Don’t mistake a port that is reachable to be an open port, an open port is a port where an active service has a connection based on it, for example port 80 for HTTP. To figure out if a port is open, the attacker will usually attempt to send a packet with the SYN flag set in order to get the server to send a response back. If the response sent by the server is a SYN ACK packet, then the port is currently open. If the response is a RST ACK packet, then the port currently has no active service running on it. If no response is received, then the port is most likely filtered to not give a response due to something like a firewall.

So in regards to how the reconnaissance traffic was searched for in the EC2 network’s, a suricata rule was made to alert of any traffic that has many SYN packets being sent. The rule can be seen below:

```
alert tcp any any -> any 1:65535 (msg:"TCP port scan detected"; flags:S; threshold: type
        both, track by_src, count 5, seconds 10; sid:1000001; rev:1;)
```

- any any: Looks at any source IP address and source port
- -> : The direction of packets
- any 1:65535: Looks at any destination IP with the destination ports of range 1 - 65535
- msg: The message printed in an alert created
- flags:S : Looks for packets with the SYN flag set
- threshold: type both, track by_src, count 5, seconds 10 : This will create a threshold for the alert to only active when 5 of the packets being searched for appear together in the span of 10 seconds

To summarise what this rule does, it checks to see if a conversation in the pcapng files has multiple SYN packets being sent from one client to another on the range of active ports. If the threshold for this has been met, the alert will activate and print the conversation to the log file. One thing to keep in mind though with this rule is the idea of false positives. A false positive is when an IDS creates an alert on something that wasn’t actually anything malicious. In the case of checking over the span of all of the pcapng files, this rule will definitely have some false positives appearing. But this does not mean that the information created by it can’t be used. By cross referencing the log file created by this script with IP addresses that are being investigated, the information of the log file can help to narrow down the search of compromised devices.

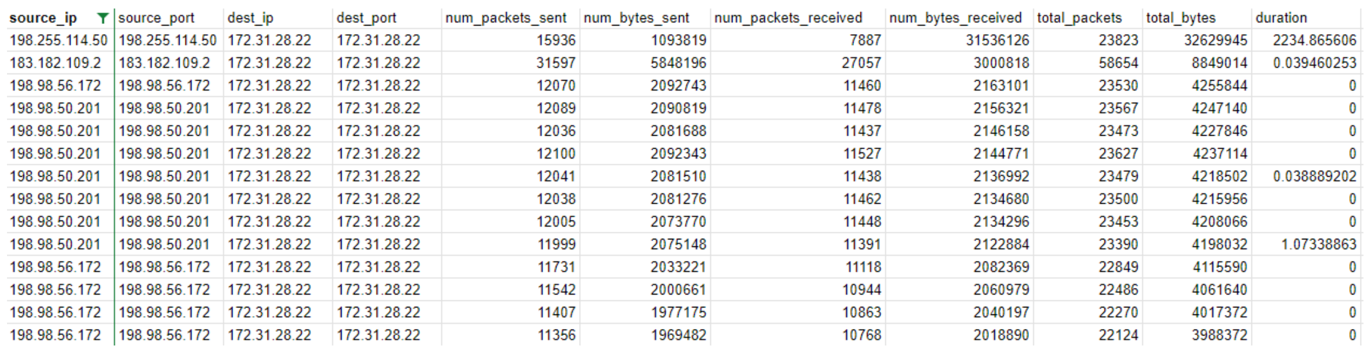
Port Knocking

Now port knocking usually is used as a security measure, as what it does is setup a sequence of closed ports that a client would try to access in sequence, allowing them to gain access to the service on it’s active port. But when used by hackers, it can become something much more malicious. To further explain how port knocking works, it works by configuring a daemon to watch the firewall log file for connection attempts to certain points, and then modifying the firewall configuration accordingly. This means that for a user to connect to a port protected by the daemon, they would need to get the correct sequence of ports. But this also means that if they were to do it by chance, they would have access to the port which the admins of a server would not want. So in terms of reconnaissance, attackers could setup multiple users to infiltrate a network and have them attempt to guess the sequence of port numbers which the server has set up. As we will see in the next section, we will talk about how this type of reconnaissance might have been what was used to crack into the server.

Malicious Traffic

	MAC Address	Hostname/Domain
183.182.109.2	02:81:c1:56:67:0b	183.182.109.2
69.30.254.140	02:81:c1:56:67:0b	69.30.254.140
103.240.182.30	02:81:c1:56:67:0b	103.240.182.30

To detect any malicious traffic, the first thing that needed to be done was to explore the areas where the total amount of data/bytes is high. To do so, I am using the excel sheet which holds all of the conversations from the .pcapng files in the capture/ folders as shown below.



source_ip	source_port	dest_ip	dest_port	num_packets_sent	num_bytes_sent	num_packets_received	num_bytes_received	total_packets	total_bytes	duration
198.255.114.50	198.255.114.50	172.31.28.22	172.31.28.22	15936	1093819	7887	31536126	23823	32629945	2234.865606
183.182.109.2	183.182.109.2	172.31.28.22	172.31.28.22	31597	5848196	27057	3000818	58654	8849014	0.039460253
198.98.56.172	198.98.56.172	172.31.28.22	172.31.28.22	12070	2092743	11460	2163101	23530	4255844	0
198.98.50.201	198.98.50.201	172.31.28.22	172.31.28.22	12089	2090819	11478	2156321	23567	4247140	0
198.98.50.201	198.98.50.201	172.31.28.22	172.31.28.22	12036	2081688	11437	2146158	23473	4227846	0
198.98.50.201	198.98.50.201	172.31.28.22	172.31.28.22	12100	2092343	11527	2144771	23627	4237114	0
198.98.50.201	198.98.50.201	172.31.28.22	172.31.28.22	12041	2081510	11438	2136992	23479	4218502	0.038889202
198.98.50.201	198.98.50.201	172.31.28.22	172.31.28.22	12038	2081276	11462	2134680	23500	4215956	0
198.98.50.201	198.98.50.201	172.31.28.22	172.31.28.22	12005	2073770	11448	2134296	23453	4208066	0
198.98.50.201	198.98.50.201	172.31.28.22	172.31.28.22	11999	2075148	11391	2122884	23390	4198032	1.07338863
198.98.56.172	198.98.56.172	172.31.28.22	172.31.28.22	11731	2033221	11118	2082369	22849	4115590	0
198.98.56.172	198.98.56.172	172.31.28.22	172.31.28.22	11542	2000661	10944	2060979	22486	4061640	0
198.98.56.172	198.98.56.172	172.31.28.22	172.31.28.22	11407	1977175	10863	2040197	22270	4017372	0
198.98.56.172	198.98.56.172	172.31.28.22	172.31.28.22	11356	1969482	10768	2018890	22124	3988372	0

Appendix[1]

A filter was placed in order to exclude any conversations where the source IP was the EC2's web server IP. This was done in order to see the conversations where a large amount of bytes were present from an outside source to the server itself. Excluding the IP addresses which have already been labelled as "benign", one address was explored was the address 183.182.109.2.

SSH

Source IP	Dst IP	Bytes Sent	Bytes Received	Total Bytes	Total Packets
183.182.109.2	172.31.28.22	5848196	3000818	8849014	58654

This conversation had a total of 8849014 bytes of data in its conversation with the webserver, with it sending 5848196 of those bytes and receiving the other 3000818. To check for any malicious intentions with this IP address, a reference of the log file created by the suricata rule for port scanning was done first. After searching for the IP address in the log file, the IP address did seem to show up quite often as a potential attacker attempting to perform a type of port scan.

```
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:50030 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:33694 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:54561 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:42494 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:38421 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:36846 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:37203 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:38160 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:53771 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:33136 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:59734 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:36722 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:58899 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:44864 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:35161 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:51573 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:57115 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:55749 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:48749 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:55987 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:51691 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:51986 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:36968 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:33441 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:42436 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:51414 -> 172.31.28.22:22
TCP port scan detected [**] [Classification: (null)] [Priority: 3] {TCP} 183.182.109.2:56987 -> 172.31.28.22:22
```

Appendix[2]

The total number of alerts that were made by this IP address was 211 entries. What can be observed by these alerts is that the IP 183.182.109.2 has multiple conversations with the web server's ssh port “22”. With the way that the rule for port scanning is set up, this amount of connections being made in sequence can be seen as suspicious, but without further analysis of the intent of these connections,

To verify the validity of the ssh connections being made by the source IP, a reference of the IP in the log files of the EC2 web server was done. Each log file in the three networks was parsed to find any traces of the Source IP using another script called “proc_logs.sh. The results of the search can be seen below.

```
type=USER_ERR msg=audit(1508573803.180:27566): pid=24628 uid=0 auid=4294967295 ses=4294967295 msg='op=PAM:bad_ident grantors=? acct="?"
exe="/usr/sbin/sshd" hostname=183.182.109.2 addr=183.182.109.2 terminal=ssh res=failed'

type=USER_LOGIN msg=audit(1508573803.180:27570): pid=24628 uid=0 auid=4294967295 ses=4294967295 msg='op=login acct="root" exe="/usr/
sbin/sshd" hostname=? addr=183.182.109.2 terminal=ssh res=failed'

type=CRYPTO_SESSION msg=audit(1508573803.892:27574): pid=24630 uid=0 auid=4294967295 ses=4294967295 msg='op=start direction=from-server
cipher=aes128-cbc ksize=128 mac=hmac-sha1 pfs=diffie-hellman-group1-sha1 spid=24631 suid=74 rport=42695 laddr=172.31.28.22 lport=22
exe="/usr/sbin/sshd" hostname=? addr=183.182.109.2 terminal=? res=success'

type=CRYPTO_SESSION msg=audit(1508573803.892:27575): pid=24630 uid=0 auid=4294967295 ses=4294967295 msg='op=start direction=from-client
cipher=aes128-cbc ksize=128 mac=hmac-sha1 pfs=diffie-hellman-group1-sha1 spid=24631 suid=74 rport=42695 laddr=172.31.28.22 lport=22
exe="/usr/sbin/sshd" hostname=? addr=183.182.109.2 terminal=? res=success'

type=CRYPTO_KEY_USER msg=audit(1508573805.144:27577): pid=24630 uid=0 auid=4294967295 ses=4294967295 msg='op=destroy kind=session fp=?
direction=both spid=24631 suid=74 rport=42695 laddr=172.31.28.22 lport=22  exe="/usr/sbin/sshd" hostname=? addr=183.182.109.2
terminal=? res=success'

type=USER_ERR msg=audit(1508573805.144:27578): pid=24630 uid=0 auid=4294967295 ses=4294967295 msg='op=PAM:bad_ident grantors=? acct="?"
exe="/usr/sbin/sshd" hostname=183.182.109.2 addr=183.182.109.2 terminal=ssh res=failed'

type=USER_LOGIN msg=audit(1508573805.144:27582): pid=24630 uid=0 auid=4294967295 ses=4294967295 msg='op=login acct="root" exe="/usr/
sbin/sshd" hostname=? addr=183.182.109.2 terminal=ssh res=failed'
```

Appendix[4]

As a reference, these logs came from the files audit.log.2 and audit.log.3 from the /ec2/log/audit directory which usually describes log files that relate to the auditing of system services. Now going back to the output of the script, in the logs tagged with the IP address 183.182.109.2, the entries with the IP address listed all are of SSH attempts made by it to the local address of the server '172.31.28.22'.

These are the parameters that are being focused on for each of the log entries for this analysis:

- type
- acct
- exe
- laddr
- addr
- res

Now if we look at the entries of the log file using these parameters as a focus, we can start to see why this traffic can be malicious. Each of these entries is attempting to connect to the server using SSH, as can be seen by the 'exe="/usr/sbin/sshd"' that is being run in each instance. To know what the server and client are in this connection, we can pay attention to the "laddr" and "addr" parameters in each of the entries which is always "laddr=172.31.28.22" and "addr=183.182.109.2", meaning the client of the connection is the outside IP address and the server being the EC2 web server. The next thing to pay attention to is the "acct" section of the entries. In here, the account of which the client is attempting to connect with is shown. Now usually, if the client were to connect to a random user it can be seen as normal traffic but something that keeps coming up in the log entries here is the attempt to ssh into the server under the account "root". The amount of entries where the "root", also known as admin, user is being attempted to be logged into is a total of 611 entries caught by the audit system. What makes this more concerning is that the "res=" parameter of each entry has shown up as "=failed", meaning that this IP is not someone who should be trying to access admin permissions of the server. This is a sign that this IP address is attempting a reconnaissance technique called "port knocking", which was discussed in the reconnaissance section of this analysis to be a type of port scanning. As we can see from the ports that the client is using to connect to the server, they are all in a random sequence of values. This could show a sign that port knocking is indeed what is occurring, but in regards to the traffic, this can also mean that someone is attempting to find out which ports on their device are able to connect to the web server, and if any of these port sequences allow them to sign in as root.

HTTP - Malware

Now going back to what the HTTP traffic in the web server usually consists of, which is GET and POST requests, to figure out if the server truly was infected with a form of malware a sucrita rule was made in order to track any HTTP GET requests in order to see if the server were to install something without knowing.

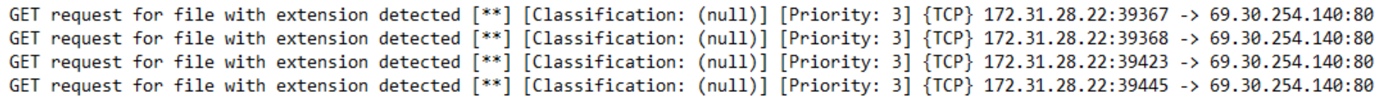
```
alert tcp 172.31.28.22 any -> any any (msg:"GET request for file with extension detected";
        content:"GET"; nocase; http_method; pcre:"/GET\s+\S+.\w{3,}\s+HTTP/\d+.\d+/i";
        sid:1000003; rev:1;)
```

- 172.31.28.22 any : Looks at any packets with the source IP 172.31.28.22 and any port
- -> : The direction of the data
- any any: Looks at any destination IP and port
- msg: The message printed when the alert is activated
- content: Searches for a specific content inside of the packets description
- nocase: Ignores uppercase lettering in the description
- http_method: Looks at packets which use the http protocol
- pcre: Used to look for a regular expression pattern in the HTTP GET request

To summarise what this rule does, it checks for any packets being sent by the server to other devices with the HTTP GET request being the type of packet to check. The perl expression checks if the get request consists of a file type by looking for the pattern of a period "." followed by 3 letters, which is the average length of most file type extensions. If a packet were to have a HTTP GET request being made by the server with a file being requested, then the rule would trigger an alert to the fast.log file of the securita IDS.

The file, http-get.log is the raw output of the alerts, which is where I started my investigation to see if any of the files alerted were to actually be "malware". In order to check the integrity of each file, a website called "www.virustotal.com" was used which checks a file using a selection of databases with known malware signatures to see if the file is similar to any known type of malware. In order to find out where these GET requests were being made, in the shell script that runs my securita rule against all of the the pcapng files in the network I made it so that it would print the current folder it was processing. Along with this, I had the contents of the alert log file "http-get.log" be catted out to the console so that when the alert of a specific conversation is made, I can cross reference the first appearance of the alert with the name of the file that was being processed.

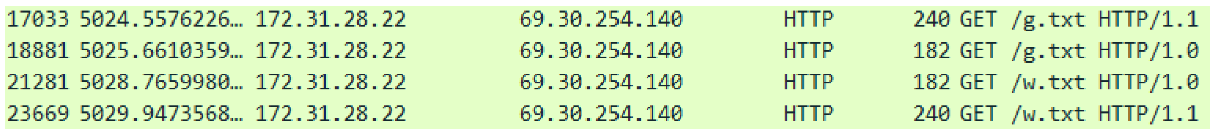
As for the findings of this analysis of GET requests, the picture below shows the alerts from which my findings will be based on.



```
GET request for file with extension detected [**] [Classification: (null)] [Priority: 3] {TCP} 172.31.28.22:39367 -> 69.30.254.140:80
GET request for file with extension detected [**] [Classification: (null)] [Priority: 3] {TCP} 172.31.28.22:39368 -> 69.30.254.140:80
GET request for file with extension detected [**] [Classification: (null)] [Priority: 3] {TCP} 172.31.28.22:39423 -> 69.30.254.140:80
GET request for file with extension detected [**] [Classification: (null)] [Priority: 3] {TCP} 172.31.28.22:39445 -> 69.30.254.140:80
```

Appendix[3]

The four alerts shown above are from a conversation that the server was having with the IP address 69.30.254.140 showed a different pattern then those of the other alerts listed. When opening the pcapng file from which the alert was made from, we can see an abnormality in the get request made. The packets from which these alerts were made are shown below.



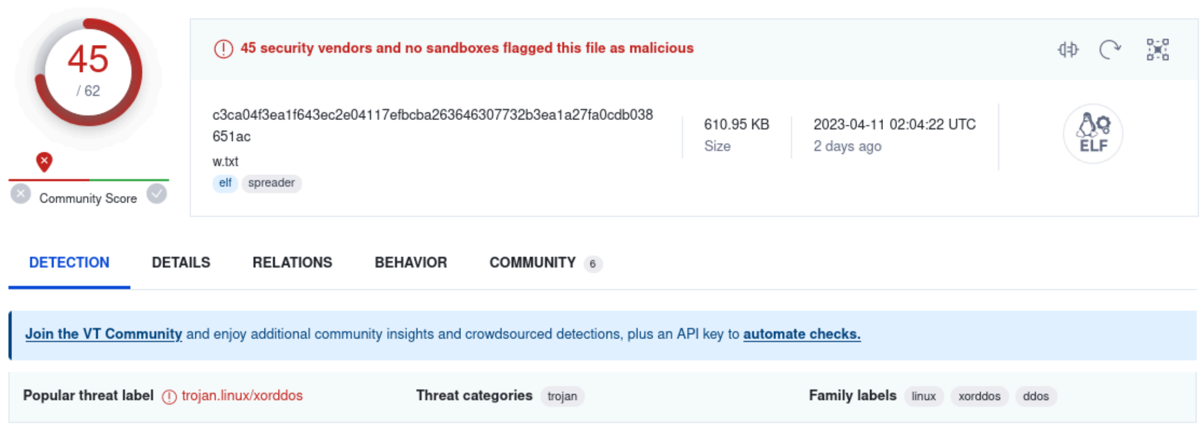
```
17033 5024.5576226… 172.31.28.22        69.30.254.140       HTTP      240 GET /g.txt HTTP/1.1
18881 5025.6610359… 172.31.28.22        69.30.254.140       HTTP      182 GET /g.txt HTTP/1.0
21281 5028.7659980… 172.31.28.22        69.30.254.140       HTTP      182 GET /w.txt HTTP/1.0
23669 5029.9473568… 172.31.28.22        69.30.254.140       HTTP      240 GET /w.txt HTTP/1.1
```

Looking at the files that are being retrieved by the server, we can see two files in total: g.txt and w.txt. Now as explained before, let's first look at the full URI from where the GET request is extracting these files from.

```
[Full request URI: http://69.30.254.140/g.txt]
```

As we can see, the server is downloading these files from the IP address 69.30.254.140, meaning that it is specifically downloading these files for no apparent reason. As these files don't follow the format of any known update or information that would usually be provided with the full URI. Now let's see what exactly these two files are by retrieving them from the capture file.

After retrieving these two files, using the testing site described before each file was run for an integrity check just like the other alerts. The pictures below show the results of the integrity test of the two files.



45 / 62

Community Score

45 security vendors and no sandboxes flagged this file as malicious

c3ca04f3ea1f643ec2e04117efbcba263646307732b3ea1a27fa0cdb038651ac

w.txt

elf spreader

610.95 KB
Size

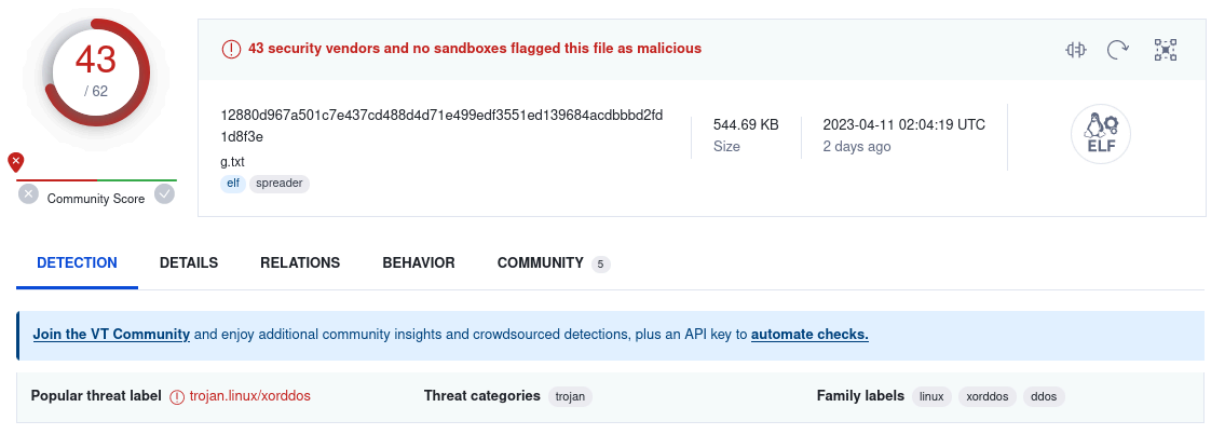
2023-04-11 02:04:22 UTC
2 days ago

ELF

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 6

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.linux/xorddos
Threat categories trojan
Family labels linux xorddos ddos



43 / 62

Community Score

43 security vendors and no sandboxes flagged this file as malicious

12880d967a501c7e437cd488d4d71e499edf3551ed139684acdbbbd2fd1d8f3e

g.txt

elf spreader

544.69 KB
Size

2023-04-11 02:04:19 UTC
2 days ago

ELF

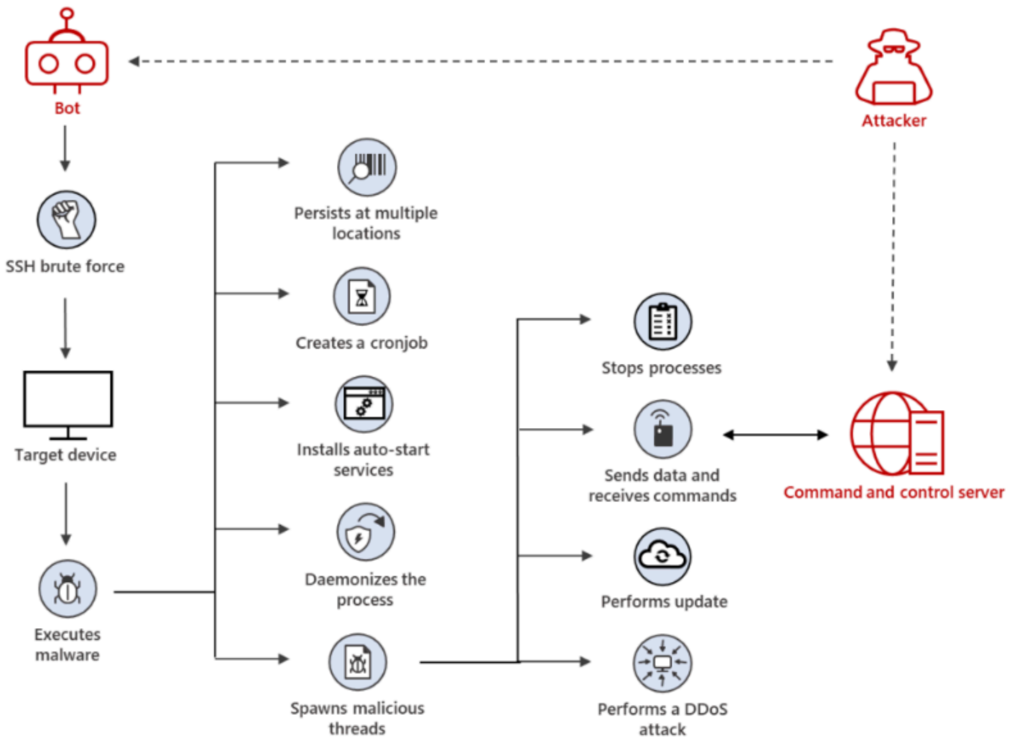
DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 5

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.linux/xorddos
Threat categories trojan
Family labels linux xorddos ddos

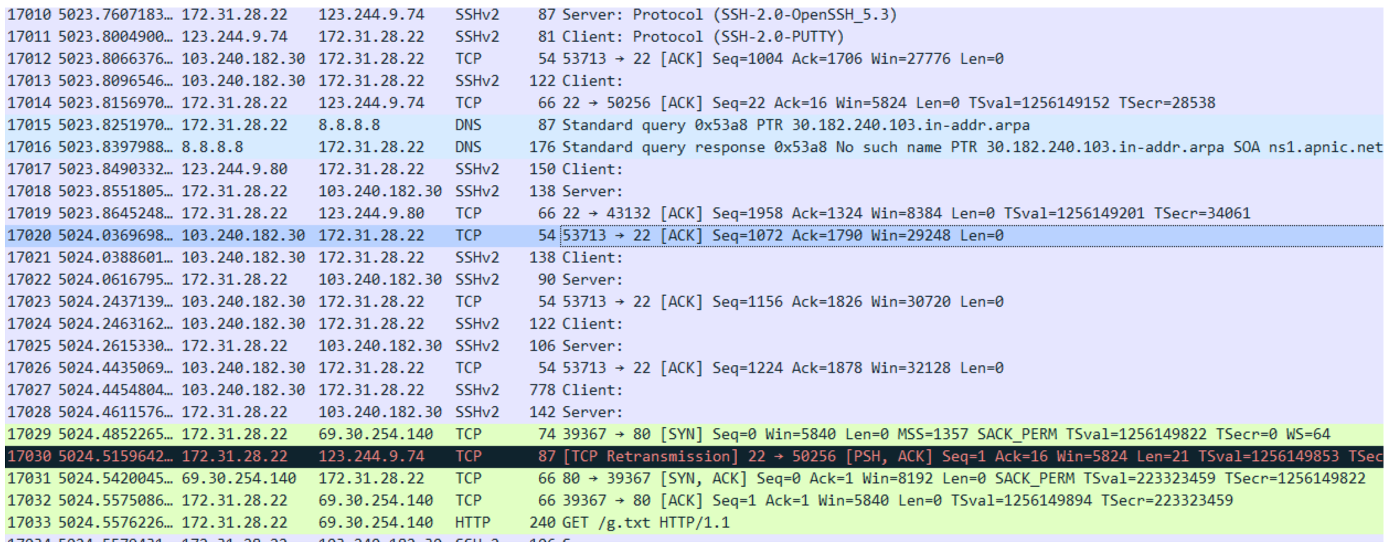
For both of the files, g.txt and w.txt, the test results stated that over 40 of the security vendors flagged the files as malicious. If we look also at the last three parameters: "Popular thread label", "Threat categories" and "Family labels" we can see what the site classifies the two files as. These files are Trojan horse viruses that affect linux systems and the exact effects of what they are capable of doing lie with the classification made by the site of "xorddos".

XorDdos is a trojan virus that targets Linux devices specifically and it works by amassing botnets that can be used to carry out distributed denial-of-service (DDos) attacks. DDos attacks are already problematic, as they are intended to bring down services of a server by overloading them but these attacks can also hide any other malicious attacks that could be happening on the server. XorDdos is also known for using SSH brute force attacks to gain remote control on target devices, which is an attractive service for attackers since the communications being done on there are all encrypted. The picture below taken from a Microsoft Security article depicts the process of how a bot is able to execute the malware on the intended device after installation.

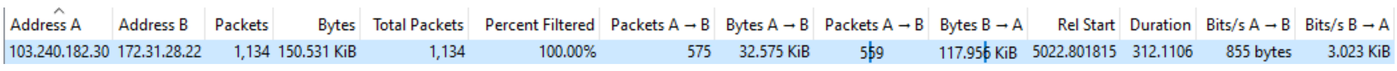


https://www.microsoft.com/en-us/security/blog/2022/05/19/rise-in-xorddos-a-deeper-look-at-the-stealthy-ddos-malware-targeting-linux-devices/

Now let's look at why the server installed this malware unintentionally by looking at the packets before the GET request took place.


```
17010 5023.7607183… 172.31.28.22   123.244.9.74   SSHv2   87 Server: Protocol (SSH-2.0-OpenSSH_5.3)
17011 5023.8004900… 123.244.9.74   172.31.28.22   SSHv2   81 Client: Protocol (SSH-2.0-PUTTY)
17012 5023.8066376… 103.240.182.30 172.31.28.22   TCP     54 53713 → 22 [ACK] Seq=1004 Ack=1706 Win=27776 Len=0
17013 5023.8096546… 103.240.182.30 172.31.28.22   SSHv2  122 Client:
17014 5023.8156970… 172.31.28.22   123.244.9.74   TCP     66 22 → 50256 [ACK] Seq=22 Ack=16 Win=5824 Len=0 TSval=1256149152 TSecr=28538
17015 5023.8251970… 172.31.28.22   8.8.8.8        DNS     87 Standard query 0x53a8 PTR 30.182.240.103.in-addr.arpa
17016 5023.8397988… 8.8.8.8        172.31.28.22   DNS    176 Standard query response 0x53a8 No such name PTR 30.182.240.103.in-addr.arpa SOA ns1.apnic.net
17017 5023.8490332… 123.244.9.80   172.31.28.22   SSHv2  150 Client:
17018 5023.8551805… 172.31.28.22   103.240.182.30 SSHv2  138 Server:
17019 5023.8645248… 172.31.28.22   123.244.9.80   TCP     66 22 → 43132 [ACK] Seq=1958 Ack=1324 Win=8384 Len=0 TSval=1256149201 TSecr=34061
17020 5024.0369698… 103.240.182.30 172.31.28.22   TCP     54 53713 → 22 [ACK] Seq=1072 Ack=1790 Win=29248 Len=0
17021 5024.0388601… 103.240.182.30 172.31.28.22   SSHv2  138 Client:
17022 5024.0616795… 172.31.28.22   103.240.182.30 SSHv2   90 Server:
17023 5024.2437139… 103.240.182.30 172.31.28.22   TCP     54 53713 → 22 [ACK] Seq=1156 Ack=1826 Win=30720 Len=0
17024 5024.2463162… 103.240.182.30 172.31.28.22   SSHv2  122 Client:
17025 5024.2615330… 172.31.28.22   103.240.182.30 SSHv2  106 Server:
17026 5024.4435069… 103.240.182.30 172.31.28.22   TCP     54 53713 → 22 [ACK] Seq=1224 Ack=1878 Win=32128 Len=0
17027 5024.4454804… 103.240.182.30 172.31.28.22   SSHv2  778 Client:
17028 5024.4611576… 172.31.28.22   103.240.182.30 SSHv2  142 Server:
17029 5024.4852265… 172.31.28.22   69.30.254.140  TCP     74 39367 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1357 SACK_PERM TSval=1256149822 TSecr=0 WS=64
17030 5024.5159642… 172.31.28.22   123.244.9.74   TCP     87 [TCP Retransmission] 22 → 50256 [PSH, ACK] Seq=1 Ack=16 Win=5824 Len=21 TSval=1256149853 TSec
17031 5024.5420045… 69.30.254.140  172.31.28.22   TCP     66 80 → 39367 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 SACK_PERM TSval=223323459 TSecr=1256149822
17032 5024.5575086… 172.31.28.22   69.30.254.140  TCP     66 39367 → 80 [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSval=1256149894 TSecr=223323459
17033 5024.5576226… 172.31.28.22   69.30.254.140  HTTP   240 GET /g.txt HTTP/1.1
```

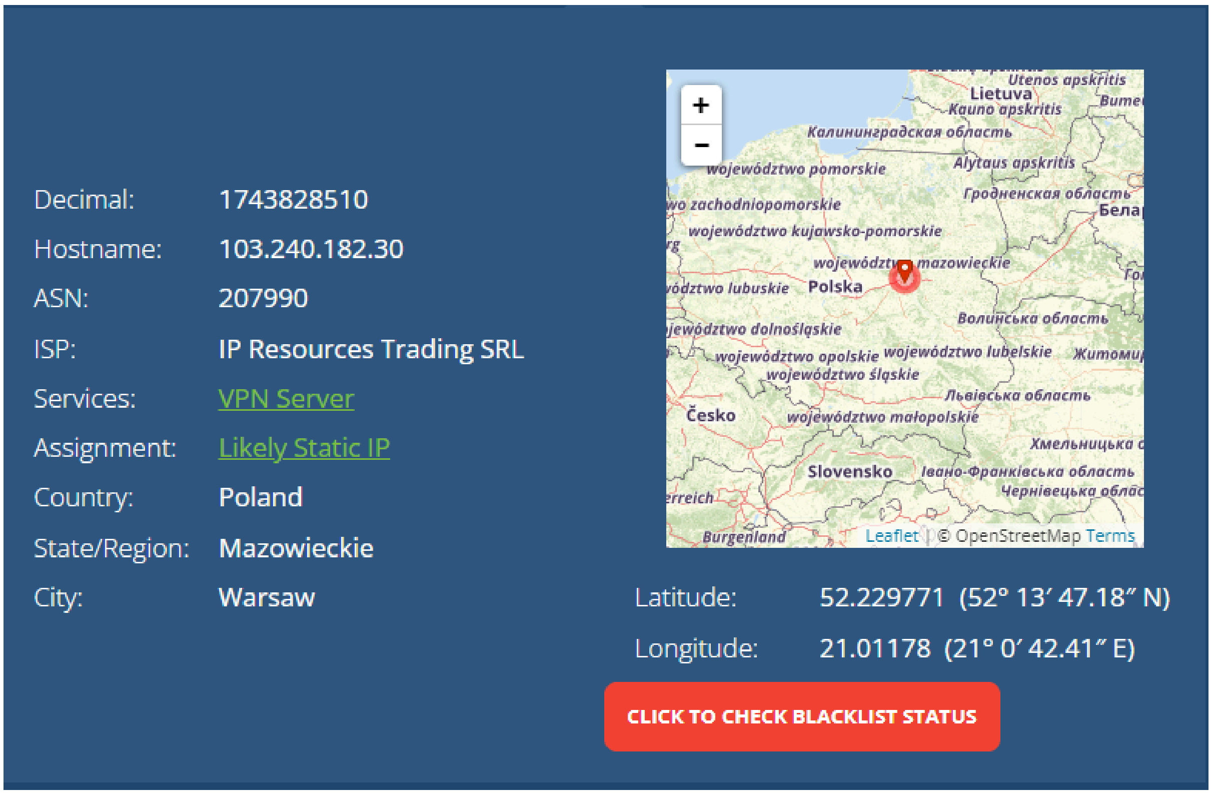
From the screenshot above, we can see the traffic before the GET request occurred, which is the last packet in the screenshot. The traffic that comes before the HTTP connection was made is comprised of a SSH conversation happening between the server ‘172.31.28.22’ and the client ‘103.240.182.30’



Address A	Address B	Packets	Bytes	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets A → B	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
103.240.182.30	172.31.28.22	1,134	150.531 KiB	1,134	100.00%	575	32.575 KiB	559	117.956 KiB	5022.801815	312.1106	855 bytes	3.023 KiB

After getting the information of the conversation between the two devices, we can see that the server was sending much more data then the client was, meaning that the server was most likely being used by the client in order to enact something. As we can see from the GET request, an assumption that can be made from this is that the client was able to gain root access to the server through some means unknown since all we have is the pcapng files and logs. After gaining the root permissions on the server, the client then had the server create a GET request to download the malware onto itself. Afterwards, if the malware were to work as we described above, the attacker then ran the executable in order to perform a Ddos attack on the server. Relating this back to the technique of port knocking, what could have been a way that this IP was able to gain access to the server was by first having other instances, such as the IP 183.182.109.2, try to first find out the correct sequence of ports to connect to. Then once the attacker was able to get root permission by some means, they decided to enact their attack by SSHing into the server as root, and perform the GET request.

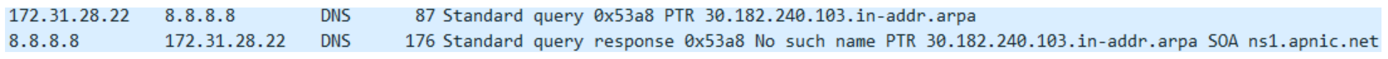
Another thing to look at is the origin of the IP address of the attacker, ‘103.240.182.30’. To do so, an IP address lookup can be done with the website www.whatismyipaddress.com and the results of searching up the attackers IP are shown below.



Decimal:	1743828510
Hostname:	103.240.182.30
ASN:	207990
ISP:	IP Resources Trading SRL
Services:	VPN Server
Assignment:	Likely Static IP
Country:	Poland
State/Region:	Mazowieckie
City:	Warsaw
Latitude:	52.229771 (52° 13′ 47.18″ N)
Longitude:	21.01178 (21° 0′ 42.41″ E)

Without knowing the actual location of the EC2 web server, it would be hard to know if the IP address of the attacker was someone who gained access to another device on the network, but from the ‘services’ section of the report made by the website on the IP we can see that the IP is associated with a ‘VPN Server’. This means that the IP address that connected to the server before the malware was downloaded is most likely not the original address of the attacker who connected to the server, meaning that this IP address was most likely spoofed.

One final thing to add to the idea of the IP address being spoofed is the DNS query that appears before the GET request.



```
172.31.28.22    8.8.8.8         DNS       87 Standard query 0x53a8 PTR 30.182.240.103.in-addr.arpa
8.8.8.8         172.31.28.22    DNS      176 Standard query response 0x53a8 No such name PTR 30.182.240.103.in-addr.arpa SOA ns1.apnic.net
```

As we can see from the screenshot, these two packets are a DNS query and response for a reverse DNS lookup of the IP 103.240.182.30 using the domain name 30.82.240.103.in-addr.apa. The server first sends a query to the DNS server 8.8.8.8 requesting the PTR(Pointer) record for the IP address. In the response, the DNS server states that there is no such name for the requested PTR record. The lack of a PTR record may indicate that the device is not configured properly, but from the SSH connection that we saw occurring during the download of the data, we can tell that IP spoofing is more likely the case.

Conclusion

Based on the analysis of the raw data from the ec2, log-server, and web networks, it is clear that the servers were targeted by an attacker who used a botnet to launch a DDoS attack. The attacker made numerous attempts to connect to the SSH server of the EC2 web server, with some attempts being successful. In addition, the audit service of the server flagged an IP address for numerous attempts to log in as the root user. These signs suggest that the attacker was attempting to gain control of the server.

One of the key findings of the analysis was the pattern of unknown IP addresses attempting to connect to the SSH server through a random sequence of ports on the client’s side. This is a common tactic used by attackers to evade detection by security systems, as it makes it difficult to identify the source of the attack. However, the attacker only attempted to connect to the server using its default SSH port 22, which suggests that the server's configuration was not properly secured.

The fact that the attacker attempted to log in as the root user is also concerning. The root user has the highest level of privileges on a system, and if an attacker gains access to this account, they could potentially gain full control of the server. While the attempts were logged as “failed”, it is clear that the attacker was persistent in their efforts to gain access to the system.

Another significant finding was the server's abnormal behaviour towards the end of the sniffer's capture files. The server made a total of 4 GET requests, which downloaded files that were classified as "malware". The malware that was downloaded was a type called “XordDos”, which when executed, performs a DDoS attack on a server, thus impacting its services and limiting its functionality. This indicates that the attacker had successfully compromised the server and was able to execute malicious code on it.

To prevent such attacks in the future, it is recommended that the SSH server be configured to use non-default ports and that strong passwords be used for all user accounts. Additionally, a firewall could be set up to block all traffic from unknown IP addresses. Regular monitoring of the server's logs and audit services could also help detect and prevent any potential attacks. It is crucial to ensure that all software and system updates are kept up to date to prevent any vulnerabilities that can be exploited by attackers. Finally, having a disaster recovery plan in place can help minimise the impact of an attack if it does occur.

In conclusion, the analysis of the raw data from the ec2, log-server, and web networks revealed a targeted attack on the servers using a botnet to launch a DDoS attack. The attacker used various tactics, such as attempting to connect to the server through random ports, brute-forcing login attempts, and executing malware on the server. It is essential that proper security measures are put in place to prevent such attacks from happening in the future. These measures include configuring the SSH server to use non-default ports, using strong passwords for all user accounts, setting up a firewall to block traffic from unknown IP addresses, and regular monitoring of the server's logs and audit services. By implementing these security measures, organisations can help prevent such attacks and minimise their impact if they do occur.

Appendix

1. Conversations Excel Sheet - https://docs.google.com/spreadsheets/d/1vHNQJTV19f9dHNlgSCLZnXNQyrv8i-hHHRldIeEbXME/edit?usp=sharing
2. Port scan alert log - https://drive.google.com/file/d/12NQAwc2H3zzooOVDXAI2ggQEx4Ky4w43/view?usp=share_link
3. 183.182.109 Log Entries - https://drive.google.com/file/d/1hQsYsxF1S1Il3pssv7S9M6-Rlywn4jGz/view?usp=share_link
4. HTTP GET Request Alert Log - https://drive.google.com/file/d/1pHBd01I5607JJndGSdymXtO7WPfDOYbN/view?usp=share_link