Assignment 11

1. **Recall that a canary is a special value that is pushed onto the stack after the return address.**
   1. How is a canary used to prevent stack smashing attacks?

Canary is a value that can either be a constant 0x000aff0d or a value depending on ret. The canary is located between the buffer and the ret, therefore when the buffer is overloaded it must overload the canary before overloading the ret and therefore we can determine if the buffer is overloaded by checking the value of the canary and if the canary is different than expected, then you have determined that the buffer have been overflowed and should terminate the program.

* 1. How was Microsoft's implementation of this technique, the /GS compiler option, flawed?

When the Canary dies on Microsoft’s implementation, the program runs the user supplied handler. But the user supplied handler can be subject to attack and the attacker can specify the handler code. Therefore this technique instead of making it safer, became a flaw for easier attacks on the program.

1. **As discussed in the text, the C function strcpy is unsafe. The C function strncpy is a safer version of strcpy. Why is strncpy safer but not safe?**

The function strncopy is “safer” because the “n” stand for the size to copy and this number is specified by the programmer, which could be a problem if the programmer specify a bigger value than the targeted buffer. Therefore this function is not completely safe.

1. **Suppose that the NX bit method of protecting against buffer overflow attacks is employed.**
   1. Will the buffer overflow illustrated in Figure 11.5 succeed?

Yes, the buffer overflow illustrated in figure 11.5 will succeed because Trudy is simply overflowing the buffer and overloading the ret value, therefore it will work since we are not executing anything on the stack.

* 1. Will the attack in Figure 11.6 succeed?

No, this attack will not succeed because the NX bit method protects the program from executing code from the no-execute stack, therefore this method will fail

* 1. Why will the return-to-libc buffer overflow example discussed in Section 11.2.1.2 succeed?

The example discussed on section 11.2.1.2 will succeed because Trudy is not trying to execute her own code, she is just modifying the ret value to execute code that is already in the program. In this example Trudy is not injecting any code, therefore the NX bit method will be no help.

1. **In addition to stack-based buffer overflow attacks (i.e., smashing the stack), heap overflows can also be exploited. Consider the following C code, which illustrates a heap overflow.**

**int main()**

**{**

**int diff, size= 8 ;**

**char \*bufl , \*buf2;**

**bufl = (char \*)malloc(size);**

**buf2 = (char \*)malloc(size);**

**diff = buf2 - bufl;**

**memset(buf2, '2', size);**

**printf("BEFORE: buf2 = %s", buf2);**

**memset(bufl, '1', diff + 3);**

**printf("AFTER: buf2 = /.s ", buf2);**

**}**

* 1. Compile and execute this program. What is printed?

BEFORE: buf2 = 22222222 AFTER: buf2 = 11122222

* 1. Explain the results you obtained in part a.

We first set buf2 to all values of ‘2’ with memset, and then we use memset to populate ‘1’s into bufl, but when we populated bufl we speficied a bigger size than bufl, therefore the ‘1’s overflowed into buf2.

* 1. Explain how a heap overflow might be exploited by Trudy

A heap overflow can be exploited by Trudy by overwriting the program’s memory with her own data and thus change the way the program works.

1. **Recall that a trojan horse is a program that has unexpected functionality.**
   1. Write your own trojan horse, where the unexpected functionality is completely harmless.

I wrote a c program that prints out “System32 deleted”, and compiled it into exe with custom icon. The exe is then renamed into mp3 and with extension hidden it looks like an mp3 file.   
Code:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main(){

printf("System32 deleted\n");

system("pause");

}

PS: after compiling it we can name it something like homework.doc.exe. FILES WERE NOT INCLUDED BECAUSE GMAIL AND HOTMAIL DETECT THEM AS VIRUS.

* 1. How could your trojan program be modified to do something malicious?

The Trojan program can be modified to actually delete files in system32 and also do a variety of changes on windows to corrupt the whole computer.

1. **It has been suggested that from the perspective of signature detection, malware now far outnumbers goodware. That is, the number of signatures required to detect malicious programs exceeds the number of legitimate programs.**
   1. Is it plausible that there could be more malware than legitimate programs? Why or why not?

Yes it is plausive, because legitimate programs are often developed with an idea to earn a profit, while malware does not need an idea, malware is simply developed with the purpose of doing bad things, stealing, vandalizing, etc.

* 1. Assuming there is more malware than goodware, design an improved signature-based detection system.

Since there are more malware than goodware, it is viable to do the signature-based detection system the other way around. Instead of comparing signature of a software with the malware database, we could develop a goodware database and compare signature with that database, therefore if the signature is not found on the goodware database, it can be considered as a possible malware.

1. **After infecting a system, some viruses take steps to cleanse the system of any (other) malware. That is, they remove any malware that has previously infected the system, apply security patches, update signature files, etc.**
2. Why would it be in a virus writer's interest to protect a system from other malware?

The main purpose of a malware is to be able run on a victim’s computer without the victim knowing. On a well written malware, the writer would want their malware to be in a stealth mode so that the victim doesn’t notice, but if the computer contains other malware, it is possible that other malware could make the victim notice that the computer is infected and thus wipe the computer. But in the other hand, if the writer develops a malware that cleanse the computer from other malware, the writer would guarantee that nothing would make the victim wipe the computer unless he find out about the malware.

1. Discuss some possible defenses against malware that includes such anti-malware provisions.

To protect a computer of such malware, we would need to have antivirus to detect malware and also as an extra layer of protection, we could have a fake malware on the computer, since the malware cleanses the computer first, we would know that it is a malware since it would delete the fake malware.

1. **Consider the code in Table 11.5, which is susceptible to a linearization attack. Suppose that we modify the program as follows:**

**int main(int argc, const char \*argv[] )**

**{**

**int i;**

**int count = 0;**

**char serial[9]="S123N456\n";**

**if(strlen(argv[l]) < 8)**

**{**

**printf("\nError---try again. \n\n");**

**exit (0);**

**}**

**for(i = O; i < 8; ++i)**

**{**

**if (argv [ 1] [i] ! = serial [i] )**

**count = count + 0;**

**else**

**count = count + 1;**

**}**

**if(count == 8)**

**{**

**printf("\nSerial number is correct!\n\n" );**

**}**

**}**

**Note that we never break out of the for loop early, yet we can still determine whether the correct serial number was entered. Is this version of the program immune to a linearization attack? Explain.**

This version is immune to linearization attack because the program never ends early and it does the exact amount of work whether the serial is correct or incorrect, therefore this version is immune to linearization attack.

1. **Modify the code in Table 11.5 so that it is immune to a linearization attack. Note that the resulting program must take exactly the same amount of time to execute for any incorrect input. Hint: Do not use any predefined functions (such as strcmp or strncmp) to compare the input with the correct serial number.**

int main(int argc, const char \*argv[] )

{

int i;

int count = 0;

char serial[9] = "S123N456\n";

if(strlen(argv[1]) < 8)

{

printf("\nError---try again. \n\n");

exit (0);

}

for(i = 0; i < 8; ++i)

{

if (argv [1] [i] != serial [i])

count = count + 0;

else

count = count + 1;

}

if(count == 8)

{

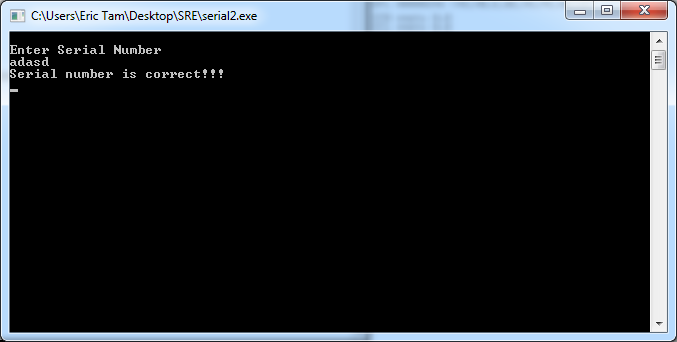
printf("\nSerial number is correct!\n\n" );

}

}

1. **Obtain the file SRE.zip from the textbook website and extract the Windows executable.**
   1. Patch the code so that any serial number results in the message "Serial number is correct!!!" Turn in a screen capture showing your results.

By changing JZ SHORT 00401075 to JMP SHORT 00401075 we were able to make the program accept any serial number we input.



* 1. Determine the correct serial number.

By checking at which line the register EAX was loaded, we were able to get the serial number “DEADbeef”



1. **Obtain the file unknown.zip from the textbook website and extract the Java class file unknown.class.**
   1. Use CafeBabe[44] to reverse this class file.

Decompiled code:

import java.io.PrintStream;

import java.util.Scanner;

class unknown

{

public static void main(String[] paramArrayOfString)

{

String str2 = "";

Scanner localScanner = new Scanner(System.in);

System.out.println("Enter a string");

String str1 = localScanner.nextLine();

int i = str1.length();

for (int j = i - 1; j >= 0; j--) {

str2 = str2 + str1.charAt(j);

}

if (str1.equals(str2)) {

System.out.println("Yes");

} else {

System.out.println("No");

}

}

}

* 1. Analyze the code to determine what the program does.

The program checks if the string entered is a palindrome.

1. **Obtain the file encrypted. zip from the textbook website and extract the file encrypted .jar. This application was encrypted using SandMark [63], with the "obfuscate" tab and "Class Encryptor" option selected and, possibly, other obfuscation options.**
   1. Generate a decompiled version of this program directly from the obfuscated (and encrypted) code. Hint: Do not attempt to use a cryptanalytic attack to break the encryption. Instead, look for an unencrypted class file. This is a custom class loader that decrypts the encrypted files before they are executed. Reverse this custom class loader and modify it so that it prints out the class files in plain text.

Decompiled EncryptedClassLoader.class, modified the code to output the unencrypted .class files, compiled and ran it (Files located at plaintext folder)

* 1. How could you make this encryption scheme more difficult to break?

Encryption can be made more difficult to break by making a second layer of encryption, by encrypting EncryptedClassLoader with a different key and using another class file that is highly obfuscated to run the encrypted – EncryptedClassLoader.

1. **Obtain the file mystery.zip from the textbook website and extract the Windows executable mystery.exe.**
   1. What is the output when you run the program with each of the following usernames, assuming an incorrect serial number in each case?
2. mark – User name must be at least 5 characters
3. markstamp – Error! Incorrect serial number (or username). Try again.
4. markkram – Error! Incorrect serial number. Try again.
   1. Analyze the code to determine all restrictions, if any, on valid usernames. You will need to disassemble and/or debug the code.

By analyzing the assembly code, we were able to find out that usernames has to be at least 5 characters in length and they must be a palindrome.

* 1. This program uses an anti-debugging technique, namely, the Windows system function IsDebuggerPresent (). Analyze the code to determine what the program does in case a debugger is detected. Why is this better than simply terminating the program?

When the program knows that the debugger is present, it will not run any operations.

* 1. Patch the program so that you can debug it. That is, you need to nullify the effect of IsDebuggerPresent().

We can nullify the effect of IsDebuggerPresent() by changing it to NOP

* 1. By debugging t he code, determine the corresponding valid serial number for each valid username that appears in part a. Hint: Debug the program and enter a username along with any serial number. At some point the program v.ill compute the valid serial number corresponding to the entered username-it does this so that it can compare to the entered serial number. If you set a breakpoint at the correct location, the valid serial number will be stored in a register, which you can then observe.
  2. Create a patched version of the code, mysteryPatch. exe that accepts any username/ serial number pair.

1. **Recall that an opaque predicate is a "conditional" that is actually not a conditional. That is, the conditional always evaluates to the same result, but it is not obvious that this is the case.**
   1. Why is an opaque predicate a useful defense against reverse engineering attacks?

Opaque predicate is useful against reverse engineering because it makes the code longer and harder to understand and therefore increasing the work of Trudy to be able to reverse engineer the program.

* 1. Give an example-different from that given in the text- of an opaque predicate based on a mathematical identity.

int x,y,z;  
…  
if( x + y + z == 0 && x\*y\*z != 0){  
….  
}

* 1. Give an example of an opaque predicate based on an input string.

String temp = in.nextLine();  
String a = temp;  
temp = “”;  
String b = a;  
a = “hi”;  
if(temp.equals(a) && temp.equals(b)){  
…  
}

References:

Worked with nirav, Jeffrey and Jay