Intro. to Computer Security: Exercise 5

Deadline: 12:00 noon on Friday the 1st of April

Application Security

This exercise looks at reverse engineering and "cracking" applications. N.B. for this exercise you will need to download a new version of the VM from:

`https://www.cs.bham.ac.uk/internal/courses/intro-comp-sec/ICS_VM2.ova`

For this exercise you should log into the new VM as user: `dan`, password: `dan!dan`. In Dan's home directory you will find two jar files, three ELF executables, the tool JD-GUI, the evaluation version of IDA pro and the example files I have demonstrated in lectures. For this exercise you need to use JD-GUI (in the tools directory) and IDA to analyse the applications and gain a complete understanding of what they do.

Part 1: Java Byte Code

The jar files employ two of the most common methods of protecting code: encryption and obfuscation. As you will see neither of these methods will stop a determined analyst.

The first jar file encrypts some of its code, however, the decryption key must be embedded in the application, and so an analyst can read the code. This is an example of "packing" which is a protection method often use by malware. This is done mainly to avoid signature based detection from malware scanners; the malware will re-encrypt itself with a different key each time it infects a computer, so making it look different each time it spreads.

The second jar file has been obfuscated, i.e., made deliberately hard to understand. This method of protection is often used by drive-by-download attack code which has been injected into a website. The point of this is to make it difficult for a casual observer to tell what the code is doing and so delaying the time it takes for anyone to realise that the code is malicious.

Question 1: exercise1.jar [4 marks]

The first Java application is a simple password check (you can run the password check jar file by typing `java -jar exercise1.jar` at the command line). Use JD-GUI to find the password for this program, describe in detail how the jar file tries to protect the password and how you found the password, explain each of the steps you took. Another version of this application (with the same password) is also running on one of the ports of the VM. Use `nmap` to find out which ports are open, investigate them and use netcat (`nc`) to connect and find the application. The version of the application listening on the port will give you a token in response to the correct password. Submit this token on the website.

Question 2: exercise2.jar [6 marks]

The second Java application opens a dialog box that asks for a registration key. Find a registration key that this application will accept. Describe in detail how the jar file tries to protect the password and how you found the password (there is no token for this application).

Part 2: ELF Binaries

Executable and Linkable Format (ELF) is the standard format for linux executables. The two ELF executables in Dan's home directory can be run from the command line by typing `./exercise-03` and `./exercise-04`. The first is a simple password check program and the second is a more complex application for viewing GPG keys.

Question 3: exercise-03 [4 marks]

The application `exercise-03` asks you to enter a password in order to be given a message. Open this application in IDA by typing `./tools/idademo64/idaq exercise-03`, examine the assembly code and run it in the IDA debugger. Work out how the password is being checked and what the message is. Describe in detail how the application checks the password and how you discovered this. In particular, describe the steps you went through and why. Another version of this application (with the same password) is also running on one of the ports of the VM (N.B. you may need to restart the VM at least once for this service to appear). Use `nmap` to find out which ports are open and netcat (`nc`) to connect to them and find the application. The version of the application listening on the port will give you a token in response to the correct password. Submit this token on the website.

Question 4: exercise-04 [6 marks]

The application `exercise-04` is a larger program to display information about public keys. This application contains a back door that can be used to get a shell. Open this application in IDA, examine it, and find the back door. This application is also running and listening on one of the ports of the VM as root. Use `nmap` to find out which port it is running on, connect to it using netcat and exploit the backdoor to get root access to the VM (N.B. you may need to restart the VM at least once for this service to appear). Describe in detail how the backdoor works and how you discovered it. In particular, describe the steps you went through and why. Once you have root access to the VM you will find a final token in the `Ex5rootToken` file, submit this to the token submission website.

Optional Extra Credit: Buffer Overflows

Marks from this question will be added to your total course work percentage for all exercises, but it cannot take your total coursework mark above 100%. The file `nameCheck` (in dan's home directory) was compiled using the command: `gcc -fno-stack-protector -z execstack -o nameCheck nameCheck.c` and the following code:

```
#include<stdio.h>

void function1(void) {
    printf("Enter your name:\n");
    char buffer[64];
    gets(buffer);
    printf("No token for you %s!\n",buffer);
    printf("By the way buffer was at: %p",buffer);
}
void function2(void) {
    // Open file
    FILE *fptr;
    fptr = fopen("overflowToken1", "r");
    // Read contents from file
    char c = fgetc(fptr);
```

```
    while (c != EOF)
    {
        printf ("%c", c);
        c = fgetc(fptr);
    }
    fflush(stdout);
    fclose(fptr);
}
int main(void) {
    function1();
    return 0;
}
```

i.e. all the memory protections have been turned off. Additionally, ASLR has been disabled on the VM.

Question A [6 marks]

Find a buffer overflow attack against the above code that will let you read the contents of the file `overflowToken1`. N.B. the program executes with the `sam` permissions and the tokens are only readable by Sam. For this question not try to execute code on the stack. Submit the token to the website and add a written description how your attack work, and how you discovered it. As well as the description, include in your answer the address on the stack of the return address for from function1, the address of function2, and what you injected to perform the attack. You may find it helpful to sketch the laptop of the stack before and after your attack.

Question B [6 marks]

Find a second buffer overflow attack against this program that will let you read the token `overflowToken2`. Submit the token to the website and add a written description how your attack work, and how you discovered it. You may find it helpful to sketch the laptop of the stack before and after your attack.

Hints

- You should work through each the examples I did in lectures which you can find in the `examples` directory i n the dan directory. The README.txt file has details of the attacks.
- When you execute a program in IDA it always executes with your permissions, therefore you cannot access the tokens when running the program in IDA.
- A program can have different stack offsets when executed in IDA vs when it is executed normally.
- The commands `while read -r line; do echo -e $line; done | ./nameCheck` and `while read -r line; do echo -e $line; done | ./idaq nameCheck` will let you run a program and enter bytes using hex notation, e.g. using these commands in the input \x00 will enter a byte of 0s into memory, instead of the ascii characters for "\" "x" "0" and "0" as you would normally get.
- The command `cat` can be more reliable than `more` for viewing files, as `more` will try to display exactly one page and may fail if it can't work out what a page is.
- You can find the shell code I used in lectures at: `https://www.cs.bham.ac.uk/internal/courses/comp-sec/shell.txt`