



Surreptitious Software

Exercise

Attacks

Searching the Executable

Christian Collberg

Department of Computer Science, University of Arizona

February 26, 2014

Software protections

In this version, `player2` fails if you have the wrong activation code:

```
> player2 0xca7ca115 10000 20000 30000 60000
Please enter activation code: 99
wrong code!
Bus error
```

(The real activation code is "42".)

Obviously, you want the program to play either without nagging you about the activation code:

```
> player2 0xca7ca115 10000 20000 30000 60000
```

or continue running regardless of what activation code you give it:

```
> player2 0xca7ca115 10000 20000 30000 60000
Please enter activation code: 999
```

Figure 1 shows the code for `player2`. Of course, in real life, the attacker won't have access to the source code of our program, just the binary!

Prerequisites

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

1. Install the following tools:

tool	url	Linux	MacOS X	Windows
gcc		✂ gcc build-essential		
gdb v7.2	ftp.gnu.org/gnu/gdb/	<ul style="list-style-type: none">• Linux: <code>./configure; make install; sudo chgrp procmod /usr/local/bin/gdb; sudo chmod g+s /usr/local/bin/gdb</code>• Mac OS X: <i>Same as for Linux</i>• Windows:		
objdump	www.gnu.org/software/binutils	✂ binutils		

```

int activation_code = 42;
uint32 play(uint32 user_key, uint32 encrypted_media[], int media_len) {
    int code;
    printf("Please enter activation code: ");
    scanf("%i",&code);
    if (code!=activation_code) {
        fprintf(stderr,"%s!\n","wrong code");
        *((int*)NULL)=99;
    }

    int i;
    for(i=0;i<media_len;i++) { ... }
}

uint32 player_main (uint32 argc, char *argv[]) {
    ....
}

int main (uint32 argc, char *argv[]) {
    ...
}

```

Figure 1: The code.

2. Download program and data files:
 - (a) `wget 'http://www.cs.arizona.edu/~collberg/tmp/ssx.zip'`
 - (b) `unzip ssx.zip`
 - (c) `cd ssx/attack-defense_attack2`
3. Build the `player2` executable which you will be working on from now on:


```
> make
```

Algorithm — Cracking by Searching the Binary

To remove the activation code check we're going to use a slightly different strategy. The latest version of `gdb` has the ability to search for a string within the executable.

We can assume that the protection code looks something like this:

```

addr1: "wrong code"
.....
read_value = scanf()
if (read_value != activation_code)
    addr2: call printf(addr1)

```

or, in pseudo assembly code:

```

addr1: .ascii "wrong code"
.....
        mov read_value, reg0
        mov activation_code, reg1
        cmp reg0,reg1
        je somewhere
addr2:  mov addr1, reg0
        call printf

```

So, we

1. search for `addr1`, the address of the string "wrong code",

2. search for *addr2*, the address where `printf` is called,
3. look backwards in the code until we find the instructions that do the check if `(activation_code != 42) ...`, and
4. patch the code as in the previous exercise.

NOTE: This will only work if the compiler generates *addr1* directly in the code! Some compilers will instead load *addr1* as an offset from a base register — then we can't find *addr2* as easily as this! On Mac OS X we can compile with `gcc -mdynamic-no-pic` to turn off this behavior. Or, we can find *addr2* using a technique you will see in the next exercise, namely setting a memory watchpoint on *addr1*.

Crack — Remove the activation code check!

Now carry out the attack:

1. Before we can start searching the binary, you need to find out where the text segment and the read only data segment start, and how long they are:

```
> objdump -x player2 | egrep 'text|Name'           # Linux
> objdump -x player2 | egrep 'rodata|Name'

> otool -l player2 | gawk '/__text/,/size/{print}' # Mac OS X
> otool -l player2 | gawk '/__cstring/,/size/{print}'
```

2. Use `gdb`'s `find` command to find the address of the string `"wrong code"`.¹

Let's call this *data* address *addr1*. Check that you have the right address:

```
(gdb) x/s addr1 here!
addr1:      "wrong code"
```

3. Now use the `find` command again, looking through the *text segment* for an instruction that uses the *addr1* address!

¹Be careful to enter the search string exactly; `gdb`'s `find` command doesn't search for partial strings.

Let's call this *code* address *addr2*.

4. Disassemble a region prior to *addr2* to verify that is the region you're looking for.

5. Now that you know both *addr1* and *addr2*, it's time to do the patching! First exit, and then re-enter `gdb`.

You now need to replace the `je` (jump equal) branch with a `jmp` (jump always). The opcode for `jmp` is `0xeb`. Show `gdb` command here:

```
(gdb) quit
> gdb -write -silent player1
```

```
do the patch here!
(gdb) quit
```

(See <http://www.itis.mn.it/linux/quarta/x86/jmp.htm>.)

6. Try the patched program! Does it work for any activation code?