FIELD GUIDE

CTF / Sec

Laeberkaes

May 19, 2020

Contents

1 Binary Challenges

1.1 Integer Overflow

For some kind of challenges it is enough to give it negative integers. To understand this we need to know that there are two different types of integers (signed and unsigned).

The only difference is that signed integers can be negative and unsigned not. For example `1101` represents an `13` unsigned integer and an `3` signed integer. So an program is vulnerable to such a integer overflow, if it passes an signed integer if only unsigned are allowed. This is often the case in some easy show challenges in CTF's. Even if the program only accepts a certain size of a integer it can get overflowed by this vulnerability.

For further information look at: `https://d3vnull.com/integer-overflow/` and `https://www.swarthmore.edu/NatSci/echeeve1/Ref/BinaryMath/NumSys.html`.

If the program checks for signed integers, there is another possibility to overflow integers. Most integers in C have the size of a 4 bytes. If you give the programa number that can't be saved in 4 bytes you get an overflow.

1.2 .bin files

For most binary files the first step is to look at them in a hex editor (like *bless*) and to run the `file` command on them to determine what kind of file it is. Some other tools you can run in the recon phase are `ltrace`, `strace`, `objdump`, `radare2`.

After this you should have a clou what you can do with this file. E. g. delete / change some bytes to turn them into an `.png` file.

2 Cryptography Challenges

2.1 XOR / Exclusive or

In these challenges you are usually given some hex values. Xoring is implemented in the most programming languages. To xor in python you can follow this example:

```
1       a = 0xc4115 # first value
2       b = 0x4cf8 # second value
3       print(hex(a ^ b))
```

The output will be `0xc0ded`. The logic behind XOR is the same like in addition of binary integers. For further information look at: `https://en.wikipedia.org/wiki/Exclusive_or`

2.2 Rare Ciphers

2.3 Substitution Cipher

In substitution ciphers the characters of the alphabet get substituted to other characters. One of the easiest is the caesar cipher, where the characters get rotated. Those kind of ciphers can be easily bruteforced with a list of common english words (`https://www.dcode.fr/monoalphabetic-substitution`)

Cipher	Encoded	Decoded
Bacon	BAABABAAB- BAABAAAABBAABBBA	stego

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Table 1: Example of an substitution cipher

2.4 RSA

Small public exponent An often used RSA challenge is the small public exponent challenge. In this challenges e, n and c are given and you have to get m.

The following equation shows how m can be calculated with small public exponents:

$$c = m^e \pmod{n} \tag{1}$$

$$m = \sqrt[e]{c} \pmod{n} \tag{2}$$

So our example will be:

$$e = 1$$

$$c = 932756572276725830865064321334454240459201116165999142 1$$

$$n = 24584123651247885275290973491257558181596763003304983826908 3$$

With these numbers the solution of the equation will be:

$$932756572276725830865064321334454240459201116165999142 1 = m^1 \pmod{n} \tag{1}$$

$$m = \sqrt[1]{932756572276725830865064321334454240459201116165999142 1} \pmod{n} \tag{2}$$

$$m = 932756572276725830865064321334454240459201116165999142 1 \pmod{n} \tag{3}$$

$$m = 932756572276725830865064321334454240459201116165999142 1 \tag{4}$$

After you have solved this equation, you have a long number. This number has to be converted to readable ascii-text. This can be accomplished with the following code in wich the number gets converted to hex. After this it gets cut into pairs, these values converted to ascii values and the values to the ascii letters. The result of all this is the flag: *abctf{b3tter_up_y0ur_e}*.

```
m = 932756572276725830865064321334454240459201116165999142 1 # this is the result
    of the equation
h = hex(m)
hex = 61626374667b6233747465725f75705f793075725f657d
print(''.join([chr(int(''.join(c), 16)) for c in zip(hex[0::2],hex[1::2])]))
```

3 Forensics Challenges

3.1 Wireshark .pcap files

In easy challenges it is enough to follow some streams and keep searching for the flag. Wireshark can find strings in the packet bytes or in its description. Further you can search for strings or hex values.

In other challenges you have to filter for connected ports. For example SSH. An SSH connection starts with `begin` and ends with `end`.

3.2 File headers

If a file cannot be opened and some relevant errors are thrown, you can give wrong file headers a shot. To solve this kind of challenges you have to open the file in a hex editor (`bless`) and have a look at the first values and compare them with expected ones of this file type (you can find those on the internet).

Filetype	Fileheader
.png	`137 80 78 71 13 10 26 10`
.jpg	`FF D8 FF`

Table 2: Examples of fileheaders

4 Miscellaneous Challenges

4.1 Difference between two files

To get the different bytes of two files you can run the command `binwalk -Wiw file1 file2`.

5 Programming Challenges

5.1 Git history

There are several possibilities how challenges can occur in CTF's. One of the most easy ones is the flag which got removed from the repository. Mostly you get a compressed file in which is the `flag.txt`. This file is part of an repository. To see older versions of the file you can run the command `git log -p`. If you are lucky enough now you will get the correct flag.

6 Pwn Challenges

7 Reversing Challenges

7.1 radare2 – disassemble

To disassemble a file with radare2 you start always the same. You first start with analyzing the file with `aaaa`. After this you can have a look at the main function with `pdf @ sym.main` or at the visual mode (entering `V!`) with `s main`.

8 Steganography

8.1 Extracting files from images

One of the easiest steganography challenges is the extraction of not password secured files. There are several commands to accomplish this (`binwalk` or `steghide`). A good toolset is *stego-toolkit* where you can run multiple tools with one command (`check_jpg` or `check_png`).

9 Web Challenges

9.1 SQLi

SQL is a language which is used to get values out of a database. For example the two following lines can be used. In the first line the two rows **username** and **email** in a table called **users** will be returned.

```
1  SELECT username, email from users;
2  INSERT into users (username, email) values ("team@ctflearn.com", "intelagent");
```

The following code will filter the table called **users** for all entries in the row **username** where the value is **word-of-search**.

```
1  SELECT * from users where username = "word-for-search";
```

Now we will start to try exploiting SQL. The following code selects all rows of a table called **users** (`SELECT * from users` ← would return all rows from all columns) and filters all results for usernames. But as condition it has `"␣or␣true` which will return all rows of the table.

```
1  SELECT * from users WHERE username = "" or true -- ";
```

9.2 POST authentication

POST form is always used if it contains personal or sensitive data. This date will not be displayed in the url. POST forms don't have a size limitation.

Easy POST authentications can be accomplished with the Linux command `curl`. The parameter `-d` will define the data wich will be sent in the POST form. The following example will send the username `admin` and the password `71urlkufpsdnlkadsf` to the webpage `http://165.227.106.113/post.php`. In further attacks this can be used for bruteforcing login screens with hydra for example.

```
curl -d username=admin -d password=71urlkufpsdnlkadsf http://165.227.106.113/post.php
```