Packet Mining for Sensitive Data & Privacy Leakage

Dave Porcello, PacketBeard Labs
Sean Gallagher, Ars Technica

Dave's Background

- PacketBeard Labs: Independent security research, packet mining, data leakage assessments, training
- Adjunct professor at Norwich University
- Founded Pwnie Express (2010)
- Pwnie Labs / R&D initiatives
- Created Pwn Plug, Raspberry Pwn, Power Pwn (DARPA)

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, etc.
- Intermediaries include your ISP, your cell carrier, governments, the dude at your local coffee shop, etc.

Primarily useful for:

- Data leakage assessments
- Privacy research
- Passive pentesting
- Network forensics
- Compliance audits (PCI, privacy regs, etc.)

How this started

- NPR's "Project Eavesdrop"
- Used a Pwn Plug to tap journalist Steve Henn
- Passive monitoring only (No MITM, crypto attacks)
- Revealed an excess of personal data
- Found new encryption leakage flaws
- Many fixes, Google security hall of fame

https://www.npr.org/sections/alltechconsidered/2014/06/10/320347267/project-eavesdrop-an-experiment-at-monitoring-my-home-office

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/*

Today's agenda

- Setting up your hunting environment
- Command line mining methods (Dave)
- GUI mining techniques (Sean)
- Hands-on "open mining" using the techniques
- Snoop on your own traffic, see what you find!

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 mining environment

Install mining tools:

```
# apt update && apt install ngrep tcpflow xplico
ssldump dsniff tshark p0f pads python-html2text
```

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: Save filtered traffic to a new file

Example 1: Save only DNS traffic to a new file:

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

Example 2: Exclude all SSL traffic:

```
# tcpdump -r $CAPFILE -w no-ssl.cap not port 443
```

- 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!

Print live web traffic to console:

```
# ngrep -d eth0 -W byline -q -t port 80
```

-d = specifies interface (use -I to read from cap file)

-W byline = honors embedded linefeeds

-q = quiet mode

-t = show timestamps

ngrep: Simple matching/regex

Grep live network traffic for "password":

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

Grep for HTTP GET/POST requests:

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

-i = ignore case

-v = inverse matching (just like grep)

ngrep: Dealing with pcap-ng

- Ngrep doesn't support pcap-ng captures
- May receive "invalid interface capture length" error
- Use "file" command to determine capture format

To convert pcap-ng to pcap format:

```
# tshark -F pcap -r capture.pcapng -w capture.pcap
```

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/
```

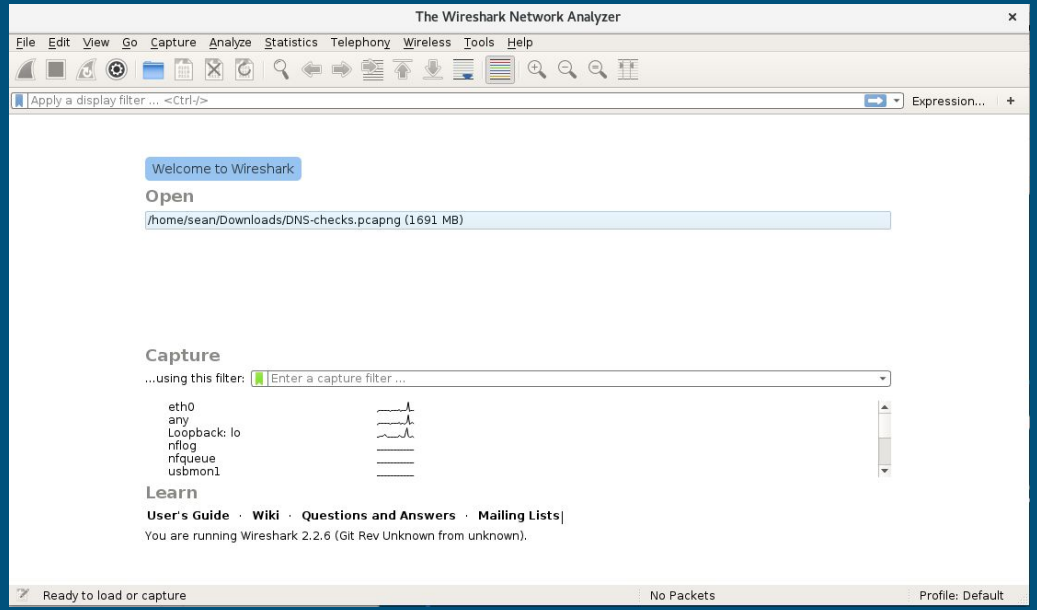
tcpflow: HTML compression/encoding

- 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*
```

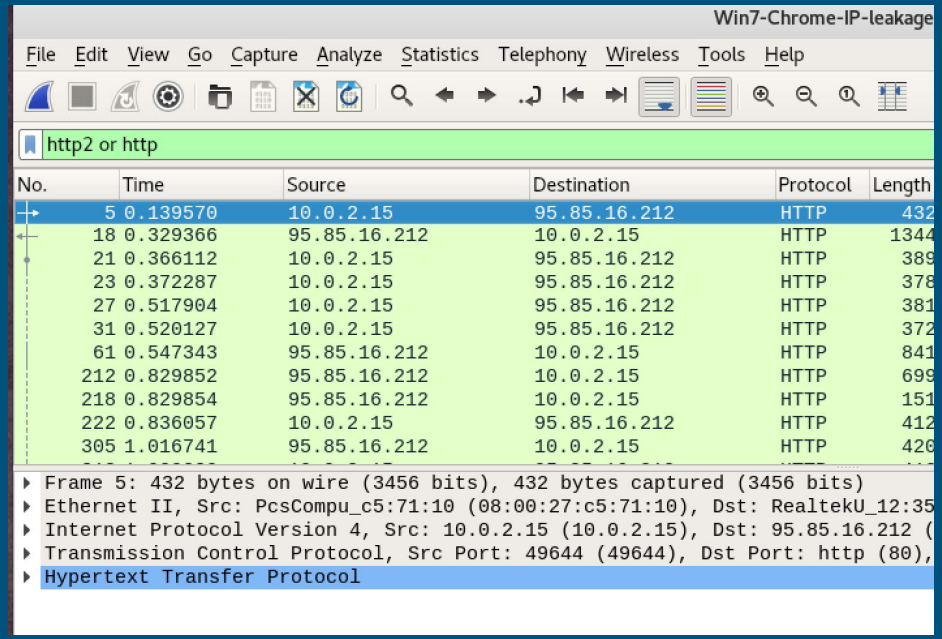
Wireshark basics - capture

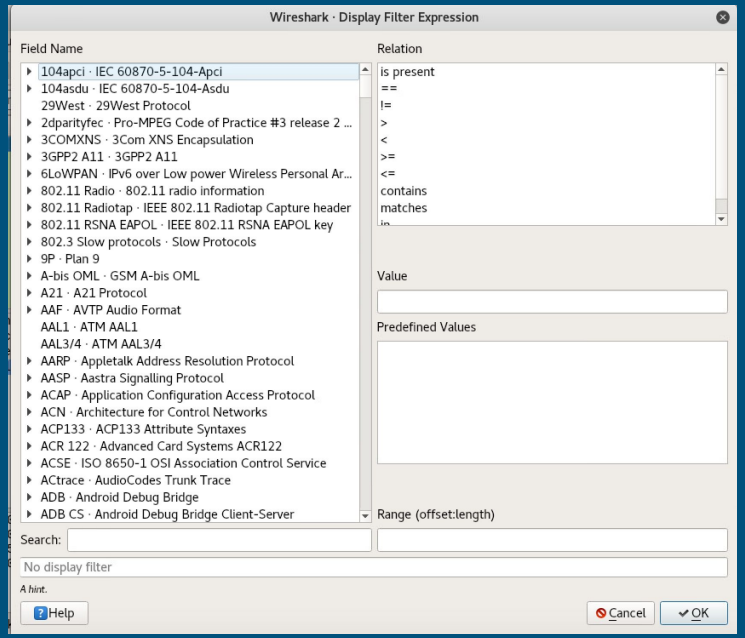
- Select an interface
- Set a filter for desired traffic
- Click the fin



Wireshark - display filtering

Can use protocol names or build boolean/perl regular expressions

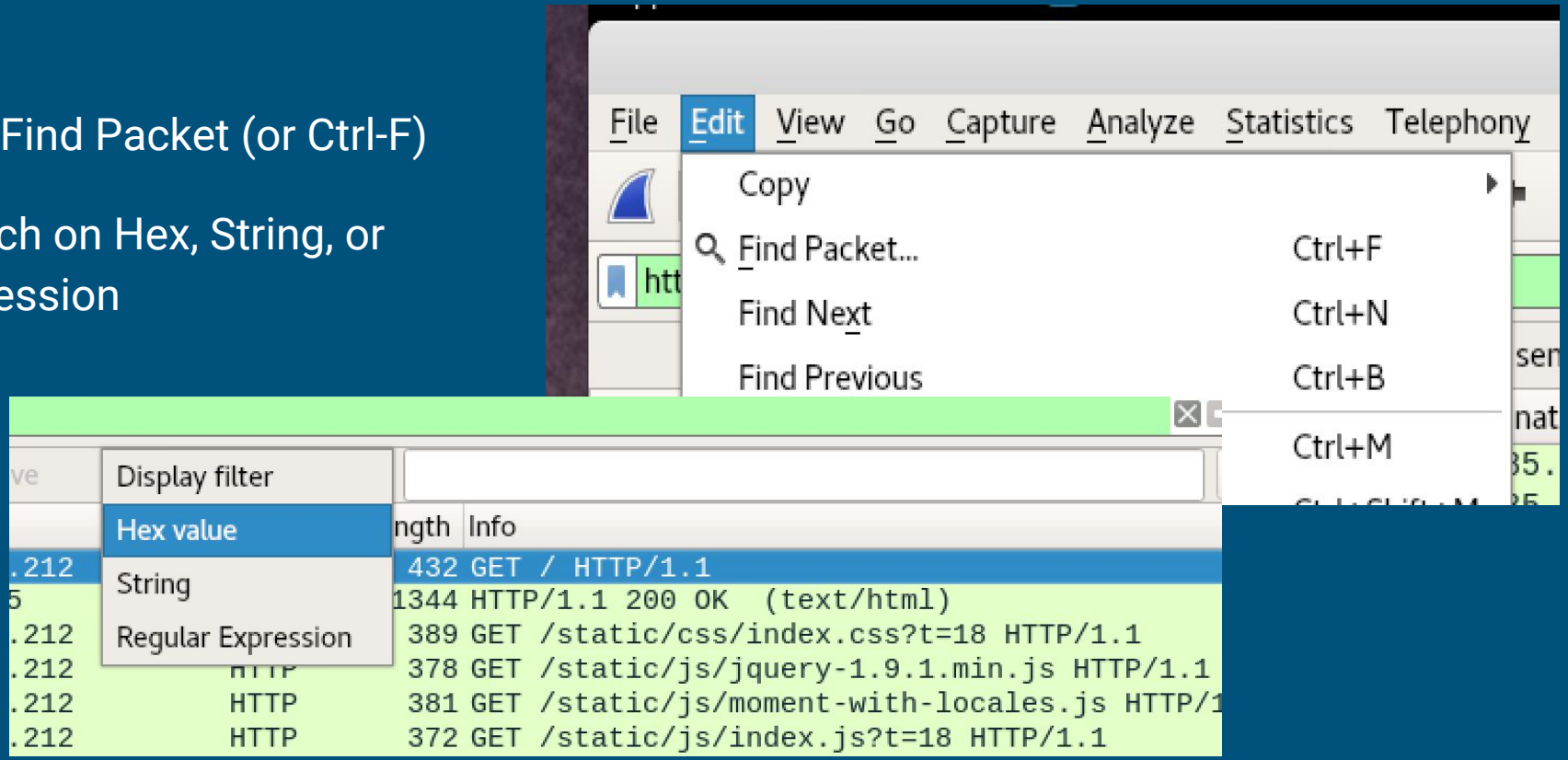




Wireshark search

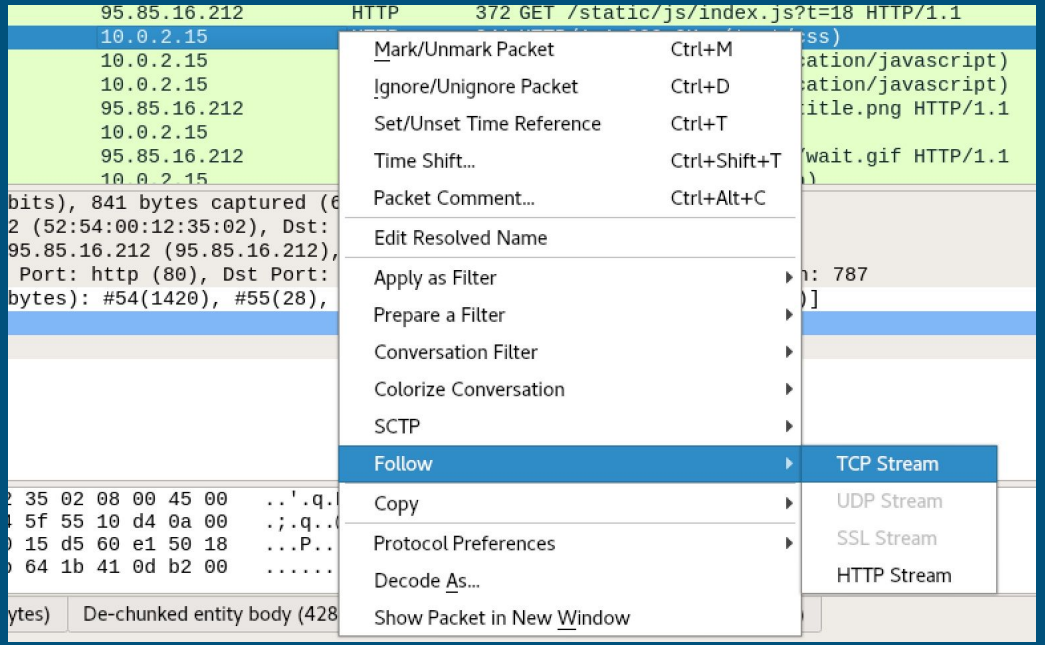
Edit/Find Packet (or Ctrl-F)

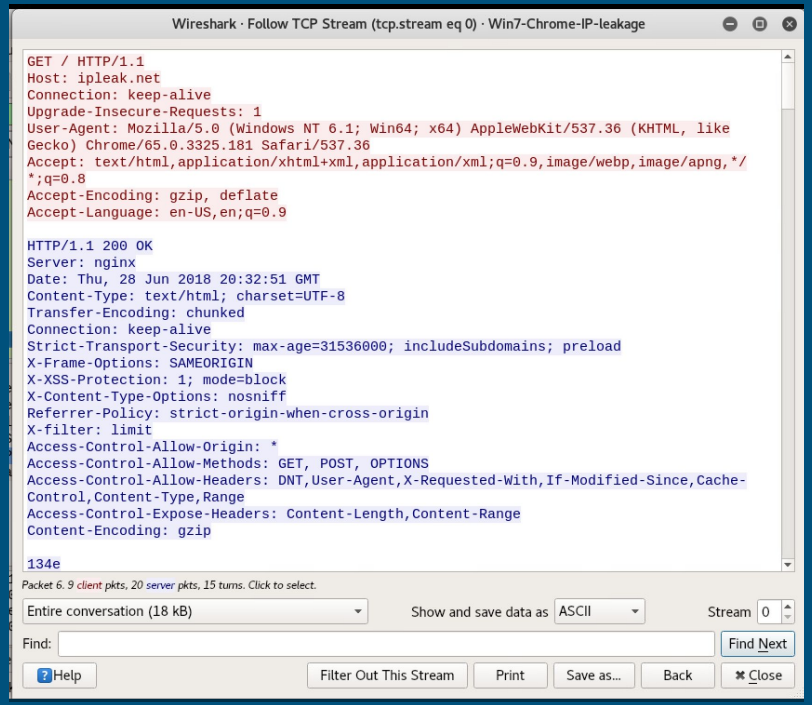
Search on Hex, String, or expression



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 bytes or "flows")
- Bro/Argus/netflow is better suited for flow stats

Connection stats: Top protocols/ports

Top IP protocols:

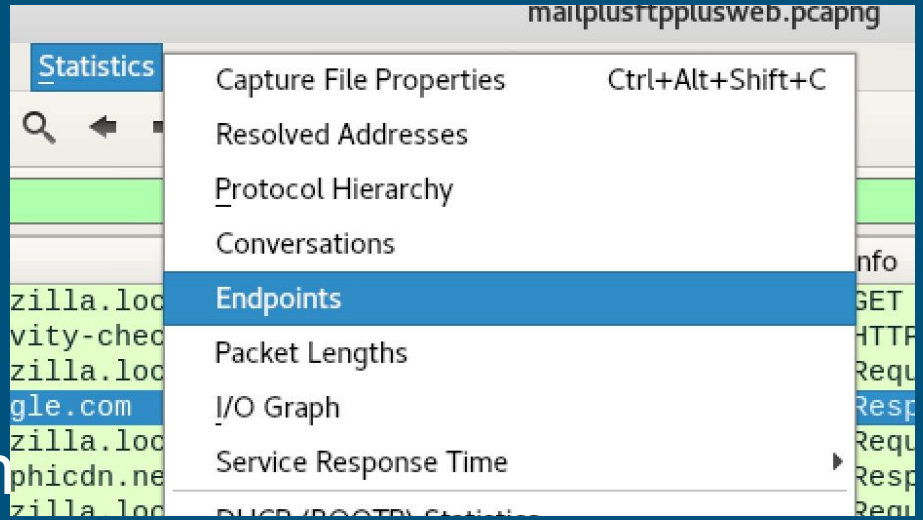
```
# tcpdump -nn -v -r $CAPFILE |grep " IP " |awk -F, '{print$6}'
|sort |uniq -c |sort -nr
```

Top 10 destination ports (based on SYN packets):

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

Connection stats (Wireshark)

- Endpoints: top network connections
- Protocol hierarchy
- 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}'
```

- Email headers, SIP traffic, web servers

- 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])' not
port 5353
```

- *(Excludes mDNS traffic)*

p0f & PADS: Passive OS/app profiling

OS/app summary via p0f:

```
# p0f -r $CAPFILE |egrep "^\| (os|app)" |sort |uniq
```

OS/app list via PADS:

```
# pads -v -r $CAPFILE -w assets.csv port 80
```

- Each subsequent run will diff against assets.csv

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]"
```

- For POST data *only*: "^<|^[a-z]"

Profiling HTTP traffic: URL timeline

URL log with timestamps:

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

- Use online URL decoders if needed

- For pentesting, 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

```
Accept-Encoding: gzip, deflate\r\n
Host: srv-2018-07-02-22.pixel.parsely.com\r\n
Connection: Keep-Alive\r\n
▸ Cookie: pid=ee6918912b427805cdec9e8c756dfb96\r\n
\r\n
[Full request URI [truncated]: http://srv-2018-07-02-22
[HTTP request 5/5]
```

HTTP traffic: 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/mobile 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!

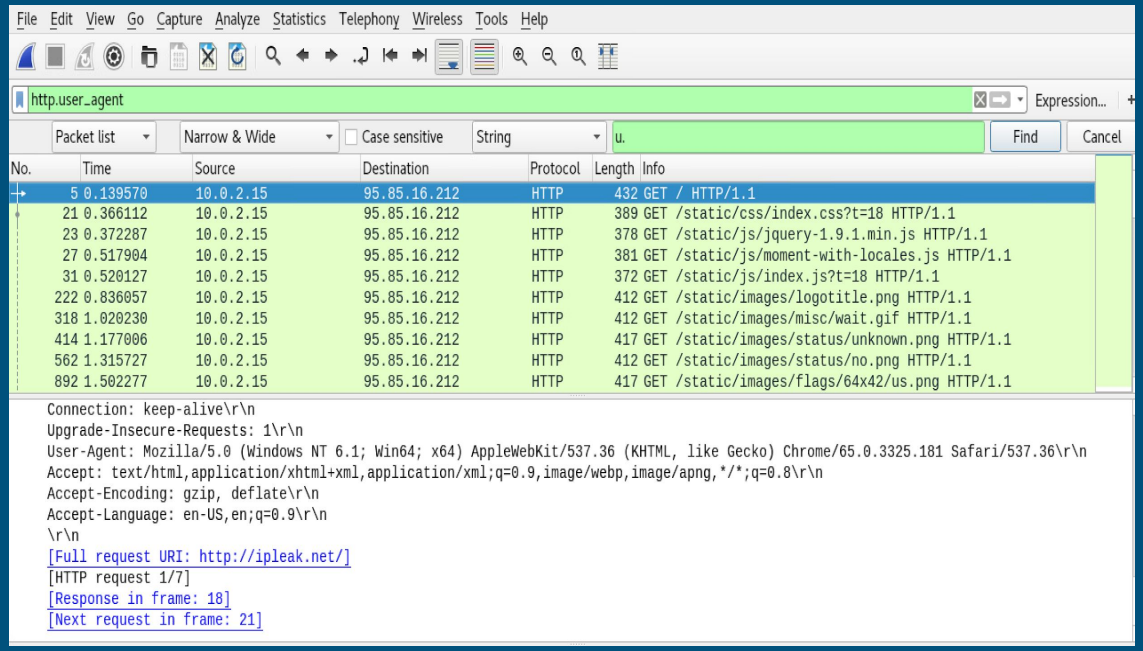
Avast/AVG stats traffic

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”)



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, emails, xml, pdfs, executables, 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 & decoding with xplico

```
# xplico -m pcap -f $CAPFILE
```

- Files & decoded data saved to xdecode/
- Supports many app-layer decoders/dissectors & some obscure protocols: "xplico -g" for list
- Front-end GUI useful for navigating extracted data

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

Content profiling: Searches & keywords

Search engine queries:

```
# ngrep -I $CAPFILE -W byline -q -t port 80 | egrep 'GET \/search\?q=' |sort |uniq
```

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 content:

```
# cat tcpflow/*.html | html2text | 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

Email traffic: headers, etc.

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

Extract emails to console:

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

Extract emails to disk:

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

- Add an ".eml" extension & open with a GUI email client to view HTML-formatted emails, images, attachments, etc.

Extracting email attachments

Show emails (without source/dest headers & newlines):

```
# tcpflow -C -0 -r $CAPFILE port 25 or port 110
```

- Look for base64-encoded attachments
- Copy base64 block to a new file, then:

```
# cat base64.txt | base64 -d >file.xxx
# file file.xxx   ### Verify file type
```

Extracting emails (Wireshark)

File - Export Objects - IMF

- Or NetworkMiner

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|^U
SER |^AUTH |login:|^Authorization:"
```

Via 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)

- NetworkMiner "Credentials" tab

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|CLASSIFIED'
```

Digging for PII (Wireshark)

Searching for credit card numbers & SSNs: regular expressions

Parsing SMB/CIFS traffic

SMB users, domains, & password hashes:

```
# tshark -nn -r $CAPFILE -V -Y tcp.port==445 |egrep
"Lan Manager Response|NTLM Response|NTLMv2
Response|Domain name|User name|Host name"
```

- Add "-T pdml" to tshark if data is truncated

Parsing SMB/CIFS traffic

SMB share & file access timeline:

```
# tshark -nn -r $CAPFILE -V -Y tcp.port==445 |egrep "Arrival Time: |Tree Id: |\[Account: |\[Domain: |\[Host: |NT Status: |Command: |GUID handle File: "
```

Carving files out of SMB traffic:

```
# tshark -nn -r $CAPFILE -q --export-objects smb,tmpfolder
```

** SMB traffic courtesy of chrissanders.org & wireshark.org*

Parsing SQL traffic

MySQL password hashes, queries, & responses:

```
# tshark -nn -r $CAPFILE -V -Y tcp.port==3306 | egrep
'Username:|Password:|Statement:|text:'
```

MSSQL queries & responses:

```
# tshark -nn -r $CAPFILE -V -Y tcp.port==1433 | egrep
"Query:|Data:|Data \[truncated\]:"
```

** SQL traffic courtesy of wireshark.org*

Hardware/mobile: Device profiling

Device info via HTTP:

```
# 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="
```

Device info via mDNS:

```
# tcpdump -nn -A -r $CAPFILE port 5353 |egrep --color
"product=|model="
```

Hardware/mobile: Windows error reporting

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

Hardware/mobile: Cell carrier info/params

Cell carrier codes:

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

Apple plist files: extract with tcpflow, decode with plistutil:

```
# grep "plist version" tcpflow/*
# apt install libplist-utils
# plistutil -i <plistfile>
```

Location tracking data

Via Apple default weather app, Wunderground, etc:

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

Via Windows default weather app:

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

Mobile apps: Android

Android apps, versions, usage, etc:

```
# ngrep -I $CAPFILE -W byline -q -t '^(GET )' port 80 | egrep "^GET |^Host:" |grep --color -A1 "ap_an="
```

Android app traffic (via Dalvik agent):

```
# ngrep -I $CAPFILE -W byline -q -t 'User-Agent: Dalvik' port 80
```

Mobile apps: Apple

Apple apps/store traffic:

```
# ngrep -I $CAPFILE -W byline -q -t port 80 | egrep -B1
"bundleId=|dpkg.ipa|^[Xx]-[Aa]pple"
```

iTunes audio downloads:

```
# ngrep -I $CAPFILE -W byline -q -t port 80 | egrep -B6
"User-Agent: AppleCoreMedia"
```

Mobile apps: Amazon Kindle & Prime

Kindle app traffic ("key=" indicates ASIN of each ebook)

```
# ngrep -q -t -I $CAPFILE -W byline | grep --color 'type="EBOK" key='
```

Prime video streaming file downloads:

```
# ngrep -q -t -I $CAPFILE -W byline | grep -B6 'Prime%20Video'
```

Mobile device profiling (Wireshark)

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

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
```

- Test with https://badssl.com

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!

.. using a known private key :)

- Many network/security appliances & embedded devices ship with default/pre-loaded private keys
- Use littleblackbox to search a capture for known keys:

```
# littleblackbox --pcap=file.pcap
```

** https://github.com/devttys0/littleblackbox*

Decrypting SSL

Pass the private key to tshark to decrypt:

```
# tshark -r SSL-decryption.pcap -q -o
"ssl.keys_list:192.168.56.101,443,http,server.pem" -z
"follow,ssl,ascii,2"
```

192.168.56.101 = SSL web server

server.pem = private key file

2 = stream number to decrypt

Decrypting SSL

- May need to fish around for the desired stream number(s)

- Note: For this to work, your capture must contain the initial TLS handshake / certificate exchange between client/server.

Packet snooping mitigation tools

- VPN/Tor *(make sure ALL traffic is VPN routed!)**
- SSH proxy through your home/office*
- HTTPS Everywhere (*"Block all unencrypted requests"*)
- Privacy Badger (& disable WebRTC in options)
- Outbound firewall rules for non-encrypted services
- DNS-over-TLS/HTTPS

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

Additional packet mining power tools

- Security Onion: Now on ELK stack!
- Bro IDS - many app-layer decoders
- Snort/Suricata rules
- Argus/netflow tools
- bulk_extractor - useful parsers, EXIF data!

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!

https://github.com/packetrat/packethunting

How to find us:

- Dave: grep8000 [at] gmail / @DavePorcello / packetbeard.blogspot.com
- Sean: sean.gallagher [at] arstechnica.com / @thepacketrat