Packet Mining for Sensitive Data & Privacy Leakage

Dave Porcello, PacketBeard
Sean Gallagher, Ars Technica

Dave's Background

- Founded Pwnie Express
- Created Pwn Plug, Raspberry Pwn, etc.
- DARPA CFT projects (Power Pwn)
- Pwnie Labs / R&D initiatives
- Pentesting, research, & training (2017+)

Sean's Background

- IT and Security Editor, Ars Technica
- Worked with Dave on Project Eavesdrop (NPR)
- Former editorial CTO, Ziff Davis Enterprise
- Former Navy officer, information security officer, and government networking contractor
- Long time testing lab rat

Overview

Techniques that expose what an intermediary can discern about an individual or organization through passive monitoring of network traffic

- Sensitive corporate data
- PII/Privacy leakage

Primarily useful for:

- Data leakage assessments
- Privacy research
- Passive pentesting
- Network forensics

How this started

- NPR's "Project Eavesdrop"
- Used a Pwn Plug to tap journalist Steve Henn
- Revealed an excess of personal data
- Found new encryption leakage flaws
- Many fixes, Google security hall of fame

Project Eavesdrop: What we found...

- Passwords
- Phone numbers
- Email addresses
- Visited domains, websites, & countries
- Images, photos, software downloads, SSL certs
- Session keys & cookies

Project Eavesdrop: What we found...

- Search keywords
- Personal interests & shopping habits
- Location data (GPS, etc.)
- VoIP/SIP phone calls
- Cell carrier parameters
- Audio recordings of NPR interviews

Project Eavesdrop: What we found...

- Hardware models & BIOS/firmware versions
- Installed OS/application versions & patch levels (including AV software)
- Running Windows processes, exe/dll versions, & connected USB devices
- MAC addresses, internal IPs, & UUIDs

2018: Everything's encrypted now, right?

- Nope.
- 62% of Alexa top million = HTTP default
- Includes hundreds of .gov sites
- Includes many mobile apps, MSNBC, Bing

** https://scotthelme.co.uk/alexa-top-1-million-analysis-february-2018/*
** https://pulse.cio.gov , https://securethe.news/sites/*

What we'll cover today

- Passive monitoring techniques only (no MITM, crypto breaking, etc.)
- GUI & cmd line methods
- Hands-on exercises
- Free, open-source tools

Prerequisites

- Kali Linux (native or VM)
  (Other distros *should* work)
- Linux cmd line experience
- Some Wireshark experience

Getting the capture file

https://github.com/packetrat/packethunting

- Also, list of commands used in this presentation (copy/paste!)

Setting up your hunting environment

Install mining tools:

```
# apt install ngrep tcpflow html2markdown
ssldump dsniff tshark
```

Set a variable for your capture file name:

```
# CAPFILE=CaptureFile.pcap
```

Tcpdump: Local capture

tcpdump -vvv -nn -i eth0 -w output.cap

Reading a capture:

tcpdump -vvv -nn -r output.cap

-vvv = very verbose

-nn = don't resolve hostnames or services

Tcpdump: Remote capture through ssh!

Capture on remote host's eth0:

```
# ssh dave@10.0.0.10 'sudo tcpdump -vUnni eth0 -w -' > output.cap
```

-U = packet buffering to avoid packet loss

Tcpdump: Basic filters

Show HTTP traffic on port 80:

```
# tcpdump -vvvAnn -i eth0 port 80
```

Show SMTP/POP3 traffic for specific host:

```
# tcpdump -vvvAnn -i eth0 'host 10.0.0.10 and port (25 or 110)'
```

Tcpdump: Basic filters

Saving only DNS traffic to a new file:

```
# tcpdump -r $CAPFILE -w dns-only.cap port 53
```

- Very helpful when dealing with large cap files!

Tcpdump: Other useful options

-C = stop capturing when cap file reaches size

-e = show MAC addresses

-A = show ASCII packet data

-X = show raw packet data in hex/ascii

-I = put interface in wireless monitor mode

ngrep: network grep!

Grep live network traffic for "password"

```
# ngrep -d eth0 -q -t -i 'password'
```

-q = quiet mode

-t = readable timestamps

-i = ignore case

ngrep: network grep!

Grep for HTTP GET/POST requests:

```
# ngrep -d eth0 -W byline -q -t '^(GET|POST)' port 80
```

More options:

-W byline = honors embedded linefeeds

-I = read from cap file instead of live interface

-v = inverse string matching (just like grep)

tcpflow: flow & file extraction

Print ASCII packet data to console:

```
# tcpflow -c -s -r $CAPFILE
```

Extract all flows, objects, & files to output folder:

```
# mkdir tcpflow
# tcpflow -a -r $CAPFILE -o tcpflow/
```

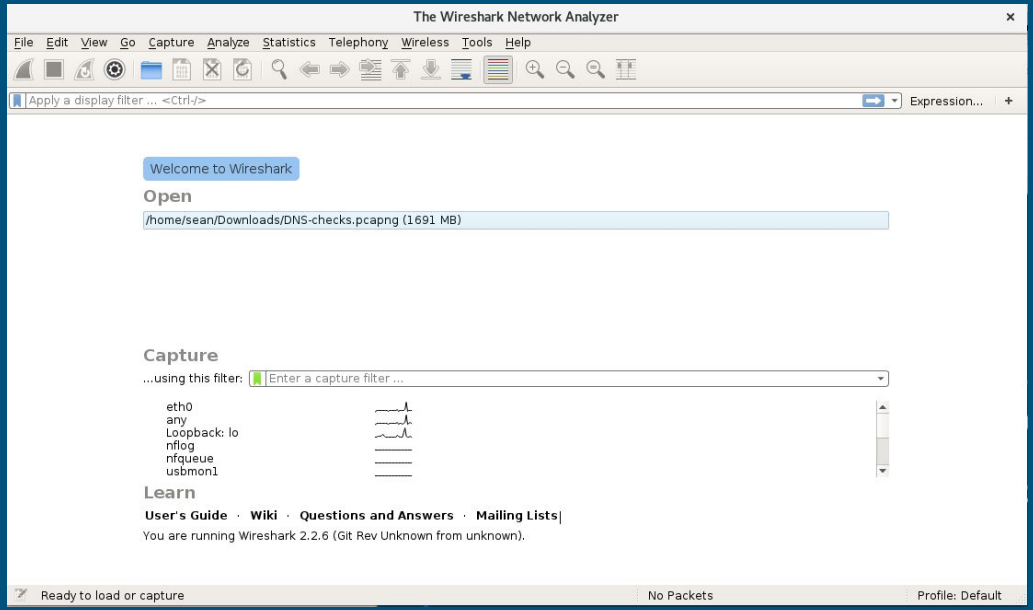
tcpflow: HTML compression/encoding

- HTML body content is often compressed/encoded
- In-line searching with ngrep/tcpflow won't work
- Use "-a" to export decompressed/decoded HTML
- Then, search HTTP body files with grep:

```
# grep 'something' tcpflow/*HTTPBODY*
```

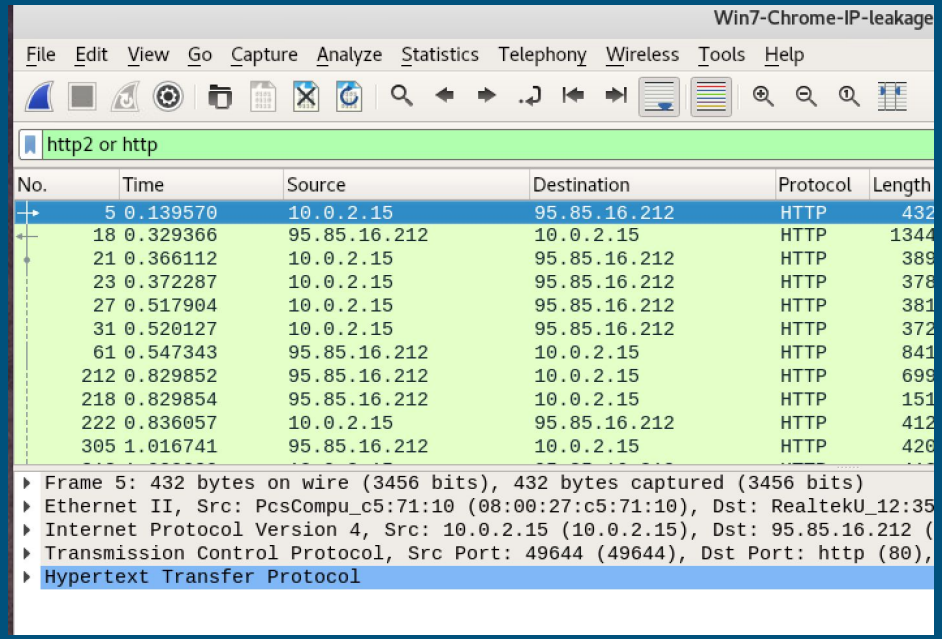
Wireshark basics - capture

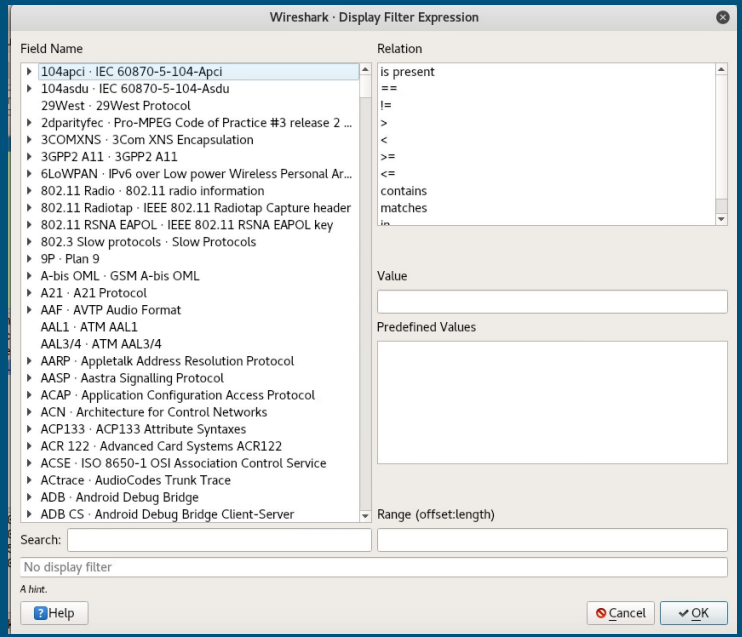
- Select an interface
- Set a filter for desired traffic
- Click the fin



Wireshark - display filtering

Can use protocol names or build boolean/perl regular expressions

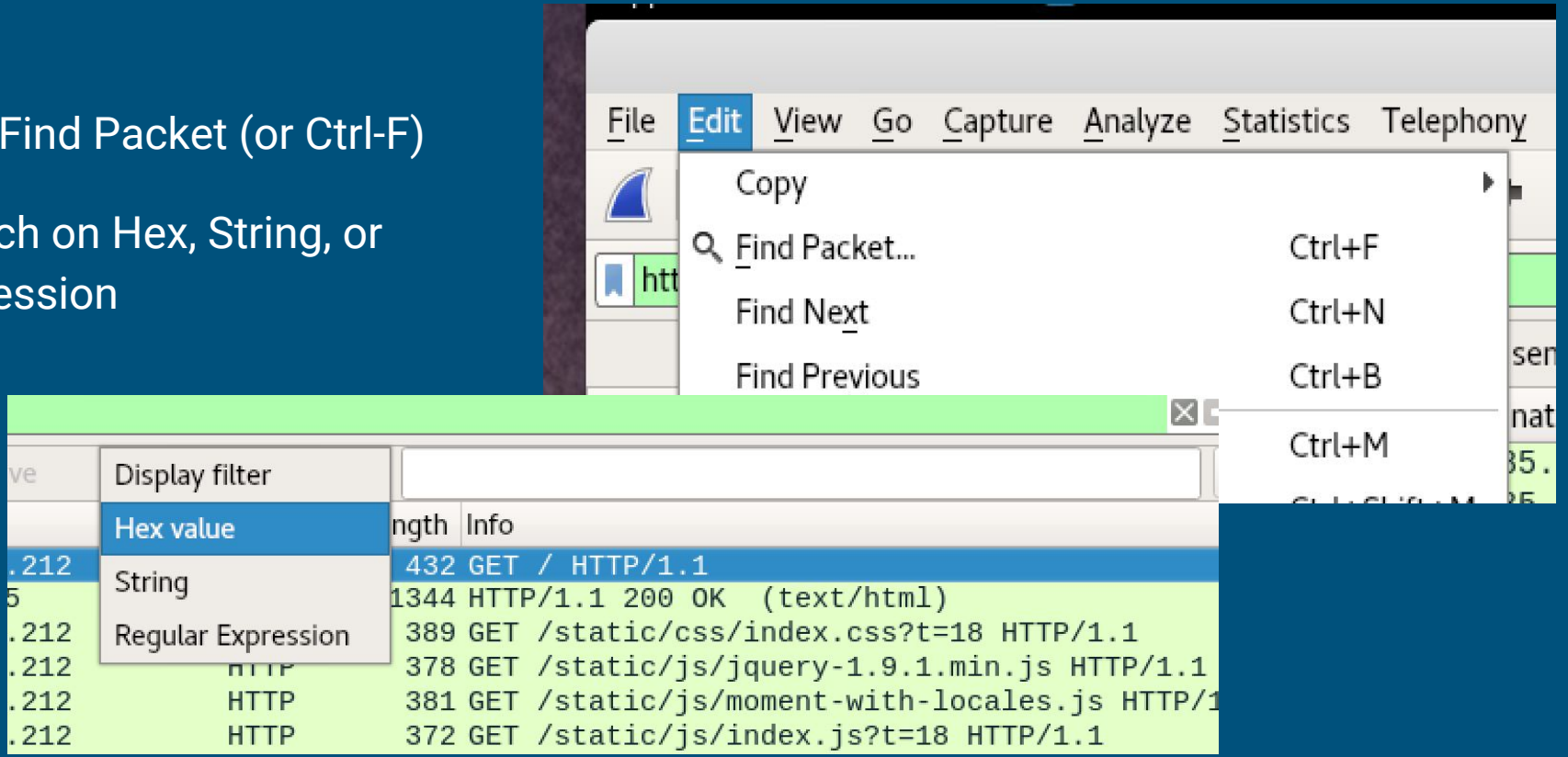




Wireshark search

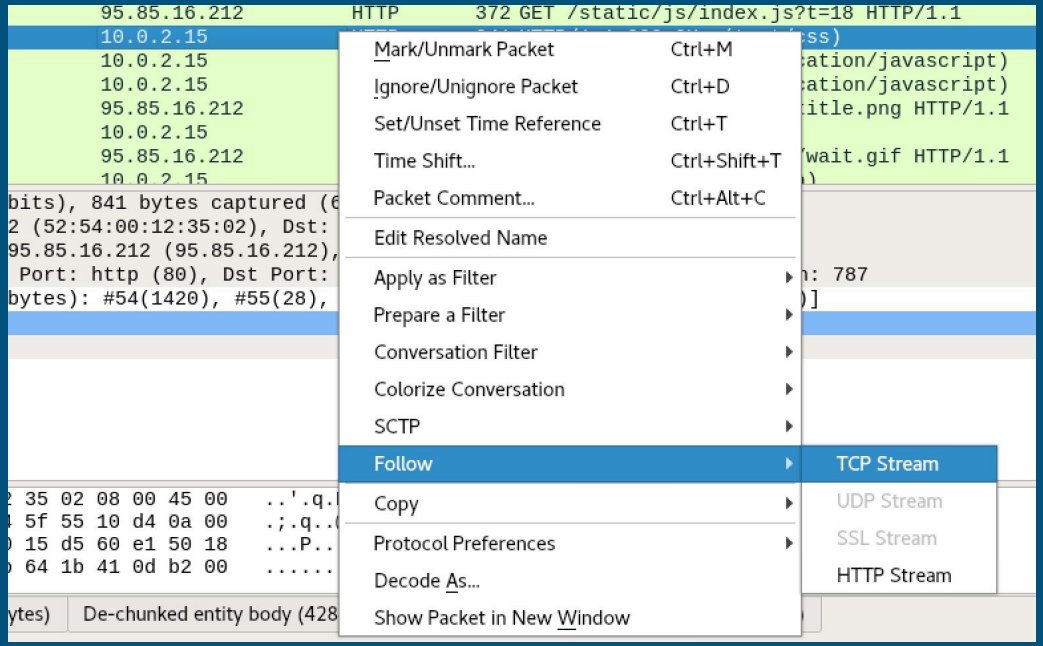
Edit/Find Packet (or Ctrl-F)

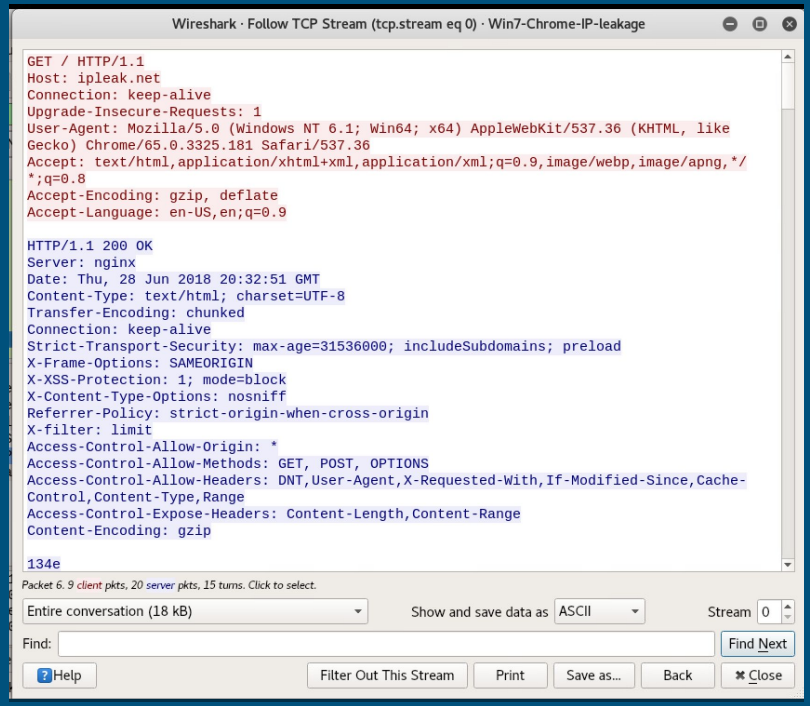
Search on Hex, String, or expression



Wireshark: Following streams

Right click on packet in stream





Connection stats: Top IPs

Top 10 source IPs:

```
# tcpdump -nn -r $CAPFILE |grep " IP " | awk '{print$3}'
|cut -d. -f -4 |sort |uniq -c |sort -nr |head
```

Top 10 destination IPs:

```
# tcpdump -nn -r $CAPFILE |grep " IP " | awk '{print$5}'
|cut -d. -f -4 |sort |uniq -c |sort -nr |head
```

Connection stats: Top connection pairs

```
# tcpdump -nn -r $CAPFILE |grep " IP " | awk
'{print$3,$4,$5}' |sort |uniq -c |sort -nr |head
```

- Note: Based on packets (not true "flows")
- Bro/Argus/Netflow is better suited for flow stats

Connection stats: Top protocols/ports

Top IP protocols:

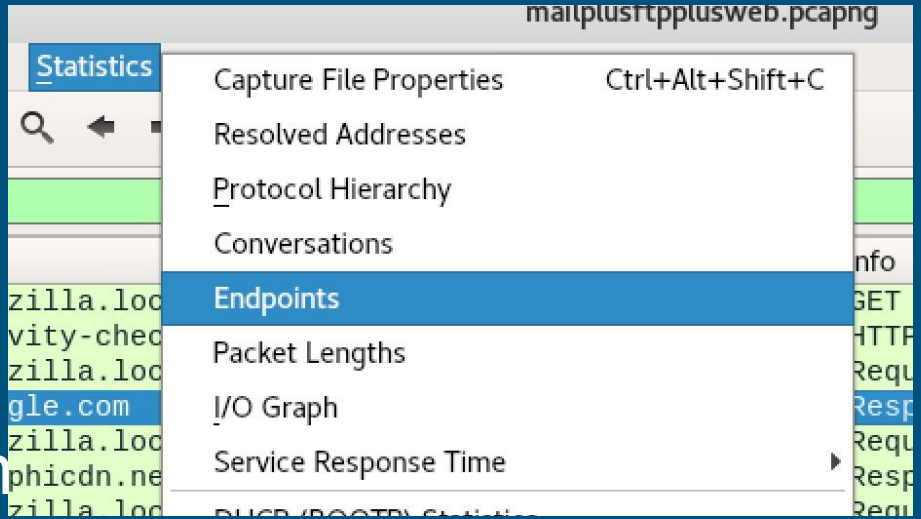
```
# tcpdump -nn -v -r $CAPFILE |grep " IP " |awk -F,
'{print$6}' |sort |uniq -c |sort -nr
```

Top 10 destination ports:

```
# tcpdump -nn -r $CAPFILE |grep " IP " | awk '{print$5}'
|cut -d. -f 5- |sort |uniq -c |sort -nr |head
```

Connection stats (Wireshark)

- Endpoints: top network connections
- Protocol heirarchy
- Conversations—IP address pairs
- Source and destination sort by count of packets



DNS digging

Top domains:

```
# tcpdump -nn -r $CAPFILE port 53 | egrep " A\? " | awk '{print$8}' | egrep -io "[a-z0-9]*\.[a-z]*\.$" | sort | uniq -ic | sort -nr | head
```

Top subdomains:

```
# tcpdump -nn -r $CAPFILE port 53 | egrep " A\? " | awk '{print$8}' |sort |uniq -c |sort -nr |head
```

DNS digging (Wireshark)

Resolved addresses (gives list of all host names)

View by destination, resolve names

NetworkMiner tool for additional crunching

Private IP leakage

Grep for private IPs in packet data:

```
# ngrep -q -t -W byline -I $CAPFILE
'10\.([0-9]{1,3}\.){2}[0-9]{1,3}|192\.168\.[0-9]{1,3}\.[0-9
]{1,3}|172\.([0-9]{1,3}\.){2}[0-9]{1,3}'
```

- ipleak.net (private IP leakage via STUN/WebRTC)

MAC address leakage

Grep for MACs in packet data:

```
# ngrep -q -t -W byline -I $CAPFILE
'([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-F])'
```

Profiling HTTP traffic: Top sites

Top 10 websites:

```
# ngrep -I $CAPFILE -W byline -q -t '^(GET|POST)' port 80 | grep "^Host:" | sort |uniq -ic |sort -nr |head
```

Top 10 referrers:

```
# ngrep -I $CAPFILE -W byline -q -t '^(GET|POST)' port 80 | egrep "^Referer: " |sort |uniq -ic | sort -nr |head
```

Profiling HTTP traffic: GETs & POSTs

Top 10 GET requests (URLs):

```
# ngrep -I $CAPFILE -W byline -q -t '^(GET )' port 80 |
grep "^GET " | sort |uniq -ic | sort -nr |head
```

HTTP POSTs (& POST data!):

```
# ngrep -I $CAPFILE -W byline -q -t '^(POST )' port 80 |
egrep "^POST|^<|^[a-z]"
```

Profiling HTTP traffic: URL timeline

URL log with timestamps:

```
# ngrep -I $CAPFILE -W byline -q -t '^(GET|POST)' port 80 |egrep "^T |^(GET|POST)|^Host:|^$"
```

- For pentesters, look for URLs on local web servers! (easier/stealthier than URL bruteforcing)

HTTP traffic: Cookies, Session IDs, etc.

Unique cookies:

```
# tcpflow -r $CAPFILE -c -s port 80 | grep -v "\.\." | grep "^Set-Cookie" | sort |uniq
```

Unique session IDs/UUIDs:

```
# tcpflow -r $CAPFILE -c -s port 80 | grep -v "\.\." | egrep -i "session.id|sessionid|session.token|SESSID|UUID|oauth|Authorization:"    ### Add some --color if needed!
```

Cookies & Session IDs with WireShark

Display filter: http.cookie

Search on text strings for “cookie”, other text



Extracting objects/files

Extract all objects/files & decode HTML:
tcpflow -a -r $CAPFILE -o tcpflow/

Breakdown by file type:
find tcpflow/ |egrep -o "\.[a-zA-Z]*$" |sort |uniq -ic |sort -nr

- Look for images, photos, audio/video files, html files, xml, pdfs, executable files, etc.

Files without extensions

- Some files may not have extensions (execs, etc.)

Example:
tcpflow/081.017.020.050.00080-010.000.002.015.52698-HTTPBODY-001

- Use file, strings, & wine to investigate

Extracting objects/files (Wireshark)

Export HTTP objects: can extract text, scripts and images

- Clicking on object takes you to drill down on packets
- Can export and save any object, file, image

User-Agent profiling

```
# ngrep -I $CAPFILE -W byline -q -t port 80 | egrep
"^User-Agent: " |sort |uniq -ic | sort -nr
```

- Look for browser versions, plugins/extensions, OS versions, & desktop apps (including AV)

User-Agent profiling: Going further

Based on "Avast" in our output, let's try:
ngrep -I $CAPFILE -W byline -q -t -i 'avast' port 80

- Avast stats traffic lists other security software, Windows Firewall status, private IP, hostname, OS/cpu details... thanks!

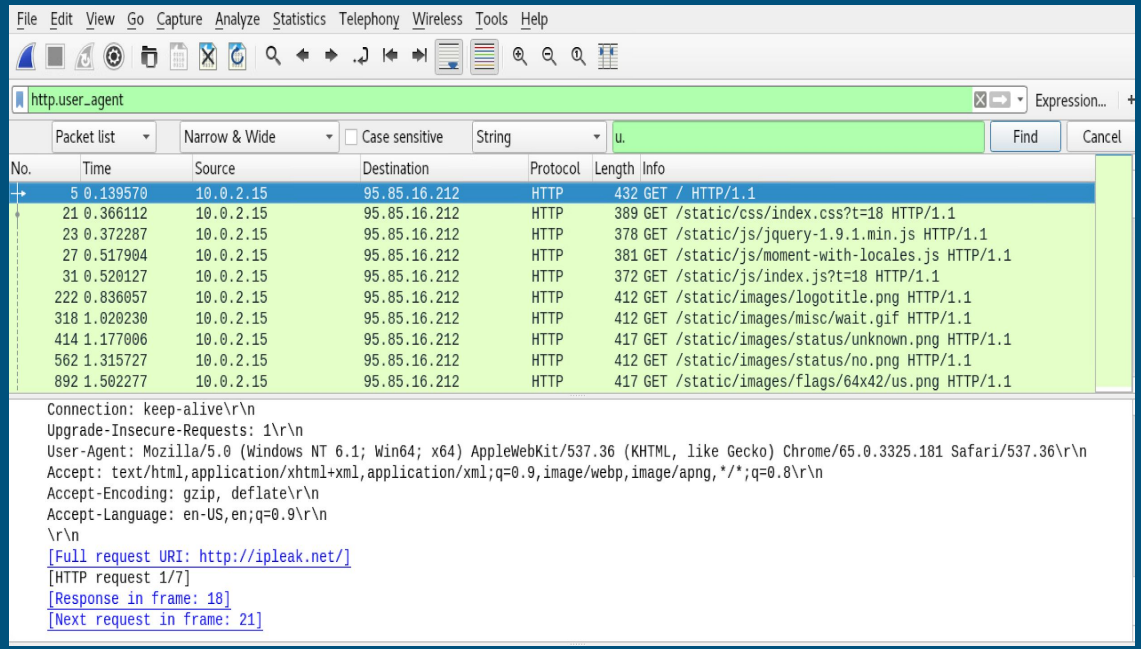
Avast/AVG stats traffic

ScAsAvastStatus=on
ScAsOtherList=AVG Antivirus,Windows Defender,
ScAsOtherStatus=on,off,
ScFwOtherList=Windows Firewall,
ScFwOtherStatus=off,
lan_addr=grep-PC
lan_ip=10.0.2.15
os=win,6,1,2,7601,1,AMD64
version=18.5.3059

User-Agent profiling (Wireshark)

Filter using http.user_agent to show all instances-gives browser/OS information

Can filter on specific agent with “contains” expression (http.user_agent contains “Mozilla”)



Content profiling: Search keywords

Bing search keywords:

```
# ngrep -I $CAPFILE -W byline -q -t
'^(GET|POST|HTTP/)' port 80 | egrep 'Referer:
http:\/\/www.bing.com\/search\?q=' |sort |uniq
```

Content profiling: URL "keywords"

URL "keyword" strings:

```
# ngrep -I $CAPFILE -W byline -q -t '^(GET|POST)' port 80 | egrep "^GET |^POST |^Referer: " | egrep -o "[a-z-]*" | egrep "[a-z-]*-[a-z-]*-" | egrep -v "(^-|-$)" |sort | uniq -ic |sort -nr |head
```

Content profiling: HTML content

Top words from HTML:

```
# cat tcpflow/*.html |html2markdown | egrep -o '\w{4,}' |sort |uniq -c |sort -nr |head -n25
```

- Great way to build a CeWL-style targeted wordlist!

Content profiling in Wireshark: search terms

Sample display filters:

http.request.uri contains “search”

Http.referrer contains “bing”

Http.request.uri contains “keyword”

Personal contact info: email addresses

Email addresses with common TLDs:

```
# tcpflow -r $CAPFILE  -c -s | egrep -i --color
'\w+@[a-zA-Z_]+?\.(com|org|net|gov|mil|edu|co|biz|info)'
```

- Regex for *any* TLD (more false positives):

```
'\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6}'
```

Personal contact info: phone numbers

"Dashed" phone numbers:

```
# tcpflow -r "$CAPFILE" -c -s port 80 | grep --color -P
"\d{3}-\d{3}-\d{4}"
```

- Dashed *or* dotted numbers (more false positives):

```
"\d{3}[-.]\d{3}[-.]\d{4}"
```

Personal contact info (Wireshark)

Search packets using [A-Z0-9._%+-]+@[A-Z0-9.-]+.[A-Z]{2,4} for e-mail addresses

[0-9][3]+-[0-9][3]+-[0-9]{4} for phone numbers

Filter using protocols to narrow false positives

Extracting emails

Extract to console:

```
# tcpflow -c -s -r $CAPFILE port 25 or port 110
```

Extract to files:

```
# tcpflow -a -r $CAPFILE port 25 or port 110  -o tcpflow/
```

Email headers, apps, & AV scanners

Senders, recipients, & email subjects:

```
# ngrep -q -t -W byline -I $CAPFILE port 25 or port 110 |egrep
"^To:|^From:|^Subject"
```

Email client apps & AV scanners:

```
# tcpflow -c -s -r $CAPFILE port 25 or port 110 |egrep -A1
"^User-Agent:|X-Antivirus" |sort -u
```

Extracting emails (Wireshark)

NetworkMiner is your best option here. Extracting emails and attachments in Wireshark is labor intensive-- grabbing data, decoding base64.

Password hunting!

FTP, Telnet, SMTP, POP3, HTTP, etc:

```
# ngrep -I $CAPFILE -W byline -q -t | egrep --color
"[Pp]assword[=:]|[Pp]ass=|[Ss]ecret=|pwd=|^PASS|^US
ER |^AUTH |login:|^Authorization:"
```

Versus dsniff:

```
# dsniff -p $CAPFILE
```

Password hunting!

Decoding HTTP Basic auth, SMTP, POP3 (base64):

```
# echo 'QWxhZGRpbjpPcGVuU2VzYW1l'  | base64 -d
```

Finding SNMP community strings:

```
# tcpdump -A -nn -r $CAPFILE port 161
```

Password hunting in SMTP/ POP (Wireshark)

Wireshark doesn't decode Base64 post-capture, but you can decode during capture:

```
tshark -o smtp.decryption:TRUE -T fields -e frame.number -e smtp.auth.username -Y smtp.auth.username -r crim.pcap
```

Display filters:

frame matches "(?i)auth plain" (will find auth or AUTH in packets)

SMTP contains "AUTH PLAIN" (base64 encoded username/password)

POP contains "AUTH PLAIN"

Password hunting in FTP, HTTP (Wireshark)

ftp.request.arg (shows all FTP requests; USER and PASS will be displayed)

http matches "[Pp]assword[=:]|[Pp]ass=|[Ss]ecret=|pwd=|^PASS|^USER |^AUTH |login:|^Authorization:" (content may be base64 encoded)

Digging for PII

Credit card numbers:

```
# tcpflow -c -s -r $CAPFILE | grep -P --color
'(6011|5[1-5]\d{2}|4\d{3}|3\d{3})[- ]\d{4}[- ]\d{4}[- ]\d{4}'
```

Social security numbers:

```
# tcpflow -c -s -r $CAPFILE | grep -P --color
'([0-6]\d\d|7[0-256]\d|73[0-3]|77[0-2])[- ]\d{2}[- ]\d{4}'
```

Digging for PII/confidential data

DOB/License/Passport numbers:

```
# tcpflow -c -s -r $CAPFILE | grep -v Cookie |egrep --color
'DOB[:= ]|[Pp]assport[:= ]|[Ll]icense number'
```

Classified/tagged documents:

```
# tcpflow -c -s -r $CAPFILE | grep -v Cookie |egrep --color -i '
CONFIDENTIAL | PROTECTED | INTERNAL USE ONLY | TOP
SECRET '
```

Digging for PII (Wireshark)

Searching for credit card numbers & SSNs: regular expressions

Hardware/mobile device profiling

Device manufacturers/models:

```
# ngrep -I "$CAPFILE" -W byline -q -t port 80 | egrep --color
"device_name=|device_type=|os_version=|dev=|X-Device-Info
:|Device:|DEVICE:|deviceId=|deviceModel=" |sort |uniq
```

Cell carrier parameters:

```
# ngrep -I "$CAPFILE" -W byline -q -t port 80 | egrep --color
"mcc=|mnc=|csc="
```

Decoding Apple iDevice plist files

- Contains cell carrier parameters, update traffic, etc.
- Extracted as XML files (via tcpflow, etc.)
- Use plistutil to decode:

```
# apt install libplist-utils
# plistutil -i <plistfile>
```

Location tracking data

GPS coordinates (Apple weather app/Wunderground/etc):

```
# ngrep -I "$CAPFILE" -W byline -q -t '^(GET|POST|HTTP/)'
port 80 |egrep "%2Clatitude%2|maxlat=|latitude="
```

Via Windows default weather app:

```
# ngrep -I "$CAPFILE" -W byline -q -t 'weather.microsoft.com'
port 80 |egrep --color "DisplayName="
```

Amazon app traffic = not encrypted..

Kindle app traffic ("key=" indicates ASIN of each ebook)
ngrep -q -t -I kindle.pcap -W byline | grep --color 'type="EBOK" key='

Prime video app traffic:
ngrep -q -t -I kindle.pcap -W byline | grep 'Prime%20Video%22%2C%22vn'

Mobile device profiling (Wireshark)

Finding mobile device details (OS, app versions, update traffic, cell carrier info, etc.)

Windows error reporting traffic

Hardware vendor, model, BIOS/firmware versions, running processes, exe/dll versions, & connected USB devices...

```
# ngrep -I "$CAPFILE" -W byline -q -t '^(GET|POST)' port 80 |egrep "^T |^GET|^Host:" |egrep -B2 "watson.microsoft.com.$"
```

- Newer/patched Windows versions now use TLS

Inspecting SSL traffic

Extract SSL certificates with tcpflow:

```
# tcpflow -a -r $CAPFILE -o tcpflow/ port 443
```

Extract SSL websites via Server Name Indication (SNI):

```
# ngrep -I $CAPFILE -q -t -W byline port 443 |egrep -o
"[a-z0-9]*\.[a-z0-9]*\.(com|org|net|gov|mil|edu|co|biz|info)"
|sort -u
```

SSL traffic: Finding weak SSL sessions

Sessions using weak cipher suites:

```
# ssldump -n -r $CAPFILE | grep "cipherSuite" | egrep -i
"RC4|MD5|EXP|NULL|_DES|ANON|64"
```

Sessions using weak SSL protocol versions:

```
# ssldump -n -r $CAPFILE | grep Version |sort -u
```

Inspecting SSL traffic (Wireshark)

Extracting SSL certificates, SNI hostnames,

Ssl.handshake.certificate (gives you every cert passed)

Ssl.handshake.extensions_server_name (SSI data)

Inspecting SSL traffic (Wireshark), cont'd

Identifying weak SSL sessions:

Ssl.handshake.ciphersuites (or ciphersuite) shows all crypto offered during client hello.

Can add known server certs to Wireshark config to decrypt traffic.

Decrypting SSL traffic..

.. using a known private key :)

```
# tshark -nn -t ad -r $CAPFILE -o
"ssl.keys_list:,443,http,server-key.pem" -V -R http
```

- Capture must contain the initial TLS handshake between client/server

Packet snooping mitigation tools

- VPNs! *(make sure ALL traffic is VPN routed!)**
- SSH proxy through your home/office*
- Enforce SSL/TLS wherever possible
- HTTPS Everywhere browser extension
- DNS-over-TLS

** NOTE: VPNs/proxies do NOT provide end-to-end encryption!*

Additional resources

http://packetlife.net/library/cheat-sheets/
https://wiki.wireshark.org/SampleCaptures
http://packetlife.net/captures/
https://github.com/chrissanders/packets
https://www.honeynet.org/challenges
https://media.defcon.org/

Questions / Open mining

- Additional questions?

Remainder of workshop:

- Open mining using these tools & techniques
- Snoop on your own traffic, see what you find!

Thank you!

How to find us:

- Dave: dave [at] rbtechvt.com / @DavePorcello
- Sean: @thepacketrat