Practical Cyber Security Fundamentals Final Project

Joshua Bittel, Logan Anderson, Nicholas Meier

April 4, 2019

Problem 1

1 Introduction

Title: Jordan Loves Orange Juice

Category: Forensics

Themes: Forensics toolkits, encodings, context clues

Difficulty: Intended difficulty of this problem is a low-medium level difficulty.

Objectives: Areas in which students will improve upon by completing this challenge:

- Ability to use common forensic toolkits
- Ability to use context clues provided to determine what direction to work towards in finding the flag

2 Overview

2.1 Explanation

- Using your knowledge of forensics, analyze this picture of Jordan to find the file and properly decode the flag. Good luck.

2.2 Prerequisite Knowledge

- Knowledge of common forensics tools
- Knowledge of common (and uncommon) encodings

3 Details

3.1 Implementation

Provided is a beautiful picture of Jordan in his glory days. Using Openstego, a file was hidden within the picture. Inside this file is a large amount of text, which is the result of encoding the flag three times using a Citrix CTX1 encoding. Once decoded, though, one will notice that the resulting string is not in English. A picture of a hand, with an Italian caption, is provided to clue individuals into the fact that the flag needs to be translated from Italian to English. Lastly though, the final word of the flag is in Hebrew. Given that the title includes Jordan's name, individuals should be able to make this connection. Once the final translation is done, the individual will be left with the flag.

3.2 Methodology

Primary steps to solving this challenge:
Step 1: Open the provided image in Openstego.
Step 2: Retrieve the hidden .txt file from within the image.
Step 3: Used the provided clues to determine which encoding is used on the flag.
Step 4: Decode the flag using the Citrix CTX1 decoder 3 times.
Step 5: Translate the resulting flag from Italian to English.
Step 6: Translate the final word of the flag from Hebrew to English.
Step 7: All done.

4 Suggested Rubric

An example rubric is provided below:

Quality of write-up	5 pts
Clarification of the task to be completed	5 pts
Successful retrieval of file from image	5 pts
Successful decoding of file	5 pts
Successful translation of flag	5 pts
Successfully identifying the flag	5 pts

Problem 2

1 Introduction

Title: Can you Break AES?

Category: Binary Exploitation/Reversing

Themes: Static Analysis, Dynamic Analysis

Difficulty: Intended difficulty is medium

Objectives: Areas in which students will improve upon by completing this challenge:

- Ability to dynamically analyze binaries
- Ability to identify vulnerabilities and reverse engineer functions in a binary.

2 Overview

2.1 Explanation

- Analyze the provided binary to extract the flag encrypted with AES-256-CBC.

2.2 Prerequisite Knowledge

- Knowledge of reverse engineering
- Knowledge of Binary Exploitation
- Knowledge of dynamic analysis

3 Details

3.1 Implementation

Flag: flag{wA$_T4is_K1nd@_eaSy?}

Analysis:

To start, running file on the two files given, flag and prob2.x gives the following output:

```
bittel@pop-os:/mnt/hgfs/Google Drive/Documents/College/Capture the Flag/ctf2019_problems/prob2$ file flag
flag: data
bittel@pop-os:/mnt/hgfs/Google Drive/Documents/College/Capture the Flag/ctf2019_problems/prob2$ file prob2.x
prob2.x: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
 for GNU/Linux 3.2.0, BuildID[sha1]=238ae228116b68ea3db88a569f28a2fa9312cce3, not stripped
bittel@pop-os:/mnt/hgfs/Google Drive/Documents/College/Capture the Flag/ctf2019_problems/prob2$
```

The flag file is listed as just binary data and the prob2.x is an executable elf binary.
Running strings on file gives no useful output and calculating the shannon entropy the flag file returns

```
Shannon entropy: 7.980911713467346
```

Which seems to suggest that the file is encrypted. By referencing the hint, it's obvious that the file is encrypted with aes-cbc 256. Running the prob2.x binary results in

```
bittel@pop-os:/mnt/hgfs/Google Dri
Usage: ./prob2.x <text to print>
```

And giving it a string as an argument results in the program printing out the argument. We can test for a string format vulnerability by giving it '%x.%x.%x.%x.%x.%x.%x.' results in:

```
'4.99625334.99626470.a09f5010.78252e78.997f7180.99625530.
```

Which indicates that there is indeed a vulnerability present. Lets run strings on prob2.x.

```
test message
ctf2019
flag
Usage: %s <text to print>
;*3$"
```

There are a few interesting strings here like 'flag' and 'ctf2019' but nothing that looks like an AES key. Lets change pace a little and debug the program with pwndbg to see what it is doing. The binary looks like it is stripped because pwndbg is not able to set a breakpoint at main and tells the user that there are no symbols found.

```
pwndbg> b main
Function "main" not defined.
pwndbg>
```

However, we know that a printf instruction happens within main so we can set a breakpoint on that function to start executing and debugging within the printf call. Using the finish command, we can finish the call to printf and start debugging in main. We can see that there is a call to a function after the printf function. Examining the registers and stack prior to executing the call instruction reveals that there are two arguments passed in, a suspicious string that looks like it might be some obsfucated key and the ctf2019 string. After entering the function, it looks like it is performing an xor operation on the suspisious string with the key ctf2019 string.

```
   0x5555555551f4    mov     eax, dword ptr [rbp - 0x1c]
   0x5555555551f7    movsxd  rdx, eax
 ▶ 0x5555555551fa    mov     rax, qword ptr [rbp - 8]
   0x5555555551fe    add     rax, rdx
   0x555555555201    xor     esi, ecx
   0x555555555203    mov     edx, esi
   0x555555555205    mov     byte ptr [rax], dl
```

Using finish to execute all the instrution in the function, we can look at the result of this function inside rax and sure enough, it's the decrypted string

```
[ REGISTERS ]
RAX  0x555555559670 ◂— '20E8B88116BDE340D86697561087761639BFD48B3132CF593F6069F7669771F182E9F9012990492EF21F98321A81B295'
```

For aes-cbc 256, there is a 256 bit key so seperating this string into
key = 20E8B88116BDE340D86697561087761639BFD48B3132CF593F6069F7669771F1
and
iv = 82E9F9012990492EF21F98321A81B295
and using *openssl aes-256-cbc -d -K*
20E8B88116BDE340D86697561087761639BFD48B3132CF593F6069F7669771F1 -iv

82E9F9012990492EF21F98321A81B295 -in flag -out flag.out to decrypt the file reveals that after running strings on the new flag.out file returns the flag.

4 Suggested Rubric

An example rubric is provided below:

Quality of write-up	5 pts
Clarification of the task to be completed	5 pts
Successful retrieval of file from image	5 pts
Successful decoding of file	5 pts
Successful translation of flag	5 pts
Successfully identifying the flag	5 pts