

Forum:[Hack Forums](#)

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

g4143 - February 17, 2012, 3:22 pm

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

In this 'part one' tutorial I'll try and explain how linux goes about handling Dll's or what they call shared objects(shared objects are GNU/Linux's Dlls) manually.

The first thing we need to familiarize ourselves with is GNU/Linux's Dynamic linking library dlfcn.h. We'll concentrate on three functions

Copied from the man pages:

```
Quote:void *dlopen(const char *filename, int flag);  
void *dlsym(void *handle, const char *symbol);  
int dlclose(void *handle);
```

dlopen()

The function dlopen() loads the dynamic library file named by the null-terminated string filename and returns an opaque "handle" for the dynamic library. If filename is NULL, then the returned handle is for the main program. If filename contains a slash ("/"), then it is interpreted as a (relative or absolute) pathname. Otherwise, the dynamic linker searches for the library as follows (see ld.so(8) for further details)

dlsym()

The function dlsym() takes a "handle" of a dynamic library returned by dlopen() and the null-terminated symbol name, returning the address where that symbol is loaded into memory. If the symbol is not found, in the specified library or any of the libraries that were automatically loaded by dlopen() when that library was loaded, dlsym() returns NULL. (The search performed by dlsym() is breadth first through the dependency tree of these libraries.) Since the value of the symbol could actually be NULL (so that a NULL return from dlsym() need not indicate an error), the correct way to test for an error is to call dlerror() to clear any old error conditions, then call dlsym(), and then call dlerror() again, saving its return value into a variable, and check whether this saved value is not NULL.

dlclose()

The function dlclose() decrements the reference count on the dynamic library handle handle. If the reference count drops to zero and no other loaded libraries use symbols in it, then the dynamic library is unloaded.

For injecting the Dll(shared object) we'll use LD_PRELOAD which allows us to load/inject our libraries/shared objects first.

More LD_PRELOAD info can be found here: http://en.wikipedia.org/wiki/DLL_injection

Now lets get started. First lets create a simple program that we'll eventually inject our very own Dll(shared object) into.

Note: create a project folder and place all files created in this tutorial in it. It just makes it easier if everything is contained in one folder.

testit.c

Code #include <stdio.h>

#include <stdlib.h>

#define ARR_SIZE 200

#define PROMPT "Enter a string->"

int main(int argc, char**argv)

{

char ca[ARR_SIZE];

fputs(PROMPT, stdout);

while (fgets(ca, ARR_SIZE, stdin))

{

fputs(ca, stdout);

fputs(PROMPT, stdout);

}

exit(EXIT_SUCCESS);

}

testit compile line

Quote:gcc testit.c -Wall -ansi -pedantic -o testit

The program(testit) we created was simple, all we needed was a looping program that didn't exit promptly so we can experience our inject Dll(shared object) eventually(not in this tutorial but soon).

Try copying and running the code, it should prompt you for some text and then display it and prompt again. When you want to exit type ctrl+c.

Now we have to extract some information from our executable testit. The first thing we need is the library paths(shared objects) and to do this we need to open a terminal and type

Quote:ldd /path_to_executable/testit

You should get an output similar to this(similar but not necessarily the same):

Quote:ldd testit

linux-vdso.so.1 => (0x00007fff7c3ff000)

libc.so.6 => /lib64/libc.so.6 (0x00007fca6f518000)

/lib64/ld-linux-x86-64.so.2 (0x00007fca6f889000)

We are interested in the path information for libc.so.6 which(for my system) is /lib64/libc.so.6

We'll need this information for our Dll(shared object).

Now, with the terminal still open, type this

Quote:objdump -d testit>testfile

which will produce an objdump of the executable testit. What we're looking for is the function

__libc_start_main which is in testfile.

testfile - partial contents

Quote:...

004004c8 <__libc_start_main@plt>:

4004c8: ff 25 5a 05 20 00 jmpq *0x20055a(%rip) # 600a28

<_GLOBAL_OFFSET_TABLE_+0x20>

4004ce: 68 01 00 00 00 pushq \$0x1

4004d3: e9 d0 ff ff ff jmpq 4004a8 <_init+0x18>

...

The function __libc_start_main is the function we'll override in our injected shared object. Why are we using an odd function like __libc_start_main? Well its just happens to be present in every executable that linux produces.

Here's some information on __libc_start_main -

http://refspecs.linuxbase.org/LSB_3.1.1/LSB-Core-generic/LSB-Core-generic/baselib---libc-start-main-.html

Now lets create/build our share object(shareobj.c) that we'll inject into our executable.

shareobj.c

Code /**64 bit system**/

#include <stdio.h>

#include <dlfcn.h>

typedef int

(*importmain)(int (*)(int, char **, char **), int, char **, void (*)(void), void (*)(void), void (*)(void), void (*)(void));

int __libc_start_main(int (*main)(int, char **, char **), int argc, char ** ubp_av, void (*init) (void), void (*fini) (void), void (*rtld_fini) (void), void (*stack_end))

{

int ans = 0;

importmain MaiN;

void * handle = dlopen("/lib64/libc.so.6", RTLD_LAZY); /*lib64/libc.so.6 from the ldd testit*/

MaiN = (importmain)dlsym(handle, "__libc_start_main");

fputs("My Injected DLL\n", stdout); /*should display if injection worked*/

ans = MaiN(main, argc, ubp_av, init, fini, rtld_fini, stack_end);

dlclose(handle);

return ans;

}

shareobj.c compile line

Quote:gcc -shared -fPIC -o shareobj.so shareobj.c -ldl -lpthread

If your having problems getting the shareobj.c working on a 32 bit system then copy the one below. Note this will only work if your libraries are the same version as mine. If this still doesn't work, then

post the results of running ldd testit and I'll show you what to do.

shareobj.c

Code `/**32 bit system**/`

`#include <stdio.h>`

`#include <dlfcn.h>`

`typedef int`

`(*importmain)(int(*) (int, char**, char**), int, char**, void(*) (void), void(*) (void), void(*) (void), void(*) (void));`

`int __libc_start_main(int(*main)(int, char **, char **), int argc, char ** ubp_av, void (*init) (void), void (*fini) (void), void (*rtld_fini) (void), void(*stack_end))`

`{`

`int ans = 0;`

`importmain MaiN;`

`void * handle = dlopen("/lib/libc.so.6", RTLD_LAZY); /*lib/libc.so.6 from the ldd testit*/`

`MaiN = (importmain)dlsym(handle, "__libc_start_main");`

`fputs("My Injected Dll\n", stdout); /*should display if injection worked*/`

`ans = MaiN(main, argc, ubp_av, init, fini, rtld_fini, stack_end);`

`dlclose(handle);`

`return ans;`

`}`

shareobj.c compile line

Quote: `gcc -shared -fPIC -o shareobj.so shareobj.c -ldl -lpthread`

Now our trail run...

First open a terminal and go to our project folder and type the following (case is important).

Quote: `LD_PRELOAD=./shareobj.so ./testit`

The output should be

Quote: My Injected Dll

Enter a string->

If the line "My Injected Dll" doesn't appear then you have some problems that we'll have to look at.

I know it doesn't look like much right now but you'll have to wait until the other pieces are complete before you see the power and flexibility of this code. If you don't understand the code completely, that's O.K. you can use it as a boiler plate once it's complete.

Also - This is the most difficult section, or the section with the most information so don't feel overwhelmed...I had to figure this all out myself.

Part two - <https://hackforums.net/showthread.php?tid=2228260>

RDCA - February 17, 2012, 6:58 pm

Well since I have a feeling that if no one responds you will not continue this, I shall respond. Try to remember that a lot more people than you think read threads without posting. I read most of this thread, and I understood some of it as well. I learned a lot as well, so thanks.

tulamba - February 17, 2012, 7:22 pm

Its a lengthy tutorial. I want to understand it first and then post the questions and comments. So it never means g4143 should not continue. You do your part, i shall do mine ;)

-Legacy- - February 18, 2012, 2:17 am

Oh boy, can't wait to try this tomorrow. I have seen so many threads with Windows DLL Injection but not one for Shared Object Injection. I personally am a Linux fan so this is kind of just awesome.

Props to you for figuring all this shit out. Will post my results tomorrow (need sleep!)

Nichole - February 18, 2012, 8:09 am

This is dynamic linking correct? Similarly in Windows to LoadLibrary? Also Could you briefly explain what a typedef is? Or how it can be used?

g4143 - February 18, 2012, 8:28 am

(Feb 18, 2012 09:09 AM)Duck Dodgers Wrote: [\[CITE\]](#) This is dynamic linking correct? Similarly in Windows to LoadLibrary? Also Could you briefly explain what a typedef is? Or how it can be used? Here you go, Google is a wonderful thing.

<http://en.wikipedia.org/wiki/Typedef>

Nichole - February 18, 2012, 8:30 am

(Feb 18, 2012 09:28 AM)g4143 Wrote: [\[CITE\]](#) Here you go, Google is a wonderful thing.

<http://en.wikipedia.org/wiki/Typedef>

Oh I've read that. From what I understand, it's renaming data types. I could be wrong.

g4143 - February 18, 2012, 2:50 pm

So has anyone tried it? Has anyone had any problems?

Nichole - February 18, 2012, 3:05 pm

I'm not on a linux at the moment so I can't try it. But this line I do not understand.

Code typedef int
(*importmain)(int(*) (int,char**,char**),int,char**,void(*) (void),void(*) (void),void(*) (void),void(*) (void));
I understand __libc_start_main is the initialization. But not the arguments.

g4143 - February 18, 2012, 4:09 pm

(Feb 18, 2012 04:05 PM)Duck Dodgers Wrote: [\[CITE\]](#) I'm not on a linux at the moment so I can't try it. But this line I do not understand.

Code typedef int
(*importmain)(int(*) (int,char**,char**),int,char**,void(*) (void),void(*) (void),void(*) (void),void(*) (void));
I understand __libc_start_main is the initialization. But not the arguments.
This is just creating a type that's like the definition of __libc_start_main.

http://refspecs.linuxbase.org/LSB_3.1.1/LSB-Core-generic/LSB-Core-generic/baselib---libc-start-main-.html

I use this to cast the result of the function `dlsym()` to the proper type 'importmain' and I use it to create a variable of type 'importmain' `Main`.

-Legacy- - February 18, 2012, 6:24 pm

(Feb 18, 2012 03:50 PM)g4143 Wrote: [\[CITE\]](#) So has anyone tried it? Has anyone had any problems?

Just tried it and worked like a charm. Now I just need to understand what was going on in the code. Great tutorial, hopefully in your next parts you will explain some things we can accomplish with this.

Pic of it working :D

Spoiler [\(Click to View\)](#)

[img]<http://www.i.imgur.com/yO1sf.png>[/img]