

Forum:[Hack Forums](#)

Thread: [\[Tutorial\] ~ The Basics of Linux Shared Library Attacks & Function Hooks](#)

Goodies - March 31, 2014, 1:20 am

Hello everyone!

Some of you know that Legion is being revamped and I have rejoined. In honor of this new world order, I decided to make a tutorial for my Linux people here :)

Note: I'm new to this and I've just decided to teach others what I know so if you see errors or wrong information in my thread, please PM me and I'll fix it!

What is a Library?

There are two "main" types of Linux Libraries are Shared Libraries and Static Libraries.

They all have the same naming convention. The first three characters should be "lib" and, when linking it in gcc or using the ld linker raw, the "lib" is not included in the filename. Static Libraries (aka: "Archives")

Extension: .a

A static library is linked at COMPILE TIME and is, therefore, static. The linker will literally incorporate the library's compiled code in the file it's compiling. This means, in terms of compiling binaries, you don't have to have an external library along with your code at runtime. However, there are many standard libraries that come in every linux windows distribution. Statically linking a library will copy ALL of its code, even if there's code you don't use.

Shared Libraries (aka: "Shared Objects")

Extension: .so

A shared library is referred to at RUNTIME and is, therefore, dynamic. If you write your own libraries that contain your own functions and plan to distribute the binary with a non-standard library, it's often best to use static so you can include this in your binary. However, having a Makefile to install the shared libraries can be a bonus so others can use your library as well. Using LD_PRELOAD, we will get programs to first use our function instead of the original.

What is LD_PRELOAD?

Essentially speaking, whatever shared library is inside of LD_PRELOAD (an environment variable) will be loaded BEFORE any other standard libraries. This is the meat and bones of this attack. We can write functions with the same name as those in any other standard library and reference the originals and store it in our own library. This is accomplished using dlsym().

What is dlopen() / dlsym()?

dlopen will open a shared library (by dynamically linking it) and will return a handle to this shared library that dlsym can then use. dlsym takes this handle and a function name (as a string literal / character array) and will return the memory address at which it is located. We can cast this memory address as a function with the same parameter types and call the original function.

Making a Very Basic Shared Library:

Okay, this is the part where we actually create a shared library. I'll be using GCC so I'll go over why we use each flag and their parameters. First, let's choose a function that is in that standard C library. So, pretty much any major function used in C other than dlsym().

I chose: strdup but I encourage you to try something else. Now, your first choice may be printf which is not bad, however, you should know about VA_ARG before you mess with with that.

Before we do this, we need to know the legitimate strdup function and how it works. It's very simple. All it does is allocate heap memory and duplicate a string literal or pointer to another variable. Simple, eh?

Run:
man strdup

We see this:
PHP Code
<?php
char *strdup(const char *s);

Okay, that's all we need! Now we can create our own function!

Filename: libHijackStrdup.c / libHijackStrdup.so
PHP Code
<?php
char *strdup(const char *s){
 return "Hello!";
}

Okay, great. So we have a function called "strdup". Let's compile it as a library and, since it's all in the standard C library, we don't need any extraneous links.

```
gcc -shared -fpic -o ./libHijackStrdup.so libHijackStrdup.c
```

Breakdown:

- shared = This let's GCC know that we are compiling a shared library.
- fpic = PIC (Position Independent Code) will allow the code to execute regardless of its position in memory allowing it to be randomized by ASLR.
- o ./libHijackStrdup.so = Output file that follows that standard naming convention of libraries.

Although, files are not usually capitalized at all xD

Now we should have our .so file. Okay, great. Now let's write a simple main.c file that will call strdup. Now, again, remember that strdup is in the standard linux library so there will NOT need to be any linking or any extraneous calls. The user will never have to conform, at all, to what we do.

main.c:
PHP Code
<?php
#include <string.h>
#include <stdio.h>
int main(){
 char *str = strdup("Something");
 printf("str = %s\n", str);
 return 0;
}

Alright, great. Let's go ahead and compile this normally:

```
gcc -o main main.c
./main
LD_PRELOAD=./libHijackStrdup.so ./main
```

Now, this seems kind of tedious to have to type "LD_PRELOAD=./libHijackStrdup.so" every time. Who, in their right mind, would do that? Well, there are tricks like:

```
echo "`pwd`/libHijackStrdup.so" > /etc/ld.so.preload
```

Adding Conditions to our Basic Shared Library:

What if we don't just want to return "Hello!" every single time? What if we want conditions?

Now, what I'm going to do is replace a word inside of the strdup command so when people pass the string literal "Muhammad" it goes to "CENSORED".

NOTE: These strings are the SAME LENGTH so I do not have to write a fancy replace function and re-declare any strings or pointers! Simply, I am going to overwrite any occurrence of "Muhammad" with "CENSORED"

For this to work, we need to actually keep the OLD version of strdup(). Well, actually, we can get around having to, but it's best practice to...

PHP Code

```
<?php
// Remember this?
char *strdup(const char *s);
// This is going to have to be edited to something like this:
static char *(*old_strdup)(const char*);
// This way, we cast the memory address returned by dlsym() as a function and can execute it, but the
// name will be changed to "old_strdup" as is the naming convention when hooking functions
```

Saving the old function is quite easy:

PHP Code

```
<?php
old_strdup = dlsym(RTLD_NEXT, "strdup");
// And that's it!!!
```

Why is it so easy?

Since the standard C library is already linked and loaded, we can call "strdup". However, note the "RTLD_NEXT". Looking at the man page, you can get a clear explanation of why I used it instead of "RTLD_LAZY". Had I used "RTLD_LAZY", it would return the memory address of my custom hook function because this library is loaded before the standard library. However, finding the NEXT occurrence (the original) is done by using "RTLD_NEXT".

Note: You MUST have:

```
#define _GNU_SOURCE
before you include <dlfcn.h>
My Final Library libHijackStrdup.c:
```

PHP Code

```
<?php
#define _GNU_SOURCE //REQUIRED for dlfcn
#include <string.h>
#include <dlfcn.h>
```

```
char *(*old_strdup)(const char*);
```

```
char *strdup(const char *s){
    old_strdup = dlsym(RTLD_NEXT, "strdup");
    char *str = old_strdup(s);
    char *c = old_strdup("CENSORED");
    char *ptr;
    short i;
    while((ptr = strstr(str, "Muhammad"))!=NULL)
    for(i=0;i<8;i++)
    *(ptr+i) = *(c+i);
    return str;
}
```

And main.c:

PHP Code

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

```
int main(){
    char *str = strdup("Hello! I am Muhammad. Nice to meet you!");
    printf("str = %s\n", str);
    return 0;
}
```

```
gcc -shared -fpic -o libHijackStrdup.so libHijackStrdup.c -ldl
echo "`pwd`/libHijackStrdup.so" > /etc/ld.so.preload
gcc main.c -o main
./main
```

And thus concludes part one of this series :)

Hooded Robin - March 31, 2014, 6:40 am

Great write-up, thanks for taking the time to put it together and share with us all :)

Ch1m4zz - March 31, 2014, 7:56 am

Yet another HQ thread from Goodies .. Thanks and keep it up

?ain - March 31, 2014, 10:40 am

New Legion.. I like that.

This is difficult, but its fun.

sp3ctrum - April 1, 2014, 1:53 pm

Goodies, serves the goods once again. Real nice write up man, lets keep this thread alive.

Goodies - *May 15, 2014, 3:09 pm*

Just in case people are still interested in this! I know I can't win the OMC prize, but I think I'd have a fighting chance with this one :) This is very interesting for anyone that knows what they're doing.

Goodies - *March 14, 2017, 1:52 pm*

Hello to any/everyone who recognizes me from this section. I used to post here quite a bit.

I'm back to being a full time Linux user (as of a month ago), so you could expect POTENTIALLY newer tutorials on things like this.

ps.fordham - *March 14, 2017, 2:30 pm*

mhh lets see if this work if u have question i pm u :-)