

Fuzzing Practical Lab

Software Security

a.a. 2022/2023

Laurea Magistrale in Ing. Informatica

Roberto Natella



Challenge

- Challenge (required):
 - Fuzz OpenSSL with **AFL** and **Address Sanitizer** (ASAN)
 - Reproduce Heartbleed
 - Interpret results, diagnose the bug
- Challenge (extra):
 - Try crashing the program without ASAN
 - Analyze the program with Valgrind
 - Run AFL with performance optimizations
 - Fix Heartbleed



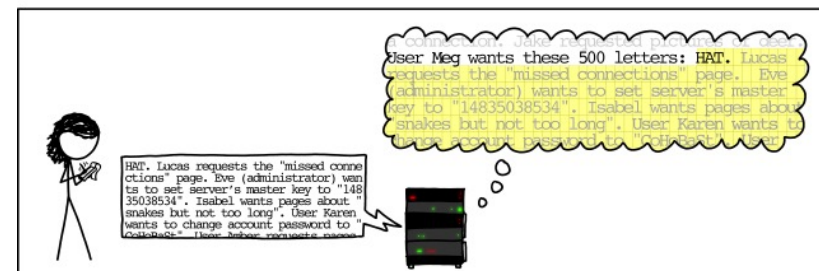
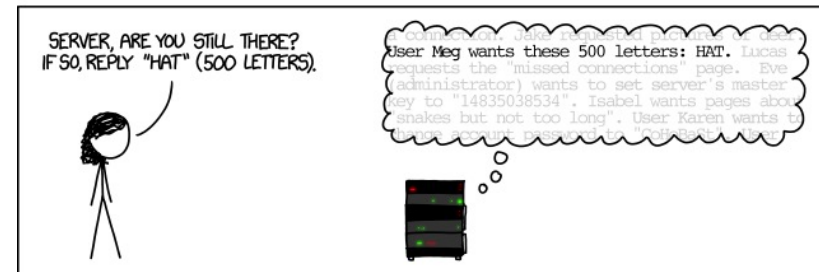
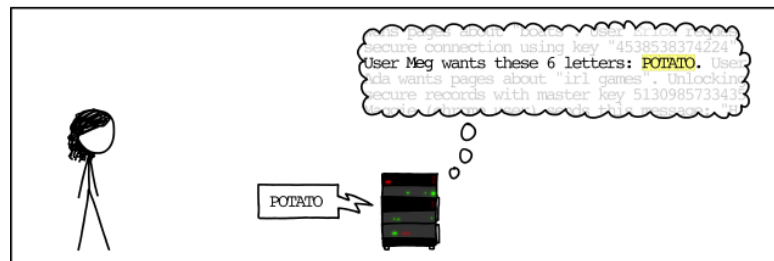
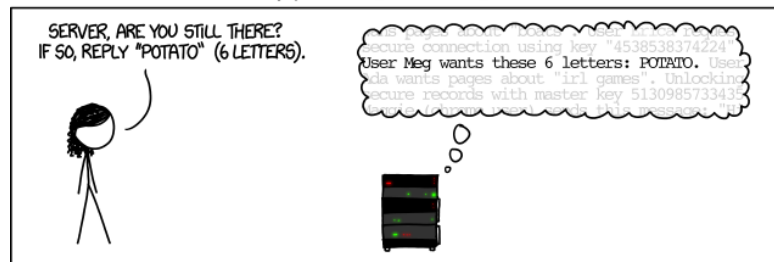
Challenge: Fuzzing OpenSSL

- **OpenSSL** è una libreria che implementa i protocolli TLS (Transport Layer Security) ed SSL (Secure Socket Layer)
- Nel 2014, è stata riscontrata una vulnerabilità **buffer over-read** ([CVE-2014-0160](#)) nel componente Heartbeat Extension
- È possibile identificarla usando AFL con ASAN
- La versione vulnerabile della libreria è la **1.0.1f**



Challenge: Fuzzing OpenSSL

HOW THE HEARTBLEED BUG WORKS:



Challenge: Fuzzing OpenSSL

- Se si utilizza la VM del corso, **AFL** è già installato
- Altrimenti, installare **AFL** usando il package manager della propria distribuzione Linux
- In Ubuntu, è disponibile il pacchetto "**AFLplusplus**"

```
$ sudo apt install afl++
```



Challenge: Fuzzing OpenSSL

- Dal repository Git del corso, nella sotto-cartella "**fuzzing/heartbleed**", prelevare il **codice sorgente della libreria OpenSSL**
- Sarà copiato nella sottocartella "**openssl**"

```
$ git pull
$ git submodule init
$ git submodule update
```

- Compilare la **libreria OpenSSL**, abilitando anche **ASAN**

```
$ cd openssl
$ <variabili di ambiente CC, CXX> ./config -d -g -no-shared
$ <variabili di ambiente ASAN> make build_libs
```



Challenge: Fuzzing OpenSSL

- You can compile the library and the test harness either using **GCC**:
 - **afl-gcc** (for C code, **CC** environment var.)
 - **afl-g++** (for C++ code, **CXX** environment var.)
- Or, you can use the **Clang** compiler suite:
 - **afl-clang-fast** (for C code, **CC** environment var.)
 - **afl-clang-fast++** (for C++ code, **CXX** environment var.)



Challenge: Fuzzing OpenSSL

- Completare il seguente programma di "test harness" (**handshake.cc**)
- **Leggere 100 byte da STDIN**, e **copiarli in un array "data" di byte**

```
int main() {
    static SSL_CTX *sctx = Init();
    SSL *server = SSL_new(sctx);
    BIO *sinbio = BIO_new(BIO_s_mem());
    BIO *soutbio = BIO_new(BIO_s_mem());
    SSL_set_bio(server, sinbio, soutbio);
    SSL_set_accept_state(server);
```

handshake.cc

Complete this part, using **read()** to copy 100 bytes from **STDIN** into an array **data[100]**

```
/* TODO: To spoof one end of the handshake, we need
    to write data to sinbio here */
```

```
BIO_write(sinbio, data, size);
```

```
SSL_do_handshake(server);
SSL_free(server);
return 0;
```

```
}
```

This function **calls the target library**, using the input data. Replace "size" with 100.

<https://stackoverflow.com/questions/15883568/reading-from-stdin>



Challenge: Fuzzing OpenSSL

- Compilare il **test harness**, utilizzando il **compilatore fornito da AFL**, e abilitando **ASAN**

```
$ <var. ASAN> <compilatore AFL C++> handshake.cc
    -o handshake
    openssl/libssl.a openssl/libcrypto.a
    -I openssl/include -ldl
```

- Per eseguire il programma, **creare un certificato fittizio** con il comando "openssl"

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



Challenge: Fuzzing OpenSSL

- Creare una cartella "input", con il **seed** per il fuzzing
- Usare come seed un **semplice file di testo** (es., un file la parola "ciao")
- Avviare AFL con il test harness
- Attendere un breve tempo (es. 5 minuti), verificare che venga rilevato un crash

```
$ afl-fuzz <parametri> -- <programma target>
```

Fare attenzione a NON usare @@, in modo che gli input siano passati **via STDIN**



Diagnose the bug (ASan)

- Eseguire nuovamente il programma "handshake" (senza AFL), passandogli in ingresso il file con l'input che causa il crash

```
$ ./handshake < file_input_crash
```

- Interpretare l'output di ASAN
 - che **tipo di errore che è stato rilevato**?
 - qual è **il punto nel codice di OpenSSL** che causa l'errore?
 - qual è **il punto nel codice di OpenSSL** che ha allocato il buffer?



Diagnose the bug (GDB)

- Eseguire ancora una volta "handshake", **via GDB**
- Se **ASan** è abilitato, inserire un **breakpoint** prima di lanciare l'esecuzione (ferma il processo prima di terminare)
- Ispezionare lo **stack frame**, ed il contenuto della variabile "**payload**"

```
gdb> set breakpoint pending on
gdb> break __asan::ReportGenericError
gdb> run < file_input_crash
...
gdb> bt
gdb> frame 2
gdb> print/x payload
```



Diagnose the bug (inputs)

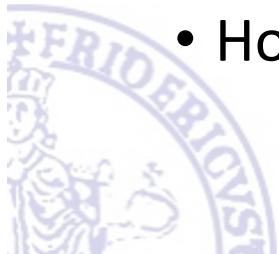
- Si confronti il contenuto della variabile "**payload**" con il **contenuto del file di input**
- Quali sono i byte dell'input che provocano l'errore?

```
$ hexdump -C file_input_crash
```



Challenge extra

- Modify the test harness to use AFL **"persistent mode"** (`__AFL_LOOP`)
 - You need to compile with **afl-clang-fast++**
 - To use **afl-g++**, you will need to add defines from https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/READEME.persistent_mode.md
- How much does it improve the **test execution throughput**?
 - Compare throughput with/without persistent mode
- How long it takes to trigger the crash?



Challenge extra

- Check that, **without ASAN**, the crash cannot be triggered. Why?
- Experiment with **Valgrind** to diagnose the crash



Challenge extra: fix Heartbleed

- L'errore è provocato dalla seguente struttura dati
- Definita in *include/openssl/ssl3.h*

```
typedef struct ssl3_record_st
{
    /*r */ int type;                /* type of record */
    /*rw*/ unsigned int length;    /* How many bytes available */
    /*r */ unsigned int off;       /* read/write offset into 'buf' */
    /*rw*/ unsigned char *data;    /* pointer to the record data */
    /*rw*/ unsigned char *input;   /* where the decode bytes are */
    /*r */ unsigned char *comp;    /* only used with decompression - malloc()ed */
    /*r */ unsigned long epoch;    /* epoch number, needed by DTLS1 */
    /*r */ unsigned char seq_num[8]; /* sequence number, needed by DTLS1 */
} SSL3_RECORD;
```



Challenge extra: fix Heartbleed

- L'errore è provocato dalla seguente struttura dati
- Definita in *include/openssl/ssl3.h*

```
typedef struct ssl3_record_st
{
    /*r */ int type;
    /*rw*/ unsigned int length; /* How many bytes available */
    /*r */ unsigned int off; /* read/write offset into 'buf' */
    /*rw*/ unsigned char *data; /* pointer to the record data */
    /*rw*/ unsigned char *input; /* where the decode bytes are */
    /*r */ unsigned char *
    /*r */ unsigned long
    /*r */ unsigned char
} SSL3_RECORD
```

Il valore di "**length**" deve essere maggiore o uguale a "**payload_length + 1+2+16**"

"data" è un vettore di byte, strutturato come:

type	payload_length	...dati...	...padding
(1 byte)	(2 byte)	(variabile)	(arrotonda a 16)



Challenge extra: fix Heartbleed

- **Correggere** il codice vulnerabile per prevenire l'attacco
 - aggiungere "return 0" se "payload_length" è eccessivamente elevato

```

unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short hbtype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);

...
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);

```



Challenge extra: fix Heartbleed

- **Correggere** il codice vulnerabile

- aggiungere "return 0" se "payload"

"s->s3->rrec" contiene la struct "SSL3_RECORD", inclusi "length" e "data" to

```
unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short hbtype;
unsigned int payload;
unsigned int padding;

/* Read type and payload */
hbtype = *p++;
n2s(p, payload);

<INPUT VALIDATION>
...

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RES;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

Scrivi in "hbtype" il primo byte ("data[0]")

Scrivi in "payload" i successivi 2 byte ("data[1], data[2]")

Se i dati sono invalidi, scartare il messaggio in questo punto, con "return 0"



More labs

- Fuzzing 101 (GitHub Security Labs)
 - WinAFL, QEMU instrumentation, fuzzing JS engines
 - <https://github.com/antonio-morales/Fuzzing101>

