Static analysis and recovery example – 13-Dec-2021

Introduction

Client: A 30 employees law firm based in the capital of the city, that provided consulting and outsourced legal analysis for different companies.

Once identified the malware as the “Japanese SEO virus” or “Japanese keyword virus”, through open sources, we have to check for documentation regarding this virus on well known databases that can provide insights into what functionality does this virus has. Later on, we will make our own assessment, studying the source code and possible avenues of approach that might have been used to actually exploit the client.

Shortly after the incident (1100 of the 29-Oct-2020) and after receiving the Incident Response, the client tried to do a recovery of the server by himself, however they had noted the following issues while trying to do so:

1. The usual login for Wordpress was not responsive (404 error)
2. They were unaware of the E-Mail associated with the recovery of the Wordpress password, and also the one associated to the billing of the server itself
3. The had no line of communication with the IT team responsible for the server backend (related to point 2)
4. They had no backups of any audio-visual material that was uploaded to the Wordpress page

Due to this issues, the recovery was impossible from their end, and they were alarmed as they were effectively locked out of their webpage.

As such, the client stated the following **objectives** to be met:

1. Find the billing and Wordpress recovery E-Mail
2. Recover all the audio-visual material
3. Delete the virus and restore functionality as soon as possible

This proved to be a challenge, a mix of OSINT and social engineering as follows.

> Using a receipt that the client had and a Whois query to confirm the information, I was able to pinpoint the name servers associated to the domain (which provided a contact E-Mail for IT issues of the server that was contracted to support the Wordpress www.example.com domain) and also the name of the individual who had bought the domain in the first place. After contacting him, he provided a series of E-Mails and password, the most important being an FTP account. [1]

Figure 1 Whois Query

With a functioning FTP account and the IT E-Mail for the server, I was able to request administrative, billing and recovery E-Mails. Also, with an FTP account to take samples, we could start analyzing what was actually happening.

Here, **objective 1** *"Find the billing and Wordpress recovery E-Mail"*, was achieved.

1 The client had outsourced the process of buying the domain to another individual, whom was not aware of the actual cyberattack and thus had not come forward with this critical information that was required for recovery.

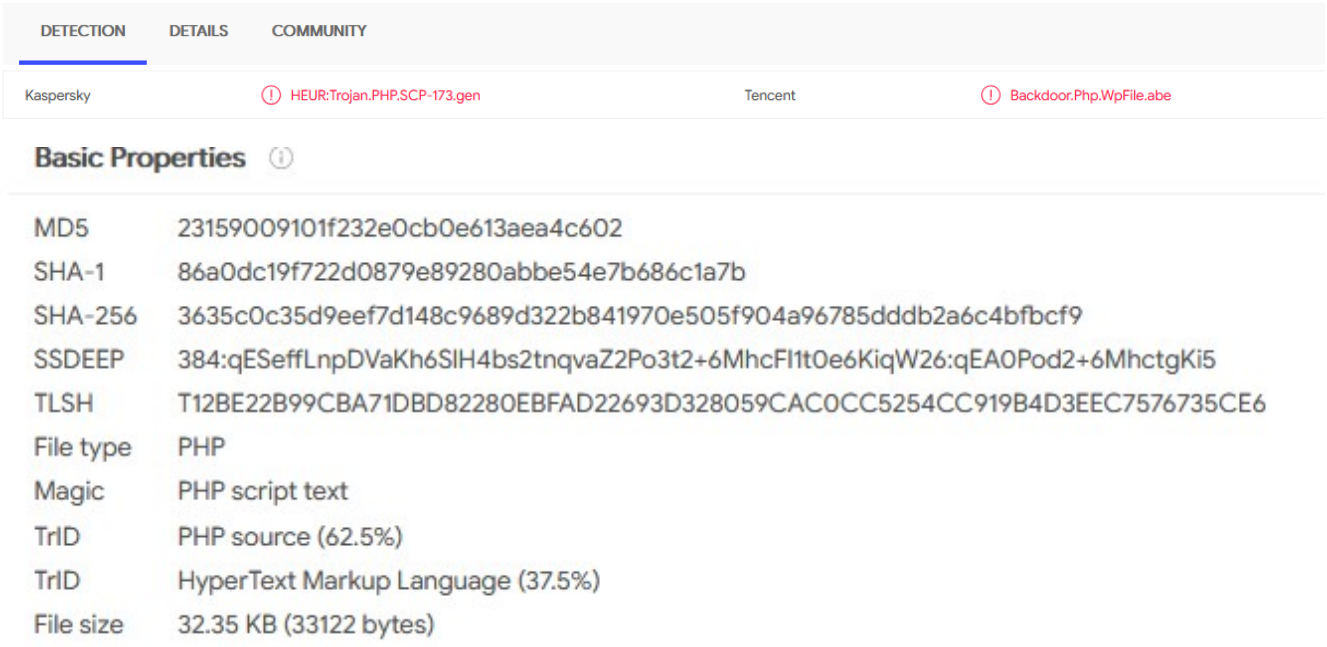
Static analysis

Object of analysis: Files inside the Apache Server that hosted the domain www.example.com

Acquiring the FTP credentials was critical to download the entire contents of the server. This was done in a secure environment inside Kali Linux, to analyze the contents further and determine the source of the virus.

The first item to be studied was index.php.[2]

Figure 2 VirusTotal analysis



The results were very interesting, as these kind of virus were already well documented, as following:

> *The malware itself is a PHP backdoor dropper that creates a custom function that uses curl and file_put_contents to download malware from a third party URL which is provided by the attacker in a HTTP request (GET or POST) to the infected website.*
>
> *As it turns out, SCP-173 is a story/meme entry from the SCP Foundation, which is a collection of creepypasta style fictional stories, and dates back to 2007. It's unknown why a hacker would choose to use it as a code comment at the end of their injection - they may just be a fan and wanted to leave an easter egg.*[3]

Also, the presence of a backdoor would be expected as to remain in control of the server, even after a clean up was performed.

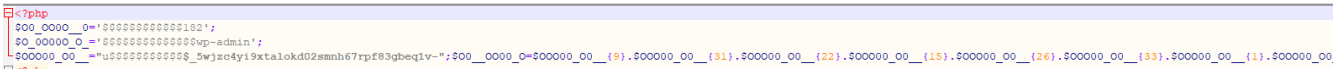
---

2 SHA-256 hash: 3635C0C35D9EEF7D148C9689D322B841970E505F904A96785DDDB2A6C4BFBCF9

3 See: https://lukeleal.com/research/posts/scp-173-malware/

When we check the source code, we can see it obfuscated:

Figure 3 Source code A



The one that we are seeing here appears to be a variant of the SCP-173, but with an apparently different obfuscation method. Here, concatenated array values defined as as “$O0_OO0O__0”, whom appear to be ruling what this script does. There is also obfuscated lines of text, using only escape characters, which make the code really difficult to read.

Figure 4 Source code B

Also, being only one long line of text doesn’t help its readability at all. After parsing it and “beautifying it”,[4] the results are much better.

Figure 5 Source code C

Results:

```php
<?php
$00_0000__0 = '$$$$$$$$$$$$182';
$O_00000_O_ = '$$$$$$$$$$$$$wp-admin';
$OOOOO_OO__ = "u$$$$$$$$$$$$_5wjzc4yi9xtalokd02smnh67rpf83gbeq1v-";
$O0__OOOO_O = $OOOOO_OO__{9} . $OOOOO_OO__{31} . $OOOOO_OO__{22} . $OOOOO_OO__{15} . $OOOOO_OO__{26} . $OOOOO
$O0O_O_0OO_ = $OOOOO_OO__{28} . $OOOOO_OO__{9} . $OOOOO_OO__{14} . $OOOOO_OO__{33} . $OOOOO_OO__{1} . $OOOOO
$O00_O_OOO_ = $OOOOO_OO__{28} . $OOOOO_OO__{9} . $OOOOO_OO__{14} . $OOOOO_OO__{33} . $OOOOO_OO__{1} . $OOOOO
$OO__0OOO0_ = $OOOOO_OO__{28} . $OOOOO_OO__{0} . $OOOOO_OO__{22} . $OOOOO_OO__{6} . $OOOOO_OO__{12} . $OOOOO
$O__0OOO_OO = $OOOOO_OO__{33} . $OOOOO_OO__{26} . $OOOOO_OO__{26} . $OOOOO_OO__{15} . $OOOOO_OO__{26} . $OOOOO
$O0_O_0O_0O = $OOOOO_OO__{6} . $OOOOO_OO__{26} . $OOOOO_OO__{33} . $OOOOO_OO__{13} . $OOOOO_OO__{12} . $OOOOO
$O00_0_OOO_ = $OOOOO_OO__{20} . $OOOOO_OO__{33} . $OOOOO_OO__{12} . $OOOOO_OO__{1} . $OOOOO_OO__{12} . $OOOOO
$OO0_00O__O = $OOOOO_OO__{32} . $OOOOO_OO__{13} . $OOOOO_OO__{20} . $OOOOO_OO__{33} . $OOOOO_OO__{24} . $OOOOO
$O_OO__0O00 = $OOOOO_OO__{32} . $OOOOO_OO__{13} . $OOOOO_OO__{20} . $OOOOO_OO__{33} . $OOOOO_OO__{24} . $OOOOO
$O_0O_0_OOO = $OOOOO_OO__{27} . $OOOOO_OO__{26} . $OOOOO_OO__{33} . $OOOOO_OO__{31} . $OOOOO_OO__{1} . $OOOOO
$O_0_0OO_0O = $OOOOO_OO__{20} . $OOOOO_OO__{12} . $OOOOO_OO__{26} . $OOOOO_OO__{1} . $OOOOO_OO__{26} . $OOOOO
$OOO0_OO_0_ = $OOOOO_OO__{28} . $OOOOO_OO__{9} . $OOOOO_OO__{14} . $OOOOO_OO__{33} . $OOOOO_OO__{1} . $OOOOO
$OO_O00_0O_ = $OOOOO_OO__{6} . $OOOOO_OO__{0} . $OOOOO_OO__{26} . $OOOOO_OO__{14} . $OOOOO_OO__{1} . $OOOOO_O
$OOO0__0O_0 = $OOOOO_OO__{27} . $OOOOO_OO__{26} . $OOOOO_OO__{33} . $OOOOO_OO__{31} . $OOOOO_OO__{1} . $OOOOO
$OOO__OO_00 = $OOOOO_OO__{6} . $OOOOO_OO__{0} . $OOOOO_OO__{26} . $OOOOO_OO__{14} . $OOOOO_OO__{1} . $OOOOO_O
$OO0O000___ = $OOOOO_OO__{0} . $OOOOO_OO__{26} . $OOOOO_OO__{14} . $OOOOO_OO__{33} . $OOOOO_OO__{22} . $OOOOO
$O0000_OO__ = $OOOOO_OO__{31} . $OOOOO_OO__{5} . $OOOOO_OO__{9} . $OOOOO_OO__{22} . $OOOOO_OO__{28} . $OOOOO_O
$OO_0_0000_ = $OOOOO_OO__{6} . $OOOOO_OO__{0} . $OOOOO_OO__{26} . $OOOOO_OO__{14} . $OOOOO_OO__{1} . $OOOOO_O
$O0__OOO0_0 = $OOOOO_OO__{6} . $OOOOO_OO__{0} . $OOOOO_OO__{26} . $OOOOO_OO__{14} . $OOOOO_OO__{1} . $OOOOO_O
$O00_O_OOO_ = $OOOOO_OO__{20} . $OOOOO_OO__{12} . $OOOOO_OO__{26} . $OOOOO_OO__{26} . $OOOOO_OO__{27} . $OOOOO
$O00_OO_O_0 = $OOOOO_OO__{20} . $OOOOO_OO__{6} . $OOOOO_OO__{13} . $OOOOO_OO__{22} . $OOOOO_OO__{17} . $OOOOO
$O_0O_O0_O0 = $OOOOO_OO__{33} . $OOOOO_OO__{11} . $OOOOO_OO__{27} . $OOOOO_OO__{14} . $OOOOO_OO__{15} . $OOOOO
$OO___00OO0 = $OOOOO_OO__{17} . $OOOOO_OO__{9} . $OOOOO_OO__{26} . $OOOOO_OO__{22} . $OOOOO_OO__{13} . $OOOOO
$O_0_0OOO0_ = $OOOOO_OO__{0} . $OOOOO_OO__{22} . $OOOOO_OO__{14} . $OOOOO_OO__{9} . $OOOOO_OO__{22} . $OOOOO_O
$O0_OOO0__0 = $OOOOO_OO__{20} . $OOOOO_OO__{12} . $OOOOO_OO__{26} . $OOOOO_OO__{20} . $OOOOO_OO__{12} . $OOOOO
$OOOO_OOO__ = $OOOOO_OO__{20} . $OOOOO_OO__{12} . $OOOOO_OO__{26} . $OOOOO_OO__{14} . $OOOOO_OO__{33} . $OOOOO
$OO_O_00_0O = $OOOOO_OO__{9} . $OOOOO_OO__{20} . $OOOOO_OO__{1} . $OOOOO_OO__{17} . $OOOOO_OO__{9} . $OOOOO_O
$O0_O_OO0_0 = $OOOOO_OO__{21} . $OOOOO_OO__{16} . $OOOOO_OO__{17} . $OOOOO_OO__{9} . $OOOOO_OO__{26};
$O00_OO_O0_ = $OOOOO_OO__{6} . $OOOOO_OO__{15} . $OOOOO_OO__{0} . $OOOOO_OO__{22} . $OOOOO_OO__{12};
$OO__O00_O0 = $OOOOO_OO__{6} . $OOOOO_OO__{23} . $OOOOO_OO__{21} . $OOOOO_OO__{15} . $OOOOO_OO__{17};
$O_00_O_O0O = $OOOOO_OO__{12} . $OOOOO_OO__{26} . $OOOOO_OO__{9} . $OOOOO_OO__{21};
$O__0O00_O0 = $OOOOO_OO__{17} . $OOOOO_OO__{13} . $OOOOO_OO__{12} . $OOOOO_OO__{33};
header('Content-Type:text/html;charset=utf-8');
```

4 Using: https://beautifytools.com/php-beautifier.php

In the line #4, we can actually see the string where every variable takes its value from. Afterwards, every single character is concatenated to provide the final result.

Figure 6 Source code D

As follows, we can see the results of each variable as we build our “strings” from the original.

```
'v', '-']
['$29wlm$uyml$_swl$', 'k$5m$ou$$$w9$m9$y', 'k$5m$2m$$$w9$m9$y', 'ku9$$$w9$m$$y$y', 'mllwl$lmowl$$92', '$lm_$m$ku9$$$w9',
 'ym$$$$im$5$i$$', 's_ymt$$m9$wzm', 's_ymt$$zm$wzm', 'olm2$lmo5_$m', 'y$l$lmo5_$m', 'k$5m$m$$y$y', '$ul5$ym$wo$', 'olm2$
i_$$x', '$ul5$$5wym', 'ul5m9$wzm', '2$$9k5_$m', '$ul5$$9$$', '$ul5$m$m$', 'y$llowy', 'y$_9z$l', 'm$o5wzm', 'z$l9_im', 'u
95$9j', 'y$ly$l', 'y$l5m9', '$y$z$l', 'ijz$l', '$wu9$', '$xiwz', '$l$i', 'z_$m']
```

This is also combined with “gibberish”, in the form of declaring these values as “NULL”, to further confuse the user.

```
('$O_OO_OOOO_=\'\',$OOOOOO_O__=NULL,
```

Another element to take into account is the following:

```
["\x4f\x4f\x5f\x4f\x5f\x4f\x5f\x30\x30\x30"]
```

These actually represent hexadecimal escape sequences[5], representing the characters of a string.

```
GLOBALS
OO_O_O_000
GLOBALS
O_0O_O0_O0
```

In this manner, the obfuscation is really complete, as the variables previously mentioned (written with really hard to discern names, that are taking its value from specific characters in a string) are now in another form of “encoding”, that is not actually encryption per se, but is formatted in a way that eludes anti-virus.

This is truly a very interesting piece of software, as it has a really thorough defense in depth to avoid being detected by automated software and human users.

Another example of this type of obfuscation can be seen here.[6][7]

5 See: https://php.watch/articles/php-character-escape-sequences-numeric-notations
6 See: https://blog.sucuri.net/2020/04/obfuscated-wordpress-malware-dropper.html
7 See: https://blog.sucuri.net/2020/01/backdoor-found-in-compromised-wordpress-environment.html

The client stated that the virus re-appeared after every attempt to delete or update Wordpress core files, and that their scans were not successful in finding any threat.

Figure 7 Client scan

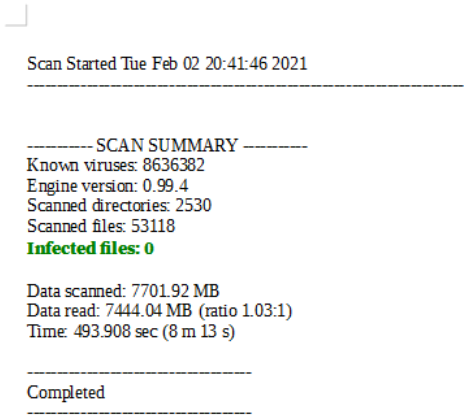
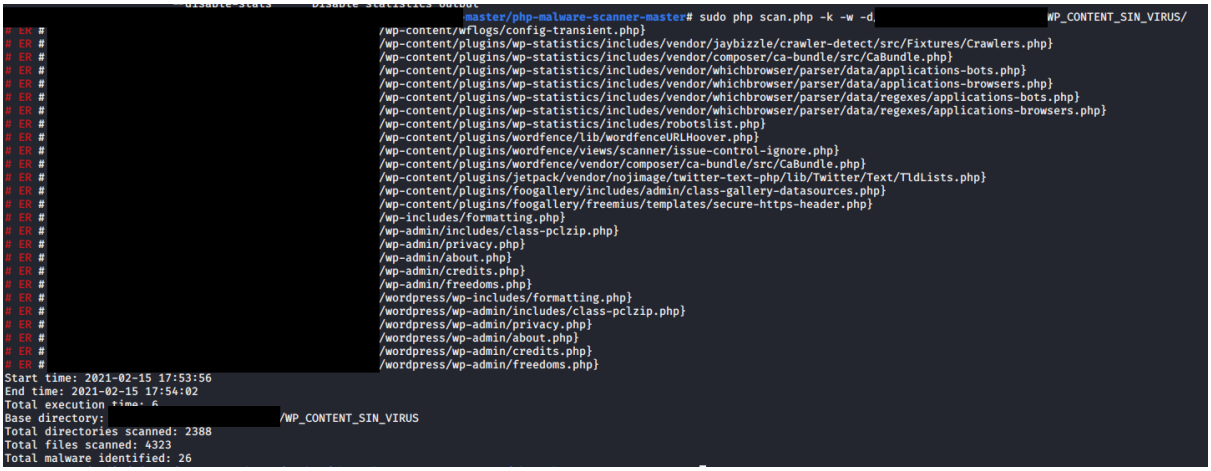


Figure 8 Analyst scan

After taking note of the malware obfuscation, and capability to evade “standard” based signature anti-virus, a specific tool had to be used. Such tool was php-malware-scanner, listed on GitHub[8] which yielded excellent results.



It can be seen that almost every .php file in the server is compromised, having obfuscated lines of code inside. It is of note, that actually takes control in wp-admin, forfeiting outside access being the reason why the client couldn’t log in.

It is advised to do a complete erasing of all the php files, to re-install Wordpress and only the necessary plug-ins for the webpage to work. The entire contents of the server, including the audio-visual information were able to be recovered through FTP and into the cyber-analyst secure HDD, just before deleting everything.

8 See: https://github.com/scr34m/php-malware-scanner

Here, **objective 2** "*Recover all the audio-visual material"*, was achieved. Having all the necessary information regarding the virus, the contents of the server (lately sanitized and only remaining the audio-visual information that was requested for safe-keeping) the reinstall of Wordpress and all packages was smooth, with the webserver up and running in less than 48 hours.

No traces of the virus remained after the complete re-install.

Here, **objective 3** *"Delete the virus and restore functionality as soon as possible"* was achieved.

Conclusion: It wasn't possible to to recover the webpage because Wordpress was compromised with a virus that had really potent obfuscation capabilities and had taken control of the admin section of Wordpress Core, and also boasted a backdoor function.