

Advanced Penetration Testing, Exploit Writing, AND Ethical Hacking

Module 1: Network Attack for Penetration Tester

Tcp stack fingerprint->change with osfuscate

Mac address impersonation->macshift(windows),ifconfig,ip(linux)

Http sniff->cain,ettercap,bettercap

Captive Portal access point->Fluxion

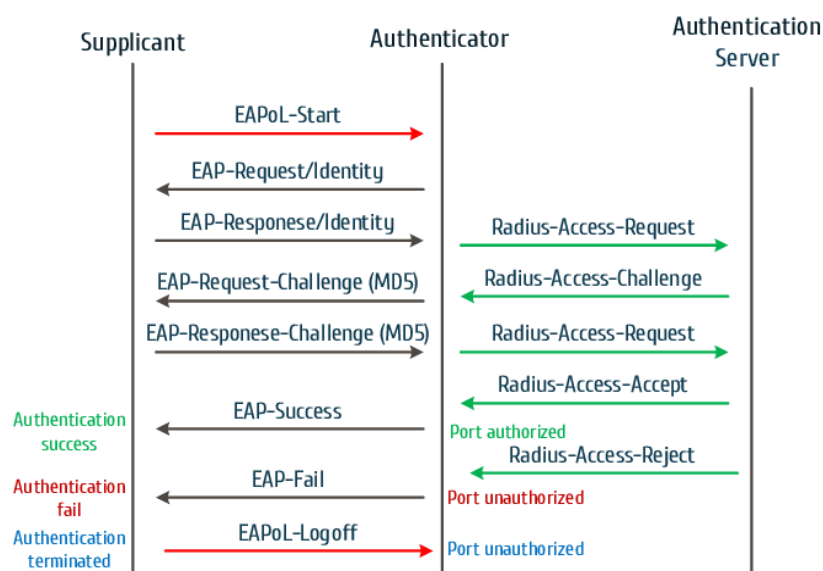
802.1x:

Authentication protocol,lan or wan

1-supplicant: the client device that wishes to connect to the LAN

2-authenticator: a network device such as a switch that provides access to the LAN

3-authentication server: a host that runs software that implement RADIUS or AAA protocol



Extensible Authentication Protocol(EAP):

Not protocol, message format

EAP-TLS->prevent MITM

EAP-MD5 Attack

[silentbridge](#)

<http://xtest.sourceforge.net/>

MAC Filtering and MAC Authentication Bypass(MAB):

All device not support 802.1x the port security exceptions

Bridge-based bypass with [silentbridge](#) or ...

VLAN Manipulation

1-Vlan hooping

<https://github.com/nccgroup/vlan-hopping---frogger>

<https://github.com/commonexploits/vlan-hopping>

<https://github.com/tomac/yersinia>

Solution:

- disable dtp
- prevent double tagging

2-Voice Vlan hooping

cdp->discovery protocol->packet interval->60s->vlan number,void vlan number

Voiphopper

Network Manipulation

1-MITM

LAN Manipulation

1-ARP Spoofing

Ettercap or bettercap

2-SMB Capture

Ettercap or bettercap

HSRP

Hot standby router protocol->ensure high-availability across multiple routers

Hsrp authentication->plaintext password->default password->cisco

Multicast hello messages include credential

Sending raw hsrp packet,active router,MITM with yarseina

<https://github.com/tomac/yersinia>

VRRP

Virtual Router Redundancy Protocol

Not include any authentication or integrity checks

MITM with Loki

https://github.com/Raizo62/Loki_on_Kali

Routing Protocols

- Discovering internal network
- Mapping internal infrastructure
- Wide-scale mitm opportunity

Ospf

Periodic multicast "hello" packets

Ospf enumeration with loki

Ospf md5 attack with loki

Also use cisco ios virtual machine

IPV6 for Penetration Testers

Ipv6 attacks can be local or remote

Arp protocol->ICMPv6 ND->spoofing or manipulation

Local ipv6 device enumeration

```
ping6 -I INTERFACE -c 5 ff02::1 > /dev/null, nmap -6 -sS -sC fc00:660:0:1::23
```

IPv6 replaces ARP with Neighbor Discovery(ND)

Remote IPv6 Discovery

```
Dig +short IN AAAA www.google.com
```

IPv6 Neighbor Impersonation MITM Attack

A->NS->all multicast nodes->identify B Mac address

B->NA->A

C->own NA with B's MAC Address->A with NA Override flag set

then

A->C->B

Sysctl -w net.ipv6.conf.all.forwarding=1

Parasite6 -IR INTERFACE

IPv6 Router MITM Attack

Node discover router with ICMPv6 router solicitation(RS)

Router respond with configuration detail with for all nodes

Attacker also claims to be a router, with a higher preference(RA Message)

IPv4 to IPv6 Proxy

<https://github.com/dukaev/ipv4-ipv6-proxy>

Section 2: Exploiting The Network

Network Exploitation

1-access to the network(check ability to manipulate clients with MITM - check)

2-techniques to exploits clients and infrastructure devices

Software Updates

1-retrieves the new update, executes the install automatically

2-sometimes update are performed over ssl(commonly updates are delivered over HTTP)

ISR Evilgrade->exploit weak software update

<https://github.com/infobyte/evilgrade>

Dns manipulation with ettercap+evilgrade

<https://www.youtube.com/watch?v=n-WdGURSDiI>

HTTPS

Mitm attack->manipulate http traffic->request to upstream https site with Ssltrp

<https://github.com/moxie0/sslstrip>

Ettercap or bettercap+ssltrp

SNMP

Snmp framework we typically have four components

1-managed device->remote device able to read or write snmp variables

2-agent->snmp service running on managed device

3-nms(network management system)->management console->collect or set snmp data on managed device and interact with agent

4-mib(management information basic)->data reported by the snmp agent

SNMP Eavesdropping: Ettercap

As MITM, ettercap will log all snmp community string observed

Ettercap -Tqm arp:remote // //

SNMP Agent Discovery

Snmp version scan->Nmap -sU -p161 -A IP

Enumerate multiple dns server->Fierce.pl -dns ri.cox.net

Snmp community string scanner:

1-genip 10.10.10.1-254 >hosts.txt

2-onesixtyone -c snmp.txt -i hosts.txt

Metasploit snmp scanner:

Use auxiliary/scanner/snmp/snmp_login

Set rhost 10.10.10.1-254

Set verbose false

Set threads 4

Exploit

Enumerate device(hostname, local user account, local routing table information, installed software, ...)->snmpcheck -t 10.10.10.10 -c NotSoPublicCommunityString

Router/Switch SNMP Control

Snmp rw access to routers and switches

Change MIB variable:

Snmpwalk -v2c -c public 10.10.10.0 system.sysLocation.0

Snmpset -v2c -c public 10.10.10.10 system.sysLocation.0 s "Foo"

<http://www.net-snmp.org/>

Also use cain and snmpblow.pl

Module2: Crypto, Network Booting Attacks, and Escaping Restricted Environments

Section1: Crypto for Pentester

Stream Ciphers

- Encrypt one bit at a time
- Encrypt length in the same as plain text(63 byte ciphertext==63 byte plaintext)
- Cipher generates a keystream
- Keystream is xor'd with plaintext to produce ciphertext

IV Considerations

We accomplish by mixing a per-packet value with each key(Initialization vector(IV))

Block Cipher

Encrypt data a block at a time

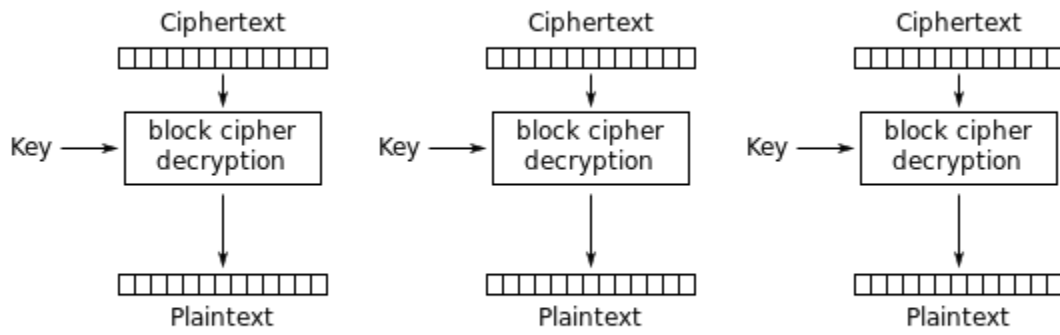
Must pad the last few bytes to an even block length(8 byte block length with 64 bytes ciphertext is 57-64 bytes plaintext)

Example: AES, DES, 3DES, Blowfish

Block Cipher Mode

Any block cipher can used with various modes(AES-CTR,3DES-CBC)

1-ECB

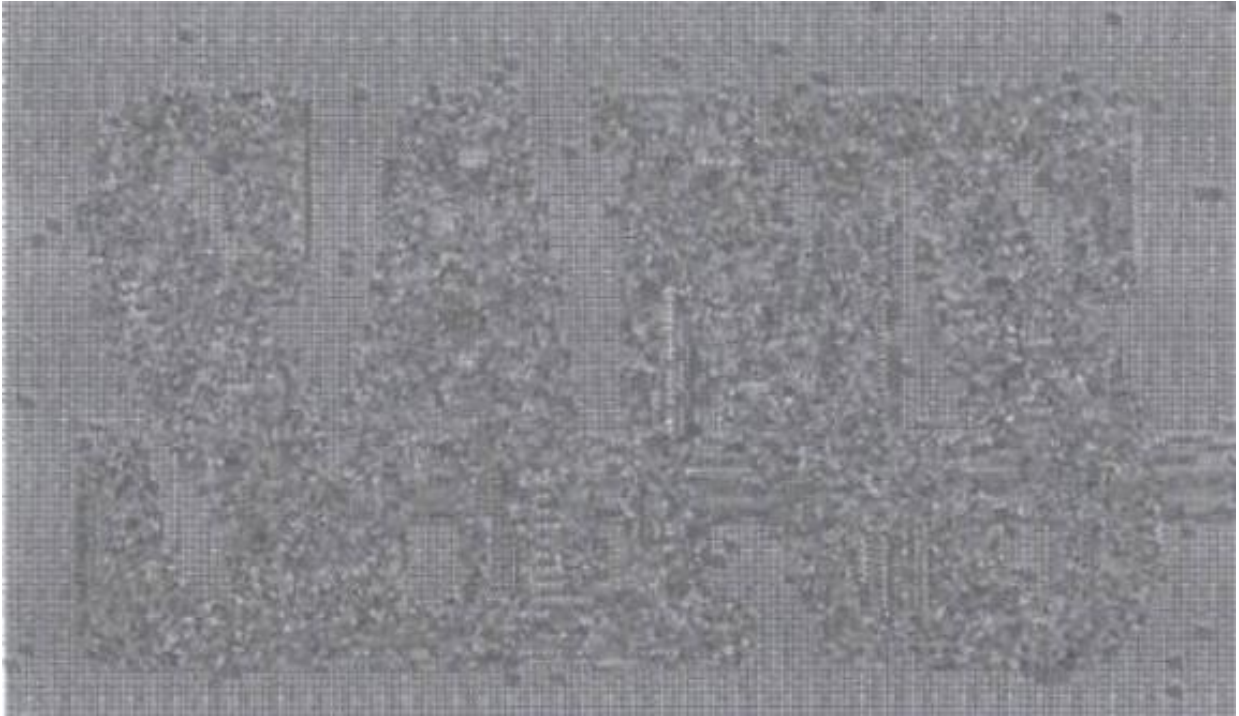


Electronic Codebook (ECB) mode decryption

Before encrypt

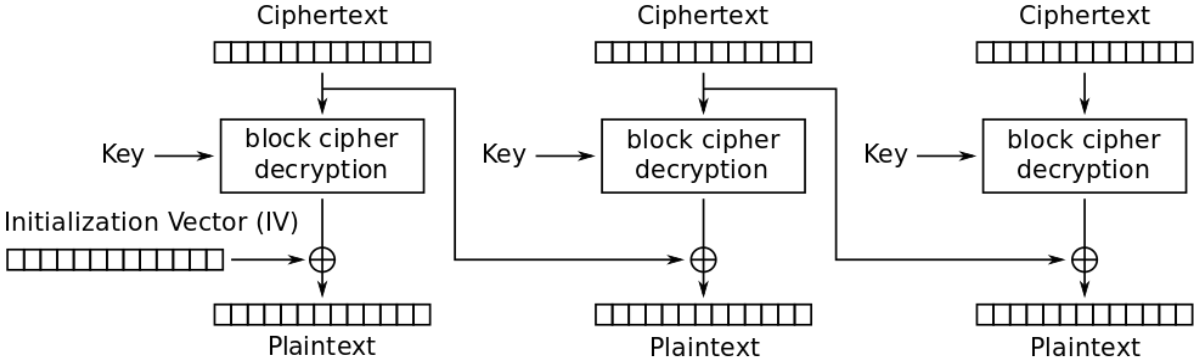


after encrypt



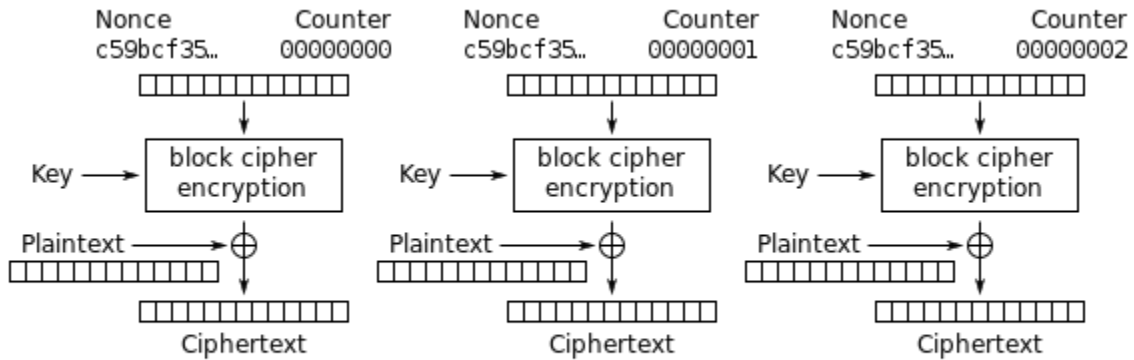
https://www.willhackforsushi.com/code/ecb_encrypt_image.zip

2-CBC



Cipher Block Chaining (CBC) mode decryption

3-CTR



Counter (CTR) mode encryption

Identifying the Algorithm

Encrypt data size divisible by 8->stream cipher, often RC4

Encrypt data size always divisible by 16->AES(128-bit block size)

Encrypt data size always divisible by 8->DES, 3DES

<https://github.com/Wind-River/crypto-detector>

<https://github.com/ashutosh1206/Crypton>

Hash Identification

Hash use a input for processing or storage

- Password storage
- Http parameter
- Message integrity checks

<https://github.com/blackploit/hash-identifier>

Extract the payload data for TCP and UDP packets

1-visualize byte distribution

Pcaphistogram->pcaphistogram.pl capture2.dump | gnuplot

2-extract data with tpick, evaluate with ent

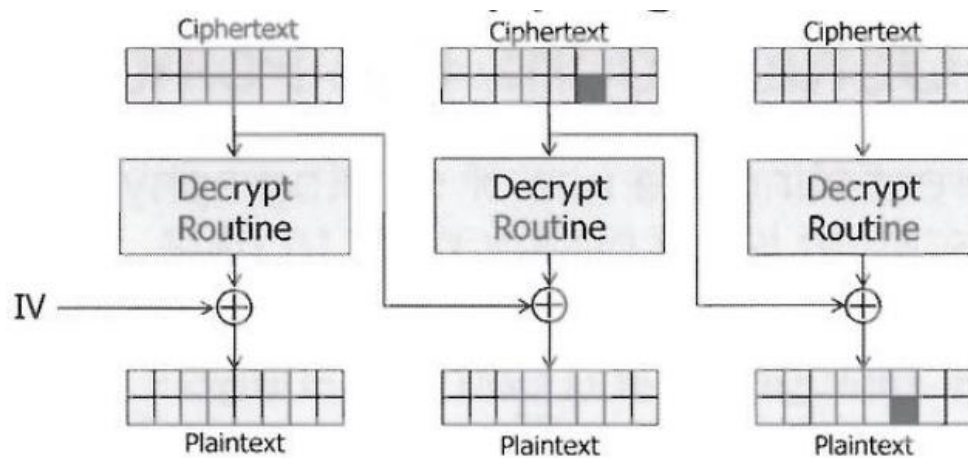
Tpick -r sample.dump -wR

Ent *.dat

3-extract data with custom scapy code, evaluate with ent

Scapy and ent

CBC Bit Flipping Attacks



- CBC decryption XOR's the decrypted data with the prior ciphertext block
 - Modified ciphertext will produce invalid plaintext (not always an issue)
- Attacker can manipulate the plaintext data by modifying prior encrypted block

Change IV

<https://github.com/GrosQuildu/CryptoAttacks>

Oracle Padding Attacks

Padding: Require for block ciphers, different methods are used.

PKCS#5/PKCS#7 Padding

Plaintext: Josh

J	o	s	h	0x04	0x04	0x04	0x04
---	---	---	---	------	------	------	------

Plaintext: Bryce

B	r	y	c	e	0x03	0x03	0x03
---	---	---	---	---	------	------	------

Plaintext: Stephen

S	t	e	p	h	e	n	0x01
---	---	---	---	---	---	---	------

Plaintext: Bernadette

B	e	r	n	a	d	e	t	t	e	0x06	0x06	0x06	0x06	0x06	0x06
---	---	---	---	---	---	---	---	---	---	------	------	------	------	------	------

Plaintext: Jennifer

J	e	n	n	i	f	e	r	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
---	---	---	---	---	---	---	---	------	------	------	------	------	------	------	------

- Pads blocks with an integer indicating number of padded bytes
- Used for simple error checking
- Even block boundaries get an added block of just padding added

PKCS#7 padding is identical to PKCS#5, for 8 or 16 byte block ciphers (PKCS#5 is only defined for 8 byte blocks)

<https://github.com/AonCyberLabs/PadBuster>

<https://github.com/KishanBagaria/padding-oracle-attacker>

<https://github.com/liamg/pax>

Hash Length Extension Attack

Function	Length
MD4	16 bytes
MD5	16 bytes
SHA	20 bytes
SHA1	20 bytes
SHA256	32 bytes
SHA512	64 bytes
RIPEND160	20 bytes
WHIRLPOOL	64 bytes

<https://github.com/stephenbradshaw/hlextend>

<https://github.com/bwall/HashPump>

https://github.com/iagox86/hash_extender

Section 2: Attacking with Network Booting

Pre-boot Environment

Includes any special enterprise keyboard/video/mouse control devices.

IPMI: Intelligent Platform Management Interface

auxiliary/scanner/ipmi/ipmi_version

<https://blog.rapid7.com/2013/07/02/a-penetration-testers-guide-to-ipmi/>

DHCP

Dhcp does not support authentication

Mitm:

Ettercap -TqM dhcp:10.10.9.9-100/255.255.0.0/8.8.8.8

PXE: Portable Execution Environment

Provide a bootable image to the client as a DHCP extension after IP address assignment.

<https://www.rapid7.com/db/modules/auxiliary/server/pxeexploit>

bootp(bootstrap protocol): client asks its ip by sending UDP,server broadcast the reply

Amended for vendor options: tftp, dhcp over bootp

Use wireshark filtering for "bootp or tftp"

PXE Attacks

1-pxe server as boot image(konboot,kali)

2-attacker needs a small infrastructure(presence along the dhcp request, malicious dhcp server, malicious tftp server)

<https://github.com/secdev/scapy/wiki/Contrib:-Code:-DhcpTakeover>

msf->pxexploit

Kon-boot

<https://blog.netspi.com/attacks-against-windows-pxe-boot-images/>

Hypervisor + VT-d + Storage Appliance

<http://sanbarrow.com/vmdk-basics.html>

Section 3: Escaping Restricted Environments

Type of restricted environments:

- *nix chroot/jail
- *nix SELinux/AppArmor
- VME(Component Virtualization)
- Windows Software Restriction Policies(SRPs)

Gols:

- escalate(become admin/root)
- escape(subvert the restrictions)

Chroot

Hide the rest of the filesystem from an application

- Is not virtualization
- Is not a security control
- Is not a jail

Chroot'ed process can still see other process

Chroot'ed process can start a new process

For example:

1-inside

Chrome /root/prison/

Nc -nvlp 99

2-from outside the jail

Ps auwx | grep nvlp

Ls -l /proc/ID | grep prison

Manipulate chroot to escape:

1-linux syscall 61(C Code, pointer to path in EBX)

2-see /proc

3-simply chdir & chroot to escape

Jail

With jails, effectively everything is virtualized and isolated except the kernel.

Jail lock a process

- To a file path
- Including forks

Configuration or administration mistake, like overlapping directory structure or moving a jailed process's working directory out from under it, could be used to escape

<https://filippo.io/escaping-a-chroot-jail-slash-1/>

<https://www.hackingarticles.in/multiple-methods-to-bypass-restricted-shell/>

Solaris Zones and Containers

Extend chroot+jail

Can have copies of the same of different kernels

Grsecurity and PAX

Grsecurity: protect linux kernel

Pax: gcc protection

- Patches to linux kernel and gcc
- Restricts and protects exploitation of common bugs
- Role Based Access Control(RBAC)

Application Restrictions

- Selinux/apparmor/applocker

Selinux limits by inode

Apparmor by filepath

Shell Restrictions

- Can be used to skirt other limits
- Only obscure as a limitation
- Shell specific in \$HOME

Virtual Machine Environment

- Network architecture often virtualize assuming it protects
- Often increase exposure to unsupported hosts

General Methodology to Escalate

1. Gain privilege by exploiting services
2. Environment variable
3. Trojan executable or libraries
4. Configuration or other input files
5. Meterpreter post modules: post/windows/escalate
6. New metasploit local exploit: exploit/[OS]/local/*
7. Depends heavily on what is installed
8. Look to obscure input

Example command to Escalate

0-find the environment and home

Ls -la

Echo *

1-Find any writable files:

Find / -type f -perm -o+w

2-Find any configuration file:

Find / -type f -name "*conf" -o -name "*cfg"

3-Find inventory of tools:

Find / -type f -perm -o+rx

4-find SUID/SGID programs

Find / -perm -2000 -o -perm -4000

Sudo -l

5-find insecure file usage

Ltrace /usr/sbin/adminapp

Manipulating Library Loading on Linux

1-check for libraries to abuse

Ldd /usr/bin/potential-escalator

2-ELF's RPATH/RUNPATH Hardcoded Libs Path

3-find libraries to control program

Env | grep 'LD_\\|RTL_'

Manipulating Library Loading on Windows

1-Find dll for vulnerable exe

Tasklist /fi "imagename eq notepad.exe" /m

Breaking Out as UID0

Chdir + chroot + traversal

Breaking Jail

- Kernel vulnerabilities
- Links or local privilege escalation

VME Attack Surface

1-Attack hypervisor directly

- custom exploitation
- what input
- what we can control

2-Attack the implementation

- weak password
- bad configuration

3-Management network

- MITM management traffic
- Password guess

4-VM Network

- Escalate or pivot to another guest
- Weakest guest to attack first
- If any vm require promiscuous mode, vswitch allows for all

5-storage network

- MITM NFS(TCP)
- MITM iSCSI MS-CHAP Authentication

Looking for Exploitable Features or Flaws

1-find usability information on custom apps

Man custom_app

Find /usr/share -name custom_app

Find /usr/local/share -name custom_app

2-try different usage arguments

/usr/bin/custom_app -h

/usr/bin/custom_app --help

/usr/bin/custom_app --DOESNTEXIST

3-break custom application with bad input

/usr/bin/custom_app 12

/usr/bin/custom_app AA

/usr/bin/custom_app A B C D E F ...

4-error messages can lead to exploitability

5-tracing library calls will show lower-level security

Ltrace /usr/bin/custom_app 10.10.10.10

6-check for blessed authorization

Sudo -l

7-privileged editors that can execute

vi/vim :!/bin/bash or :shell

Ed ed !/bin/bash

Nano Ctrl+W

8-privileged commands with arguments

Custombackup ` /bin/bash`

Export \$IFS=":."; custombackup file:`id`

Other Tools to Help Escalate

1-penetestermonkey's exploit-suggester

2-PenturaLab's Linux_exploit_suggester

3-Pentestmonkey's unix-privesc-check

4-Tobias Klein's checksec.sh

5-LinEnum

6-Enum4linux

Section 4: Windows Restricted Desktops

Restricted Desktops

Type of restrictions

1-physical security controls

- BIOS
- Boot
- Encrypted disks
- HW disabling

2-network and internet controls

- Web filtering, host-level firewalls/HIDS/HIPS

3-windows controls

- Group Policy Objects(GPO)
- Application Black/Whitelisting(applocker)
- Third-party software replacing explorer.exe

Third-party software: RES PowerFuse, Secure Desktop, SiteKiosk

Software Restriction Policies

Policy rule are based on:

- Certificate: software publisher certificate used to digitally sign the file
- Hash: a cryptographic fingerprint of the file
- Zone: from which IE zone the file was downloaded
- Path: path where the file is stored

Unrestricted: if the application falls into the scope of the rule, execution is allowed

Disallowed: if the application falls into the scope of the rule, execution in not allowed

Escaping Restricted Desktops

- Breaking out of authorized applications
- Transferring files and tools
- Executing custom code

- 1-work with most windows accessories and games
- 2-GPO hides files/directories
- 3-third-party application as a file transfer tools(notepad,ms paint,..)
- 4-nslookup as a file transfer tool
- 5-using debug.exe to re-create an exe/dll
- 6-ikat toolkit
- 7-using runas

Exploits:

- 1-browser exploits(/server/browser_autopwn auxiliary)
- 2-using runas
- 3-file-format exploits(/server/file_autopwn auxiliary)
- 4-Using a Dynamic Link Library(msfvenom+msfpayload,...)
- 5-Microsoft office macros
- 6-using the windows api
- 7-powershell

Module 3: Python, Scapy, and Fuzzing

Section1 : Product Security

Product security testing

- Network Access Control(NAC)
- Intrusion Prevention System(IPS)
- Antivirus(AV)
- Voice over IP(VoIP)
- Smartphone
- Countless others

Initial Questions

- Type of product
- Size of deployment
- Location of deployment
- Data element stored
- User access level
- Business driver

Testing Environment

- VMware and images of company OS build
- hardware(laptops, switch, cables)
- Disassemblers and debuggers
- Fuzzing tools(sulley, packetfu, custom)
- Scripting language(python, ruby)
- sniffers(wireshark, tcpdump)

Vulnerability Discovery

- Corporate disclosure policy
- Appropriate contacts
- Severity and impact
- Remediation efforts

Types of Disclosure

- Full disclosure: detail made public, possibly with a exploit
- Limited disclosure: existence of problem publicized
- Responsible disclosure: analyst works with vendor to disclose after resolution

Section2 : Python for Pentester

1-basic types

2-string slicing

3-string concatenation

4-lists

5-control(if/elif/else,for)

6-useful python built-in's(func)

7-exceptions

8-modules(sys, os)

9-introspection(dir, help, type, globals)

10-useful snippets(<https://github.com/jmortega/python-pentesting>)

11-python 2 to python3(2to3)

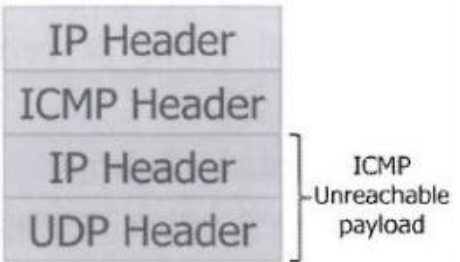
Leveraging Scapy

1-scapy packet layering

```

>>> p = IP(src="10.10.10.10", dst="10.10.10.70")
>>> p /= ICMP(type=3,code=0)
>>> p /= IP(src="10.10.10.70", dst="10.10.10.10")
>>> p /= UDP(sport=135,dport=135)
>>> p
<IP frag=0 proto=icmp src=10.10.10.10
dst=10.10.10.70 |<ICMP type=dest-unreach
code=network-unreachable |<IP frag=0 proto=udp
src=10.10.10.70 dst=10.10.10.10 |<UDP
sport=loc_srv dport=loc_srv |>>>>

```



- Scapy provides the blocks for assembling packets
- You assemble them as needed by appending each object

2-scapy protocol support

<https://scapy.readthedocs.io/en/latest/>

3-useful script

- Intercepting packets with scapy
- Packet capture interaction
- Sniffer channel over wireless
- Ipv6 router discovery

Section3: Fuzzing Introduction and Operation

What is fuzzing

1. Testing mechanism that send malformed data to a well-behaving protocol implementation
2. Research technique that has shows great success in identifying vulnerabilities
3. Essential part of a software development lifecycle for secure products

Fuzzing requirements

1. Documentation: a source of information about the target being evaluated
2. Target: one or more targets to evaluate
3. Tools: fuzzing tools or programmatic harness to leverage to building tools
4. Monitoring: method to identify when a fault is reached on the target
5. Time, patience, creativity

Techniques - Static Test Cases

- During information collection, analyst identifies individual tests
- Test case stored as a file that can be sent to target, often binary file
- Lots of up-front development time

- Limited by creativity of analyst
- Easy to reproduce tests across systems

Techniques - Randomized

- Start with a valid frame
- Selected portions replaced with randomized data
- Infinite run-time process, limited code coverage due to random nature of data

Techniques - Mutation

- No protocol analysis, just a sample data for mutation
- Mutates one byte/short/long at a time through entire data

Tools: taof

Techniques - Intelligent Mutation

- Describes a protocol and tests permutations
- Lots of up-front time analyzing protocol
- Often consists of a protocol “grammar” and describing the operation and framing

Vulnerability:

- Directory traversal(..../..../..../..../etc/passwd)
- Command injection(system())

Read more: <http://fuzzing.org/>

Section4: Building a Grammar with Sulley

Sulley as a fuzzing framework

<https://github.com/OpenRCE/sulley>

<https://github.com/jtpereyda/boofuzz>

1-http get request initialization

2-http get request immutable values

3-http get request delimiters

4-http get request strings

5-http get request numbers

6-http get request finishing up the grammar

7-http get request counting mutations

8-http get request estimating runtime

9-http get request displaying mutations

Session Agent

Netmon: capture libpcap files for each mutation

Procmon: monitor process for faults, restarting as needed

Vmcontrol: start, stop, and reset guest; take, delete and restore snapshots

Post-Mortem Analysis

Pcap_cleaner: remove all files without crash data(even non-pcap!)

Crashbin_explorer: navigate, examine and graph crash data

Section5: Fuzzing Block Coverage Measurement

Block Coverage

- Wxpython+mysql+ida pro+ida python plugin+uDRAW+paimei
- pstalker

NEED MORE INFORMATION!!!

Module4: Exploiting Linux for Penetration Testers

Section1: Introduction Memory

Physical Memory

Process registers:

- Hard coded variables
- Fastest

Process cache:

- Data cache - L1 & L2 cache
- Instruction cache & tlb's
- Pre-fetches data from ram

Random Access Memory(RAM): volatile memory the loses the information it holds when its host is powered off

Processor Register

1-General purpose registers - 32bit

- EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP

2- General purpose registers - 64bit

- RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP + R8-R15

3-Segment Register

- CS, DS, SS, ES, FS, GS
- Often used to reference memory locations

4-Flags register - Mathematical Operations

- Zero flag, Negative flag, Carry flag, etc.

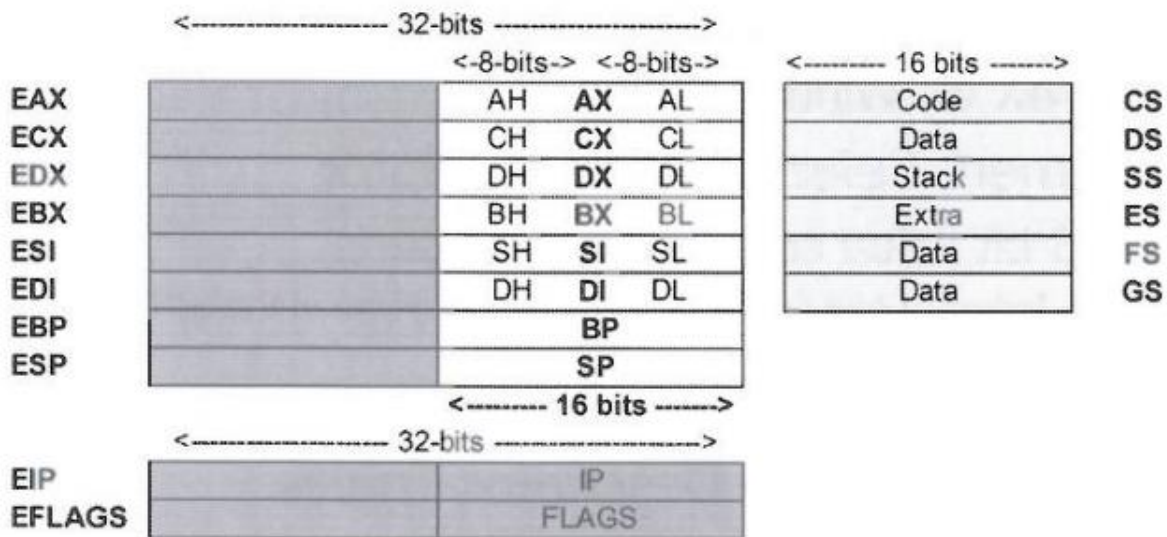
5-Instruction Pointer(IP)

6-Control registers

- CR0 - CR4
- CR3 holds the start address of the page directory

General Purpose Registers

- EAX/RAX - Accumulator Register - "imul eax,4"
Designed to work as a calculator
- EDX/RDX - Data Register - "add eax, edx"
Works with eax on calculations
Pointer to input/output points
- ECX/RCX - Count register - "mov ecx, 10"
Used often with loops
- EBX/RBX - Base register - "inc ebx"
General purpose register
- The lower 16-bits of the 32-bits general purpose registers
- ESI/RSI - Source index
Pointer to read location during string operations and loops
- EDI/RDI - destination index
Pointer to write locations during string operations and loops
- ESP/RSP - stack pointer - "movl %esp,%ebp"
Holds the address of the top of the stack
- EBP - base pointer - RBP is used for general purpose
Serves as an anchor point for the stack frame



Segment Registers

- Segment register functionality and types
 - * nix vs. windows usage
- Segment selector and descriptor
- Global and local descriptor tables
- CS - Code segment
- SS - Stack segment
- DS - Data segment
- ES - extra segment
- FS - extra data segment
- GS - extra data segment

Memory models

- Real mode
 - 64-bit systems still start in this mode
- Protected mode
 - Support for virtual memory up to 4gb and beyond
- Long mode for x64 systems

Virtual Memory

1. Physical memory
 - Physical address extension(pae) can support up to 64gb
2. Virtual / Linear addressing
 - Supports 4gbs of virtual address space on a 32-bit system

Paging

- Process of allowing indirect memory mapping

- Linear addressing is mapped into fixed-sized pages
Most commonly 4kb
- Page file may not be needed or used on 64-bit systems
- Context switching and the process control block(PCB)
Register values for each process are stored in the pcb and loaded during context switching

Paging vs. Swap

- Non-recently access pages are copied over to disk
- Page faults
- Swapping an entire process
- Windows memory optimization

Object Files

1. Code segment
Fixed size segment containing code
2. Data segment
Fixed size segment containing initialized variables
3. BSS segment
Fixed size segment containing uninitialized variables
4. Heap
Segment for dynamic and/or large memory requests
5. Stack segment
Procedure stack for the process

Calling Conventions

- Define how functions receive & return data
Parameters are placed in registers or on the stack
Define the order of how this data is placed
Includes adjusting the stack pointer during or after function epilog to advance over arguments
- Most common calling conventions
Cdecl - caller places parameters to called function from right to left and the caller tears down the stack
- Stdcall - parameters placed by caller from right to left, and the called function responsible for tearing down the stack

Tools: GNU Debugger(GDB)

Section2: GDB & Syntax

Gdb Useful commands

- Disass <function>
- Break < function>
- Print \$eip
- x/<number>i<mem address>
- Info
- C or continue - continues execution after a breakpoint
- Si - step one instruction

- Backtrace or bt
- Set disassembly-flavor <intel or att>
- Info breakpoints & delete breakpoints
- Run

X86 assembly language

- Low-level programming language
- Optimized for processor manipulation
- Ideal for:
 - Device drivers
 - Video games requiring hardware access
 - Allows faster access to hardware
 - Where speed is critical

AT&T vs. Intel syntax

• AT&T

sub \$0x48, %esp

mov %esp,%ebp

src dest

- \$ = Immediate Operand
- % = Indirect Operand
- () = Pointer

• Intel

sub esp, 0x48

mov ebp, esp

dest src

- [] = Pointer

• Size of Operands

– AT&T uses the last character in the name of the instruction such as b for byte, w for word, or l for long

- movl \$0x8028024,(%esp)

– Intel uses "byte ptr", "word ptr", or "dword ptr"

- mov DWORD PTR [esp],0x8028024

Section3: Linkers & Loaders

- Linkers vs. loaders
Linker link a function name to its actual location
Loaders load a program from storage to memory
- Symbol resolution
Resolving the function's address during runtime
- Relocation
Address conflict may require relocation
- Name mangling(not to be confused with overloading)

ELF

- Executable and linking format
- Executable & relocatable files
Can be mapped directly into memory at runtime
Allows for relative addressing to remain while changing the load address
- Shared objects
Used primarily to house shared function
- Procedure linkage table(PLT)
Read-only table produced at compile-time which holds all necessary symbols needing resolution
Resolution performed when a request is made for the function(lazy linking)
- Global offset table(GOT)
Writable memory segment to store pointers
Update by the dynamic linker during symbol resolution

Tool: objdump

Objdump

- Objdump -d
Disassemble an object file
- Objdump -h
Displays section headers
- Objdump -j <section name>
Allow you to specify section
Objdump -j .text -d ./<prog_name>

Tool: readelf

Readelf

- Tool to display ELF object file information
- Display information on ELF headers and sections; i.e. GOT, PLT, Location information

ELF Demonstration

1-main func
Disas main

2-show on the second image
Objdump -d -j .text memtest | grep puts

3-show on the slide image

objdump -R memtest

4-view all section

Readelf -t memtest

5-within the .got.plt section

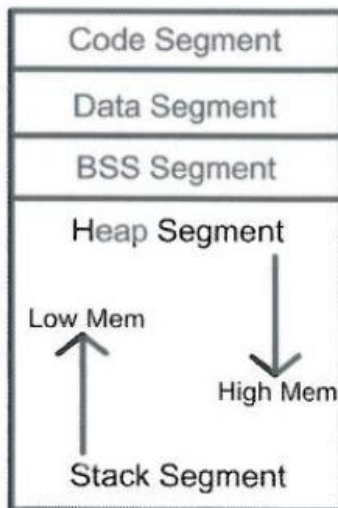
Readelf -x 22 memtest

Section4: Dynamic Linux Memory

Memory - The Heap

What is a heap?

- Dynamic memory allocated at program runtime
Memory allocating functions are used to request resources
- Allocation time is not finite
- Memory is freed by
Program code
Garbage collector
Program termination



Dynamically Allocated
Memory

1. Code Segment holds executable instructions
2. Data Segment stores global and static variables
3. BSS Segment stores uninitialized counterparts
4. Heap Segment is used for all other program variables

Erickson, Jon. "Hacking, The Art of Exploitation."
San Francisco: No Starch Press, 2003

Malloc Implementation

Heap manager used by the program

Interface to sbrk() and mmap()

malloc() - allocates a chunk of memory

realloc() - decreases or increases amount of space allocated

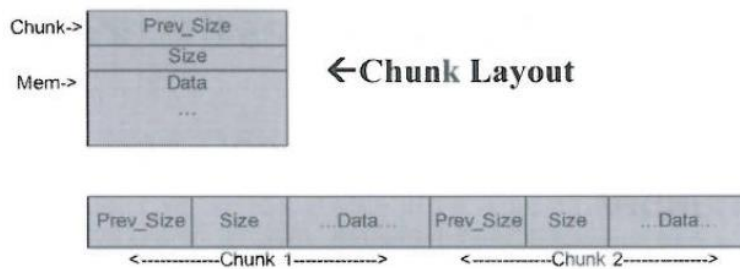
free() - frees the previously allocated chunk

calloc() - initialized data as all 0's

Dmalloc

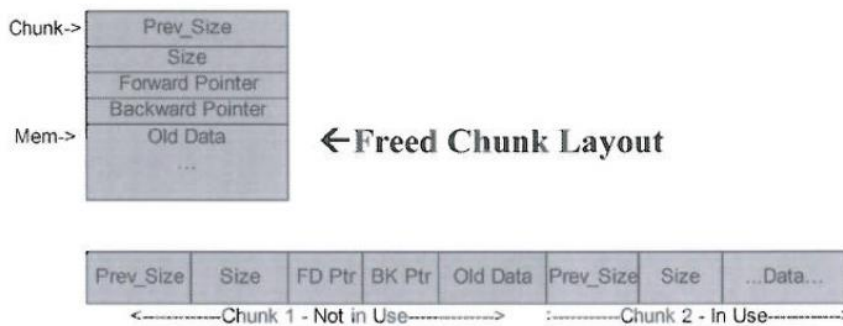
Used by many linux variants as the primary memory allocator

Includes malloc(),realloc(),free() and some utility routines



Adjacent Chunks in Memory

* Concept taken from <http://www.phrack.com/issues.html?issue=57&id=9>

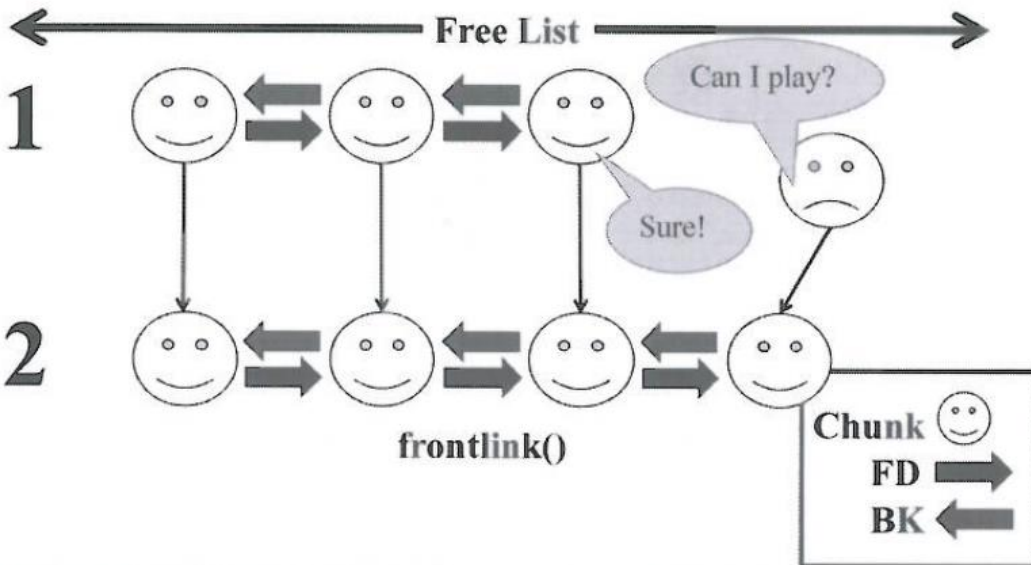
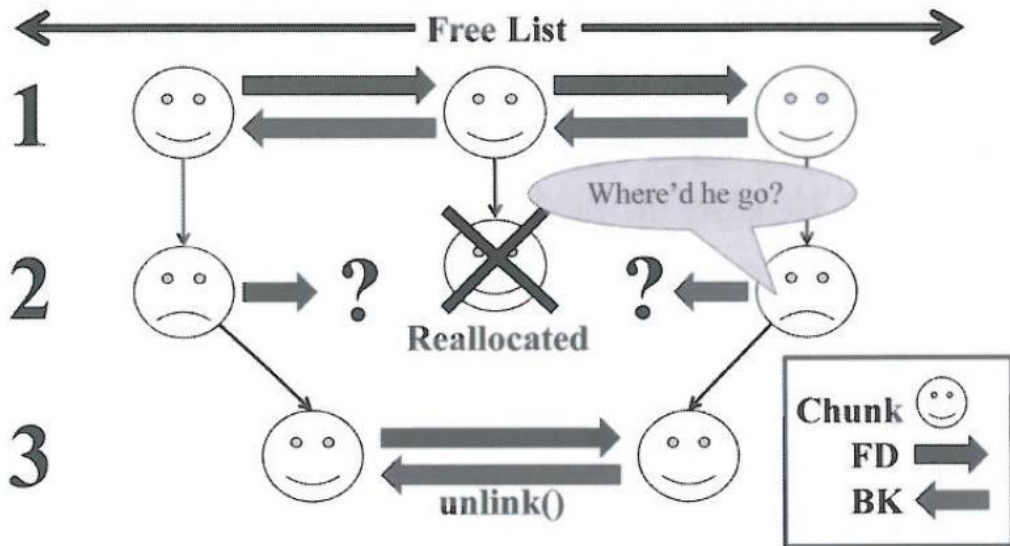


Adjacent Chunks in Memory

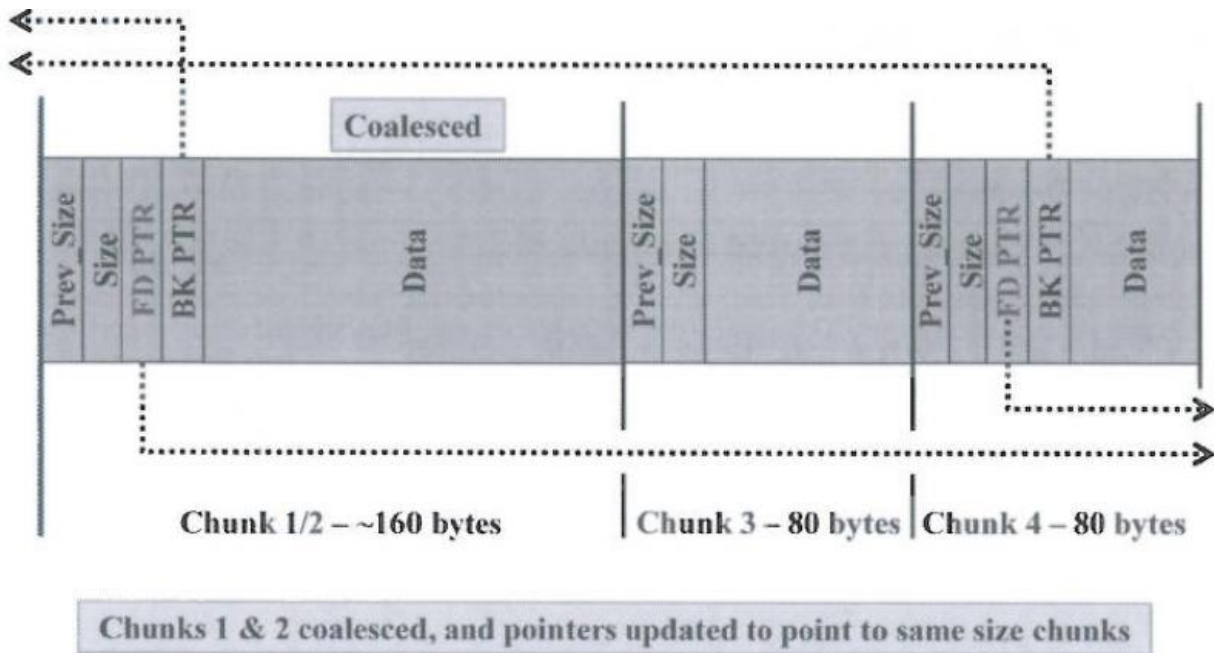
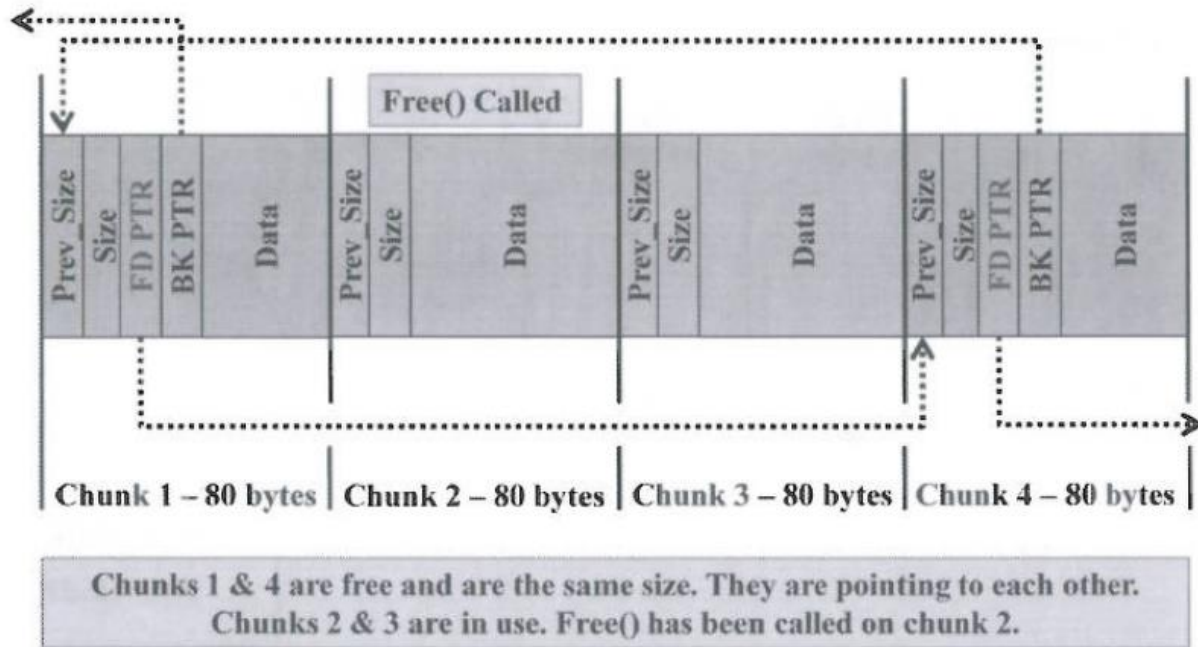
unlink() & frontlink()

The unlink() function removes chunks from a doubly-linked list

The frontlink() function inserts new chunk into a doubly-linked list
 unlink() is called by free() when an adjacent chunk is also unused



Unlink & Coalescing Process



Ptmalloc: based on dlmalloc

Tcmalloc: thread-caching malloc

Jemalloc: jason evan's malloc

Tool: ltrace

Section5: Introduction to Shellcode

Shellcode - code to spawn a shell

Injected into a program during exploitation and serves as the “payload”

System Calls

- Force the program to call functions on your behalf
- Communicate between user mode and kernel mode(ring 0)
- Arguments are loaded into processor registers and an interrupt is made. On 32-bit x86 EAX holds the desired system call number
EBX, ECX and EDX hold arguments usually in alphabetical order
- Each system call must be well-understood prior to writing the assembly code

Common system calls

<http://man7.org/linux/man-pages/man2/syscalls.2.html>

Creating Shellcode

We need to restore rights

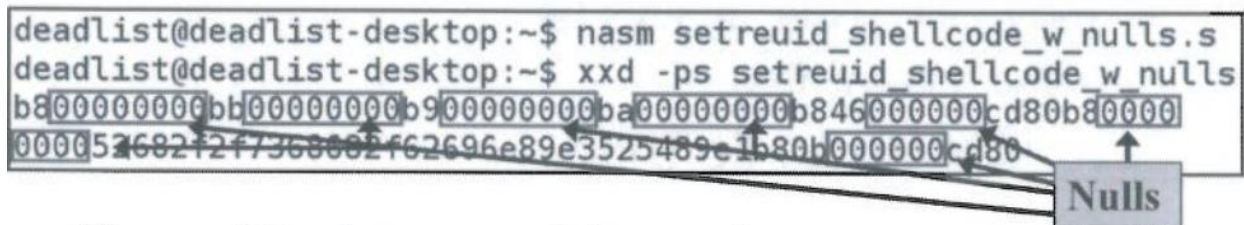
- setreuid() -> system calls

<http://shell-storm.org/shellcode/>

Tool: The Netwide Assembler(NASM)

- Use nasm to assemble ...

```
deadlist@deadlist-desktop:~$ nasm setreuid_shellcode_w_nulls.s
deadlist@deadlist-desktop:~$ xxd -ps setreuid_shellcode_w_nulls
b800000000b00000000b90000000ba00000000b84600000cd80b80000
000052682f2f7368002f62696e89e3525489e1b80b00000cd80
```



- Use xxd to dump machine code
 - -ps flag dumps hex only
- There are many null bytes!
 - Most string functions will fail
- Shellcode is also 56 bytes! Too large!

Remove null bytes:

- Xor eax, eax

- Sub eax, eax
- Mov eax, eax
- Inc eax / dec eax

Section6: Smashing the Stack

Stack Exploitation on Linux

Goals of stack overflows:

- Privilege escalation
- Getting shell
- Bypass authentication
- Overwrite
- Much more...

Steps:

1. Finding privileged programs
2. Trigger a segmentation fault
3. Determine the size of the buffer
4. Redirection execution to granted

<https://medium.com/@maximilianomeyer1/hack-the-box-pwn-little-tommy-429edd9cb105>

Stripped Programs

The strip tool removes symbol tables

Some popular return-to-xxx methods:

- Ret2strcpy & ret2gets
Potentially overwrite data at any location
- Ret2sys
The system() function executes the parameter passwd with /bin/sh
- Ret2plt
Return to a function loaded by the program

Many functions take in arguments that you can place on the stack

Return Oriented Programming

- Rop is the successor to return-to-libc style attack
- Rop can be multi-staged or turning-complete
Injection of code may or may not be required
Jump Oriented Programming(JOP) technique can perform a similar goal through a gadget dispatcher to avoid stack dependency and ESP advancement

Gadget

Are simply sequence of code residing in executable memory, usually followed by a return instruction

Rope without Returns

Using pop instructions and jmp *(reg)'s can achieve the same goal as return

Section7: Advanced Stack Smashing

Linux stack protection

- 4-byte value placed on the stack
- Protect the return pointer(RP), saved frame pointer(SFP) and other stack variables

Canaries and Security Cookies

Linux uses the term canaries and windows uses security cookies

Common type of stack protection:

1-stack smashing protector
Formerly known as prepolice

Integrated with gcc on many platform

Support different type of canaries

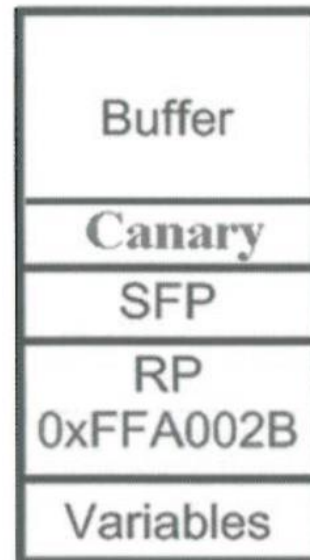
2-stackguard

Integrated with older version of gcc

Uses a terminator canary

Type of canaries

- **Types of Canaries:**
 - Terminator Canary
 - 0x00000aff & 0x000aff0d
 - Random Canary
 - Random 4-byte value protected in memory.
 - HP-UX's /dev/urandom
 - Hash of Return Pointer
 - Null Canary
 - 0x00000000



Defeating stack protection

- Bypassing a terminator canary on ubuntu
- Normally seems to default to \x00000aff
- Some programs have custom canaries
- Overwriting SFP
- Multiple writes with strcpy() or gets()

Real life example: ProFTPD 1.3.0(stack overflow discovered, terminator canary is repaired, aslr is defeated, local and remote exploit version released)

Linux Address Space Layout Randomization(ASLR)

- Stack and heap addressing is randomized
- mmap() is randomized
- Most significant bits are not randomized
- PaX patch will increase randomization

Enable aslr(full randomization): echo 2 > /proc/sys/kernel/randomize_va_space

Enable aslr(conservative randomization): echo 1 > /proc/sys/kernel/randomize_va_space

Disable aslr: echo 0 > /proc/sys/kernel/randomize_va_space

Defeating ASLR

- Data leakage
 - Format string bugs
- Locating static values
 - Not everything is always randomized
 - Procedure linkage table(PLT)

- Providing more bits to the randomization pool increases security

Tool: ldd

Linux-gate.so.1

- Virtual dynamically-linked shared object (VDSO)
- Consistently loaded at 0xffffe000
- Used for virtual system calls
 - A gateway between user mode and kernel mode
 - Work with SYSENTER & SYSEXIT
 - Faster method than invoking int 0x80

Other Opcodes of Interest

- Ret-to-ESP
 - jmp/call esp - 0xffd4 or 0xffe4
- Ret-to-EAX
 - Useful when a pointer is returned via eax to the calling function
 - jmp/call eax - 0xffe0 or 0xffd0
- Ret-to-Ret
 - Return repeatedly down the stack until you control the location
 - Ret instruction - 0xc7**2.6.20 has these in linux-gate.so.1
- Ret-to-Ptr
 - During some sef-fault, registers hold addresses on the stack we can control; e.g. 0x41414141 may show up in a register
 - Ret-to-Ptr in eax, edx, edi, esi, etc.

Checking For BoF

```
deadlist@deadlist-desktop:~$ ./aslr_vuln AAAA
I'm vulnerable to a stack overflow... See if you can hack me!

deadlist@deadlist-desktop:~$ ./aslr_vuln `python -c 'print "A" *100`
I'm vulnerable to a stack overflow... See if you can hack me!

Segmentation fault (core dumped)
```

Checking with GDB

48 A's overflows

```
(gdb) run `python -c 'print "A" *48`
Starting program: /home/deadlist/aslr_vuln `python -c 'print "A" *48`
stack overflow... See if you can hack me!

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) p $esp
$1 = (void *) 0xbf8d6800
```

ESP is changing with each run

Ret2Buffer is unlikely

```
(gdb) run `python -c 'print "A" *48`
Starting program: /home/deadlist/aslr_vuln `python -c 'print "A" *48`
I'm vulnerable to a stack overflow... See if you can hack me!

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) p $esp
$3 = (void *) 0xbf9ad100
```

No Where to Return

- The stack randomized with each run of the program
- System libraries and functions are randomized with each run
- 20-bits is used for randomization
- Brute-force not a good solution

Wrapping the Target Program

- Exec family function(execl(), execlp(), execl(), execv(), execvp())
- Replace the current process image with a new process image

Module5: Exploiting Windows for Penetration Testers

Section1: Introduction to Windows Exploitation

Windows has two access mode:

- Kernel mode: core operating system components, drivers
- User mode: application code, drivers

Kernel memory is shared between processes

32-bit windows provides 2gb of virtual memory to the kernel and 2gb to the user

64-bit windows provides 7tb or 8tb to the kernel and 7tb or 8tb to the user

Windows vs. Linux

- Linking and loading
Elf vs. pe/coff
got/plt vs. iat/eat
- Windows api
Windows application programming interface
- Structured exception handling - seh
Global exception handler
Try and except/catch blocks
- Threading
fork() vs. threads

Linking & Loading - PE/COFF

- Windows object file format
- Two primary formats(executable format, dynamic link libraries(dll))
- Import address table
- Export address table
- .reloc section

pe/coff primary section

- Dos executable file
Mz header - "4d 5a"
- Signature
Pe signature - pe\0\0
- Header
0x014c - intel 386 requirement
- Optional header
0x010b - pe32 format | 0x020b - pe64 format
Image size
Rva offset
Stack and heap requirements
- Section table
Ascii section names(.text, .idata, .rsrc, etc)
Memory location of section
4096 byte boundary alignment
- Lazy linking
Similar to plt and got relationship
Symbols are not resolved until first call

Tool: ollydbg, immunity debugger, pedump

Ollydbg

- Binary analysis
- Register contents, procedures, api calls, patching and more

Immunity debugger

- Free debugger based of of ollydbg
- Extensive development work focused on reverse engineering and exploit development

- Support python scripting
- Combines command-line and gui

PEDUMP

- Pe file examination
- Display pe header
- Display section tables
- Display symbol tables

The Windows API

- Set of compiled functions and services provided to windows application developers
- Provides services such as network services, registry access, command-line services
- You must ask the os to perform most routines

Thread Information Block(TIB)/Thread Environment Block(TEB)

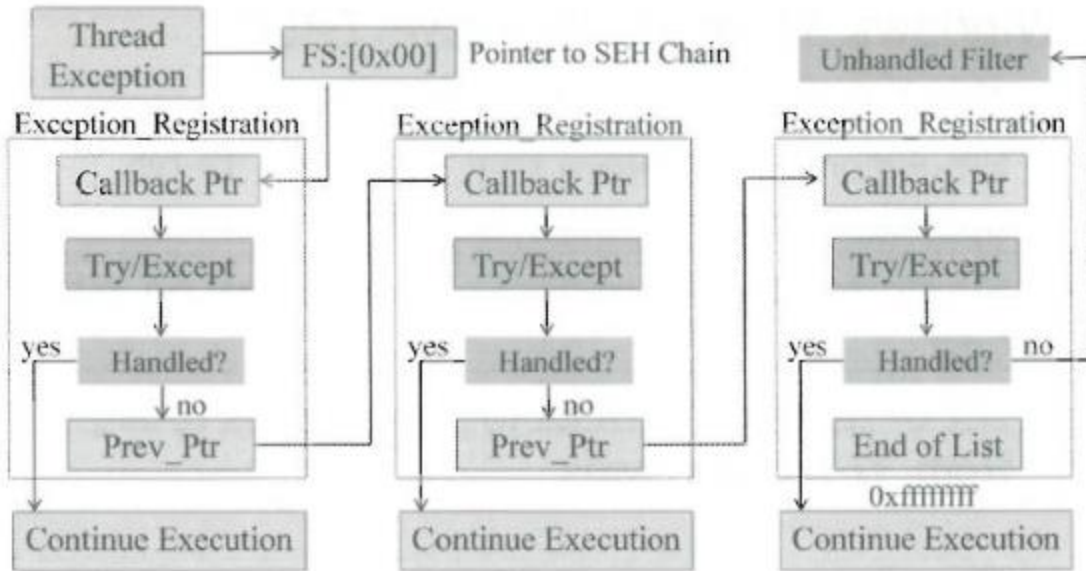
- Store information about current thread
 - FS:[0X00] pointer to seh chain
 - FS:[0X30] address of peb
 - FS:[0X18] address of tib
- Takes away the requirement to make an api call to get structural data
- Each thread has a tib

Process Environment Block(PEB)

- Structure of data with process specific information
 - Image base address
 - Heap address
 - Imported modules
 - Kernel32.dll is always loaded
 - Ntdll.dll is always loaded
- Overwriting the pointer to
 - RL_CRITICAL_SECTION is a common attack
 - The peb is located at 0x7ffdf000 *randomization*
 - 0x7ffdf020 holds the fastpeblock pointer
 - 0x7ffdf024 holds the fastpebunblock pointer

Structured Exception Handling(SEH)

- Callback function
 - Allow the programmer to define what happens in the event of an exception such as print a message and exit of fix issue
- Chain of exception handlers
 - FS:[0x00] points to the start of the SEH chain
 - List of structures is walked until finding one to handle the exception
 - Once one is found, the list is unwound and the exception registration structure at FS:[0x0] points only to the callback handler
- UnhandledExceptionFilter is called if no other handlers handle the exception



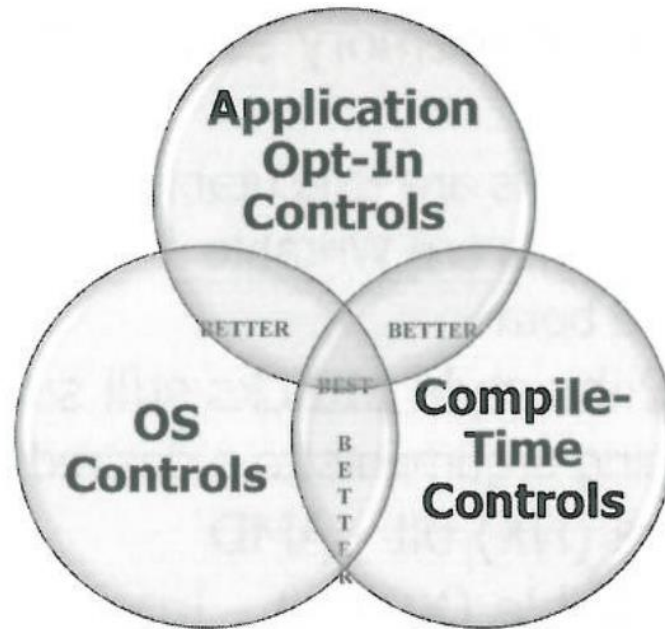
WOW64

Windows 32-bit on windows 64bit

Set of user-mod dlls to handle calls to and from 32-bit processes

Section2: Windows OS Protections and Compiler-Time Control

Exploit mitigation Controls



1-Application Opt-In controls

Exploit mitigation control supported by the os

Include ASLR participation, DEP participation and

2-OS-Controls

Compile program to participate in a control

Include ASLR, hardware DEP, and several others.

3-Compile-Time Controls

Controls that are added in during compile-time

Include canaries or security cookie, application ASLR, SafeSEH, and

Linux Write XOR Execute

Marks areas in the memory as writable or executable

- Code segment are executable
- Data segments are writable
- Cannot be both

No execute (NX) bit - AMD

eXecute disable (XD) bit - Intel

Data Execution Prevention

- Marks pages as non-executable(e.g. stack, heap, raises an exception if execution is attempted)
- Hardware based by setting the execute disable(XD) bit on intel
Amd uses the no execute (NX) bit
- Can be manually disabled in system properties
- Software dep is supported even if hardware dep is not supported
Software dep only prevents seh attacks with safeseh

SafeSEH

- Build a table of trusted exception handlers during compile-time
- Will not pass control to an address that is not in the table
- Third-party programs & dlls may cause a problem

Structured Exception Handling Overflow Protection(SEHOP)

- Adds a validation frame at the bottom of the stack
- Prior to an exception handler being called, the SEH chain is walked to verify that the validation frame is at the end
- Defeating this control would require one to create a fake validation frame in the attack which points to ntdll!FinalExceptionHandler

PEB Randomization(PEB Discussed Shortly)

- The PEB has 16 possible locations
0X7FFD0000,0X7FFD1000,.....,0X7FFDF000
- Randomization runs separately from Address Space Layout Randomization(ASLR) on later version

Heap Cookies

- 8-bits in length (256 possible values)
- Can be guessed 1/256 tries on average
- Placed directly after the "Previous Chunk Size" field

Safe Unlinking

- Similar to the update to early GLIBC unlink() usage on linux; e.g. dlmalloc
- Much better protection than 8-bit cookies
- Combined with cookies and PEB randomization, exploitation is difficult

Low Fragmentation Heap(LFH)

- 32bit cookie
- Allocate blocks in predetermined size ranges by putting block into blocks
128 buckets total

Enhanced Mitigation Experience Toolkit(EMET)

- Adds additional or more strict exploit mitigation controls to the windows os
- Introduces controls such as "Mandatory ASLR"

Windows Kernel Hardening

- First 64kb of memory cannot be mapped, so no more null pointer dereferencing
- Guard pages added to the kernel pool
- Improved aslr
- Kernel pool cookie
- C++ vtable protection or ie
- ROP/JOP protection
- ForceASLR, sehop, more aggressive cookies

Section3: Windows Overflows

mona.py

Pycommand for immunity debugger

Helpful for

- Rop gadget
- Exploit mitigation control scanning(DEP, ASLR, SafeSEH, etc)
- Easily search for trampolines and code reuse

Useful commands

- Update: !mona update
- Search for trampolines and other code reuse blocks: !mona jmp -r esp -m <module_name>
- Search for seh overwrite code sequences: !mona seh -m <module_name>
- Set up the working folder to where output is written: !mona config -set working_folder <PATH/%p>
- Display loaded modules and protections: !mona modules
- Generate a pattern to determine buffer size: !mona pattern_create <N>
- Pattern locator: !mona pattern_offset <pattern>
- Find rop gadget: !mona rop

Finding a “jmp esp” or “call esp”:

Select one of the dll's not participating in aslr or rebase, like Configuration.dll, and run the following command: !mona jmp -r esp -m Configuration.dll

Functions that can disable DEP

- VirtualAlloc() - create new memory region, copy shellcode and execute
- HeapCreate() - same as above, but requires more api chaining
- SetProcessDEPPolicy() - change the dep policy for whole process
- NtSetInformationProcess() - changes the dep policy for process
- VirtualProtect() - change access protection of the memory page where you shellcode resides
- WriteProcessMemory() - write shellcode to a writable and executable location and execute

VirtualProtect() Method

- Rop requires familiarity with the desired function and practice fixing broken chains
- Rop can be used to set up the arguments to VirtualProtect() on the stack or in registers

```
BOOL WINAPI VirtualProtect(
    __in LPVOID lpAddress,
    __in SIZE_T dwSize,
    __in DWORD flNewProtect,
    __out PDWORD lpflOldProtect
);
```

Stack Pivoting

- Method to move the position of esp from the stack to an area such as the heap
xchg/mov esp, eax
Ret
Register such as eax pointing to rop code on the heap, so we pivot esp to take advantage of pop's and push's
- Works hand-in-hand with ROP and JOP
Not always necessary with stack overflows
If doing seh overwrite, you may not have enough space below on the stack to hold all of your code
You can use gadget that subtracts a number of bytes from the stack pointer to get to a location where you have more space

Section 5: Tools to help build gadgets

- Mona.py - an immunity debugger PyCommand
- White phosphorus - Immunity Canvas commercial exploit pack
- Ropeme - linux gadget builder
- Metasploit RopDB - more flexible ROP chains

Section 6: Building a Metasploit Module

Metasploit Template

Sample file: sample.rb

/framework3/documentation/samples/modules/exploits/sample.rb

/opt/metasploit/msf3/documentation/samples/modules/exploits/sample.rb

<https://github.com/lattera/metasploit/blob/master/documentation/samples/modules/exploits/sample.rb>

Section 7: Windows Shellcode

Shellcode on Windows

- Still commonly used to spawn shells

- Can do much more, such as adding user accounts, dll injection, viewing files, meterpreter, etc.

Shellcode is specific to processor type

- X86, arm, powerpc, etc. assembled code

Location of libraries and functions can be tricky on windows

- System calls on linux are consistent, but not on windows
- Changes between oss and service packs can cause problems

Sockets not directly available through system calls

- You must go through an api to load the library and call the appropriate function

Accessing kernel resources

Dlls are loaded into running processes

- We are forced to use the windows api to make system calls
- Kernel32.dll, kernelbase.dll, and ntdll.dll are always loaded, but we must first locate them
- We must also determine a way to wald through the loaded modules eat to find a desired function

Locating kernel32.dll

- Load additional modules with LoadLibraryA() and GetProcAddress()
LoadLibraryA() allows us to load libraries(return a handle to the base address)
GetProcAddress() allows us to get the functions address inside the Dll(base address of the dll holding the function is passed as an argument, as well as the desired function name)
- Process Environment Block(PEB)
The peb holds a list of loaded modules
- SEH unhandled exception handler points to a function within kernel32.dll
- Checkout "Win32 Assembly Components" by the last stage of delerium

Locating GetProcAddress()

- GetProcAddress()'s RVA changes often between os releases and service packs
- We can find this by walking the export address table
- You can walk the table and compare the desired function to the list

Loading Modules and APIs

- Any module can be loaded into the processes address space with LoadLibraryA()
- Specific APIs/Function can be resolved with GetProcAddress()
- You have a portable method to locate the addresses and are not bound to one os or service pack

Multi-stage Shellcode

For when there's not enough space to fit all of your shellcode

1. Execute a first-stage loader
Allocate memory with `VirtualAlloc()`, read additional shellcode coming over the connection, and execute
2. Open sockets can be walked with `getpeername()` in `ws2_32.dll`
Locate the file descriptor
Redirect `cmd.exe` to the existing file descriptor/socket
3. Egg hunting shellcode is a technique to use when you can get additional shellcode to execute loaded somewhere in memory, prepended with a tag

Resources

- [SANS Sec660](#)
- [SANS Sec760](#)
- [CTF](#)
- [HackTheBox](#)