

Forum:[Hack Forums](#)

Thread: [Linux C++/C DLL injection + Code Injection Tutorial - Part Two](#)

g4143 - February 19, 2012, 9:35 am

Main Page - <https://hackforums.net/showthread.php?tid=2220892>

Part Two - Preparing the injected code for both 32/64 bit GNU/Linux systems.

This part is preparing the package to be injected. Its basically shell code what we're preparing here.

Note: I know GNU/Linux (Intel/AMD) operates with both a 32 and a 64 bit environments so I'll prepare our package for 64 bit environment and let the user, if needed prepare the package for a 32 bit environment. The steps will be the same just use the 32 assembler code instead of the 64 bit assembler code. So if you have a 32 bit system then use the code that follows the 64 bit...Both 32 and 64 bit assembler code follows the same instructions.

64 bit assembler code

hello.s

Code .section .data

.section .bss

.section .text

.global _start

_start:

pushq %rbp

movq %rsp, %rbp

pushq %rax

pushq %rdi

pushq %rsi

pushq %rdx

call tohere

tohere:

popq %rsi

.equ myoff, . - mydata

addq \$-myoff + 0x1, %rsi

movq \$1, %rax

movq \$1, %rdi

movq \$mylen, %rdx

syscall

popq %rdx

popq %rsi

popq %rdi

popq %rax

```
movq  %rbp, %rsp
popq  %rbp
ret
```

mydata: .ascii "Injected Code!\n"

.equ mylen, . - mydata

The compile lines for hello.s are below

Quote:as hello2.s -o hello2.o

ld hello2.o -o hello2

Once you have copied and compiled the code, run it and you should have the following output

Code Injected, Code!

Segmentation fault

And yes I know the program 'Seg faults' but that's O.K. because we're going to strip down this code.

In a terminal run the objdump command and it will produce a text file named 'textfile' that contains a dump of the code hello.

Code objdump -D hello>testfile

Here's what textfile(objdump result) should look like.

Code hello: file format elf64-x86-64

Disassembly of section .text:

00400078 <_start>:

```
400078: 55      push  %rbp
400079: 48 89 e5  mov  %rsp,%rbp
40007c: 50      push  %rax
40007d: 57      push  %rdi
40007e: 56      push  %rsi
40007f: 52      push  %rdx
400080: e8 00 00 00 00  callq 400085 <tohere>
```

00400085 <tohere>:

```
400085: 5e      pop   %rsi
400086: 48 81 c6 28 00 00 00  add  $0x28,%rsi
40008d: 48 c7 c0 01 00 00 00  mov  $0x1,%rax
400094: 48 c7 c7 01 00 00 00  mov  $0x1,%rdi
40009b: 48 c7 c2 0f 00 00 00  mov  $0xf,%rdx
4000a2: 0f 05    syscall
4000a4: 5a      pop   %rdx
4000a5: 5e      pop   %rsi
4000a6: 5f      pop   %rdi
4000a7: 58      pop   %rax
4000a8: 48 89 ec  mov  %rbp,%rsp
4000ab: 5d      pop   %rbp
4000ac: c3      retq
```

004000ad <mydata>:

```
4000ad: 49 6e    rex.WB outsb %ds:(%rsi),(%dx)
```

```

4000af:  6a 65      pushq $0x65
4000b1:  63 74 65 64    movslq 0x64(%rbp,%riz,2),%esi
4000b5:  20 43 6f      and   %al,0x6f(%rbx)
4000b8:  64 65 21 0a    fs and %ecx,%fs:%gs:(%rdx)

```

Now armed with this info(textfile) you'll have to open hello in a hex editor and remove all the hex values except what's listed in textfile...The final product should look something like below(Note: this is what it'll look like in the hex editor).

hexeditor result

```

Code 55 48 89 E5 50 57 56 52 E8 00 00 00 00 5E 48 81 C6 28 00 00
00 48 C7 C0 01 00 00 00 48 C7 C7 01 00 00 00 48 C7 C2 0F 00
00 00 0F 05 5A 5E 5F 58 48 89 EC 5D C3 49 6E 6A 65 63 74 65
64 20 43 6F 64 65 21 0A 00

```

Congratulations, hello now contains our shell script that we'll inject into a running process. Well we'll inject it after two more tutorials.

The 32 bit version follows32 bit version

hello.s

Code .section .data

.section .bss

.section .text

.global _start

_start:

pushl %ebp

movl %esp, %ebp

pushl %eax

pushl %ebx

pushl %ecx

pushl %edx

call tohere

tohere:

popl %ecx

.equ myoff, . - mydata

addl \$-myoff + 0x1, %ecx

movl \$4, %eax

movl \$1, %ebx

movl \$mylen, %edx

int \$0x80

popl %edx

popl %ecx

popl %ebx

popl %eax

movl %ebp, %esp

popl %ebp

ret

```
mydata: .ascii "Injected Code!\n"
.equ mylen, . - mydata
compile lines
```

Code as hello2.s -o hello2.o
ld hello2.o -o hello2
Output from the 32 bit code.

Code Injected, Code!
Segmentation fault
The objdump line.

Code objdump -D hello>testfile
The contents of textfile

Code hello: file format elf32-i386

Disassembly of section .text:

```
08048054 <_start>:
8048054: 55      push  %ebp
8048055: 89 e5    mov   %esp,%ebp
8048057: 50      push  %eax
8048058: 53      push  %ebx
8048059: 51      push  %ecx
804805a: 52      push  %edx
804805b: e8 00 00 00 00    call 8048060 <tohere>

08048060 <tohere>:
8048060: 59      pop   %ecx
8048061: 81 c1 20 00 00 00    add  $0x20,%ecx
8048067: b8 04 00 00 00    mov  $0x4,%eax
804806c: bb 01 00 00 00    mov  $0x1,%ebx
8048071: ba 0f 00 00 00    mov  $0xf,%edx
8048076: cd 80      int  $0x80
8048078: 5a      pop   %edx
8048079: 59      pop   %ecx
804807a: 5b      pop   %ebx
804807b: 58      pop   %eax
804807c: 89 ec      mov  %ebp,%esp
804807e: 5d      pop   %ebp
804807f: c3      ret

08048080 <mydata>:
8048080: 49      dec   %ecx
8048081: 6e      outsb %ds:(%esi),(%dx)
8048082: 6a 65     push  $0x65
8048084: 63 74 65 64    arpl  %si,0x64(%ebp,%eiz,2)
8048088: 20 43 6f      and  %al,0x6f(%ebx)
804808b: 64 65 21 0a    fs and %ecx,%fs:%gs:(%edx)
Once you strip the hello file it should look like the output below..We're just stripping everything out
```

of hello except what's in the textfile displayed above.

```
Code 55 89 E5 50 53 51 52 E8 00 00 00 00 59 81 C1 20 00 00 00 B8
04 00 00 00 BB 01 00 00 00 BA 0F 00 00 00 CD 80 5A 59 5B 58
89 EC 5D C3 49 6E 6A 65 63 74 65 64 20 43 6F 64 65 21 0A 00
```

Now that you've created the shell script(hello), you'll have to wait for the next to tutorials to inject it.

Wow the responses are overwhelming...

To give you an idea where we're going with this, check out the following code. It basically calls the shell code we created above.

The 64 bit example - Only for 64 bit GNU/Linux operating systems.

```
Code #include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
```

```
typedef void (*pfunc)(void);
```

```
unsigned char mya[] = {
    0x55,0x48,0x89,0xE5,0x50,0x57,0x56,0x52,0xE8,
    0x00,0x00,0x00,0x00,0x5E,0x48,0x81,0xC6,0x28,
    0x00,0x00,0x00,0x48,0xC7,0xC0,0x01,0x00,0x00,
    0x00,0x48,0xC7,0xC7,0x01,0x00,0x00,0x00,0x48,
    0xC7,0xC2,0x0F,0x00,0x00,0x00,0x0F,0x05,0x5A,
    0x5E,0x5F,0x58,0x48,0x89,0xEC,0x5D,0xC3,0x49,
    0x6E,0x6A,0x65,0x63,0x74,0x65,0x64,0x20,0x43,
    0x6F,0x64,0x65,0x21,0x0A,0x00
};
```

```
int main(int argc, char**argv)
{
    pfunc tfunc = (pfunc)mya;
    unsigned long addr1 = (unsigned long)mya & ((0UL - 1UL) ^ (BUFSIZ - 1));/*find page*/

    mprotect((void*)addr1, 1, PROT_READ|PROT_WRITE|PROT_EXEC);

    tfunc();
    exit(EXIT_SUCCESS);
}
```

The 32 bit example - Only for 32 bit GNU/Linux operating systems.

```
Code #include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
```

```
typedef void (*pfunc)(void);
```

```
unsigned char mya[] = {
    0x55,0x89,0xE5,0x50,0x53,0x51,0x52,0xE8,
    0x00,0x00,0x00,0x00,0x59,0x81,0xC1,0x20,
```

```

0x00,0x00,0x00,0xB8,0x04,0x00,0x00,0x00,
0xBB,0x01,0x00,0x00,0x00,0xBA,0x0F,0x00,
0x00,0x00,0xCD,0x80,0x5A,0x59,0x5B,0x58,
0x89,0xEC,0x5D,0xC3,0x49,0x6E,0x6A,0x65,
0x63,0x74,0x65,0x64,0x20,0x43,0x6F,0x64,
0x65,0x21,0x0A,0x00
};

```

```

int main(int argc, char**argv)
{
    pfunc tfunc = (pfunc)mya;
    unsigned long addr1 = (unsigned long)mya & ((0UL - 1UL) ^ (BUFSIZ - 1));/*find page*/

    mprotect((void*)addr1, 1, PROT_READ|PROT_WRITE|PROT_EXEC);

    tfunc();
    exit(EXIT_SUCCESS);
}

```

-Legacy- - February 20, 2012, 11:43 pm

Why the hell has no one posted? Just because you don't understand the code here, doesn't mean you should not post. I have to go to sleep but once again, I will try this out in the morning. Looks bloody complicated (probably because I don't know assembly) but just the concept is fantastic. Great work on this and did you come up with this all from scratch?

Will post results when I can.

g4143 - February 21, 2012, 4:12 am

(Feb 21, 2012 12:43 AM)-Legacy- Wrote: [\[CITE\]](#) Why the hell has no one posted? Just because you don't understand the code here, doesn't mean you should not post. I have to go to sleep but once again, I will try this out in the morning. Looks bloody complicated (probably because I don't know assembly) but just the concept is fantastic. Great work on this and did you come up with this all from scratch?

Will post results when I can.

Yep, I pretty much figure out everything from scratch.

Terror32 - February 21, 2012, 4:14 am

Very good tutorial keep the good work.Thank you for that.

Nichole - February 21, 2012, 5:30 am

I'm definitely looking forward to trying this out! Going to go back to your part one tutorial though, but I'll try it later today when the Linux is all set up!

[H]ell - February 21, 2012, 6:52 am

This is to much information for kids on this forum to understand.
I advise you to go to another forum.

It's what I did.

g4143 - February 21, 2012, 7:23 am

(Feb 21, 2012 07:52 AM)[H]ell Wrote: [\[CITE\]](#) This is to much information for kids on this forum to understand.

I advise you to go to another forum.

It's what I did.

You are generally right but there is some hidden talent here.

So which forum?

tulamba - February 21, 2012, 8:20 am

g4143 dude

I have bookmarked it. I will give your thread its due share of importance and respect as my background is computer hardware. I saw your post in another thread where you had recommended one member to learn c and assembly. I am wondering if there is a book that addresses both aspects of it. To me, learning c without knowing assembly is like looking at only one side of the picture.

Again, its my personal opinion, you and every body can disagree :)

Kind Regards,

Dbof - February 21, 2012, 8:25 am

(Feb 21, 2012 07:52 AM)[H]ell Wrote: [\[CITE\]](#) This is to much information for kids on this forum to understand.

I advise you to go to another forum.

It's what I did.

So where did you go to? I was actually looking for a similar forum, but I have never found anything.
PM me ?

g4143 - February 21, 2012, 8:51 am

(Feb 21, 2012 09:20 AM)tulamba Wrote: [\[CITE\]](#) g4143 dude

I have bookmarked it. I will give your thread its due share of importance and respect as my background is computer hardware. I saw your post in another thread where you had recommended one member to learn c and assembly. I am wondering if there is a book that addresses both aspects of it. To me, learning c without knowing assembly is like looking at only one side of the picture.

Again, its my personal opinion, you and every body can disagree :)

Kind Regards,

This is one thing I agree with 100 percent. Learn C and the when you have enough questions about how C handles things then start learning general assembler.

A sound knowledge of general assembler programming will set a C programmer free.

About books that address both C/ASM, I think it still has to be written.