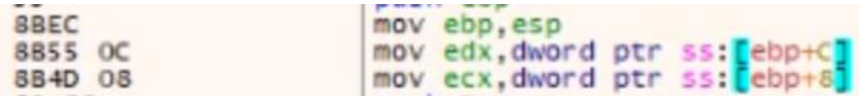
CSE410/510
Lab3

Lab3
Issued: June 17th, 2021
Due: June 14th, 2020 11:59PM

Name: Kamalesh Ram Chandran Govindaraj

Exercise 1
This question relates to the opcode presentation from class. Here are three opcodes with their representative assembly code from one of the malware samples. In the space below, explain what each line of assembly code instruction is doing. Note, 'ss:' simply indicates it is a stack pointer :



Score: / 2 pts

	Explanation of What it is Doing
Line 1	ESP is the current stack pointer. EBP is the base pointer for the current stack frame. So moving ESP to EBP, EBP now points to the top of your stack, and ESP will point to the next available byte on the stack.
Line 2	Ebp+c has the second argument of the function. Using the assembly code, we are moving the second argument to the edx register.
Line 3	Ebp+8 has the first argument of the function. Using the assembly code, we are moving the first argument to the ecx register.

Exercise 2
In this exercise we will modify the assembly code to alter program execution. You **must** use the Kali VM for this exercise. It simply won't work any other way. It will be more difficult than the other problems. Keep an open mind - there are different ways to come up with a solution. Here is what you need to do.

First enable NAT access for the Kali VM. You will need to install a library and that requires access to the Internet. Don't worry, you won't be running any malware. Next, start Kali and login. Once logged in, open a terminal window and issue the following commands:

```
> sudo apt update
> sudo apt install libc6-dev:i386
```

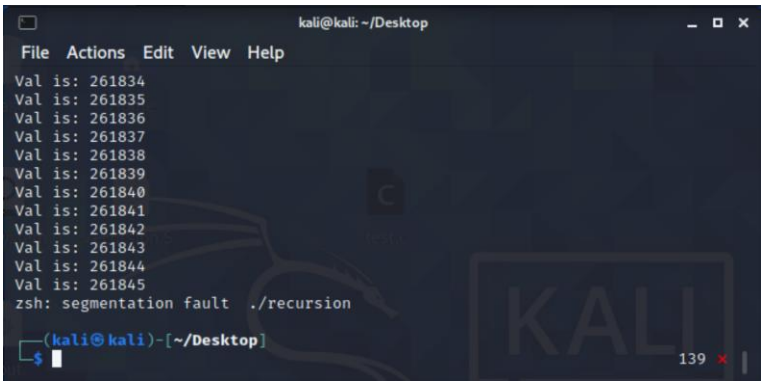
This will allow you to compile C code as 32 bit.

Next, download these two files from Piazza:

recursion.s
recursion

One ends with the .s extension and is assembly code - the other is an executable. The assembly code and executable are a program with recursion. And the recursion is not very good! Copy the files to the Kali machine's Desktop. Run the executable file on Kali like the following (you may need to `chmod a+x` on the file to get it to run):

It will segment fault like the following:



You are to do the following:

1. Edit the assembly file and add an `if` conditional. The method `recursion` takes in an `int` argument and increments it before calling itself. Your `if` conditional should examine this `int` argument after it is incremented but before it calls itself and, if its value is 200, it should exit the method immediately, returning a value of 0.
2. To build an executable from assembly, use this command:

```
> gcc -m32 recursion.s -o recursion
```

Again, think about this one and different ways to tackle it. Also, make a copy of the assembly file **before** you edit it. You will want to do that for certain as you might make a mistake! Note, later in the course you will modify larger programs. This is a dry run - though the tools we use will make life a lot easier.

Score: / 25 pts

```
        .file   "recursion.c"
        .text
        .section.rodata
.LC0:
        .string "Val is: %d\n"
        .text
        .globl  recursion
        .type   recursion, @function
recursion:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        pushl   %ebx
        subl    $4, %esp
        .cfi_offset 3, -12
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        cmpl    $200, 8(%ebp)
        jne     .L2
        movl    $0,%eax
        jmp     .L1
.L2:
        subl    $8, %esp
        pushl   8(%ebp)
        leal    .LC0@GOTOFF(%eax), %edx
        pushl   %edx
        movl    %eax, %ebx
        call    printf@PLT
        addl    $16, %esp
        movl    8(%ebp),%eax
        addl    $1,%eax
        subl    $12, %esp
        pushl   %eax
        call    recursion
        addl    $16, %esp
.L1:
        movl    -4(%ebp), %ebx
        leave
        .cfi_restore 5
        .cfi_restore 3
        .cfi_def_cfa 4, 4
        ret
```

```
        .cfi_endproc
.LFE0:
        .size   recursion, .-recursion
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        leal    4(%esp), %ecx
        .cfi_def_cfa 1, 0
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        .cfi_escape 0x10,0x5,0x2,0x75,0
        movl    %esp, %ebp
        pushl   %ecx
        .cfi_escape 0xf,0x3,0x75,0x7c,0x6
        subl    $4, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        subl    $12, %esp
        pushl   $0
        call    recursion
        addl    $16, %esp
        movl    $0, %eax
        movl    -4(%ebp), %ecx
        .cfi_def_cfa 1, 0
        leave
        .cfi_restore 5
        leal    -4(%ecx), %esp
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE1:
        .size   main, .-main
        .section.text.__x86.get_pc_thunk.ax,"axG",@progbits,__x86.get_pc_thunk.ax,comdat
        .globl  __x86.get_pc_thunk.ax
        .hidden__x86.get_pc_thunk.ax
        .type   __x86.get_pc_thunk.ax, @function
__x86.get_pc_thunk.ax:
.LFB2:
        .cfi_startproc
        movl    (%esp), %eax
        ret
        .cfi_endproc
.LFE2:
        .ident  "GCC: (Debian 10.2.1-6) 10.2.1 20210110"
        .section.note.GNU-stack,"",@progbits
```

Exercise 3

Look at the following assembly code. Fill in the missing instructions in assembly. The red lines let you know where to input the answer:

Score: / 10 pts

Code	Assembly

`int two(int c){`
`  if(c < 0)`
`  return 1;`
`  else`
`  return 0;`
`}`
 	
`int one(int a){`
`if(two(a)){`
`  return -a;`
`}`
`else{`
`  return a;`
`}`
	two:
	
	pushl %ebp
	movl %esp, %ebp
	movl 8(%ebp), %eax
	cmpl $0, 8(%ebp)
	jns .L2
	movl $1, %eax
	jmp .L3
	
	.L2:
	movl $0, %eax
	
	.L3:
	popl %ebp
	ret
	
	one:
	pushl %ebp
	movl %esp, %ebp
	pushl %ebx
	movl 8(%ebp), %ebx
	pushl 8(%ebp)
	call two
	movl %ebx, %edx
	je .L5
	movl 8(%ebp), %eax
	negl %eax
	je .L6
	negl %edx
	
	.L5:
	movl 8(%ebp), %eax
	
	.L6:
	movl %edx, %eax
	addl $4, %esp
	popl %ebx
	popl %ebp
	ret

CSE410/510
Lab3

October 7th, 2020

Exercise 4
This exercise will be an introduction to IDA Pro and review some of the features touched on in class on Thursday. Use the clean FlareVM you exported at the beginning of Lab 2. Set it to Host-only Adapter. You will not run malware but let's be safe. Download this file from Piazza.

Homework3.zip

The password for this file is 'infected' without the quotes. Inside the zip file is the following Windows EXE:

0003a317ff8c4c6ed4531f5cc3bdde7a8f54b7f978a2308361a2b09efa096dc4

This is malware first detected in May of 2020, so it is fairly recent. You may search VirusTotal for the hash to find out more information.

This part of the exercise will get you familiar with program execution tracing to find the trail of function calls related to the use of a particular string. IDA Pro is on the desktop. The icon looks like this:

Open IDA Pro Educational. Next, open the file from the zip using IDA Pro. So, you will need to choose a 'New' analysis and, as the file does not end in .EXE, you will need to select All Files when you go to select the malware sample. Select it and load it in, it will say it 'crashed' but it grabbed enough content to be useful. Open the 'Strings window' and find the following string, '\\mssrv.exe", shown below in the pic to avoid confusion as it is defined a couple of times. You want this exact instance **with the leading Windows file separator**:

.rdata:0040...	00000005	C	*.*
.rdata:0040...	0000000B	C	\\mssrv.exe
.rdata:00411...	0000001E	C	\v\v\n\n\t\t\t\t\

Answer the following questions:

1. Search the internet for "mssrv.exe". Does this look like a safe, Windows standard, .exe? Answer yes/no and also include a paragraph description of why you answered yes/no.

Score: / 5 pts
No, Mssrv.exe is a Trojan PWSteal.Drorar. It monitors user Internet activity and private information. It sends stolen data to a hacker site. This has been identified as a program that

is undesirable to have running on your computer. This consists of programs that are misleading, harmful, or undesirable.

2. In class we learned that IDA lets you use cross references and xrefs to relate addresses together to trace the application's control flow. Using these techniques, you will need to trace where "\\mssrv.exe" is utilized. That is, list the name of the function, like "sub_401530", where the string is used.

Score: / 5 pts

sub_401340 in line (push offset aMssrvExe_0;"\\mssrv.exe")

3. The function you have identified in question 2 lists the directory where the file "\\mssrv.exe" is written. Please supply the *logical* Windows name (like "Program File") for the directory and *how* you determined that value. For the *how* part include a couple sentence description. Note, the location where the file is to be written is created in this function but is then passed to another function to do the actual write.

Score: / 5 pts

The function in question 2 calls the function sub_401200. This has a subkey with a path "Software\\Microsoft\\Windows\\CurrentVersion\Run". Logical windows name of the directory is Software because it is the root of the path. Software is the directory where the file "\\mssrv.exe" is written.

4. This function you identified in question 2 calls another function that writes "mssrv.exe" to disk. What is the name of that function?

Score: / 5 pts

sub_401D90 in line (call sub_401D90)

5. The next two questions will be hard. They will require you to both trace the program and to consult the Microsoft documentation to understand DLL imported functions. The function called in question 4 has two CreateFileA calls from Kernel32 DLL. What file does the first CreateFileA invocation (the first one from the top down of the function) open and what file does the second CreateFileA invocation call open? I am looking for a paragraph description - feel free to include pics if you would like. Note, you will need to trace backward and identify where arguments to functions are initially populated and what value they have. Ok, here is one hint, this will help at one point where you look at the assembly in the context of the Microsoft documentation: https://stackoverflow.com/questions/20331517/setting-a-value-to-null-in-assembly

Score: / 10 pts

CreateFileA function creates or opens an existingfile. The function sub_401D90 has two CreateFileA functions. In the first invocation it has these arguments with respective values. hTemplateFile-0, dwFlagsAndAttributes-0, dwCreationDisposition-3, lpSecurityAttributes-0,

dwShareMode-1, dwDesiredAccess-80000000h. The first argument of the function has the file name. If the file doesn't exist, the function creates the file with all the argument value. The second invocation it takes these arguments with respective value. hTemplateFile-0, dwFlagsAndAttributes-80h, dwCreationDisposition-2, lpSecurityAttributes-0, dwShareMode-1, dwDesiredAccess-0C0000000h. This opens the file created by the first invocation and writes the mssrv.exe is written in it.

6. Based on your answer to question 5 - where do you believe the content for mssrv.exe was obtained? That is, the file was dropped by this malware but where did it get the content to write to it? Answer where you believe the content came from with a paragraph description of why.

Score: / 5 pts

The sub routine sub_401D90 call the CreateFileMapping and CreateFileMapping functions. The CreateFileMapping function returns a handle to the file mapping object. This handle will be used when creating a file view so that you can access the shared memory. Processes calling CreateFileMapping for an existing object receive a handle to the existing object. You can imagine MapViewOfFile as a malloc+memcpy of the file you are opening, nothing more. So MapViewOfFile normally just chooses an address where it can fit the file view's bytes continuously in memory. This is how mssrv.exe is getting the content to write