# Objective 6: Shellcode Primer

IT Security professionals may be required to dissect machine language code (reverse engineers), or to create it (exploit developers.) This challenge teaches us to write simple machine language applications.

The Shellcode Primer is next to Ruby Cyster in Jack’s office, but first we will need to visit Chimney Scissorsticks in the NetWars area on the top of Santa’s castle.

|  |  |
| --- | --- |
| Text  Description automatically generated |  |

## Terminal: Holiday Hero

This is another training adventure in using the Chrome or Firefox web developer tools. It will teach us how to manipulate cookies and how to manipulate JavaScript variables using the webdev console. It is important for use to know that cookies persist between sessions, but variables do not; we will have to reset the session midway, so the order will be important to us.

This game is based on the Guitar Hero game that was first released in 2005.

|  |  |
| --- | --- |
| A picture containing text  Description automatically generated | Text  Description automatically generated |
|  |

## Assignment

Win the Holiday Hero game to get the achievement and hints from Chimney Scissorsticks. It is a two-player game, so you will need a partner if you want to just play the game. Of course, you will learn more if you hack the game.

Use your browser webdev skills to win the game.

### Step 1 question

Notice Chimney’s words, *“Single player mode? I heard it can be enabled by fiddling with two client-side values, one of which is passed to the server.”*

There are many ways for a browser to pass values to a server, but we are looking for the most common one (Hint: It can be persistent.) Find the value that the Holiday Hero game is passing to the server and change it to a value that may let us play without another person.

### Step 1 answer

A very common way that the browser and server pass values to each other is through cookies. In Chrome you can find the cookie under the Application tab.

Graphical user interface, application

Description automatically generated

In Firefox, it is under the Storage tab.

Graphical user interface, application

Description automatically generated

Change the value from false to true.

Graphical user interface, application

Description automatically generated

### Step 2 question

Now you need to find the second variable that is not sent to the server. In is in the JavaScript on the browser. Use the webdev tool to find the JavaScript source code and find a variable that will help you enter single player mode. Note that the JavaScript you need will not appear in the browser until you have the game running (sleigh and holiday hero in its frame.)

### Step 2 Answer

In Chrome, source code is found under the Sources tab. You will have to dig for it, as it is in a different site from the main page, 2021.kringlecon.com. You want hero.kringlecastle.com. Note that “pretty-print” makes the code easier to read. Either the button or the curly brace icon ‘{}’ will work.

Graphical user interface, text, application

Description automatically generated

In Firefox, source code is found under Debugger > Sources. There is a ‘{}’ pretty-print icon in Firefox.

Graphical user interface, text, application

Description automatically generated

Use search (control-F) to find a variable that may help us.

In Chrome we find this.

Graphical user interface, text, application, Word

Description automatically generated

And in Firefox, this.

Graphical user interface, text, application, Word

Description automatically generated

In either case, we want to set single\_player\_mode = 1 to turn it on.

### Step 3 question

For the final step, the order is important.

1. Set the cookie; single\_player:true
2. Reload the frame or refresh the browser so the new game starts with the cookie
3. Set the context for webdev. The easiest way is to right-click Inspect in the game.
4. Change to the console and enter single\_player\_mode = 1
5. Turn your game console on and win the game

There’s a lot of detail here because many people had difficulty and it is not easy to explain. Two things are critical. If the iframe or browser is reloaded/refreshed, the variable change in console goes away and the cookie stays put. The iframe (Holiday Hero game) and main (Kringlecon) sites have different contexts, and the console change will only work if you are in the iframe context.

### Step 3 answer

The cookie is already set to true, so now reload the frame.

My Chrome once had “Reload Frame” here, but I seem to have lost it. I hit F5/refresh on the site instead.  
Graphical user interface, application

Description automatically generated

I did not lose Reload Frame in Firefox, though.

Graphical user interface, text, application, chat or text message

Description automatically generated

In both browsers, click Inspect with the cursor inside the game iframe.

Graphical user interface, text, application

Description automatically generated

Select the Console tab, enter single\_player\_mode = 1, hit Enter. This works in Chrome or Firefox. In Firefox you also need to click run.  
Graphical user interface, text, application

Description automatically generated

If all is aligned, you should see the computer join as Player 2, and the right-hand console should turn on.  
A screenshot of a video game

Description automatically generated

Turn on the left-hand console and play! You need to get over 80% and your score goes down every time you miss.

Graphical user interface, website

Description automatically generated

## Hints after solving the Holiday Hero

Chimney has three hints you can use in the Shellcode primer.

|  |  |
| --- | --- |
| A green screen with white text  Description automatically generated with medium confidence  A green screen with white text  Description automatically generated with medium confidence  Text  Description automatically generated | Text  Description automatically generated |

I am sure you especially like the hint that says, “Read the directions!”

## Objective 6: Shellcode primer

The goal of this objective is to familiarize you with machine language. People who reverse engineer malware or create exploits need to understand assembly language.

Graphical user interface, text, application, timeline

Description automatically generated

Since the questions are on the web site, we will just give you the answers and explanations.

### Step 1 Introduction

The idea here is to play with the debugger. Instead of writing code just click Execute to see the code in the debugger. Select different commands on the left side and watch the before and after values change to get a feel for what’s going on. Pay careful attention to how the push and pop commands change the stack and the register. We will not use push in the Shellcode Primer, but pop will be important.

Graphical user interface, text, application

Description automatically generated

### Step 2 Loops

For this step also, just click execute and play with the debugger. The important concept for the next steps you need is how labels work. The label is a pointer to a location in the code and is defined with labelname: (the colon is essential.) When a label is used, in jnz or call for example, there is no colon (call labelname).

### Step 3 Getting Started

This one is simple.

|  |  |
| --- | --- |
| Graphical user interface, text  Description automatically generated | Graphical user interface, text, application  Description automatically generated |

### Step 4 Returning a Value

Note that mov rax, 1337 puts 1337 into the rax register and not the other way around.

|  |  |
| --- | --- |
| Text  Description automatically generated | Graphical user interface, text  Description automatically generated with medium confidence |

Text

Description automatically generated

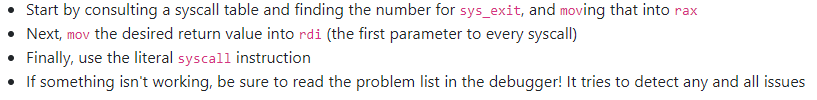
### Step 5 System Calls

System calls are different from what you are used to in higher level languages. In assembly language you load the arguments into registers, including the system call number for the function you want to execute.

A screenshot of a computer screen

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| Graphical user interface, text, application  Description automatically generated | Graphical user interface, text, application, email  Description automatically generated |



### Step 6 Calling Into the Void

When assembly language enters a function, it needs to know where it came from so it can return when the function is complete. Before it enters, it puts the current value of the instruction pointer on the stack with a push. When the function exits, it pops that value and returns execution to the place that called the function. It is important that a function leaves the return value on the top of the stack when it completes. Here you crash the program by leaving a bogus value where the return value should be.

|  |  |
| --- | --- |
| Graphical user interface, text, application, chat or text message  Description automatically generated | A screenshot of a computer  Description automatically generated with low confidence |

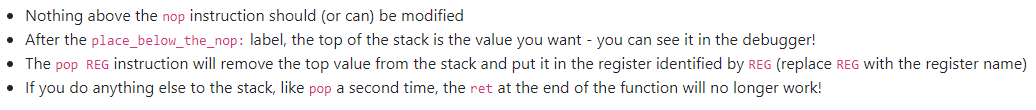


### Step 7 Getting RIP

Read this section entirely and carefully. It talks about how a call places the current value of the instruction pointer (rip, or address in memory currently being executed) on the stack so it knows where to return. A ret takes that value off the stack to return to where the call was.

This is a trick we will use to access a string that we put into memory. We wouldn’t need the trick if we had an assembly language compiler, but we are editing code by hand.

|  |  |
| --- | --- |
| Graphical user interface, text, application, email  Description automatically generated | Text  Description automatically generated |



### Step 8 Hello, World!

Now we will use the trick from step 7 to put a string into memory and learn its address. Once we know the address we can write it to the screen, use it to open a file, etc. Again, this is a trick we use because we do not have a compiler to do this work for us. The steps are:

* call labelname ; this puts the current address (rip) on the stack. It is also the address of your string
* db ‘Hello World or whatever your string is’,0
* labelname:
* pop someregister ; this puts the rip with the address of your string into the register you chose

|  |  |
| --- | --- |
| Graphical user interface, text, application  Description automatically generated | A screenshot of a computer  Description automatically generated with medium confidence |



### Step 9 Hello, World!!

This time you will write “Hello, World!!” to the screen. This step adds the concept of file descriptors. When the computer reads or writes, it needs to know where to read or write so you give it a file descriptor. The first three are 0 for stdin, 1 for stdout, and 2 for stderr (stdin is usually the keyboard and stdout the monitor.) When sys\_open opens a file it assigns a file descriptor to that file, a number that is 3 or higher. See <https://en.wikipedia.org/wiki/File_descriptor> .

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated with medium confidence Graphical user interface, text, application, email  Description automatically generated | Graphical user interface, text, application, email  Description automatically generated |

Text

Description automatically generated

### Step 10 Opening a File

Here you do the call, db, label:, pop rdi trick to put the address of the string containing the file path into the rdi register to tell sys\_read where to look.

The file descriptor or file handle will be an integer >= 3, since 0, 1, and 2 are already taken. In my case, it shows “Process exited cleanly with exit code 3” so the descriptor is 3.

|  |  |
| --- | --- |
| Graphical user interface, text, application  Description automatically generatedGraphical user interface, text  Description automatically generated | Graphical user interface, text, application, email  Description automatically generated |



### Step 11 Reading a File

This is the graduation exercise. Take the code for opening a file and add code to read the file and write it to standard out.

Note: if you add to the code in sections, you may get this error.  


If you do, fill in the exit code at the end before working on the rest of the code. That will avoid the error.

; TODO: Call sys\_exit

mov rax, 60

mov rdi, 0

syscall

|  |  |
| --- | --- |
| Text  Description automatically generated Graphical user interface, text, application, email  Description automatically generated | Graphical user interface, text, application  Description automatically generated  Graphical user interface, text, application  Description automatically generated |
| Graphical user interface, text, application, email  Description automatically generated |

## Hints after solving the Shellcode Primer

Ruby has several hints that will be useful for the next objective.

|  |  |
| --- | --- |
| A picture containing text  Description automatically generated  A green screen with white text  Description automatically generated with medium confidence | Text  Description automatically generated |
| Text  Description automatically generated with low confidence  <https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks> | Text  Description automatically generated with low confidence |