Breaking Into CTFs

@ikuamike

Agenda

1. Introduction
2. What is a CTF
3. Why play a CTF
4. Types of CTFs
5. CTF Categories
6. Conclusion

@ikuamike

Introduction

Michael Ikua - Lead Penetration Tester, Silensec
- CTF Player @fr334aks

Kelvin Njau - Security Consultant, Silensec
- CTF Player @fr334aks
- @k0imet_

Trevor Saudi - Technical Content Creator, CYBERRANGES
- CTF Player @fr334aks
- @trevorsaudi

@ikuamike

What is a CTF

A Capture The Flag competition is a cybersecurity contest where individuals/teams tackle challenges for points.

In most cases the one with the most points at the end wins, in other cases you get a position on a long standing leaderboard.

Why play a CTF

1. For hands on practice of your cybersecurity skills
2. To get exposed to new areas of interest in cybersecurity
3. For fun, some challenges pose fun puzzles to solve
4. To win prizes/giveaways
5. To network with different professionals/enthusiasts
6. To boost your resume as a means to get a start in the cyber security industry

Type of CTF Competitions

1. Jeopardy
2. Boot2Root
3. Attack and Defend
4. Wargames
5. CyberRanges

@ikuamike

Jeopardy

This is the most common type of CTF competition where you compete in different categories to capture points. Mainly revolves around conference events or meetups.

Platforms:

- Just follow CTFtime (ctftime.org)
- HackTheBox (ctf.hackthebox.com)
- CYBERRANGES (app.cyberranges.com)
- CyberSpaceKenya (ctfroom.com)

Boot2Root

This type is mostly pentesting focused where you have a stand-alone victim machine and you're supposed to perform an attack and gain root privileges to complete the challenge.

Platforms:

- HackTheBox Machines (app.hackthebox.eu/machines)
- Vulnhub Machines (vulnhub.com)
- TryHackMe Rooms (tryhackme.com)
- Offensive Security Proving Grounds
- etc...

@ikuamike

Attack and Defend

You compete by attacking opponents infrastructure and simultaneously defend your own infrastructure.

Platforms:

- HackTheBox Battlegrounds (app.hackthebox.eu/battlegrounds/lobby)
- TryHackMe King of the hill (tryhackme.com/games/koth)
- Some CTF Events

Wargames

These are progressive challenges where you get the key/password to the next challenge by solving the current.

Platforms:

- Overthewire (overthewire.org/wargames/)
- Underthewire (underthewire.tech/wargames)

CyberRanges

These are challenges with near realistic environments with interconnected machines that put a more real world spin to ctfs.

Platforms:

- CYBERRANGES (app.cyberranges.com)
- HackTheBox Pro labs (app.hackthebox.eu/prolabs)
- TryHackme Networks (tryhackme.com/hacktivities)
- Offensive Security OSCP labs

@ikuamike

CTF Categories

1. Web Application Security
2. Mobile Application Security
3. Forensics
4. Steganography
5. Reverse Engineering
6. Binary Exploitation
7. Miscellaneous

@ikuamike

Web Application Security

Introduction

A proper foundational knowledge on the protocols used in web applications as well as the different technology stacks is important when figuring out web application security.

OWASP

Open Web Application Security Project is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted.

@ikuamike

OWASP Top 10

https://www.owasptopten.org/the-release-of-the-owasp-top-10-2021

@ikuamike

Web CTF Challenge

https://ctfroom.com
Vault: HTML & PHP Runs the Web

@ikuamike

HTTP Basics Refresher

HTTP Request Message Format

Method, Resource, HTTP Version
Zero or more header lines
Blank line
Optional Message Body

```
GET / HTTP 1.1
Host: 127.0.0.1
User-Agent: curl/7.68.0
Accept: */*
```

@ikuamike

HTTP Basics Refresher

HTTP Request Methods

GET - Request a resource or information
HEAD - Like GET but only for the header and not the information
POST - Sends information to the server for a certain action
OPTIONS - Asks the server what methods it supports

@ikuamike

HTTP Basics Refresher

HTTP Response Message Format

HTTP Version, Response code
Zero or more header lines
Blank line
Optional Message Body

```
HTTP/1.1 200 OK
Server: Apache/2.4.7
Content-Length: 1150
Content-Type: text/html
```

@ikuamike

HTTP Basics Refresher

HTTP Response Codes

2 (success) - The request was successfully received, understood and accepted.
3 (redirection) - Further action needs to be taken in order to complete the request.
4 (client error) - The request contains bad syntax or cannot be fulfilled.
5 (server error) - The server failed to fulfill an apparently valid request.

Career Relevance

1. Application Security Engineer
2. Bug Bounty Hunter
3. Pentester
4. Web Developer

@ikuamike

Mobile Application Security

Introduction

When it comes to mobile apps we mainly think of two mobile platforms; android and ios.

For today, I'll focus on android.

@ikuamike

Android Application CTF Challenge

https://github.com/sajjadium/ctf-archives/blob/main/HacktivityCon/2021/mobile/To_Do/todo.apk

Tools

apktool

jadx-gui

@ikuamike

Android Application Structure

Android apps come with an extension of .apk which is **A**ndroid **P**ac**K**age. This APK is just a zip file with assets and compiled code. Typically it looks like this when you unzip:

```
myapp.apk
├── AndroidManifest.xml
├── META-INF/
├── classes.dex
├── lib/
├── res/
└── resources.arsc
```

Android Application Structure

AndroidManifest.xml

This is a compressed version of the AndroidManifest.xml file which contains all of the basic application information such as the package name, package version, externally accessibly activities and services, minimum device version, and more. The compressed version of this file is not humanly readable, but there are a couple of tools that are able to decompress it, most notably being **apktool**.

Android Application Structure

META-INF/

The META-INF/ folder is essentially a manifest of metadata information including the developer certificate and checksums for all the files contained within an APK. If you were to try and make changes to an APK without removing and re-signing this folder, you would get an error when installing the modified version.

Android Application Structure

classes.dex

The classes.dex file (sometimes there are multiple) contains all the compiled bytecode of an Android application. This is what is decompiled into Java source files.

Android Application Structure

resources.arsc

The resources.arsc file contains metadata about the resources and the XML nodes of the compiled resource files like XML layout files, drawables, strings, and more. It also contains information about their attributes (like width, position, etc) and the resource IDs, which are used globally by both Java and XML app files in the app. This file is compressed into a binary form that is read into memory during runtime. Apktool can also decompress these files and output them into a humanly-readable format for you to explore.

@ikuamike

Android Application Structure

res/

The “res” folder contains compressed binary XML versions of the resource XML files that are paired with the resources.arsc file during runtime to read images, translations, etc. These XML files are in the same binary format as the AndroidManifest.xml file and can be easily decoded with apktool.

Android Application Structure

lib/

Not all Android apps contain a lib/ folder, but any app with native C++ libraries will. Within this folder, you will find different folders per-architecture, each one containing .so files specifically compiled for that target architecture such as “armeabi-v7a” and “x86”. This is also why you cannot install an app on an x86 device without it providing x86-compiled libs (Google for “INSTALL_FAILED_NO_MATCHING_ABIS”).

Android Application Structure

assets/

Any other files that may be needed by the app appear here. Additional native libraries or DEX files may be included here. This can happen especially when malware authors want to try and “hide” additional code, native or Dalvik, by not including it in the default locations.

Career Relevance

1. Bug Bounty Hunter
2. Penetration Tester
3. Application Security Engineer
4. Mobile Developer

Forensics

@k0imet_

Steganography

@k0imet_

Binary Exploitation

Practical Session

- ROPemporium ret2win - https://ropemporium.com/challenge/ret2win.html

Tools

- Ghidra
- pwntools
- gdb

@trevorsaudi

What Is Binary Exploitation

- In a CTF context, binary exploitation involves finding vulnerabilities such as buffer overflows, heap overflows, format string bugs in programs and exploiting them in order to get your flag.
- Most commonly, these vulns result in an RCE that enables you to run arbitrary command on the remote server
- In CTF context, we are provided a binary file, known as ELF (executable and linkable format)
- Mostly known as Pwn category in challenges
- Career : malware analyst, bug bounty, network security, application security

@trevorsaudi

Reverse Engineering

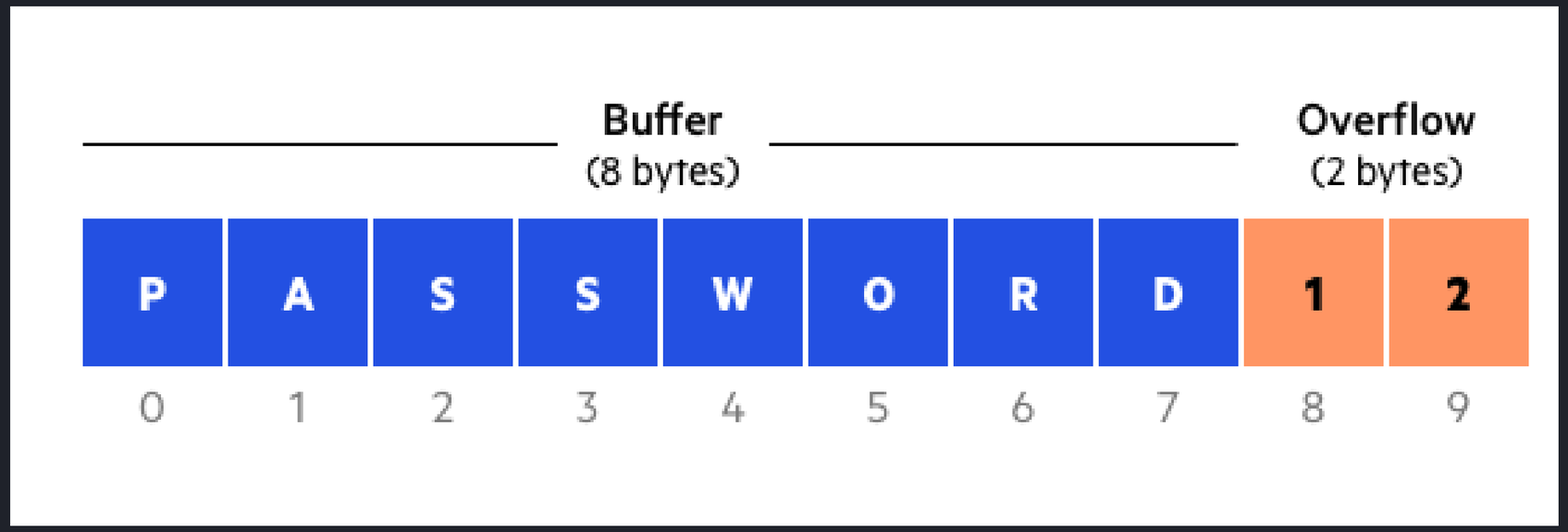
Introduction

- The goal involves understanding the functionality of a given program by decompiling or disassembling the binary and making sense of the output
- Career: similar to binex

Buffer Overflows

- The most basic forms of binary exploitation take place in the stack
- Buffers are temporary memory regions that temporarily store data during memory operations
- A buffer overflow vuln is one where the volume data being supplied in the buffer exceeds the storage capacity of the buffer
- This causes the program to write to adjacent memory locations causing crashes

Buffer Overflows



@trevorsaudi

Impact of the BOF attack

- By overwriting the memory of an application, we can hijack the execution flow of a program and possibly send malicious instructions to the program
- In the context of CTFs, our goals usually are:
  - 1. Jump to a function defined in memory which gives us a shell (ret2win)
  - 2. Use ROP to get our own shell in the server
- We have stack-based BOFs and Heap-based ones

Impact of the BOF attack

- Stack is more common, leverages stack memory which exists during execution of a function
- Heap is harder to exploit and involves flooding the memory space allocated for a program beyond memory used for current runtime operations
- The common languages involved which are more vulnerable are C and C++ since they lack safeguards

How the stack works

- The stack is a region of computer memory that stores temporary variables created by a function
- It is a LIFO (last in first out) data structure and uses principles such as push and pop
- Push - adds an element to the stack
- Pop - removes the last element that was added
- It grows backwards into memory

Registers used in the stack

EBP - base pointer
EIP - instruction pointer

- The base pointer marks the address of start of a function’s stack frame(area managed by that function) while the instruction pointer marks the address of the next instruction to be executed in memory

Registers used in the stack

64	32
RAX	EAX
RBX	EBX
RCX	ECX
RDX	EDX
RDI	EDI
RSI	ESI
RBP	EBP
RSP	ESP

@trevorsaudi

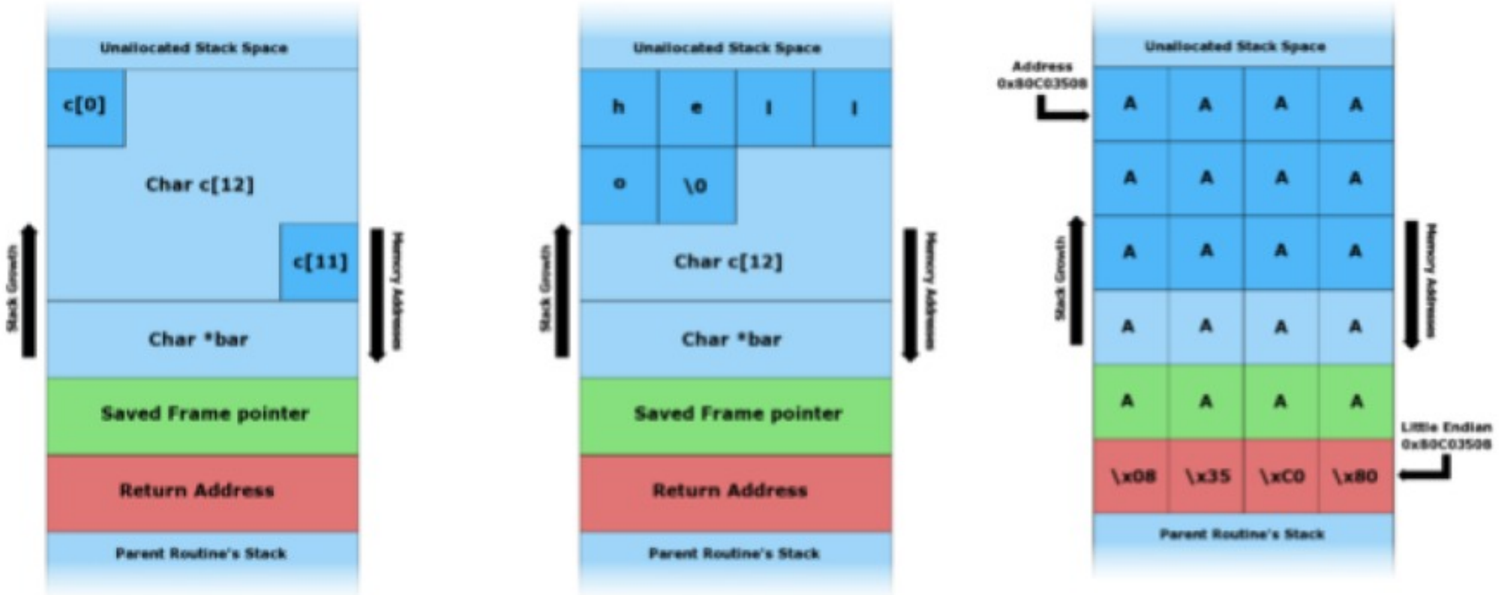
Registers used in the stack

- When we run this code, it takes an argument from the command line and copies to a local stack variable c
- The code works well only if you supply input less than 12 characters. (11 characters and below)
- The program stack looks as follows
- Variables get loaded first
- Then we have the frame pointer, essentially the EBP which is the address of the current instruction being executed

Registers used in the stack

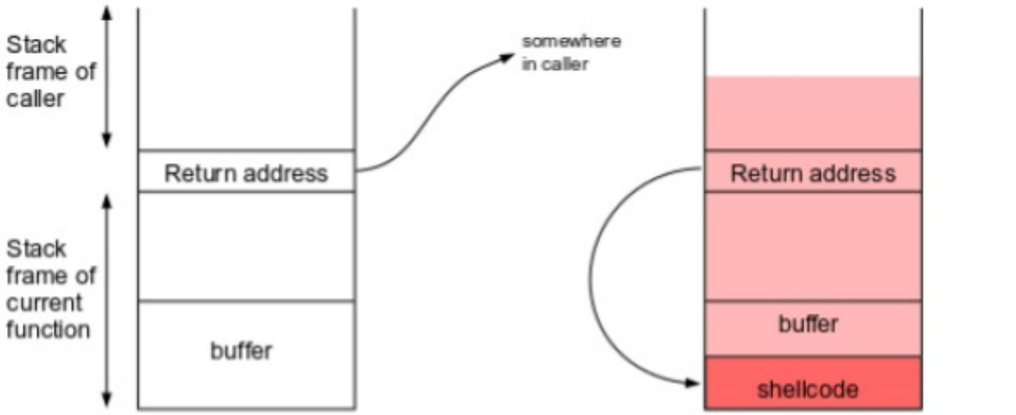
- Then the return address is where the stack will return to after executing this function

Registers used in the stack



@trevorsaudi

Registers used in the stack



Normal memory layout

After overflow
All red values written by the attacker
Return address now points to shellcode

@trevorsaudi

Endianness

- Endianness is the order or sequence of bytes in computer memory
- Expressed as big-endian or little-endian
- Bigf endian stores the most significant byte first while little endian stores the least significant value first (reversed order of bytes)
- Eg 0xdeadbeef will be represented as 0xefbeadde in little endian format

ret2win

- This is the most basic binex challenge
- It involves a binary where there is a win function , and once you direct execution flow to that function you complete the challenge and get the flag
- To successfully exploit this, you will need to overwrite the EIP(32-bit) or RIP(64-bit), with a specific value (normally little-endian version of the return function’s address)

Summary solution

- Study the reversed binary using tools like ghidra to understand how the program works - if source is provided use that instead
- Fuzzing - supply large input to the program till it crashes
- Find the offset - at what point does the input crash the program (EIP)
- Find the win function address
- Put together a script and get your flag

Summary solution

Snippet solution script

- https://gist.github.com/trevorsaudi/3cfa6f4037dd6a1e684416b1f2e94aef

References & Resources

- https://ir0nstone.gitbook.io/notes/types/stack/ret2win
- https://en.wikipedia.org/wiki/Stack_buffer_overflow
- https://github.com/Zeyad-Azima/Offensive-Resources#exploit-development
- https://github.com/r0hi7/BinExp

Wargames to practice on

- ROPemporium
- Protostar
- Picogym
- Pwnable.kr
- Pwnable.tw
- Pwnable.xyz