Software Security 09

Format strings vulnerabilities

Gianluca Amato
Università di Chieti-Pescara

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Vulnerabilità delle stringhe di formato

- Le vulnerabilità della stringa di formato (format strings vulnerabilities) sono una serie di bug
  - identificati nella seconda metà degli anni 2000;
  - selative alle *stringhe di formato* che si trovano come primo parametri di varie funzioni C (`printf`, `sprintf`, `fprintf`, ...);
  - consentono all'attaccante di corrompere la memoria (come in un buffer overflow)

```
printf("The answer is %d!",42);
```

Uso delle stringhe di formato

- Le stringhe di formato consentono di
  - Convertire vari tipi nella loro rappresentazione come stringa.
  - Specificare il formato della rappresentazione.
  - Cambiare il contenuto delle variabili !!
- Quest'ultima possibilità dà origine, se non usate correttamente, a vulnerabilità molto serie!
- Vedremo che questo tipo di vulnerabilità ricorda molto quelle di tipo SQL injection e Cross-site scripting.

Vulnerabilità

- Consideriamo il seguente programma

echo.c

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc > 1) {
        printf(argv[1]);
        printf("\n");
    } else {
        printf("Please, give me a value to echo.\n");
    }
    return 0;
}
```

Vulnerabilità

- Possiamo osservare che nel programma il primo argomento di linea di comando viene passato a `printf` nello spazio normalmente dedicato alla stringa di formato

```
(argc > 1) {
 printf(argv[1]);
 printf("\n");
```

- Cosa accade se nella riga di comando vengono passate stringhe contenenti il simbolo % ?

```
$ ./echo %s%s%s%s%s%s%s%s
Segmentation fault (core dumped)
```

Vulnerabilità

- Possiamo far andare in crash il programma aggiungendo un numero adeguato di `%s`
  - Per ogni `%s`, la funzione `printf` cerca il successivo parametro (puntatore alla stringa da stampare) nello stack o nei registri.
  - La funzione `main` non ha messo niente di specifico né nello stack né nei registri
    - I valori presi da printf sono quindi valori che stanno lì per altri motivi
  - Se uno di questi valori punta ad una zona di memoria non allocata, si genera un errore di *segmentation fault*.

Vedere il contenuto dello stack

- La stessa vulnerabilità si può usare per leggere il contenuto dello stack, con lo specificatore di formato `%x`
  - Lo specificatore `%x` legge un valore dallo stack e lo visualizza in esadecimale.
  - Possiamo usare `%`*n*`x` (con *n* intero) per stampare il valore su *n* cifre
  - Possiamo usare `%`*n*`$x` (con *n* intero) per stampare il valore dell'*n*-esimo parametro (senza dovere scrivere tutti i `%x` precedenti)

Vedere il contenuto dello stack

- Consideriamo la seguente variazione:

echo2.c

```c
#include <stdio.h>

int main(int argc, char** argv) {
    long long value1 = 0xabababababababab;
    long long value2 = 0xcdcdcdcdcdcdcdcd;
    long long value3 = 0xefefefefefefefef;

    if (argc > 1) {
        printf(argv[1]);
        printf("\n");
    } else {
        printf("Please, give me a value to echo.\n");
    }
    return 0;
}
```

Vedere il contenuto dello stack

- Possiamo usare le format string vulnerabilities per vedere il contenuto dello stack.
- Nei sistemi a 32 bit, possiamo leggere le tre variabili `value1`, `value2` e `value3` con

```
$ ./echo2-32 "%x %x %x %x %x %llx %llx %llx"
f7f3c000 0 0 0 0 efefefefefefefef cdcdcdcdcdcdcdcd abababababababab
```

- I primi cinnque `%x` visualizzano solo semplici valori di padding aggiunti all'inizio di `main` per far sì che la cima dello stack fosse allineata a 16 byte.

Vedere il contenuto dello stack

- Nei sistemi a 64 bit la procedura è simile ma la stringa è leggermente diversa.

```
$ ./echo2 "%llx %llx %llx %llx %llx %llx %llx %llx %llx %llx %lx"
7fff29a3a678 7fff29a3a690 402e00 7fbf01674680 7fbf01676000 7fff29a3a678 200000000 0 efefefefefefefef cdcdcdcdcdcdcdcd abababababababab
```

- In questo caso:
  - I primi cinque `%llx` visualizzano i valori dei registri rsi, rdx, rcx, r8, r9 che, secondo le convenzioni, contengono i primi parametri
    - rdi contiene la stringa di formato
  - Il sesto e settimo `%llx` contengo i valori di argc e argv che la funzione main si salva nello stack quando viene chiamata
  - L'ottavo `%llx` visualizza un dato per il padding
  - Seguono gli `%llx` che visualizzano le variabili

Corruzione della memoria

- La vulnerabilità delle stringhe di formato consente anche di modificare la memoria.
  - Lo specificatore `%n` interpreta il proprio parametro come un indirizzo di memoria, nella quale la printf salverà il numero di caratteri stampati.
  - Se riusciamo a fornire nello stack in corrispondenza del parametro `%n` un valore a nostra scelta, possiamo modificare qualunque locazione di memoria.

Corruzione della memoria

changeit.c

```c
#include <stdio.h>

int flag;

int main(int argc, char** argv) {
    if (argc < 1) {
        printf("Please, enter your name!\n");
    }

    printf(argv[1]);

    if (flag) {
        printf("\n\nYou win!\n");
    } else {
        printf("\n\nTry again.\n");
    }
}
```

Corruzione della memoria

- Notate che la variabile `flag` è globale
  - non si trova quindi nello stack ma nel segmento dati
- L'idea dell'attaco è:
  - Individuare la posizione in memoria della variabile flag
  - Fornire una stringa in input che:
    - Contiene questo indirizzo
    - Legge con `%x` tutti i valori nello stack fino ad arrivare alla posizione che contiene che questo indirizzo
    - Termina con un `%n`

Posizione di `flag`

- La posizione di `flag` la si può individuare disassemblando il codice.
- In questo caso, poiché la variabile è globale e il binario contiene la tabella dei simboli, possiamo direttamente usare readelf.
  - `readelf -a changeit-32 | grep flag`
  - Otteniamo:
    - `37: 0804b014     4 OBJECT  GLOBAL DEFAULT   25 flag`

Gianluca Amato
Università di Chieti-Pescara

https://cybersecnatlab.it

Software Security 09

Format strings vulnerabilities

FINE