CTFs: Exploiting for fun and profit

by

Andrea Bellino - Andrea Iuorio

November 26, 2021

Outline

Introduction

What are CTFs?

- A Capture The Flag is a computer security competition
- Participants compete in challenges, gaining points and trying to obtain the highest score
- Each challenge goal consists on finding a specific piece of hidden text, aka *the flag*
- Different types of challanges, requiring different skills
  - web, reverse engineering, binary exploitation, crypto, forensics...

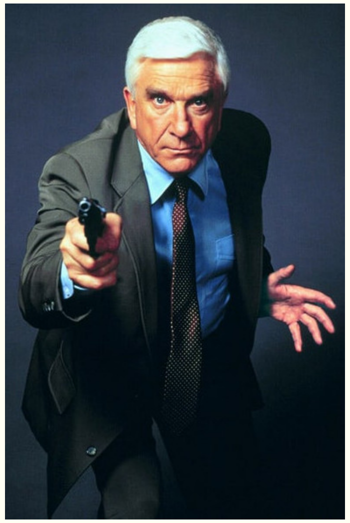
Why CTFs?

- Enhancement of your problem solving and creative thinking
- Challanges always teach you something new
- Acknowledgment of highly required skills
- It's fun!

Spoiler

- There is no secret book or magic bullet to make you good at it
- The only way to improve and be better is to practice (and study)

Ready to play?



Web Exploitation

Web Exploitation

Web applications often serve dynamic content, use databases, and rely on third-party web services

- **SQL Injection**: an application takes input from a user and doesn't check if it doesn't contain additional SQL
- **Cross Site Scripting**: a user can type on input form a JS code to be executed by the target application
- **Authentication & Authorization**: unauthorized resources can be accessed even when UI doesn't expose them
- Demo time

Binary exploitation

Binary exploitation

In this category we have to exploit an application (running on a remote server) for opening a shell.

Usually we have a copy of the binary (or the source code) we will need to exploit.

- A lot of server applications have their binaries available/open source
- It's still possible to search vulnerabilities even if we don't have the binary
  - Security by obscurity is always a bad idea!!

greeting (Tokyowesterns 2016)

Host : pwn2.chal.ctf.westerns.tokyo Port: 16317

Note: To prevent from DoS attacks, output length is limited in 131072 characters.

And we have a copy of the binary

- We identify a vulnerability by analyzing the binary
- We use the vulnerability (exploit) to open a shell

printf

int printf (const char * format, ...)

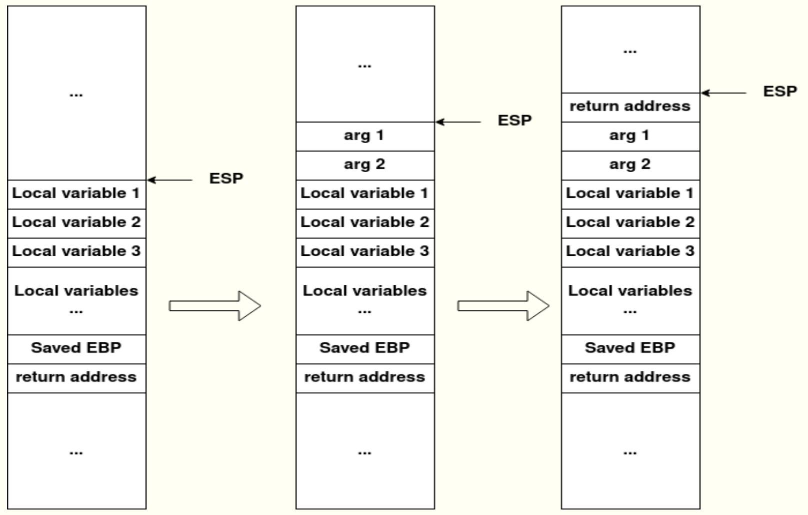
Writes the C string pointed by format to the standard output (stdout). If format includes format specifiers (subsequences beginning with %), the additional arguments following format are formatted and inserted in the resulting string replacing their respective specifiers.

printf

int printf (const char * format, ...)

- %x => Unsigned hexadecimal integer

x86 calling convention

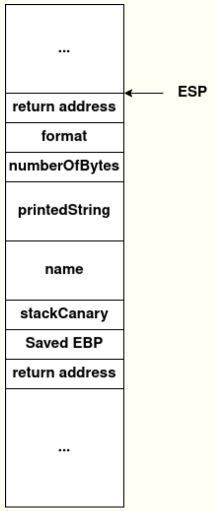


printf

int printf (const char * format, ...)

- %x => Unsigned hexadecimal integer
  - We can use it to read the memory
- %n => Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.

x86 calling convention



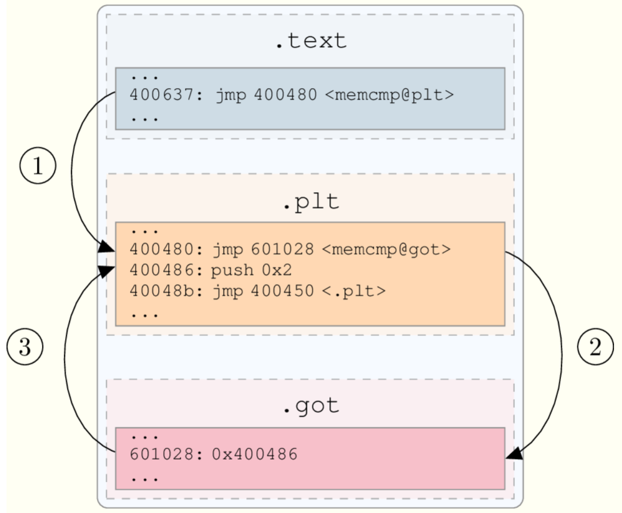
printf

int printf (const char * format, ...)

- %x => Unsigned hexadecimal integer
  - We can use it to read the memory
- %n => Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.
  - We can use it to write the memory

Now, how do we use this vulnerability for open a shell?

Dynamic linking


.text
...
400637: jmp 400480 <memcmp@plt>
...
.plt
...
400480: jmp 601028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
.got
...
601028: 0x400486
...
1
2
3

- Call to PLT
- Jump to address present in GOT
- Default GOT is next address in PLT
- .plt writes in GOT real address for next call
  - Lazy binding

Our exploit

- Using the system C function we can execute any commands => system("/bin/sh") will open a shell
- We need a library function call after the printf
- But there aren’t any!
- Solution => write the pointer of a main instruction address into .fini_array

Exploit to-do list

- Write 0x8048614 into .fini_array (0x08049934)
- Write 0x8048490 (system PLT) into strlen GOT (0x08049a54)
- Send the string "/bin/sh"

- These numbers are pretty big...
- But we can write them two bytes at the time!
- Instead of doing it manually, let's write a simple script.

Lession learned

Never, ever trust the user input!

- Every input represent an occasion for an attacker
- Always check and filter any inputs, never use them as-is
- Use an allowlist approach instead of a blocklist one

Reverse Engineering

Reverse Engineering

- A process of taking a compiled program and converting it back into a more human readable format.
- The goal is to understand its functionality for various purposes..
  - how some piece of closed source software works
  - malware analysis, to identify deeper issues
  - find vulnerabilities and stuff

Forensic

Forensic

Application of investigation and analysis techniques to gather data recovery

- File format identification and metadata
- Analysis of dump from memory, disk or network connection
- Steganography
- Demo time

Conclusions

Conclusions

- You can’t improve the security of your application by hiding stuff
- Even simple mistakes can be exploit by attackers
- CTFs are a great way to learn new stuff and to improve as a software developer
- Have we already say they are fun?

Resources

- Where can I find writeups?
  - https://ctftime.org/writeups
- Which tools should I use?
  - This list contains (probably) all you need ;)
- Where can I practice?
  - PicoCTF
  - Overthewire
  - ...and many other platforms!
- ...any YouTube channel?
  - John Hammond
  - Live Overflow