Surreptitious Software

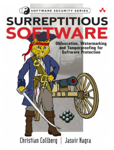
Exercise

Attacks Techniques

Christian Collberg
Department of Computer Science, University of Arizona
February 26, 2014

Learning about the executable (Linux)

p. 72



1. objdump prints out information about an executable file. It has lots of options, depending on what you want. The -T option prints the dynamic symbols:

```
> objdump -T player2
DYNAMIC SYMBOL TABLE:
00000000      DF *UND*  00000039  GLIBC_2.0   printf
00000000      DF *UND*  0000002b  GLIBC_2.0   atoi
00000000      DF *UND*  00000024  GLIBC_2.0   fprintf
00000000      DF *UND*  00000020  GLIBC_2.0   time
```

2. objdump can also disassemble:

```
> objdump -d player2 | head

080483f4 <.init>:
 80483f4:       55                        push   %ebp
 80483f5:       89 e5                     mov    %esp,%ebp
 80483f7:       83 ec 08                  sub    $0x8,%esp
```

3. objdump gives you the start address:

```
> objdump -f player2 | grep start
start address 0x080484f0
```

4. objdump gives you the address and size of the string (read-only date) and text segments:

```
> objdump -x player2 | egrep 'rodata|text|Name'
Idx Name          Size      VMA       LMA       File off  Algn
 11 .text         00000508  080484f0  080484f0  000004f0  2**4
 13 .rodata       00000075  08048a14  08048a14  00000a14  2**2
```

Learning about the executable (Mac OS X)

On Mac OS X we have to use otool instead of objdump for some operations.

1. To print the dynamic symbols:

```
> objdump -T player2
```

2. To disassemble:

```
> otool -t -v player2
```

3. To get the start address:

```
> otool -t -v player2 | head
```

4. To get the address and size of the string and text segments:

```
otool -l player2 | gawk '/__text/,/size/{print}'
otool -l player2 | gawk '/__cstring/,/size/{print}'
```

NOTE: `otool` sometimes displays addresses like "00000bd0", and sometimes like "0000000100000bd0." Inside `gdb` use "0x100000bd0" since this is the actual virtual address. (Avoid leading zeros since this indicates an octal address.)

Tracing the executable

1. `ltrace` traces library calls:

```
> ltrace -i -e printf player2
[0x804884e] printf("hash=0x%x\n", 0x478a1c90hash=0x478a1c90)    = 16
tampered!
[0x8048702] printf("Please enter activation code: ") = 30
Please enter activation code:
```

2. `strace` traces system calls:

```
> strace -i -e write player2
[110425] write(1, "hash=0x478a1c90\n", 16hash=0x478a1c90) = 16
[110425] write(2, "tampered!\n", 10tampered!) = 10
[110425] write(1, "Please enter activation code: ",... ) = 30
```

Gdb

1. To start `gdb`:

```
gdb -write -silent --args player2 0xca7ca115 1000 2000 3000 4000
```

2. The latest version of `gdb` (7.0 and above) has the new `find` command which searches for a string in an executable:

```
(gdb) find startaddress, +length, "string"
(gdb) find startaddress, stopaddress, "string"
```

NOTE: Note that you have to give the *entire* string you're looking for — `find` doesn't do partial searches. I believe it looks for the string *including the null character at the end*, so any trailing spaces, tabs, etc. have to be included in the search.

You can also search for bytes, words, etc.

3. To set a breakpoint at a particular address:

```
(gdb) break *0x......
(gdb) hbreak *0x......
```

`hbreak` sets a hardware breakpoint which doesn't modify the executable itself.

NOTE: Note that on x86-64, the program must be started before you can set a hardware breakpoint!

4. To set a watchpoint at a particular address:

```
(gdb) rwatch *0x......
(gdb) awatch *0x......
```

`rwatch` only checks for reads of the location.

NOTE: Note that on x86-64, the program must be started before you can set a hardware watchpoint!

5. To disassemble instructions:

```
(gdb) disass startaddress endaddress
```

or, if you only want to see a certain number (here, 3) of instructions:

```
(gdb) x/3i address
(gdb) x/i $pc
```

The second command prints the instruction at the current address,

6. To examine a data word (`x=hex`,s=string, d=decimal, `b=byte`,...):

```
(gdb) x/x address
(gdb) x/s address
(gdb) x/d address
(gdb) x/b address
```

You can hit return multiple times to examine consequtive locations.

7. To print register values:

```
(gdb) info registers
```

8. To examine the callstack:

```
(gdb) where
(gdb) bt         -- same as where
(gdb) up         -- previous frame
(gdb) down       -- next frame
```

9. To step one instruction at a time:

```
(gdb) display/i $pc
(gdb) stepi
(gdb) si        -- same as stepi
(gdb) nexti     -- like step, but don't step into functions
(gdb) ni        -- same as nexti
```

The `display` command only has to be set once. It makes sure that `gdb` prints the instruction it's stepping over.

10. To modify a value in memory:

```
(gdb) set {unsigned char}address = value
(gdb) set {int}address = value
```

Patching executables with gdb

Cracking an executable proceedes in these steps:

1. find the right address in the executable,
2. find what the new instruction should be,
3. modify the instruction in memory,
4. save the changes to the executable file.

This process is called *patching*.

`gdb` can patch the executable for us, but it is very picky about how to go about it. There are two ways to start the program to allow patching:

method 1:

```
> gdb -write -q player1
```

method 2:

```
> gdb -q player1
(gdb) set write
(gdb) exec-file player1     # reload the file!
```

`gdb` doesn't allow us to patch the executable when it is running. It's therefore best to:

1. Run the program under `gdb` and find the address of the instruction you want to patch.
2. Exit `gdb`.
3. Start `gdb` again using one of the two methods above.
4. Make the patch and exit:

```
(gdb) set {unsigned char} 0x804856f = 0x7f
(gdb) quit
```

Prerequisites

Before working the exercise make sure you download, install, and build the following:

tool	url	Linux	MacOS X	Windows
gcc		✠ `gcc build-essential`		
gdb v7.2	`ftp.gnu.org/gnu/gdb/`			
	• **Linux:** `./configure; make install; sudo chgrp procmod /usr/local/bin/gdb; sudo chmod g+s /usr/local/bin/gdb` • **Mac OS X:** *`Same as for Linux`* • **Windows:**			
objdump	`www.gnu.org/software/binutils`	✠ `binutils`		