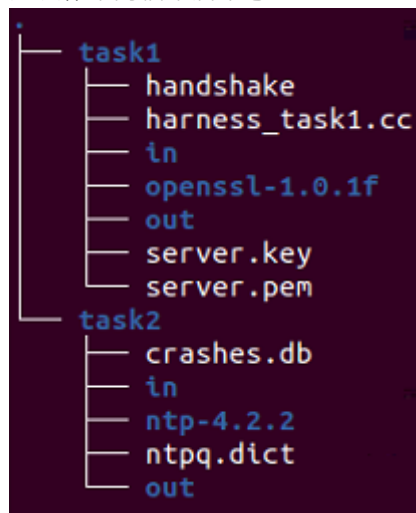


# 網安實務 Hw2

網工所碩一 309552005 吳偉誠

用到的工具:AFL(該HW的練習工具)、afl-collect(Task2我用於分析多個crashes data的)

這次作業的檔案層示意圖:



## Task1 - CVE-2014-0160

### 撰寫harness

基於助教提供的harness模板，我僅須對data、size加一點code即可:

原:

```
27 int main() {
28     static SSL_CTX *sslctx = Init();
29     SSL *ssl = SSL_new(sslctx);
30     BIO *rbio = BIO_new(BIO_s_mem());
31     BIO *wbio = BIO_new(BIO_s_mem());
32     SSL_set_bio(ssl, rbio, wbio);
33     SSL_set_accept_state(ssl);
34
35     /* TODO: To spoof one end of the handshake, we need to write data to rbio
36      * here */
37     BIO_write(rbio, data, size);
38
39     SSL_do_handshake(ssl);
40     SSL_free(ssl);
41     return 0;
42 }
43
```

在TODO的地方加上:

```
uint8_t data[100] = {0};
size_t size = read(STDIN_FILENO, data, 100);
if (size == -1) {
    err();
}
```

而err() 為:

```
void err() {
    ERR_print_errors_fp(stderr);
    exit(1);
}
```

多加的code思維:

因為data與size在原先程式並未定義  
因此我們僅需宣告其值，並加個錯誤判斷即可

## 解題步驟:

在task1資料夾底下解開openssl-1.0.1f.tar.gz

基本上跟助教提供的參考步驟類似。

### 1. 用ASAN編openssl:

在openssl-1.0.1f資料夾底下:

```
$ chmod +x config
$ AFL_USE_ASAN=1 CC=afl-clang-fast CXX=afl-clang-fast++ ./config -d -g
$ make
```

### 2. 編譯harness:

跟harness同層底下terminal:

```
$ AFL_USE_ASAN=1 afl-clang-fast++ harness_task1.cc -o handshake openssl-
1.0.1f/libssl.a openssl-1.0.1f/libcrypto.a -I openssl-1.0.1f/include -ldl
```

編完會生出一個"handshake"的執行檔

### 3. 創建一個虛擬證書:

使用OpenSSL創建例如512位RSA密鑰。該證書僅在Fuzzing測試期間使用，因此它的安全性與否無關緊要

```
$ openssl req -x509 -newkey rsa:512 -keyout server.key -out server.pem -days 365 -
nodes -subj /CN=a/
```

### 4. 跑Fuzz:

在task1底下創建兩個Dir: `in` `out`，用於當AFL的輸入與輸出

`in` 資料夾底下放入 `clienthello`:用於客戶端發送到服務器以初始化secure session的標準SSL hello的訊息，也作為AFL的seeds。

```
$ afl-fuzz -i in -o out -m none -t 5000 ./handshake
```

```
wulearn@wulearn-MS-7B24: ~/Desktop/NS_hw2/task1

american fuzzy lop 2.57b (handshake)

process timing | overall results
  run time : 0 days, 0 hrs, 2 min, 5 sec | cycles done : 0
  last new path : 0 days, 0 hrs, 0 min, 35 sec | total paths : 14
  last uniq crash : 0 days, 0 hrs, 0 min, 27 sec | uniq crashes : 1
  last uniq hang : none seen yet | uniq hangs : 0
cycle progress | map coverage
  now processing : 2 (14.29%) | map density : 5.62% / 5.78%
  paths timed out : 0 (0.00%) | count coverage : 1.07 bits/tuple
stage progress | findings in depth
  now trying : interest 32/8 | favored paths : 10 (71.43%)
  stage execs : 70/255 (27.45%) | new edges on : 9 (64.29%)
  total execs : 38.0k | total crashes : 1 (1 unique)
  exec speed : 297.2/sec | total tmouts : 0 (0 unique)
fuzzing strategy yields | path geometry
  bit flips : 2/192, 0/189, 0/183 | levels : 2
  byte flips : 0/24, 0/21, 0/15 | pending : 12
  arithmetics : 2/1341, 0/1025, 0/173 | pend fav : 8
  known ints : 0/88, 1/418, 0/380 | own finds : 13
  dictionary : 0/0, 0/0, 0/0 | imported : n/a
  havoc : 9/33.8k, 0/0 | stability : 100.00%
  trim : 90.12%/18, 0.00%

[cpu000: 57%]
```

大約跑了一分多，會出一個uniq crashes

```
american fuzzy lop 2.57b (handshake)

process timing | overall results
  run time : 0 days, 0 hrs, 44 min, 20 sec | cycles done : 7
  last new path : 0 days, 0 hrs, 1 min, 49 sec | total paths : 68
  last uniq crash : 0 days, 0 hrs, 13 min, 11 sec | uniq crashes : 2
  last uniq hang : none seen yet | uniq hangs : 0
cycle progress | map coverage
```

第二個出現在跑了3X分鐘時(後來分析發現跟第一個的錯誤是一樣的)

## 5. 觀看crash訊息:

\$ ./handshake < out/crashes/該id

如:

```
wulearn@wulearn-MS-7B24:~/Desktop/NS_hw2/task1$ ./handshake < out/crashes/id\:000000\,sig\:06\,src\:000000\,op\:havoc\,rep\:8
=====
==3517956==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x000000433267 bp 0x7ffe3a7d2c10 sp 0x7ffe3a7d23d0
=====
Files
#0 0x433266 in memcpy (/home/wulearn/Desktop/NS_hw2/task1/handshake+0x433266)
#1 0x4cf4b8 in tls1_process_heartbeat /home/wulearn/Desktop/NS_hw2/task1/openssl-1.0.1f/ssl/t1_lib.c:2586:3
#2 0x5169b1 in ssl3_read_bytes /home/wulearn/Desktop/NS_hw2/task1/openssl-1.0.1f/ssl/s3_pkt.c:1092:4
#3 0x51873d in ssl3_get_message /home/wulearn/Desktop/NS_hw2/task1/openssl-1.0.1f/ssl/s3_both.c:457:7
#4 0x4f7dfe in ssl3_get_client_hello /home/wulearn/Desktop/NS_hw2/task1/openssl-1.0.1f/ssl/s3_srvr.c:941:4
#5 0x4f5ddb in ssl3_accept /home/wulearn/Desktop/NS_hw2/task1/openssl-1.0.1f/ssl/s3_srvr.c:357:9
#6 0x4de94f in SSL_do_handshake /home/wulearn/Desktop/NS_hw2/task1/openssl-1.0.1f/ssl/ssl_lib.c:2564:7
#7 0x4c71d2 in main /home/wulearn/Desktop/NS_hw2/task1/harness_task1.cc:47:3
#8 0x7f1838d50b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-start.c:308:16
#9 0x41c50d in _start (/home/wulearn/Desktop/NS_hw2/task1/handshake+0x41c50d)

0x629000009748 is located 0 bytes to the right of 17736-byte region [0x629000005200,0x629000009748)
allocated by thread T0 here:
```

在ASAN底下:

紅字部分可以看到有:heap-buffer-overflow的漏洞

註:這邊只截圖第一個crash的結果當代表

## 6. 對照CVE-2014-0160的回報:

CVE-ID
<b>CVE-2014-0160</b> <a href="#">Learn more at National Vulnerability Database (NVD)</a> • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description
The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug.

是**buffer over-read**的問題

因此可以驗證復現成功

## 漏洞分析

在執行Heartbeat服務時，因為函式庫沒有事先檢查所需要的回傳資料大小，可能是為了加快SSL溝通的速度，一律都傳回64KB大小的資料，而所回傳的資料，也並沒有驗證使用者是否有權限可以接收。

也就是說，當執行Heartbeat服務時，假設，如果原先只需回傳2KB的資料，但因為沒有事先確認原本所需資料的大小，當OpenSSL函式庫預設回傳64KB的資料，執行Heartbeat服務的伺服器就會把暫存在記憶體中其他62KB的資料，一併回傳。

簡單來說，就是最一般的資料權限與檔案大小處理不當，造成使用Heartbeat服務時lib會將存在記憶體中隨意位址的62KB回傳，又因為沒有權限的檢查，而造成每個人都可以藉由重複利用一直取得server端的記憶體資料。

-Reference:<https://ithelp.ithome.com.tw/articles/10157104>

## Task2 - CVE-2009-0159

### 解題步驟:

也是在task2底下解開ntp-4.2.2.tar.gz

由助教提示的需在ntp/ntp.c的main()進行改寫，剩下指令也跟助教給的類似。

因此我先將ntp.c裡的main()改成:

```
int ntpmain(
    int argc,
    char *argv[])
{
#ifdef __AFL_HAVE_MANUAL_CONTROL
    __AFL_INIT();
#endif
    int datatype = 0;
    int status = 0;
    char data[1024 * 16] = {0};
    int length = 0;
#ifdef __AFL_HAVE_MANUAL_CONTROL
    while (__AFL_LOOP(1000))
    {
#endif
        datatype = 0;
        status = 0;
        memset(data, 0, 1024 * 16);
        read(0, &datatype, 1);
        read(0, &status, 1);
        length = read(0, data, 1024 * 16);
        cookedprint(datatype, length, data, status, stdout);
#ifdef __AFL_HAVE_MANUAL_CONTROL
    }
#endif
}
```

```
#endif
    return 0;
}
```

改寫成具有stdin讀取數據類型、狀態和數據

輸出文件作為stdout

測試network program的常見方法-隔離測試解析器之類的目標功能

而與AFL一起連動的方法是參考這篇:

[https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/README\\_persistent\\_mode.md](https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/README_persistent_mode.md)

然後我又在nextvar的函數開頭加上:

```
memset(vname, 0, sizeof(vname));
memset(vvalue, 0, sizeof(vvalue));
```

確保這些static變數不會保留上一次運行的資料

## 1. 編譯ntpq:

在ntp-4.2.2資料夾底下:

```
$ CC=afl-clang-fast ./configure && AFL_HARDEN=1 make -C ntpq
```

## 2. 跑Fuzz:(使用助教提供的dict，來增加找到的路徑數量)

在task2資料夾下創兩個資料夾:in、out 作為fuzz的輸入輸出。

並輸入 `$ echo aaa > in/testcase` 作為testcase

之後下:

```
$ afl-fuzz -i in -o out -x ntpq.dict ntp-4.2.2/ntpq/ntpq
```

```
american fuzzy lop 2.57b (ntpq)

process timing
  run time   : 0 days, 0 hrs, 4 min, 14 sec
  last new path : 0 days, 0 hrs, 0 min, 0 sec
  last uniq crash : 0 days, 0 hrs, 0 min, 5 sec
  last uniq hang : none seen yet
  cycle progress
    now processing : 277* (75.48%)
    paths timed out : 0 (0.00%)
  stage progress
    now trying : havoc
    stage execs : 6720/24.6k (27.34%)
    total execs : 5.71M
    exec speed  : 22.4k/sec
  fuzzing strategy yields
    bit flips : 33/143k, 15/143k, 11/142k
    byte flips : 0/17.9k, 0/14.2k, 0/13.8k
    arithmetics : 47/804k, 0/162k, 0/39.7k
    known ints  : 11/75.0k, 4/374k, 1/591k
    dictionary  : 13/378k, 9/527k, 7/66.2k
    havoc       : 288/2.20M, 0/0
    trim        : 12.36%/6765, 18.88%

overall results
  cycles done : 7
  total paths : 367
  uniq crashes : 74
  uniq hangs  : 0

map coverage
  map density : 0.26% / 0.78%
  count coverage : 3.89 bits/tuple

findings in depth
  favored paths : 62 (16.89%)
  new edges on  : 92 (25.07%)
  total crashes : 95.8k (74 unique)
  total tmouts  : 0 (0 unique)

path geometry
  levels : 16
  pending : 91
  pend fav : 0
  own finds : 366
  imported : n/a
  stability : 99.22%

[cpu001: 55%]
```

> 可以發現不到5分鐘就有74個uniq crashes

### 3. 分析crash

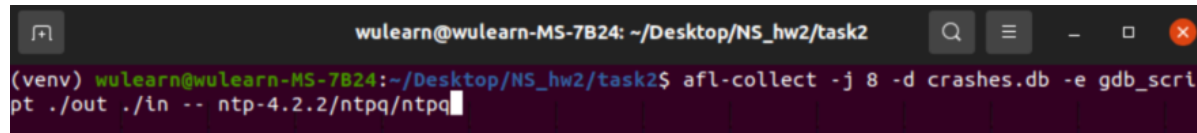
由於找到的crashes的資料太多，所以我才使用最前面提到的工具:afl-collect:

- 可以快速且自動判別各個結果造成的可能  
安裝afl-collect是參考這篇:<https://gitlab.com/rc0r/afl-utils>

要觀看crash分析的話就在task2資料夾內下這指令:

```
afl-collect -j 8 -d crashes.db -e gdb_script ./out ./in -- ntp-4.2.2/ntp/ntp
```

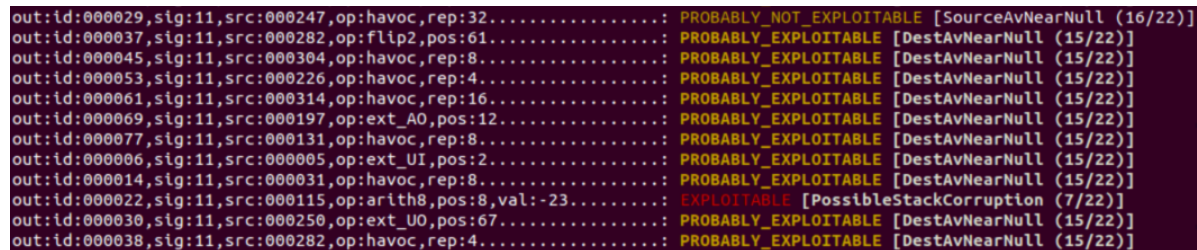
如圖:



```
wulearn@wulearn-MS-7B24: ~/Desktop/NS_hw2/task2
(venv) wulearn@wulearn-MS-7B24:~/Desktop/NS_hw2/task2$ afl-collect -j 8 -d crashes.db -e gdb_script ./out ./in -- ntp-4.2.2/ntp/ntp
```

註:我afl-collect是裝在虛擬機上

結果:



```
out:id:000029,sig:11,src:000247,op:havoc,rep:32.....: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
out:id:000037,sig:11,src:000282,op:flip2,pos:61.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000045,sig:11,src:000304,op:havoc,rep:8.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000053,sig:11,src:000226,op:havoc,rep:4.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000061,sig:11,src:000314,op:havoc,rep:16.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000069,sig:11,src:000197,op:ext_A0,pos:12.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000077,sig:11,src:000131,op:havoc,rep:8.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000006,sig:11,src:000005,op:ext_UI,pos:2.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000014,sig:11,src:000031,op:havoc,rep:8.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000022,sig:11,src:000115,op:arith8,pos:8,val:-23.....: EXPLOITABLE [PossibleStackCorruption (7/22)]
out:id:000030,sig:11,src:000250,op:ext_U0,pos:67.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
out:id:000038,sig:11,src:000282,op:havoc,rep:4.....: PROBABLY_EXPLOITABLE [DestAvNearNull (15/22)]
```

會告訴每個crash的可能性

其中值得注意的是id:000022 (可能有Stack的錯誤)

註:後續也有相關的PossibleStackCorruption問題，但也跟上述問題一樣，就沒特別列出來了

這邊我們先用ASAN再重新編一次程式:(在ntp-4.2.2資料夾底下)

```
$ CC=afl-clang-fast ./configure && AFL_USE_ASAN=1 make -C ntp
```

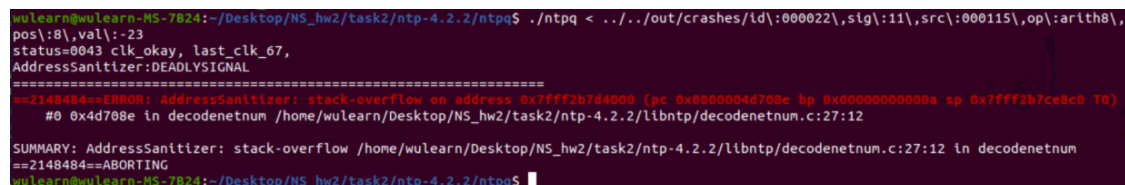
ASAN是讓我們方便觀看是出現啥問題

不事先用ASAN編再跑AFL:是因為這樣AFL就不能跑起來了

這邊我們拿id:000022 拿來實際跑看看:

在ntp-4.2.2/ntp/內下指令\$ ./ntp < ../../out/crashes/該id名稱

如圖:



```
wulearn@wulearn-MS-7B24:~/Desktop/NS_hw2/task2/ntp-4.2.2/ntp$ ./ntp < ../../out/crashes/id:000022,sig:11,src:000115,op:arith8,
pos:8,val:-23
status=0043 clk_okay, last_clk_67,
AddressSanitizer:DEADLYSIGNAL
=====
==2148484==ERROR: AddressSanitizer: stack-overflow on address 0x7ffff2b7d400 (pc 0x0000004d708e bp 0x00000000000a sp 0x7ffff2b7ce8c T0)
#0 0x4d708e in decodenetnum /home/wulearn/Desktop/NS_hw2/task2/ntp-4.2.2/libntp/decodenetnum.c:27:12
SUMMARY: AddressSanitizer: stack-overflow /home/wulearn/Desktop/NS_hw2/task2/ntp-4.2.2/libntp/decodenetnum.c:27:12 in decodenetnum
==2148484==ABORTING
wulearn@wulearn-MS-7B24:~/Desktop/NS_hw2/task2/ntp-4.2.2/ntp$
```

可以看到有stack-overflow的漏洞

### 4. 對照CVE-2009-0159的回報:

CVE-ID
<b>CVE-2009-0159</b> <a href="#">Learn more at National Vulnerability Database (NVD)</a> • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description
Stack-based buffer overflow in the cookedprint function in ntpq/ntp.c in ntpq in NTP before 4.2.4p7-RC2 allows remote NTP servers to execute arbitrary code via a crafted response.

確實有:Stack-based buffer overflow問題

### 漏洞分析

其有問題的code是在ntpq.c底下的cookedprint()函數:

```
case 0C:
    if (!decodeuint(value, &uval))
        output_raw = '?';
    else
    {
        char b[10];
        (void)sprintf(b, "%03lo", uval);
        output(fp, name, b);
    }
    break;
```

buf空間給的不夠多(可以輸入不包含NULL的11個字節)

因此將buf空間加大(11+NULL=12) · 並改用為更安全的snprintf函數:

```
else
{
    //char b[10];
    char b[12];
    //(void)sprintf(b, "%03lo", uval);
    (void)snprintf(b, sizeof(b), "%03lo", uval);
    output(fp, name, b);
}
break;
```