5 Returning from ELF to Libc

by Rebecca “Bx” Shapiro

Dear friends,

As you may or may not know, demons lurk within ELF metadata. If you have not yet been introduced to these creatures, please put this paper down and take a look at either our talk given at 29C3[9], or our soon-to-be released WOOT publication (in August 2013).

Although the ability to treat the loader as a Turing-complete machine is Pretty_Neat, we realize that there are a lot of useful computation vectors built right into the libraries that are mapped into the loader and executable’s address space. Instead of re-inventing the wheel, in this POC sermon we’d like to begin exploring how to harness the power given to us by the perhaps almighty libc.

The System V amd64 ABI scripture[10] in combination with the eglibc-2.17 writings have provided us ELF demon-tamers with the mighty useful IFUNC symbol. Any symbol of type IFUNC is treated as an indirect function – the symbol’s value is treated as a function, which takes no arguments, and whose return value is the patch.

The question we will explore from here on is: Can we harness the power of the IFUNC to invoke a piece of libc?

After vaguely thinking about this problem for a couple of months, we have finally made progress towards the answer.

Consider the exit() library call. Although one may question why we would want to craft metadata that causes a exit() to be invoked, we will do so anyway, because it is one of the simplest calls we can make, because the single argument it takes is not particularly important, and success is immediately obvious.

To invoke exit(), we must lookup the following information when we are compiling the crafted metadata into some host executable. This is accomplished in three steps, as we explain in our prior work.

1. The location of exit() in the libc binary.
2. The location of the host executable’s dynamic symbol table.
3. The location of the host executable’s dynamic relocation table.

To invoke exit(), we must accomplish the following during runtime:

1. Lookup the base address of libc.
2. Use this base address to calculate the location of exit() in memory.
3. Store the address of exit() in a dynamic IFUNC symbol.
4. Cause the symbol to be resolved.

... and then there was exit()!

Our prior work has demonstrated how to accomplish the first two tasks. Once the first two tasks have been completed at runtime, we find ourselves with a normal symbol (which we will call symbol 0) whose value is the location of exit(). At this point we have two ways to proceed: we can

(1) have a second dynamic symbol (named symbol 1) of type IFUNC and have relocation entry of type R_X86_64_64 which refers to symbol 0 and whose offset is set to the location of symbol 1’s values, causing the location of ext() to be copied into symbol 1,

-or-

(2) update the type of the symbol that already has the address of exit() to that it becomes an IFUNC. This can be done in a single relocation entry of type R_X86_64, whose addend is that which is copied to the

[9] `https://www.youtube.com/watch?v=dnLYoMIBhpo`

[10] `http://www.uclibc.org/docs/psABI-x86_64.pdf`

first 8 bytes of symbol 0. If we set the addend to `0x0100000a00000000`, we will find that the symbol type will become 0x0a (IFUNC), the symbol shndx will be set as 01 so the IFUNC is treated as defined, and the other fields in the symbol structure will remain the same.

After our metadata that sets up the IFUNC, we need a relocation entry of type R_X86_64_64 that references our IFUNC symbol, which will cause exit() to be invoked.

At this moment, you may be wondering how it may be possible to do more interesting things such as have control of the argument passed to the function call. It turns out that this problem is still being researched. In eglibc-2.17, at the time the IFUNC is called, the first argument is and will always be 0, the second argument is the address of the function being called, and the third argument the addressed of the symbol being referenced. Therefore at this level exec(0) is always called. It will clearly take some clever redirection magic to be able to have control over the function's arguments purely from ELF metadata.

Perhaps you will see this as an opportunity to go on a quest of ELF-discovery and be able to take this work to the next level. If you do discover a path to argument control, we hope you will take the time to share your thoughts with the wider community.

Peace out, and may the Manul always be with you.