EFFUSION – A NEW SOPHISTICATED INJECTOR FOR NGINX WEB SERVERS

Andrew Kovalev, Konstantin Otrashkevich, Evgeny Sidorov & Andrew Rassokhin
Yandex, Russia

This article is a continuation of our research into modern methods of web malware distribution, the initial results of which were presented at VB2013. In our presentation, we spoke about three modern approaches used by attackers to embed malicious code into HTTP responses [1]. One of those approaches was the use of web-server modules for malware distribution, and as an example of this we described malicious modules for an Apache web server.

In this article we will describe ‘Effusion’ – a new piece of malware that uses a similar approach, but for an Nginx web server, and which was used in a massive infection campaign in the third quarter of 2013.

All the data for this article has been obtained with the use of Yandex’s anti-virus system [2].

INTRODUCTION

The methods by which malicious code is distributed in drive-by download attacks are constantly evolving. One of the first methods to be used involved adding malicious code to static content (HTML templates, JavaScript files, etc.) – however, after some time such modifications became easy for anti-virus products to detect using signatures. To complicate signature analysis, attackers began to use obfuscation and encryption techniques. In response to this, anti-virus products started to employ JavaScript emulators (sandboxes), which did a better job of detecting malicious code in web pages. The next stage in the evolution of drive-by downloads involved modifying the source code of content management systems (CMS) such as Joomla, WordPress, DLE, etc. The malicious code began checking the referer (e.g. for referral from SERPs) and the user-agent (the code is not displayed to search bots, mobile redirects, etc.), as well as the user session (to determine whether the user is an administrator) and, depending on the results, deciding whether or not to insert malicious code into the web page. However, it has become a straightforward task for the majority of webmasters to remove such an infection from their web servers – in fact, there are even special scripts that help with this task [3].

The next step of the evolution involved embedding a piece of malicious code into the body of an HTTP response. This approach is heavily employed today, and is the method used by Effusion – which injects malicious code into the HTTP responses of Nginx web servers.

Malware representation

At the end of November 2013, we received several calls for help from webmasters who were having difficulty removing infections from their websites. During the investigation of these incidents we found (and analysed) two malicious samples.

We also discovered that the attackers had modified the /etc/init.d/nginx script in order to load a malicious shared object with the name ‘/usr/lib/libnginx.so’ into the Nginx address space. The shared object was loaded using the LD_PRELOAD technique. Part of the modified script is shown in Figure 1.

![Figure 1: Part of the modified /etc/init.d/nginx script.](image)

We found that the malware is represented by only a single shared object. We analysed two such shared objects with the following MD5 hashes:

- 9f1796452a20fca0093d7a4954efad2d
- f26ac64f927b0f445cd3f19d91294624

We checked the samples using the VirusTotal service – the results are shown in Table 1.

<table>
<thead>
<tr>
<th>Hash</th>
<th>Date</th>
<th>VirusTotal results</th>
</tr>
</thead>
<tbody>
<tr>
<td>9f1796452a20fca0093d7a4954efad2d</td>
<td>2013-12-05</td>
<td>1/48</td>
</tr>
<tr>
<td>9f1796452a20fca0093d7a4954efad2d</td>
<td>2013-11-25</td>
<td>1/48</td>
</tr>
<tr>
<td>f26ac64f927b0f445cd3f19d91294624</td>
<td>2013-12-05</td>
<td>0/48</td>
</tr>
<tr>
<td>f26ac64f927b0f445cd3f19d91294624</td>
<td>2013-11-27</td>
<td>0/48</td>
</tr>
</tbody>
</table>

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

We found that the ELF headers in the samples were corrupted in order to complicate their analysis. The first sample was detected only by Avira’s AntiVir product, which detected it as HEUR/ELF.Malformed. The second sample was not detected by any product. The samples were compiled for the x64 platform with the ‘-fPIC’ key.
Analysis of the initialization process

The LD_PRELOAD technique allows the shared object to be the first to be loaded and allows it to hook different functions easily. If a standard library function is reimplemented in such an object, it will be replaced by that of the shared object. The malicious sample contains its own implementation of the setsid function, so this function is invoked by Nginx instead of the original one.

The reimplementation of the setsid function is used for the initialization of the malicious sample. First, the dlsym function is executed in order to obtain the address of the original setsid function, then the original setsid is executed. Next, the sample checks whether initialization has already been performed. If initialization is required, it will continue with the execution. The initialization process involves the following steps:

1. The base address is obtained by searching for the ELF signature in memory. A type of ABI (application binary interface) and a file class of the shared object are obtained from the ELF header and stored in memory for future use.
2. The malicious configuration, stored in the data segment, is decrypted and parsed. If the configuration contains a particular filename, then this file will be opened and mapped into the process memory, and additional configuration information (an array with blacklisted IP addresses) will be loaded. If there is no such a filename, a part of memory will be allocated via the mmap function call. This memory will be used for inter-process communications.
3. The process is cloned via a call to the fork function and the child process will be used for remote control, system process monitoring and root activity detection functions.
4. The addresses of the zlibVersion and inflateInit2 functions are obtained via the corresponding dlsym function calls and stored in memory. They will be used for the processing of compressed HTTP responses.

An overview of the hooking of the setsid function is shown in Figure 2.

During the loading of the shared object, an initialization function, _init_proc, is executed. In this function, the ngx_http_copy_filter_init function is hooked by replacing its address in the ngx_http_copy_filter_module_ctx structure; the address of the reference to this function in the structure is hard-coded in the shared object and differs from sample to sample.

The hooking of ngx_http_copy_filter_init in turn embeds pointers to the custom HTTP header and HTTP body filters (defined in the shared object) into Nginx’s filter chain. These functions will be executed during the processing of the HTTP response header and HTTP response body, respectively. The embedding is performed by replacing the values of the global variables ngx_http_top_header_filter and ngx_http_top_body_filter in Nginx’s memory using addresses of special functions in the shared object. The original values of these variables are stored in memory and will be used in the embedded filters. Additional information about the handlers and filters in the Nginx web server can be found in [4]. Figure 4 shows a typical HTTP request processing cycle in Nginx – the filter chain in which the functions are embedded is underlined.

The embedded functions will be used for analysis of HTTP traffic and for injection of malicious code. The addresses for the replacements (in other words, the addresses of global references ngx_http_top_header_filter and ngx_http_top_body_filter) are also hard-coded in the shared object. This completes the initialization process. An overview of the hooking of ngx_http_copy_filter_init is shown in Figure 5.
Code injection

The two filters embedded into Nginx’s filter chain are used to provide code injection and remote control functions. Let’s start with the malicious HTTP header filter. During the execution of the filter the following steps are performed:

1. The ctx field of the ngx_http_request_t structure (the parameter of the original function) is obtained and checked.
2. If the pointer to ctx is NULL, then 160 bytes of memory will be allocated and the pointer to the memory area will be assigned to ctx. A special marker, 0xDEADBEEF, will be written into the memory.
3. The ctx memory is checked for the presence of the 0xDEADBEEF marker. If the marker is not found, the hook will execute the original ngx_http_top_header_filter and will exit after execution.
4. The filter performs several checks. For example, it checks whether the request method is ‘GET’, whether the content length is a non-zero value, whether the status code is 200, etc. If any of these checks fail, the execution of the hook will be interrupted and the original function will be invoked.
5. If an HTTP request contains the ‘Pragma’ header and remote control is allowed by the current configuration, then the filter will attempt to process it as a management request.
6. If it is not a management request, the filter performs more checks. It checks whether the current time value is greater than a particular value, that the client IP address isn’t blacklisted and malicious code hasn’t already been injected into the HTTP response for this client, that the URI doesn’t contain certain forbidden substrings listed in the configuration, that the processed HTTP response has a Content-Type header with a proper value, that the client has a proper user-agent and referer headers, that root isn’t logged on, and that a forbidden process isn’t being run. If the processed HTTP header is suitable for code injection, information about it will be stored in the ctx field.

7. The original filter is executed.

Remote control functions

Effusion’s remote control is accomplished via a specially crafted HTTP request that must contain the ‘Pragma’ header. During the processing of such a request, the value of
the ‘Pragma’ header is decoded from BASE64, then the first eight bytes of decoded data are decrypted and the first four bytes of the eight-byte block are checked for the presence of the 0xDEADBEEF marker. The last four bytes in this eight-byte block denote the remote command. The available types of command are shown in Table 2.

<table>
<thead>
<tr>
<th>The last DWORD value in first eight bytes</th>
<th>Description of remote command</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001h</td>
<td>Get status of the malware</td>
</tr>
<tr>
<td>10002h</td>
<td>Update malware configuration</td>
</tr>
<tr>
<td>10003h</td>
<td>Resume code injection</td>
</tr>
<tr>
<td>10004h</td>
<td>Pause code injection</td>
</tr>
<tr>
<td>10005h</td>
<td>Backconnect to remote server</td>
</tr>
</tbody>
</table>

Table 2: The remote commands available.

The other part of data is the payload, which is encrypted only in the case of update malware configuration messages. For example, if the attacker wanted the malware to perform a backconnect and route his commands to an opened root shell, he would send an HTTP request with the ‘Pragma’ header and the value of this header must be in the following form:

BASE64_ENCODE(
XTSA_ECB_ENCRYPT(key, 0xDEADBEEF||0x10005)||IP address||Port
)

where ‘key’ is the encryption key which is stored in the data segment of the sample; the backconnect is performed in the child process which appears after the call to the fork function during the initialization of the shared object. An overview of the remote control function is shown in Figure 6.

Monitoring of the processes in the system and detection of root activity

The malware has functions for scanning the list of running processes and for detection of root activity in the system. Such functions are implemented in order to protect the shared object from anti-rootkit software such as rkhunter, and from being detected by the server administrator. The functionality acts in the child process which appears after the call to the fork function during the initialization of the shared object.

While monitoring system processes, the shared object reads the content of the /proc directory. Each record is examined to determine whether it is a number or a string. After that, a path to a command line for each process is obtained in the form of '/proc/%d/cmdline', then the values of the command lines are read and checked for the presence of forbidden process names. If a forbidden process is being run, the shared object stops acting.

As for the detection of root activity, the malware obtains the IDs of the processes being run, then for each process the status is read ('/proc/%d/status'). Next, a UID is obtained from each status and compared with zero. If there is a process whose status contains a zero UID, then for that process opened file descriptors are obtained by reading '/proc/%d/fd'. After that the malware searches through opened file descriptors for those that contain the 'pts' substring, and the modification time of such descriptors is obtained via a call to the lstat function. Eventually, if the difference between the current time value and the value of the modification time is less than a constant set in the configuration, the malware decides that root is logged in and stops acting.

Configuration decryption algorithm

Every sample we analysed contained initial configuration stored in the data segment in an encrypted form. The decryption key is also stored in the data segment. The first byte of the encryption key is used as an offset inside the data segment array and is used to find a valid start address of the ciphertext.

At first, 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 8 shows pseudo code of the decryption algorithm.

We analysed this code and found that this is an implementation of the XTEA encryption algorithm [5, 6] with the number of rounds equal to 11; the mode of operations is ECB [7, 8]. Different encryption keys are used in different samples. We developed a special tool for the decryption of such configurations [9].

Configuration of the shared object can be updated via specially crafted HTTP requests – the XTEA algorithm in ECB mode is also used for data decryption in such requests.

Format of the configuration

Examples of the initial configuration and updated configuration of the samples are presented in Figures 9 and 10. The first part of the configuration contains special flags and offsets to data in the rest of the file.

The configuration format is described in Table 3.

None of the samples we analysed contained malicious code for injection in their initial configuration – such malicious code appeared only after an update of the configuration via special management HTTP requests.

BLACK MARKET

Effusion appeared on the black market on 13 November 2013, costing $2,500 – it is sold only to a limited number of verified customers. Its author also developed ‘Trololo_mod’, a malicious module for an Apache web server.

According to a seller on one of the underground forums, the product is distributed in binary form and doesn’t need developer packages to be installed on the target server. An attacker just needs to run the builder that will install the malware; the process takes between 60 and 180 seconds. The malware doesn’t require a C&C server for its activity.

INFECTION CAMPAIGN

As stated at the beginning of this article, Effusion was used in a massive infection campaign which started in the middle
of November 2013. Figure 11 shows the number of infected
hosts and their appearances in Yandex search results (with alerts) on a day-by-day basis.

The victims were servers hosting moderately popular
websites. Effusion was used to embed code which loaded
malicious content from web resources with URLs in the
following format:

\[ hxxp://rdomn[0-9]{8,11}.hopto.me \]

In order to embed harmful code, Flash objects were also
used. Eventually, users were redirected to a landing page
of a Nuclear exploit kit, and a piece of ransomware was
installed onto their systems.

**CONCLUSION**

Effusion is the most sophisticated injector for *nix systems
that we have come across. In a nutshell, it has the following
peculiarities:

- ELF headers are modified in order to complicate analysis.
- Modified functions similar to strlen, inet_addr, etc. are
  used instead of regular ones.
The XTEA algorithm (11 rounds) in ECB mode is used for encryption/decryption.

- Hash tables are used in order to avoid performance issues.
- There are functions that monitor forbidden processes.
- Advanced techniques are used for checking root activity.
- Updated configuration is stored only in RAM and is never dumped to disk.

The appearance of this malware confirms the fact that attackers are moving from the practice of infecting individual files to infecting the executable files of web servers. The old infection methods are gradually coming to nought, clearing a way for modern hi-tech methods of malicious code embedding which are hard to detect using traditional approaches. Yandex uses a traffic analysis approach to detect such types of infection: an anti-virus robot browses web pages, emulates legitimate user behaviour and analyses HTTP responses, so harmful code injected into web pages can be detected. The SafeBrowsing API [10] can be used to check whether a particular site is infected, and additional information about detected malicious code is available at [11].

REFERENCES


CALL FOR PAPERS

VB2014 SEATTLE

*Virus Bulletin* is seeking submissions from those wishing to present papers at VB2014, which will take place 24–26 September 2014 at the Westin Seattle hotel, Seattle, WA, USA.

The conference will include a programme of 30-minute presentations running in two concurrent streams. Unlike in previous years, the two streams will not be distinguished as ‘corporate’ and ‘technical’, but instead will be split into themed sessions covering both traditional AV issues and some slightly broader aspects of security:

- Malware & botnets
- Anti-malware tools & methods
- Mobile devices
- Spam & social networks
- Hacking & vulnerabilities
- Network security

Submissions are invited on topics that fall into any of the subject areas listed above. A more detailed list of topics and suggestions can be found at http://www.virusbtn.com/conference/vb2014/call/.

SUBMITTING A PROPOSAL

The deadline for submission of proposals is **Friday 7 March 2014**. Abstracts should be submitted via our online abstract submission system. You will need to include:

- An abstract of approximately 200 words outlining the proposed paper and including five key points that you intend the paper to cover.
- Full contact details.
- An indication of which stream the paper is intended for.

The abstract submission form can be found at http://www.virusbtn.com/conference/abstracts/.

One presenter per selected paper will be offered a complimentary conference registration, while co-authors will be offered registration at a 50% reduced rate (up to a maximum of two co-authors). *VB* regrets that it is not able to assist with speakers’ travel and accommodation costs. Authors are advised that, should their paper be selected for the conference programme, they will be expected to provide a full paper for inclusion in the VB2014 Conference Proceedings as well as a 30-minute presentation at VB2014. The deadline for submission of the completed papers will be 10 June 2014, and potential speakers must be available to present their papers in Seattle between 24 and 26 September 2014.

Any queries should be addressed to editor@virusbtn.com.