A Deep Dive Into Traffic Fingerprints using Wireshark

Luca Deri <deri@ntop.org>, @lucaderi
Ivan Nardi<ivan@ai2m.eu>, @i_nardi

About Us

- Luca is the founder of the ntop project that develops open source network traffic monitoring applications. All code is available at https://github.com/ntop
- Ivan is a network and software engineer at AI2M, where they develop data retention and traffic analysis systems. He has been involved in DPI for more than 10 years and he is helping Luca (and Toni) to maintain nDPI.

Agenda

1. Introduction to Fingerprint
2. Passive Fingerprinting
3. Protocol Fingerprinting
4. Obfuscated Protocols Fingerprinting

nDPI in a nutshell

- An open-source library providing:
  - deep packet inspection engine for network visibility: protocol classification, metadata extraction, flow risks computation
    - basic blocks for a cyber-security application
    - flow risks: an indication that in the flow there is something unusual/dangerous to pay attention to
      - ~60 different flow risks: self-signed certificate, possible SQL/RCE injection, suspicious DGA domain, invalid character in SNI...
  - algorithms for data analysis: data forecasting, anomaly detection, clustering and similarity evaluation, (sub-)string searching and IP matching, probabilistic data structures,...
- Available on GitHub, no license required

nDPI & Wireshark

- nDPI can be used with Wireshark via extcap functionality + Lua scripting
  - The extcap interface is a versatile plugin interface that allows external binaries to act as capture interfaces directly in Wireshark
  - A simple way to have nDPI results into Wireshark/tshark GUI as "first-citizen" objects
  - Details: talk by Luca at SharkfestEU 2017
- Example: 100_extcap_tls_mismatch.pcapng (with extcap)

Part I:
Introduction to Fingerprinting

What is a Network Fingerprint

- Fingerprinting refers to the process of identifying and gathering specific information about a system or network to create a *unique* traffic profile or “fingerprint”.
- The term "unique" needs to be interpreted:
  - Family: this DHCP packet is generated by an iOS device.
  - Application: this TLS flow is generated by the Trickbot malware.
- References
  - https://medium.com/@nayanchaure601/os-fingerprinting-ab5c4d70ec22
  - https://medium.com/thg-tech-blog/fingerprinting-network-packets-53ee32ddf07a

How can I Use a Fingerprint?

- It can then be used to identify and categorise different devices, applications, or users based on their specific characteristics and behaviours.
- Typical use cases:
  - Label network traffic with an application. Example: this HTTPS connection was made by Apple Safari.
  - Network segmentation: fingerprint DHCP packets to automatically assign outdated Windows hosts to specific VLANs.
  - Cybersecurity: detect unusual behaviour or traffic patterns that are unexpected for specific hosts (e.g. label a device as an iPad and detect it uses services typical of Android devices)

What We'll Not Cover in This Talk

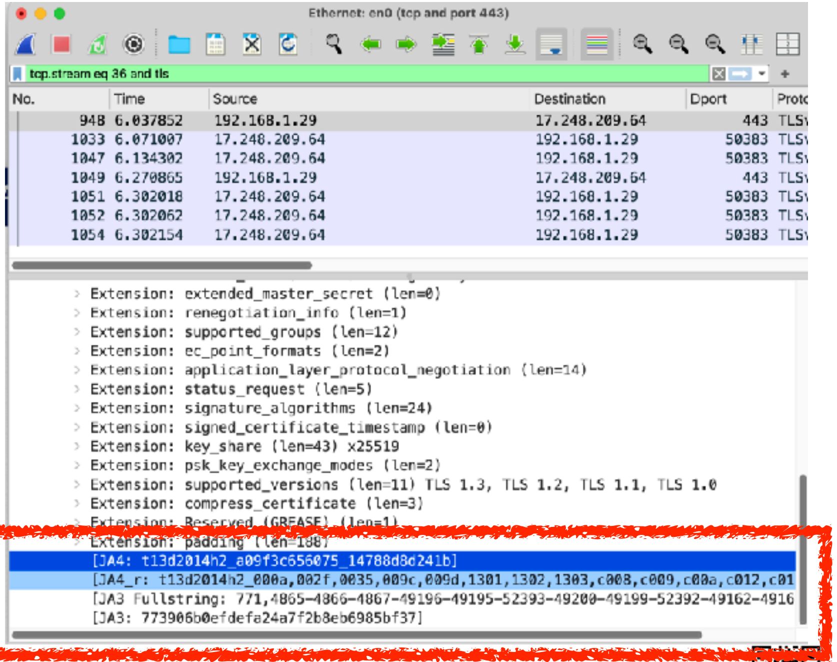
- There are two type of fingerprint
  - Initial flow fingerprint (this talk)
  - Post-connection behavioural fingerprint (not this talk)

- Behavioural analysis is used in particular in cybersecurity for detecting malware.Tool/paper examples:
  - Cisco Joy: https://github.com/cisco/joy
  - Cisco Mercury: https://github.com/cisco/mercury
  - Cloudflare, JA4 Signals, https://blog.cloudflare.com/ja4-signals/
  - L. Deri, A. Sartiano, Monitoring IoT Encrypted Traffic with Deep Packet Inspection and Statistical Analysis, Proceedings of CITST-2020

Active vs Passive [1/2]

Fingerprints can be determined using passive or active probing techniques with usual pro (no traffic, no fingerprints) / cons (traffic is injected in the network, hence we're not invisible).

- Passive
  Fingerprints are calculated by passively observing network traffic and producing the fingerprint according to "de-facto" techniques (e.g. JA3/JA4).
- As shown later, fingerprinting encrypted traffic has interesting features as ciphers and extensions ease fingerprint calculation.



Active vs Passive [2/2]

- Active fingerprinting is implemented by actively sending packets to a target machine in order to receive a response.
- Port scan can be considered a basic fingerprinting technique as it can be used to determine the operating system or read the version of specific services (e.g. read the HTTP server version and use it to find vulnerabilities) for attacking it.
- Some active fingerprinting tools:
  - nmap a popular network scanner including host discovery and service and operating system detection.
  - JARM a TLS server fingerprinting application developed by Salesforce. It provides the ability to identify and group malicious TLS servers on the Internet.

Advantages and Limitations

- Passive fingerprinting is useful when conducting network reconnaissance or monitoring network behaviour over extended periods as it is:
  - Non-intrusive nature
  - Able to gather information without alerting the target.
- However, passive fingerprinting has limitations
  - It may not provide as detailed or accurate information as active fingerprinting since it relies solely on observed behaviours (e.g. in TLS 1.3 server hello and certificate are encrypted and thus they cannot be used albeit very useful).
  - Some techniques may be subject to noise or interference, impacting the reliability of the gathered information.

Fingerprinting Families: Can Happen Anywhere

End System

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

App (RTP/RTSP fingerprint Meet vs Teams vs Zoom)

Network Library Fingerprinting (TLS/QUIC)

SNMP, NetBIOS

TCP/IP stack (Linux vs Win vs macOS)

IEEE 802.11

CDP Protocol, ARP

Fingerprinting Methods

- Protocol Fingerprint
  - Analyse a specific protocol (e.g. DHCP fingerprint, or TCP behaviour for OS fingerprinting) in order to compute the expected fingerprint. Example: Window hosts do not set the Timestamps option in TCP SYN packets.
- Content Fingerprint
  - Create the fingerprint based on the content of specific protocol. Examples:
    - HTTP User-Agent
    - Android vs iOS vs Windows can be passively detected looking at DNS domain names queries (e.g. thinkdifferent.us and connectivitycheck.android.com)
    - Firefox connects via TLS to firefox.settings.services.mozilla.com

Using Fingerprinting in Real Life

- Browser fingerprinting
  Collects information about a web browser and device where it's running on including browser type, version, operating system, screen resolution, installed plugins. This creates a unique "fingerprint" that can be used to track the user across different sessions and websites.
- Policy Enforcement (OS/Device Fencing)
  Restrict to specific VLANs/block old/specific devices/OSs by looking at the device MAC address or initial DHCP request. This technique plays an important role in securing OT (Operational Technology) networks.
- Traffic Prioritisation
  Disable specific traffic (e.g. Zoom Video) in case of limited available bandwidth.

Fingerprinting in Cybersecurity

- Fingerprinting plays a crucial role in cyber security as it helps in detecting threats, securing networks, and implementing targeted security measures.
- Defenders:
  - Match malware signatures (e.g. TLS fingerprint or SSL certificate hash) and block malicious traffic.
  - Prevents massive scanners from exploring network services.
- Attackers
  - Use fingerprinting to detect flaws (e.g. CVEs) that can be used to attack the system.
  - During reconnaissance, identify application/OS version in order to target attacks towards weak victims.

Fingerprints in MITRE

- MITRE Adversarial Tactics, Techniques, and Common Knowledge (Att&ck) is a knowledge base that tracks cyber adversary tactics and techniques. Fingerprinting is listed under Techniques / Enterprise / System Information Discovery
- MITRE Common Attack Pattern Enumerations and Classifications (CAPEC™) is dictionary of known patterns of attack employed by adversaries to exploit known weaknesses in cyber-enabled capabilities. CAPEC-224 covers fingerprinting.

Likelihood Of Attack

High

Typical Severity

Very Low

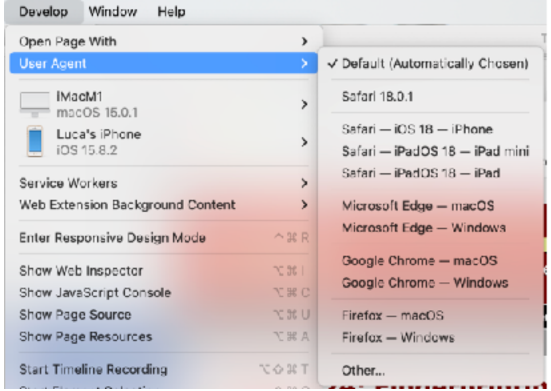
Relationships

Nature	Type	ID	Name
ParentOf	S	312	Active OS Fingerprinting
ParentOf	S	313	Passive OS Fingerprinting
ParentOf	S	541	Application Fingerprinting

View Name	Top Level Categories
Domains of Attack	Software
Mechanisms of Attack	Collect and Analyze Information

False Positives vs False Negatives

- Definitions:
  - False positives occur when a system or network is wrongly classified, leading to unnecessary security measures or alerts.
  - False negatives occur when a threat or vulnerability goes undetected due to an inaccurate or incomplete fingerprint.
- Caveats:
  - Traffic fingerprints are subject to false positives as sometimes it is very simple to mimic devices/apps in order to circumvent protections.
  - Fingerprints cannot be be 100% accurate: in the early days of fingerprints, tools pretended to identify the exact OS and version leading to long device databases. This is no longer possible due to network stack randomisation (TCP sequence numbers, MAC addresses, ephemeral ports etc).
  - Two different devices/OSs/applications can share the same/similar fingerprint. This is because they can use the same TLS library or the same OS family in different flavours (iOS, vs iPad OS, vs macOS)
  - Network traffic can be forged, so fingerprints need to be carefully used as attackers can inject packets to trick defenders.



- The collection and analysis of personal data, such as device or browser fingerprints, may raise privacy concerns and require compliance with relevant regulations.
- Organisations must ensure that their fingerprinting practices adhere to applicable laws and regulations, while also respecting user privacy and providing transparency in how their data is used and protected.
- Tracking users using fingerprints is not desirable for users but widely used in the industry. Companies are periodically introducing new features to prevent/limit them: macOS, Android, and many browsers (Safari, Firefox, Chrome) support "Do Not Track" feature, even if it is often disabled by default (e.g. in Chrome)

Part II
Passive Fingerprinting

Prior Art: Some OS Fingerprinting Tools

- Passive Fingerprinting
  - p0f
  - prads (Passive Real-time Asset Detection System)
  - SATORI: Python rewrite of passive OS fingerprinting tool.

- Active Fingerprinting
  - nmap: network scanner featuring OS fingerprinting
  - Ettercap / NetworkMiner: network forensics tools able to determine OS type/version
  - XProbe2: remote active operating system fingerprinting tool

Status
- Not Actively Developed/Maintained
- Stand-by/Maintenance
- Actively Developed

p0f Overview [1/2]

p0f generates TCP signatures in the format below that are then mapped against a signature database (currently mostly outdated).

```
sig = ver:ittl:olen:mss:wsize,scale:olayout:quirks:pclass
```

```
ver         - signature for IPv4 ('4'), IPv6 ('6'), or both ('*').
ittl        - initial TTL used by the OS.
olen        - length of IPv4 options or IPv6 extension headers
mss         - maximum segment size, if specified in TCP options
wsize       - TCP window size
scale       - window scaling factor, if specified in TCP options or '*'
olayout     - comma-delimited layout and ordering of TCP options, if any
quirks      - comma-delimited properties and quirks (e.g. ENC or dont't
              fragment) observed in IP or TCP
pclass      - payload size classification: '0' for zero, '+' for non-zero,
              '*' for any.
```

p0f Overview [2/2]

Example of TCP signatures

```
.-[ 192.168.1.117/54868 -> 213.19.144.104/443 (syn) ]-
|
| client   = 192.168.1.117/54868
| os       = Mac OS X
| dist     = 0
| params   = generic fuzzy
| raw_sig  = 4:64+0:0:1460:65535,5:mss,nop,ws,nop,nop,ts,sok,eol+1:df:0
|
`----
```

Using the same approach p0f can also fingerprint application protocols such as HTTP.

```
.-[ 192.168.1.7/53251 -> 184.25.204.10/80 (http request) ]-
|
| client   = 192.168.1.7/53251
| app      = ???
| lang     = English
| params   = none
| raw_sig  = 1:Host,Accept=[*/*],Accept-Language=[en-US;q=1],Connection=[keep-
alive],Accept-Encoding=[gzip, deflate],User-Agent:Accept-Charset,Keep-Alive:Argo/9.1.0
(iPhone; iOS 10.2; Scale/2.00)
|
```

How to Create a Fingerprint

As seen with p0f, creating a fingerprint is usually <u>not rocket science</u> if the following principles are satisfied:

- Extract protocol/application unique characteristics.
- Ignore parameters that are random (e.g. TLS GREASE*), request-specific (e.g. a hostname or the SNI).
- Concat parameters after transformations (e.g. sort) to make the string fingerprint and avoid the fingerprint to be circumvented.
- Optionally hash the fingerprint to create a fixed-length fingerprint string.

*GREASE (Generate Random Extensions And Sustain Extensibility), a mechanism to prevent extensibility failures in the TLS ecosystem. It reserves a set of TLS protocol values that may be advertised to ensure peers correctly handle unknown values.

Hash Functions Primer

- A hash function is used to map arbitrary long data into a fixed size ("compress") string of bytes.
- Hash functions Properties:
  - Uniformity: distribute uniformly data across a finite domain (e.g. 0 ... $2^{32}-1$).
  - Collision Resistance: it should be difficult to find two different inputs that produce the same hash value.
  - Avalanche effect: a small change in the input should produce a significantly different hash value.

Hash vs Raw Fingerprint

- Due to the nature of hash functions, fingerprints that use them are designed for <u>equality matching</u> (e.g. identify malware X whenever its fingerprint is detected in traffic)
- Raw (i.e. un-hashed) fingerprints have variable length however they can be used for similarity matching (e.g. TCP stack fingerprint X is similar to fingerprints produced by Windows systems)

(Un-hashed) Fingerprint Similarity

- Due to the nature of hash functions, only un-hashed fingerprints can be searched for similarity matching as follows:
  - Transform fingerprint string into a vector of numbers, a.k.a. word embedding in AI parlance: "the representation is a real-valued vector that encodes the meaning of the word in such a way that the words that are closer in the vector space are expected to be similar in meaning" (source Wikipedia).
  - Use labelled data (e.g. pre-classified traffic) to create a database of fingerprints and search for similarity (K-NN, K Nearest Neighbour).
  - Vector databases are able to index numerical vectors and search for similarity using approximate nearest neighbourhood algorithms with the goal of finding the closest database match to the searched vector.

Part III
Protocol Fingerprinting

Fingerprints and Wireshark

In the following slides, we'll show some Lua scripts we developed and that are available at

- https://github.com/ntop/nDPI/tree/dev/wireshark

nDPI / wireshark /

Add file ···

lucaderi Added further TCP fingerprints ✓ 9c0e4c5 · 5 days ago History

Name	Last commit message	Last commit date
..		
sfeu24	wireshark: lua: add script for QUIC fingerprints [...	last month
sharkfest_scripts	fixed lua errors in non-iec104 packets (#1209)	3 years ago
tshark	Performed some grammar and typo fixes (#2511)	3 months ago
README.md	Performed some grammar and typo fixes (#2511)	3 months ago
download-fuzz-traces.sh	shell: reformatted, fixed inspections, typos (#25...	3 months ago
ndpi.lua	Added further TCP fingerprints	5 days ago

TCP/IP Stack Fingerprinting [1/3]

- As discussed earlier, TCP/IP stack fingerprinting is one of the most popular methods for detecting the OS from network traffic.
- Unfortunately there is <u>no single standard/representation</u> hence there are various formats produced by the many available fingerprint tools.
- As Wireshark does not natively features a TCP/IP stack fingerprint, we have developed one as part of our contribution.
- The fingerprint format is the following
  <TCP Flags>_<TTL>_<TCP Win>_SHA256(<Options Fingerprint>)

```
-- Normalize TTL
ip_ttl = tonumber(ip_ttl)
if(ip_ttl <= 32) then       ip_ttl = 32
elseif(ip_ttl <= 64)  then ip_ttl = 64
elseif(ip_ttl <= 128) then ip_ttl = 128
elseif(ip_ttl <= 192) then ip_ttl = 192
else ip_ttl = 255       end
```

Note:
- The fingerprint is computed on the SYN (req) packet
- For IPv6 we use Hop Limit instead of TTL

TCP/IP Stack Fingerprinting [2/3]

```
> Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
> Ethernet II, Src: HonHaiPrecis_6e:8b:24 (00:16:ce:6e:8b:24), Dst: ASUSTekCOMPU_40:76:ef (00:15:f2:40:76:ef)
> Internet Protocol Version 4, Src: 192.168.0.114 (192.168.0.114), Dst: 192.168.0.193 (192.168.0.193)
v Transmission Control Protocol, Src Port: 1137, Dst Port: 21, Seq: 0, Len: 0
    Source Port: 1137
    Destination Port: 21
    [Stream index: 0]
    [Stream Packet Number: 1]
  > [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3753095934
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0111 .... = Header Length: 28 bytes (7)
  > Flags: 0x002 (SYN)
    Window: 16384
    [Calculated window size: 16384]
    Checksum: 0x2963 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  v Options: (8 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted
      > TCP Option - Maximum segment size: 1460 bytes
      > TCP Option - No-Operation (NOP)
      > TCP Option - No-Operation (NOP)
      > TCP Option - SACK permitted
  > [Timestamps]

0000  00 15 f2 40 76 ef 00 16  ce 6c 8b 24 08 00 45 00   ···@v··· ·n·$··E·
0010  00 30 a7 e3 40 00 80 06  d0 60 c0 a8 00 72 c0 a8   ·0··@··· ·`···r··
0020  00 c1 04 71 00 1[illegible] 02   ···q···· ······p·
0030  40 00 29 63 00 00 02 04  05 b4 01 01 04 02         @·)c···· ······
```

- Raw: 2_128_32768_0205B4010104
- Hashed: 2_128_32768_44bd01ba086e

TCP/IP Stack Fingerprinting [3/3]


```
> Frame 85: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface unknown, id 0
> Ethernet II, Src: Intel_a8:1f:ec (3c:a9:f4:a8:1f:ec), Dst: TechnicolorD_e0:86:62 (20:b0:01:e0:86:62)
> Internet Protocol Version 4, Src: 192.168.1.128 (192.168.1.128), Dst: 89-96-108-170.ip12.fastwebnet.it (89.96.108.170)
v Transmission Control Protocol, Src Port: 35830, Dst Port: 8080, Seq: 0, Len: 0
    Source Port: 35830
    Destination Port: 8080
    [Stream index: 5]
    [Stream Packet Number: 1]
  > [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 510107882
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x002 (SYN)
    Window: 64240
    [Calculated window size: 64240]
    Checksum: 0x4bd1 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  v [Timestamps]
      [Time since first frame in this TCP stream: 0.000000000 seconds]
      [Time since previous frame in this TCP stream: 0.000000000 seconds]
v ntop Extensions
    TCP Fingerprint: 2_64_64240_1a766bf8a57a

0000  20 b0 01 e0 86 62 3c a9  f4 a8 1f ec 08 00 45 00    ····b<· ······E·
0010  00 3c db 5c 40 00 40 06  d7 2c c0 a8 01 80 59 60   ·<·\@·@· ·,····Y`
0020  6c aa 8b f6 1f 90 1e 67  a0 ea 00 00 00 00 a0 02   l······g ········
0030  fa f0 4b d1 00 00 02 04  05 b4 04 02 08 0a e4 36   ··K····· ·······6

TCP Fingerprint (ntop.tcp_fingerprint)                  Packets: 113 · Displayed: 5 (4.4%)
```

TCP/IP Stack Fingerprinting Findings [1/2]

While studying the TCP fingerprints we have noted some facts.

Windows

- Does not use the timestamp (8) option.
- Has a default TTL of 128, vs 64 used on Linux etc.

iOS/iPadOS/macOS (Intel)

- Send SYN+ECE+CRW. Others (including macOS Silicon) just SYN.
- Options (iOS but not iPadOS) end with a double EOL.

```
⌄ Options: (24 bytes), Maximum segment size, No-Operation (
  › TCP Option - Maximum segment size: 1460 bytes
  › TCP Option - No-Operation (NOP)
  › TCP Option - Window scale: 5 (multiply by 32)
  › TCP Option - No-Operation (NOP)
  › TCP Option - No-Operation (NOP)
  › TCP Option - Timestamps: TSval 1148500268, TSecr 0
  › TCP Option - SACK permitted
  › TCP Option - End of Option List (EOL)
  › TCP Option - End of Option List (EOL)
```

TCP/IP Stack Fingerprinting Findings [2/2]

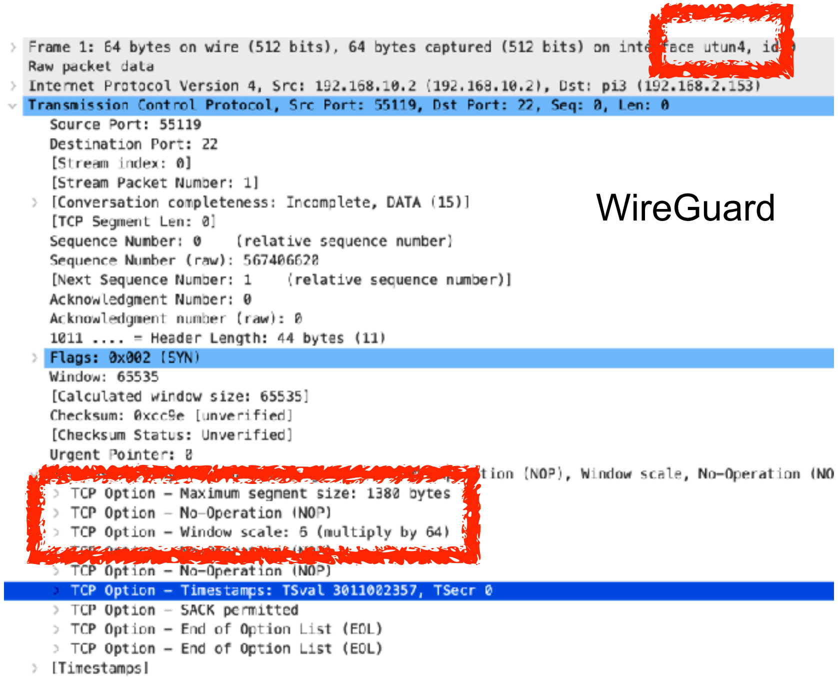
- macOS/iPadOS/iOS are similar but not identical
  - macOS Intel (SYN+ECE+CRW) and AppleSilicon (SYN) are different so you can fingerprint the platform with the TCP/IP stack.
  - iPadOS and iOS are similar but not identical.
- A single OS/device can have multiple fingerprints. Example iPadOS: 194_64_0_d29295416479, 194_64_65535_d29295416479, 2_64_65535_d29295416479, 194_64_65535_d3a424420f2a

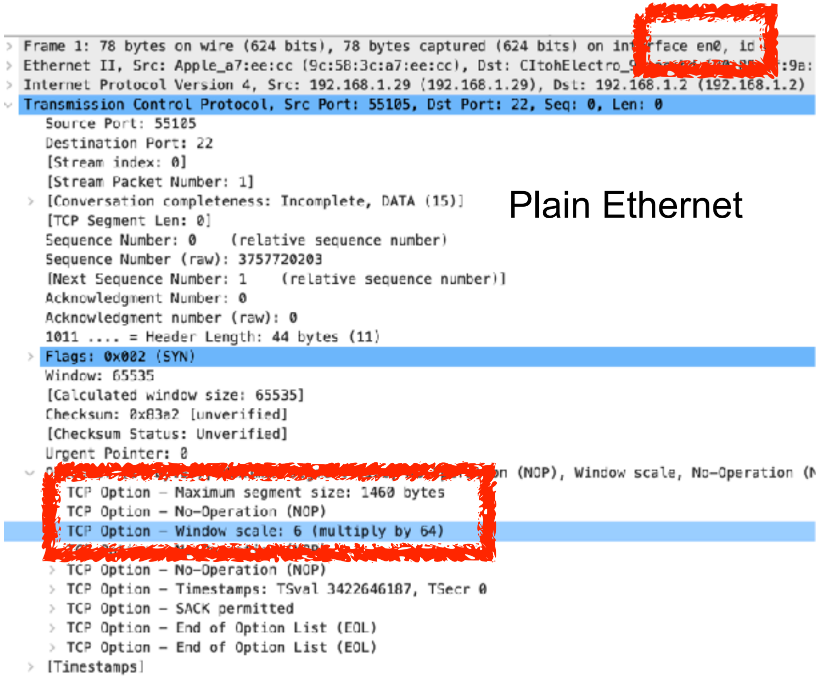
- Using the TCP/IP stack fingerprint it is possible to find out the OS of embedded devices

[Android Scanner]

[Linux Wireless Label Base Station]

Fingerprint produced by tools such as nmap and p0f were mostly created for identifying the host OS, but they offers interesting side-properties....





Same client host (macOS) connected to two Raspberry Pi: one over a VPN (Wireguard) and the other over plain Ethernet. Different MSS and Window Scale Factor

TCP/IP Stack Fingerprinting and Cybersecurity


```
> Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480
> Ethernet II, Src: 76:ac:b9:35:30:da (76:ac:b9:35:30:da), Dst:
> Internet Protocol Version 4, Src: 192.168.10.145 (192.168.10.
v Transmission Control Protocol, Src Port: 49175, Dst Port: 888
    Source Port: 49175
    Destination Port: 8888
    [Stream index: 0]
    [Stream Packet Number: 1]
  > [Conversation completeness: Incomplete (35)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 253744456
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x002 (SYN)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x5297 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

https://zmap.io/

```
> Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
> Ethernet II, Src: 76:ac:b9:35:30:da (76:ac:b9:35:30:da), Dst: PCSSyste
> Internet Protocol Version 4, Src: 192.168.10.145 (192.168.10.145), Dst
v Transmission Control Protocol, Src Port: 46998, Dst Port: 8888, Seq: 0
    Source Port: 46998
    Destination Port: 8888
    [Stream index: 0]
    [Stream Packet Number: 1]
  > [Conversation completeness: Incomplete (35)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1163206847
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x002 (SYN)
    Window: 1024
    [Calculated window size: 1024]
    Checksum: 0xd56b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

https://github.com/robertdavidgraham/masscan

HowTo Manipulate TCP Options

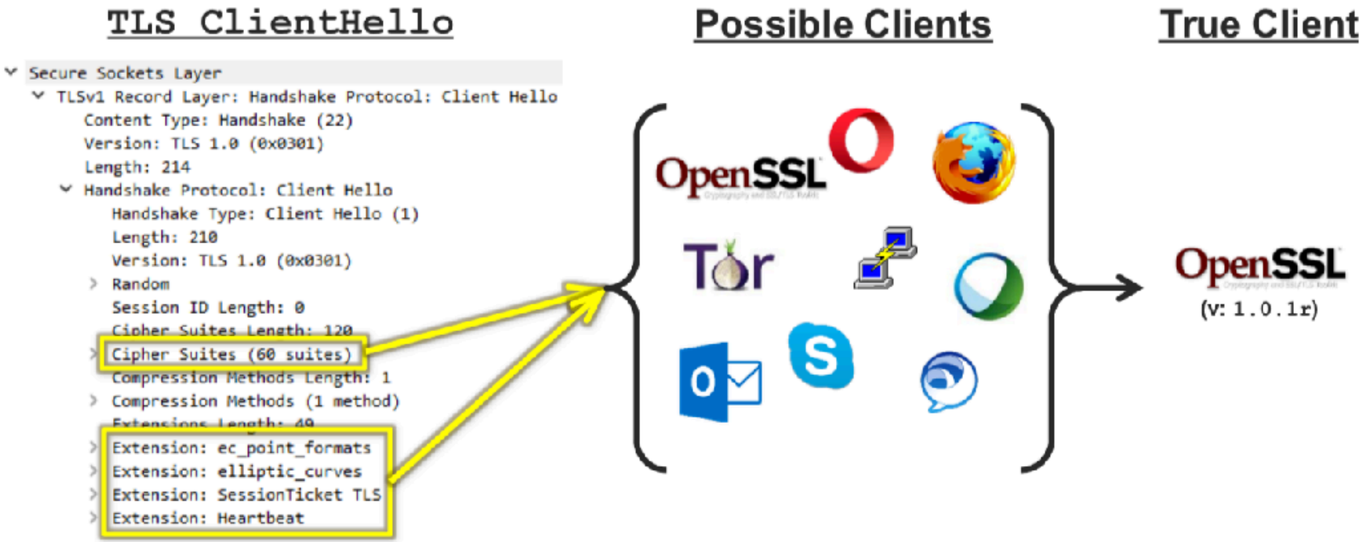
- Using sockets it is possible to manipulate a few (but not all) TCP options using the setsockopt() call.
- All supported options are listed in /usr/include/netinet/tcp.h

- Example:

```
int mss = 576;
int result = setsockopt(lsock, IPPROTO_TCP, TCP_MAXSEG, &mss, sizeof(mss));
if (result != 0) {
   perror(0);
   return 1;
 }
```

TLS/QUIC Fingerprinting [1/3]

- Contrary to the TCP/IP stack (usually) part of the kernel, for TLS/QUIC encoder/decoder is implemented by a user-space library hence every application sitting on the same OS can potentially use different fingerprints.



TLS/QUIC Fingerprinting [2/3]

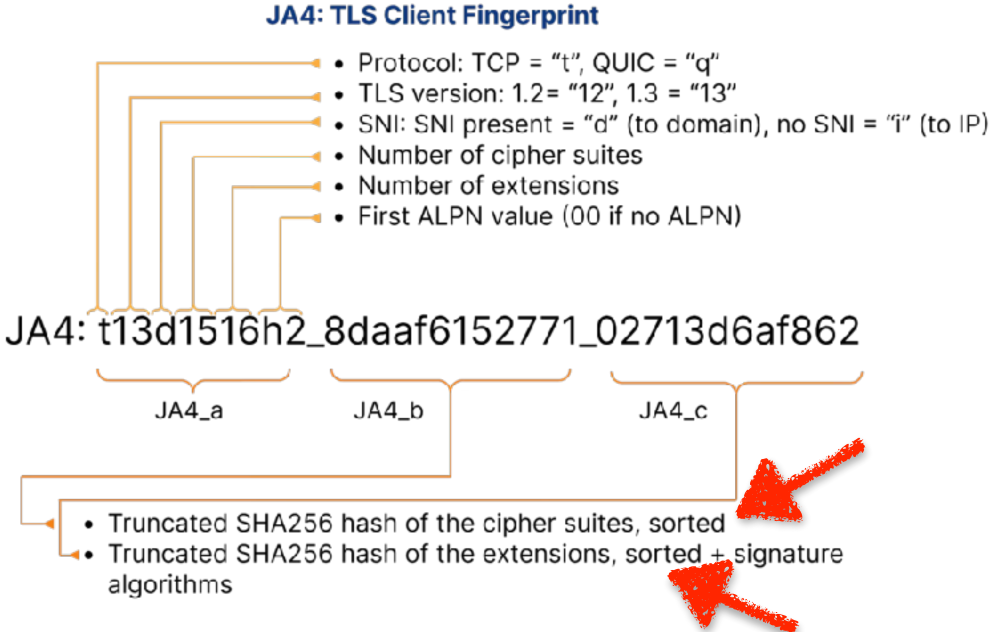
- JA3 was the first popular fingerprint for SSL/TLS was invented by Salesforce in 2017 with goal to produce fingerprints that could be easily shared for threat intelligence.
- Two fingerprints: JA3 (client) and JA3S (server). They are created concatenating the following fields in the same order they are received in the TLS Client Hello (JA3) and TLS Server Hello (JA3S):

  `TLSVersion,Ciphers,Extensions,EllipticCurves,EllipticCurvePointFormats`

  skipping GREASE (Generate Random Extensions And Sustain Extensibility) extensions.
- JA3 has been replaced by JA4 as in 2023 Google started to randomise extensions to prevent JA3 detection thus jeopardising it.

TLS/QUIC Fingerprinting [3/3]

- JA4 is the JA3 successor and it comes with additional fingerprints named JA4+ (e.g. for TCP, HTTP, SSH...). While JA4 for client fingerprinting has been released under BSD 3-Clause, all other are patent pending and subject to license. Wireshark implements only JA4.



JA3/JA4 in Wireshark


```
> Internet Protocol Version 4, Src: 172.16.2.185 (172.16.2.185), Dst: 192.168.2.142 (192.168.2.142)
> Transmission Control Protocol, Src Port: 52494, Dst Port: 3389, Seq: 20, Ack: 20, Len: 173
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 168
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 164
        Version: TLS 1.2 (0x0303)
      > Random: 5cef9e29f7050d16a1a2391d64625681e3425a70bdd5045db0a23d3db250dbe7
        Session ID Length: 0
        Cipher Suites Length: 44
      > Cipher Suites (22 suites)
        Compression Methods Length: 1
      > Compression Methods (1 method)
        Extensions Length: 79
      > Extension: server_name (len=18) name=192.168.2.142
      > Extension: supported_groups (len=8)
      > Extension: ec_point_formats (len=2)
      > Extension: signature_algorithms (len=18)
      > Extension: status_request (len=5)
      > Extension: signed_certificate_timestamp (len=0)
      > Extension: extended_master_secret (len=0)
        [JA4: t12d220700_0d4ca5d4ec72_3304d8368043]
        [JA4_r: t12d220700_000a,002f,0035,003c,003d,009c,009d,00ff,c008,c009,c00a,c012,c013,c014,c023,c024,c027,c028,c02b,
        [JA3 Fullstring: 771,255-49196-49195-49188-49187-49162-49161-49160-49200-49199-49192-49191-49172-49171-49170-157-1
        [JA3: e4d448cdfe06dc1243c1eb026c74ac9a]
```

Browser Fingerprints: <cipher list>

chrome 129 / brave 1.70.126 / opera 113 / edge 129
002f,0035,009c,009d,1301,1302,1303,c013,c014,c02b,c02c,c02f,c030,cca8,cca

Android chrome 111
002f,**0035,009c,009d,1301,1302,1303,c013,c014,c02b,c02c,c02f,c030,cca8**,cca9

macOS firefox 131
002f,**0035,009c,009d,1301,1302,1303,**c009,c00a**,c013,c014,c02b,c02c,c02f,c030,cca8**,cca9

Windows firefox 131
002f,0035,009c,009d,1301,1302,1303,c009,c00a**,c013,c014,c02b,c02c,c02f,c030,cca8**,cca9

safari iOS 15/18 - macOS
000a,**002f**,**0035,009c,009d,1301,1302,1303**,c008,c009,c00a,c012,**c013,c014,c02b,c02c,c02f,c030,cca8,**cca9

Browser Fingerprints: <extensions list>

chrome 129 / brave 1.70.126 / opera 113 / edge 129
0005,000a,000b,000d,0012,**0017**,001b,0023,**002b,002d,0033**,4469,fe0d,ff0

Android chrome 111
0005,000a,000b,000d,0012,0015,**0017**,001b,0023,**002b,002d,0033**,4469,ff01

macOS firefox 131
0005,000a,000b,000d,0017,001c,0022,0023,**002b,002d,0033**,fe0d,ff01

Windows firefox 131
0005,000a,000b,000d,0017,001c,0022,0029,**002b,002d,0033**,fe0d,ff01

safari iOS 15/18 - macOS
0005,000a,000b,000d,0012,0015,**0017**,001b,**002b,002d,0033**,ff01

Browser Fingerprints: <signatures list>

chrome 129 / brave 1.70.126 / opera 113 / edge 129
0403,0804,0401,0503,**0805,0501,0806,0601**

Android chrome 111
0403,0804,0401,0503,**0805,0501,0806,0601**

macOS firefox 131
0403,0503,0603,**0804,0805,0806**,0401,**0501,0601**,0203,0201

Windows firefox 131
0403,0503,0603,**0804,0805,0806**,0401,**0501,0601**,0203,0201

safari iOS 15/18 - macOS
0403,0804,0401,0503,0203,**0805,0805,0501,0806,0601**,0201

Repeated

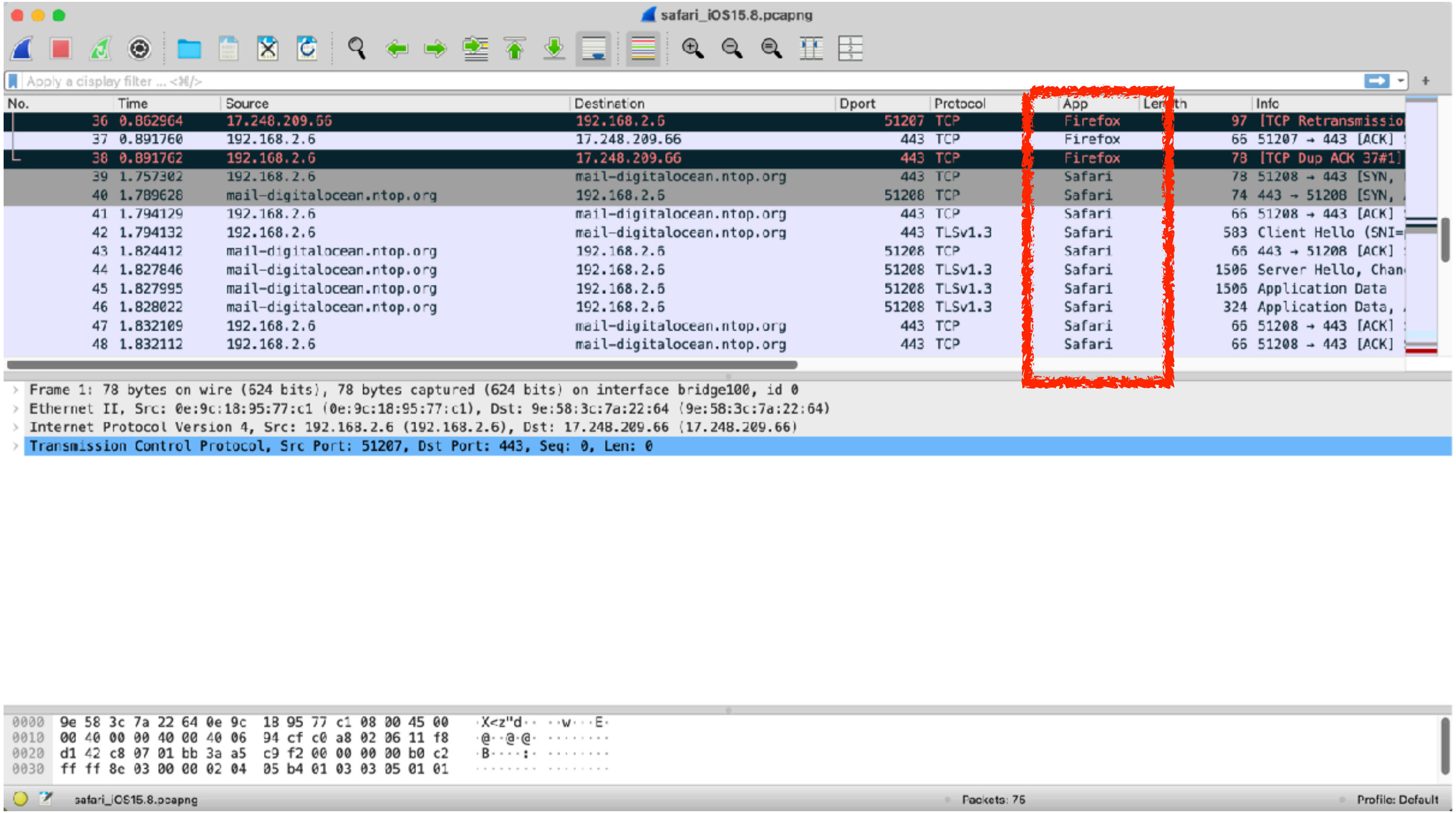
Browser Fingerprints in Wireshark [1/2]


```
local ja4_db = {
    ['02e81d9f7c9f_736b2a1ed4d3'] = 'Chrome',
    ['07be0c029dc8_ad97e2351c08'] = 'Firefox',
    ['07be0c029dc8_d267a5f792d4'] = 'Firefox',
    ['0a330963ad8f_c905abbc9856'] = 'Chrome',
    ['0a330963ad8f_c9eaec7dbab4'] = 'Chrome',
    ['168bb377f8c8_a1e935682795'] = 'Anydesk',
    ['24fc43eb1c96_14788d8d241b'] = 'Chrome',
    ['24fc43eb1c96_14788d8d241b'] = 'Safari',
    ['24fc43eb1c96_845d286b0d67'] = 'Chrome',
    ['24fc43eb1c96_845d286b0d67'] = 'Safari',
    ['24fc43eb1c96_c5b8c5b1cdcb'] = 'Safari',
    ['2a284e3b0c56_12b7a1cb7c36'] = 'Safari',
    ['2a284e3b0c56_f05fdf8c38a9'] = 'Safari',
    ['2b729b4bf6f3_9e7b989ebec8'] = 'IcedID',
    ['39b11509324c_ab57fa081356'] = 'Chrome',
    ['39b11509324c_c905abbc9856'] = 'Chrome',
    ['39b11509324c_c9eaec7dbab4'] = 'Chrome',
    ['41f4ea5be9c2_06a4338d0495'] = 'Chrome',
    ......
```

Missing JA4_a

[ndpi.lua]

Browser Fingerprints in Wireshark [2/2]



RDP (Remote Desktop Protocol) [1/3]

- RDP is a proprietary protocol created by Microsoft to graphically connect to hosts on a LAN.
- Until version 5.2 (WinXP) the protocol was not encrypted, but today almost all communications are over TLS.
- Until the protocol was unencrypted it was possible to create a fingerprint using RDP attributes such screen resolution, keyboard language etc., thing that is no longer possible with TLS.
- However JA4 can be the solution as we could use it to fingerprint RDP traffic.

RDP (Remote Desktop Protocol) [2/3]

No.	Source	Destination	Dport	Protocol	Length	Info
1	172.16.2.185	192.168.2.142	3389	TCP	68	52494 → 3389 [SYN, ECE, CWR] S
2	192.168.2.142	172.16.2.185	52494	TCP	56	3389 → 52494 [SYN, ACK] Seq=0
3	172.16.2.185	192.168.2.142	3389	[illegible]	[illegible]	[illegible]
4	172.16.2.185	192.168.2.142	338	RDP	63	Negotiate Request
5	192.168.2.142	172.16.2.185	5249	RDP	63	Negotiate Response
6	172.16.2.185	192.168.2.142	3389	[illegible]	[illegible]	[illegible]
7	172.16.2.185	192.168.2.142	338	TLSv1.2	217	Client Hello (SNI=192.168.2.14
8	192.168.2.142	172.16.2.185	5249	TLSv1.2	1223	Server Hello, Certificate, Ser
9	172.16.2.185	192.168.2.142	338	TCP	44	52494 → 3389 [ACK] Seq=192 Ack
10	172.16.2.185	192.168.2.142	3389	TLSv1.2	170	Client Key Exchange, Change Ci

```
> Frame 7: 217 bytes on wire (1736 bits), 217 bytes captured (1736 bits)
> Null/Loopback
> Internet Protocol Version 4, Src: 172.16.2.185 (172.16.2.185), Dst: 192.168.2.142 (192.168.2.142)
> Transmission Control Protocol, Src Port: 52494, Dst Port: 3389, Seq: 20, Ack: 20, Len: 173
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 168
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 164
        Version: TLS 1.2 (0x0303)
      > Random: 5cef9e29f7050d16a1a2391d64625681e3425a70bdd5045db0a23d3db250dbe7
        Session ID Length: 0
        Cipher Suites Length: 44
      > Cipher Suites (22 suites)
        Compression Methods Length: 1
      > Compression Methods (1 method)
        Extensions Length: 79
      > Extension: server_name (len=18) name=192.168.2.142
      > Extension: supported_groups (len=8)
```

RDP (Remote Desktop Protocol) [3/3]

- Thanks to JA4, it is possible to detect/fingerprint RDP attackers
  - RDP client contacts many different hosts
  - Often short-living sessions (host scan)
- Example of RDP scan/attacks detected on a service provider network:

Time

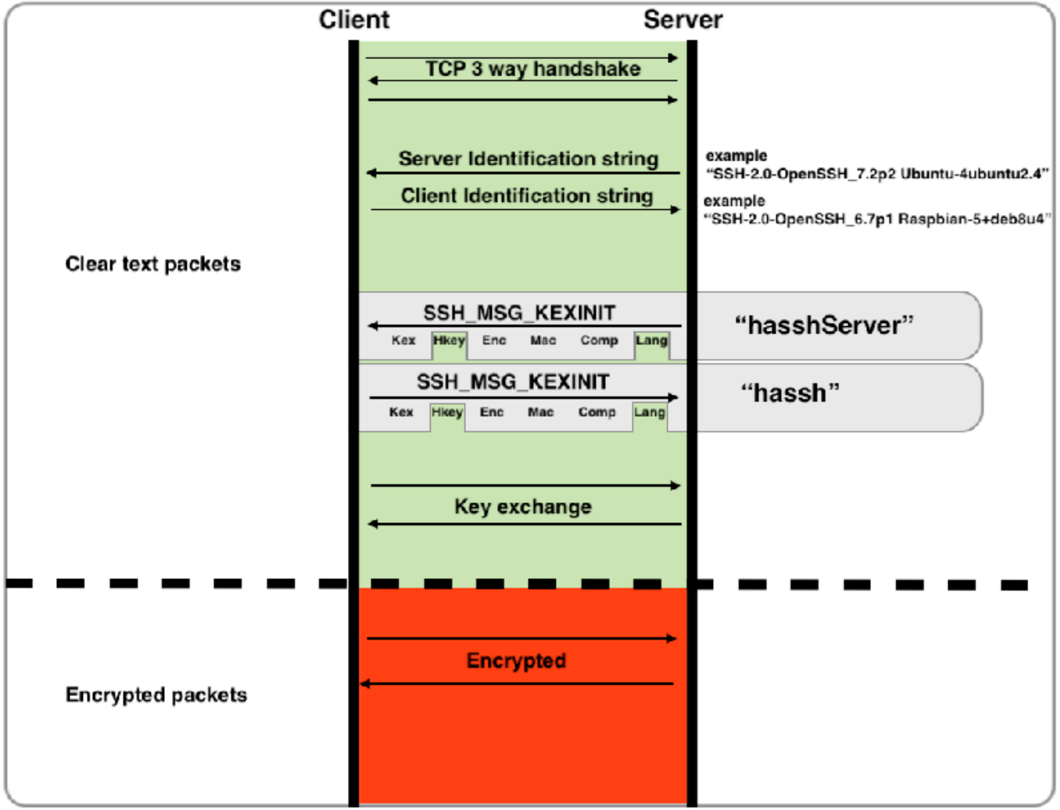
First packet:	2024-10-18 21:53:19
Last packet:	2024-10-18 21:53:46
Elapsed:	00:00:26

```
JA4                                    # Flows
t12i280600_bbd4f008d9b2_f28add8e7af0   2328 [57.3 %]
t10i410400_eeb9a0269cf9_282f11336259   1192 [29.4 %]
t12i210600_76e208dd3e22_f28add8e7af0   238 [5.9 %]
t12i080500_723ebca51f63_dccb52d5fcaf   158 [3.9 %]
t10i550400_59f835f43fe7_282f11336259   134 [3.3 %]
```

SSH

- HASSH is a network fingerprinting standard created by Salesforce which can be used to identify specific client and server SSH implementations.
- Fingerprints can be easily stored, searched and shared in the form of an MD5 fingerprint.
- They can be computed for both client and server and are useful to detect changes in SSH client software/configuration.
- As with JA4:
  - HASSH defines two fingerprints: one flow SSH client, and one for SSH server.
  - JA4+ includes a patented fingerprint names JA4SSH whose goal is to fingerprint traffic rather than client/server.

SSH Negotiation



https://github.com/salesforce/hassh

HASSH Client Fingerprint

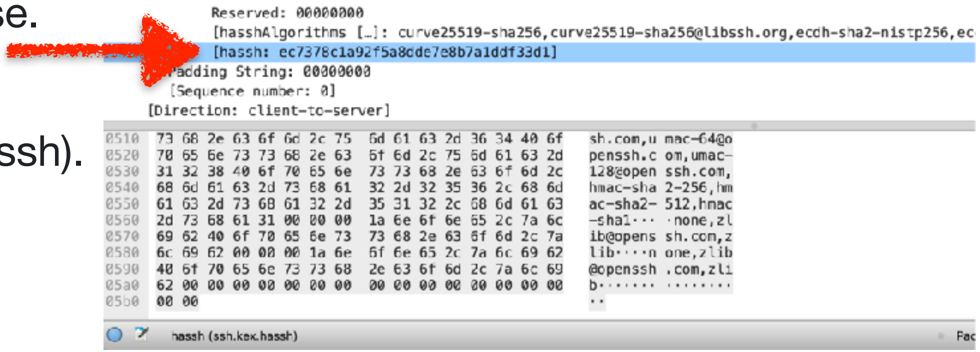
Function	Algorithms seen in SSH_MSG_KEXINIT packets
Key Exchange methods	`curve25519-sha256@libssh.org,diffie-hellman-group-exchange-sha256,ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1,diffie-hellman-group14-sha1,diffie-hellman-group14-sha256,diffie-hellman-group15-sha512,diffie-hellman-group16-sha512,diffie-hellman-group17-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256@ssh.com,diffie-hellman-group15-sha256,diffie-hellman-group15-sha256@ssh.com,diffie-hellman-group15-sha384@ssh.com,diffie-hellman-group16-sha256,diffie-hellman-group16-sha384@ssh.com,diffie-hellman-group16-sha512@ssh.com,diffie-hellman-group18-sha512@ssh.com`
Encryption	`aes128-cbc,aes128-ctr,aes192-cbc,aes192-ctr,aes256-cbc,aes256-ctr,blowfish-cbc,blowfish-ctr,cast128-cbc,cast128-ctr,idea-cbc,idea-ctr,serpent128-cbc,serpent128-ctr,serpent192-cbc,serpent192-ctr,serpent256-cbc,serpent256-ctr,3des-cbc,3des-ctr,twofish128-cbc,twofish128-ctr,twofish192-cbc,twofish192-ctr,twofish256-cbc,twofish256-ctr,twofish-cbc,arcfour,arcfour128,arcfour256`
Message Authentication	`hmac-sha1,hmac-sha1-96,hmac-md5,hmac-md5-96,hmac-sha2-256,hmac-sha2-512`
Compression	`zlib@openssh.com,zlib,none`

Concatenating these algorithms together with a delimiter of ";" and MD5 the resulting string, gives the hassh client fingerprint.

HASSH Server Fingerprint

Function	Algorithms seen in SSH_MSG_KEXINIT packets
Key Exchange methods	`diffie-hellman-group-exchange-sha256,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1`
Encryption	`aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,aes192-cbc,aes256-cbc,arcfour,rijndael-cbc@lysator.liu.se`
Message Authentication	`hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160,hmac-ripemd160@openssh.com,hmac-sha1-96,hmac-md5-96`
Compression	`none,zlib@openssh.com`

- At https://github.com/0x4D31/hassh-utils/blob/master/hasshdb you can find a large SSH fingerprint database.
- As of today, Wireshark supports HASSH (ssh.kex.hassh).



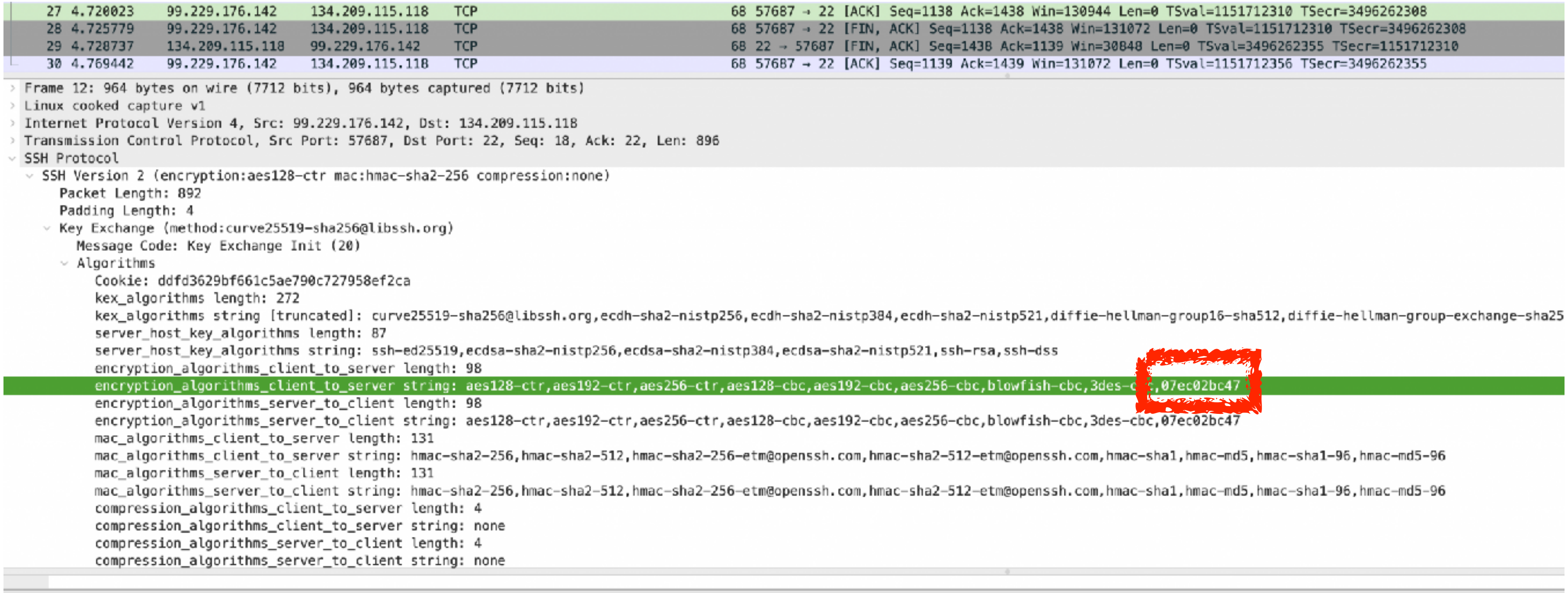
What Problems HASSH Addresses ? [1/2]

- HASSH adds contextual information to packet header information.
- The HASSH client is used to fingerprint the client, and thus:
    - Allow blocking clients outside of the "allowed set".
    - Detect exfiltration if data when using SSH clients with multiple distinct hashes.
    - NAT won't shield different SSH clients as they can now be detected with this technique.
    - Identify specific client versions.

What Problems HASSH Addresses ? [2/2]

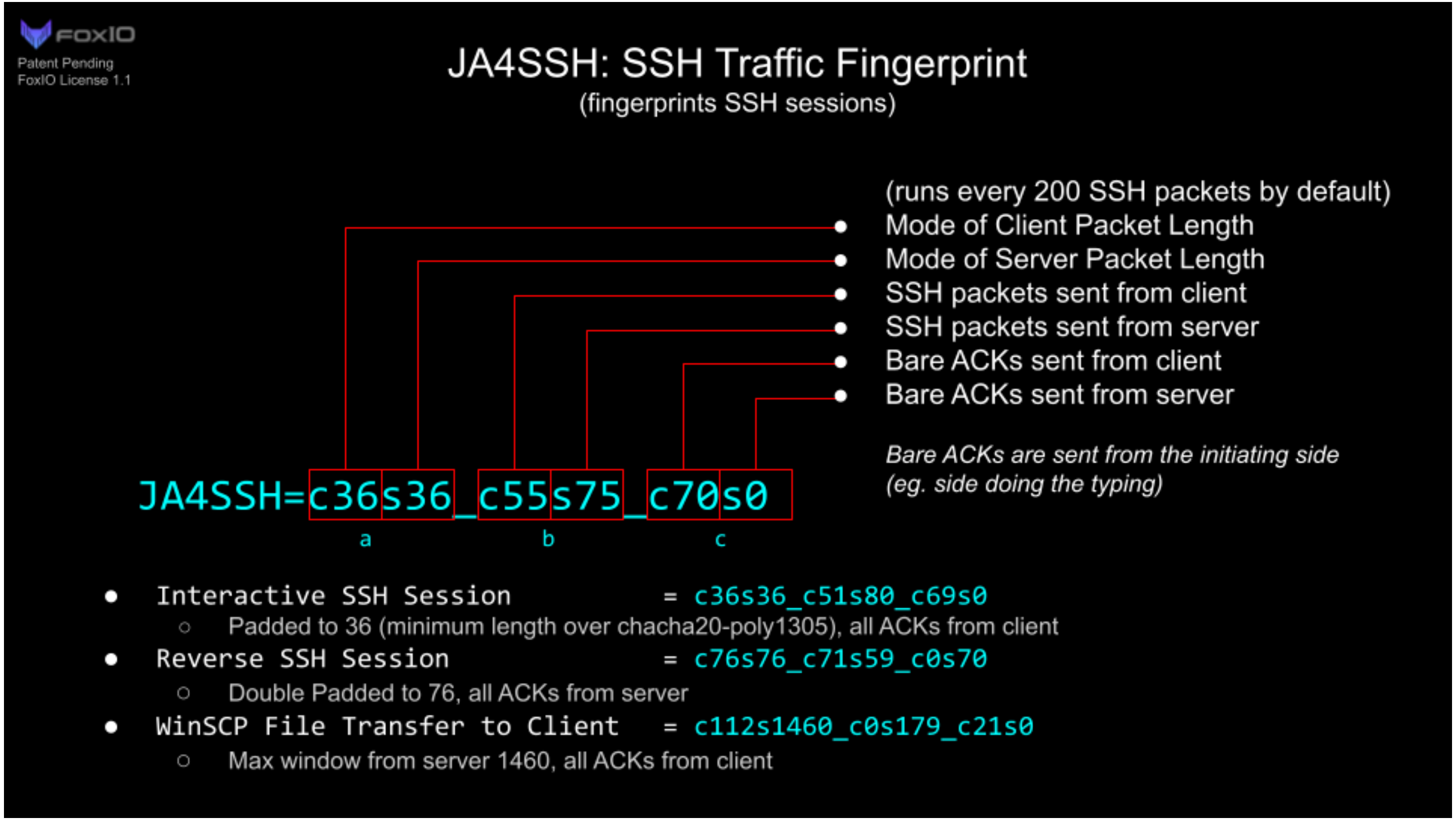
- The HASSH server can be used to detect if the server configuration is insecure or different from the past.
- In IoT or datacenter where configurations are static (or at least under strict control), fingerprint should be predictable.
- Same as HASSH client it can be used to block insecure servers, or detect unexpected changes in server configuration.

Evading SSH Fingerprinting (HASSH) with Arbitrary Ciphers



As with JA3, SSH implementations advertise random encryption algorithms in order to evade fingerprinting

JASSH



https://github.com/FoxIO-LLC/ja4/blob/main/technical_details/README.md

DHCP [1/2]

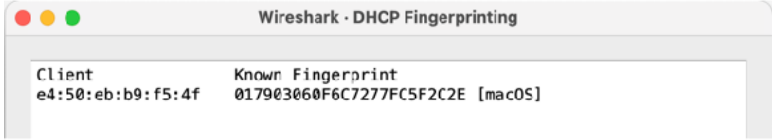
- DHCP does not have a "standard" fingerprint as JA4 or HASSH, but clients can be easily fingerprinted analysing DHCP options.
- In particular it is possible to list all options id's and list the parameters request list.

```
> User Datagram Protocol, Src Port: 68, Dst Port: 67
v Dynamic Host Configuration Protocol (Request)
    Message type: Boot Request (1)
    Hardware typ
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x39776de7
    Seconds elapsed: 4
  > Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0 (0.0.0.0)
    Your (client) IP address: 0.0.0.0 (0.0.0.0)
    Next server IP address: 0.0.0.0 (0.0.0.0)
    Relay agent IP address: 0.0.0.0 (0.0.0.0)
    Client MAC address: Apple_b9:f5:4f (e4:50:eb:b9:f5:4f)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  v Option: (55) Parameter Request List
      Length: 12
      Parameter Request List Item: (1) Subnet Mask
      Parameter Request List Item: (121) Classless Static Route
      Parameter Request List Item: (3) Router
      Parameter Request List Item: (6) Domain Name Server
      Parameter Request List Item: (15) Domain Name
      Parameter Request List Item: (108) IPv6-Only Preferred
      Parameter Request List Item: (114) DHCP Captive-Portal
      Parameter Request List Item: (119) Domain Search
      Parameter Request List Item: (252) Private/Proxy autodiscovery
      Parameter Request List Item: (95) LDAP [TODO:RFC3679]
      Parameter Request List Item: (44) NetBIOS over TCP/IP Name Server
      Parameter Request List Item: (46) NetBIOS over TCP/IP Node Type
  > Option: (57) Maximum DHCP Message Size
  > Option: (61) Client identifier
  > Option: (50) Requested IP Address (192.168.16.41)
  > Option: (51) IP Address Lease Time
  > Option: (12) Host Name
  > Option: (255) End
    Padding: 00000000000000
```

DHCP [2/2]

We have fingerprinted popular DHCP devices and embedded them in the ndpi.lua script.

```
local fingerprints = {
    ['017903060F77FC'] = 'iOS',
    ['017903060F77FC5F2C2E'] = 'macOS',
    ['0103060F775FFC2C2E2F'] = 'macOS',
    ['017903060F6C7277FC5F2C2E'] = 'macOS',
    ['0103060F775FFC2C2E'] = 'MacOS',
    ['0603010F0C2C51452B1242439607'] = 'HP LaserJet',
    ['0603010F42430D2C0C'] = 'HP LaserJet',
    ['01032C06070C0F16363A3B45122B7751999A'] = 'HP LaserJet',
    ['060FFC'] = 'Xerox Printer',
    ['0103063633'] = 'Windows',
    ['0103060F1F212B2C2E2F79F9FC'] = 'Windows',
    ['0103060F1F212B2C2E2F7779F9FC'] = 'Windows',
    ['0102060C0F1A1C79032128292A77F9FC11'] = 'Windows',
    ['010F03062C2E2F1F2179F92B'] = 'Windows',
    ['0103060C0F1C2A'] = 'Linux',
    ['011C02030F06770C2C2F1A792A79F921FC2A'] = 'Linux',
    ['0102030F060C2C'] = 'Apple AirPort',
    ['01792103060F1C333A3B77'] = 'Android',
}
```



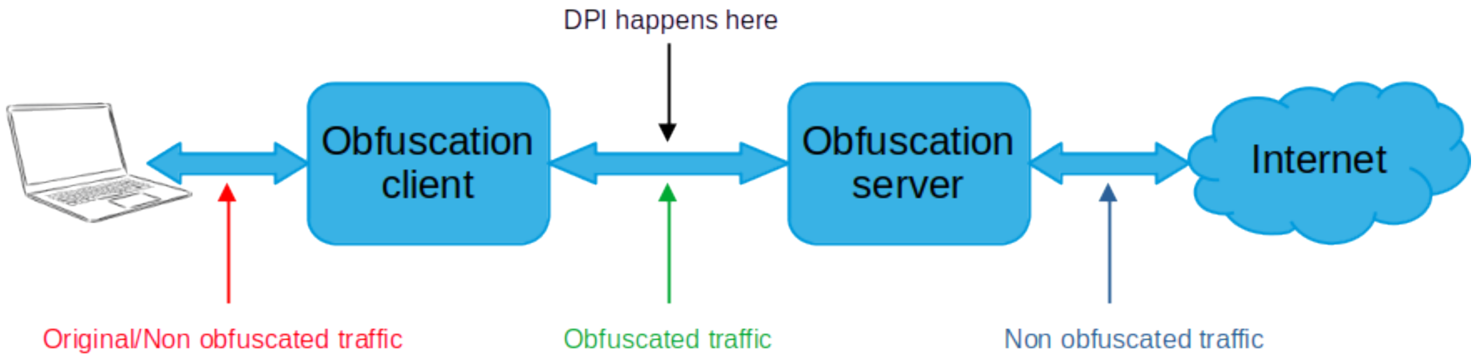
Part IV

Obfuscated Protocols Fingerprinting

Obfuscated traffic

- Traffic content is almost always encrypted, but traffic type/classification (or some traffic characteristics) can be usually inferred anyway
  - This is basically the goal of any DPI/Network Visibility/Firewall system
- Techniques to avoid detection are often called "obfuscation": the general idea is to make the traffic "look like" something else, hiding its true nature
  - if it works, the obfuscated traffic will be detected as something else (different traffic type or classification): the DPI system has been fooled/bypassed
  - blending in with standard and allowed traffic, it increases DPI systems error rate and operational costs in computation, time and money.
- There are two general strategies to obfuscate the traffic:
  - to mimic some content that it is allowed, like TLS
    - example: encapsulate the traffic in a TLS tunnel
  - to randomize the flow content, making it dissimilar to anything that it is specifically blocked
    - example: (fully) encrypt (again) the traffic, removing any plaintext info (or magic word or common patterns)

Obfuscated traffic: general schema



- Is a VPN an obfuscation technique? No, it isn't, even if it does "hide" your traffic *content*
  - All VPN services/apps (with their default configuration) are a simple wrapper over OpenVPN, Wireguard or IPSec
    - all of these protocols are easily detectable
    - Using a VPN you "hide" your traffic content, but you don't obfuscate it
- Sometimes you might want to obfuscate the VPN traffic itself!
  - All VPN apps have at least one option to enable some kind of obfuscation
- Example: 110_general_openvpn_over_tls.pcapng
- Example: 111_general_shadowsocks.pcapng

Fingerprints of obfuscated traffic

- We will show you that even obfuscated traffic can be easily fingerprinted/identified
  - We might not be able to detect the "real" (i.e. original) traffic type, but it is usually enough to know that some kind of obfuscated algorithm has been used
  - Obfuscated traffic is (very) suspicious per se
- Compared to the fingerprints Luca talked about, these new fingerprints:
  - are still cheap to calculate, even if they require more than 1 packet per flow
  - might be a more complex object than a simple string or number
- Three major user cases:
  - Fingerprint of obfuscated OpenVPN
  - Fingerprint of obfuscated TLS handshakes
  - Fingerprint of Fully Encrypted Protocols
- We implemented these logics in nDPI in an efficient way, allowing us to identify (some) obfuscated flows with good precision and low false positives rates, using minimum resources, at scale and in real time with live traffic
  - Wireshark (via extcap) will be used to show the final results and the raw fingerprints; these fields can be filtered or you can collect some statistics about them, as usual

Obfuscated traffic: disclaimer

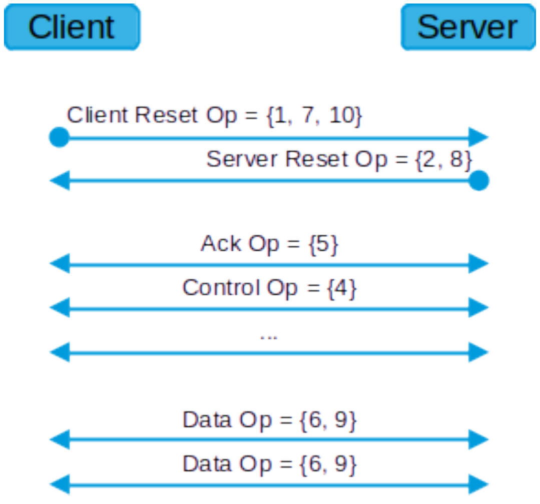
- Detecting obfuscated traffic might be a sensitive topic; different people might have different opinions about it. However:
  - the techniques we will talk about are based on academic papers publicly available and presented at major conferences
  - responsible disclosure: all involved parties have been notified before papers publication
- We are not the authors of these papers
- In our tests we used some VPN apps and some V2Ray protocols (ShadowSocks, VMess, Trojan,...) with their default/simplest configurations and without enabling advance features.
  - The original papers have some considerations/results about these more complex configurations.
  - Tradeoff between ease of deployment and obfuscation efficiency
  - V2Ray is still (one of) the best choice if you need to obfuscate your own traffic

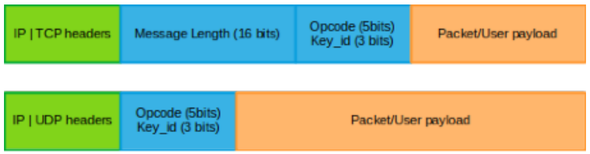
Fingerprint of OpenVPN

- OpenVPN is Open to VPN Fingerprinting, Xue et al.,USENIX Security '31, 2022

OpenVPN: overview

- One of the most used (and old) protocol for creating VPN
- A dozen of different messages; the first byte of the OpenVPN header is the message type (i.e. opcode)
- Initial handshake with a TLS-style exchange of key materials
- Example: 120_openvpn_plain_tcp.pcapng





OpenVPN: XOR patch

- OpenVPN protocol is easily detectable (via message types)
- In 2013, a patch adding obfuscation capability to OpenVPN has been proposed
  - Simply XOR with a shared key
  - Not accepted by upstream maintainers: no proper evaluation of security risks/claims
- Nonetheless, in the following years this patch has been included in a lot of different proprietary VPN apps
- At least from 2020 it is well known that this patch has a fatal flow: the first byte of each message is always encrypted with the same byte of the key. Therefore, in a connection, the same opcode would be always mapped/encrypted to the same value
- The paper (from 2022) found that 34 out of 41 “obfuscated” VPN configurations are vulnerable to this (and similar) "bug"

OpenVPN: fingerprint

- Basic idea:
  - The set of the opcodes of the first packets of a standard OpenVPN flow is quite peculiar:
    - one (different) opcode (i.e. resets) per direction only at the very beginning
    - real handshake with a few different opcodes (i.e. ack/control/...)
    - from one packet forward, the opcode is always the same (i.e. data)
  - Because of the XOR patch flaw, an obfuscated OpenVPN flow has a "similar" set, i.e. a set with the same cardinality
- The fingerprint is the ordered collection of the first byte of the initial packets
- Example: 121_openvpn_udp_obfuscated.pcapng (with and without extcap)

OpenVPN: results

- We tested all the VPN apps vulnerable to this heuristic according to the original paper
- All of them (but one) are still vulnerable
- Our implementation detect these flows with TPR = ~100%
- What about false positives?

OpenVPN: results

Controlled traffic without obfuscated OpenVPN flows	Matches/Total flows	Real traffic from an ISP	Matches/Total flows
Firefox (random sites)	0/4053	TCP	0/544066
Chrome (random sites)	0/6746	UDP	20/559791
Android (random apps, web, games, calls)	0/3315	TCP (443 only)	4/3429006
Edge (random sites)	0/7372	UDP (no 53, 443, 2152, 4500)	40/298849
iPhone (random apps, web)	0/3224	UDP(443 only)	25/988079
Office span port (Win, Linux, VM, Phones)	0/3968	STUN	0/106488
		IPv6 (no 53, 443)	2/879107
		RTP and DTLS	6/8452

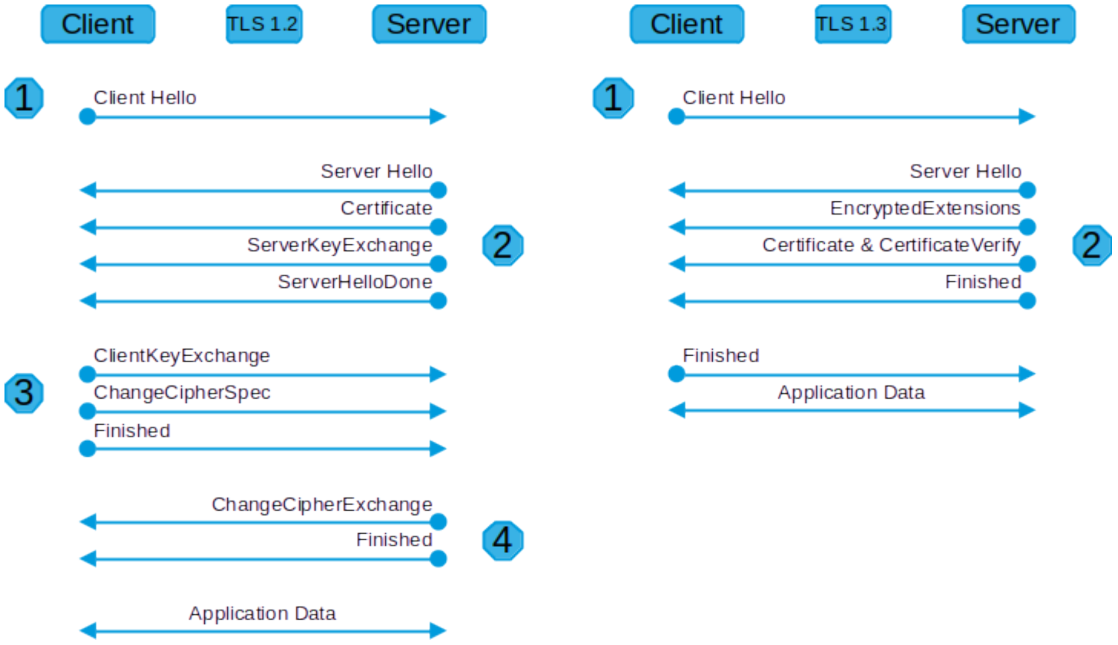
- FPR (worst case) =
  - ~1*10^-5 (with 10 pkts per flow)
  - ~3*10^-6 (with 20 pkts per flow)

Fingerprint of obfuscated TLS

- Fingerprinting Obfuscated Proxy Traffic with Encapsulated TLS Handshakes, Xue et al.,USENIX '24

TLS: handshake

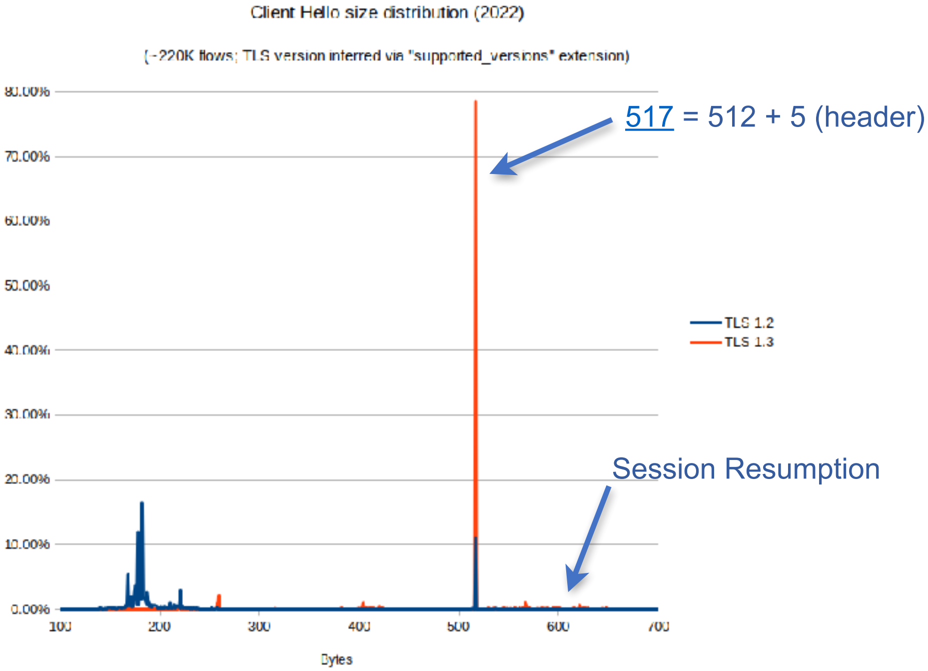
- Different messages exchanged in multiples TCP packets
- Burst/flight: consecutive packets sent in the same direction
- Only "full" handshake, no session resumption or 0RTT

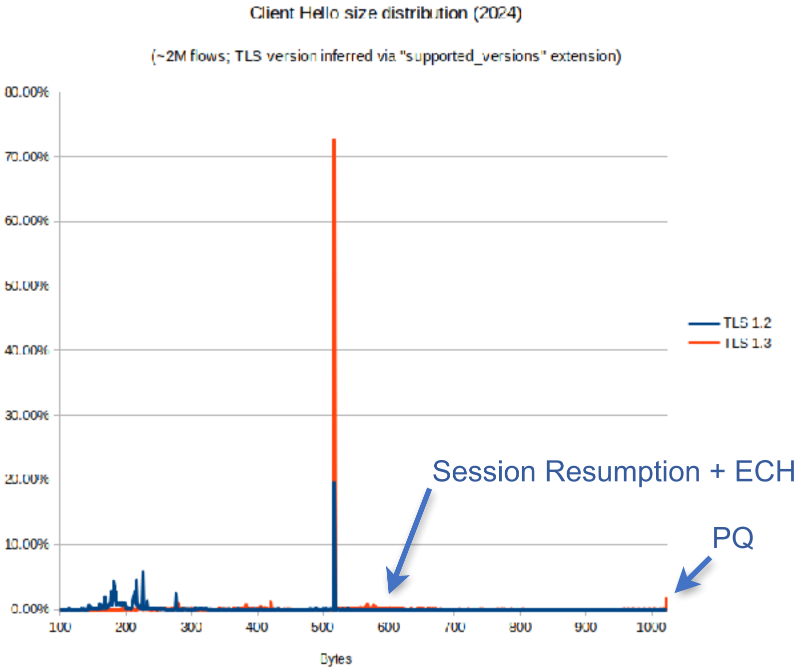


Obfuscated TLS fingerprinting

- Terminology:
  - "packets/bytes distribution" == distribution/histogram of packets size and number
  - "burst/flight distribution" == packets/bytes distribution of all the packets belonging to the same burst
- Basic idea:
  - the packets/bytes distribution of a (plain) TLS handshake (i.e. bursts distribution) is quite unique
  - this fingerprint is still detectable if the handshake is encrypted/ proxied/obfuscated/tunneled
- The fingerprint is the packets/bytes distribution of the initial bursts

TLS: statistics





TLS: new features

- CH size and its characteristics haven't really changed for a long time, since 2013
- In the last years, two new features have been developed and also deployed:
    - Post-Quantum algorithms
    - Encrypted Client Hello
- These two extensions have a significant impact on CH size

TLS: Post Quantum algorithms

- Idea: deploy today new cryptographic algorithms secure against future quantum computers
  - For details: "Real-world post-quantum TLS in Wireshark" by Peter Wu, SharkFestUS-24
- Effects: CHs (and SHs) are significant bigger (~1200 more bytes) because of the new crypto key material

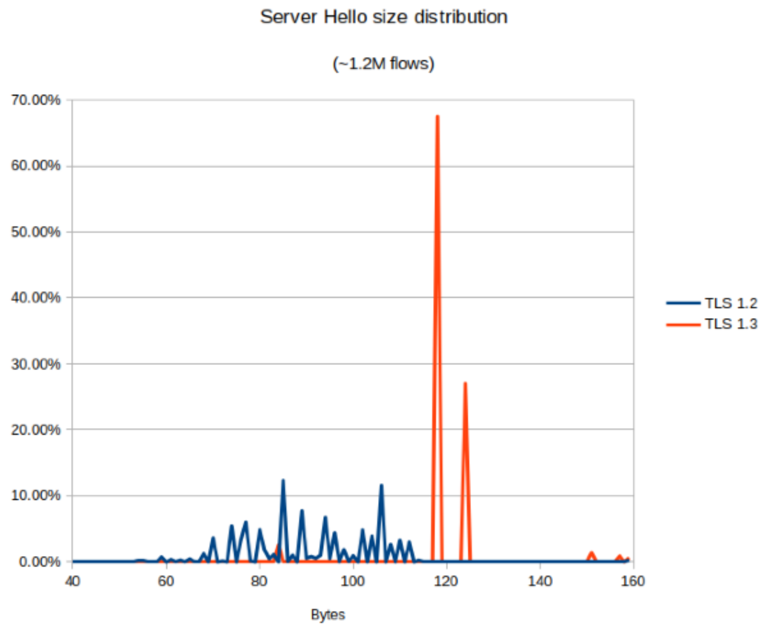
TLS: Encrypted Client Hello

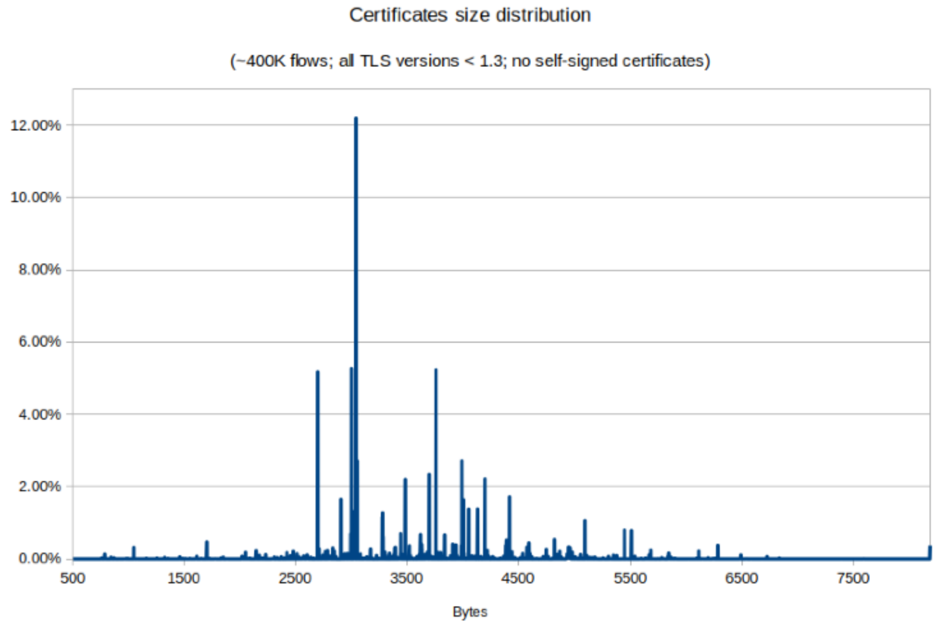
- ClientHello messages are sent in cleartext. ECH is a new TLS extension allowing sensitive information (SNI, ALPNs) to be sent "encrypted" by the client
- Goal: to ensure that connections to servers/sites in the same anonymity set are indistinguishable from one another.
  - Basically some kind of "legal"/"allowed" domain fronting
- Two CHs are involved:
  - Outer CH: in cleartext, with a SNI referring to the anonymity set
  - Inner CH: encrypted, with the "real"/hidden SNI

TLS: Encrypted Client Hello

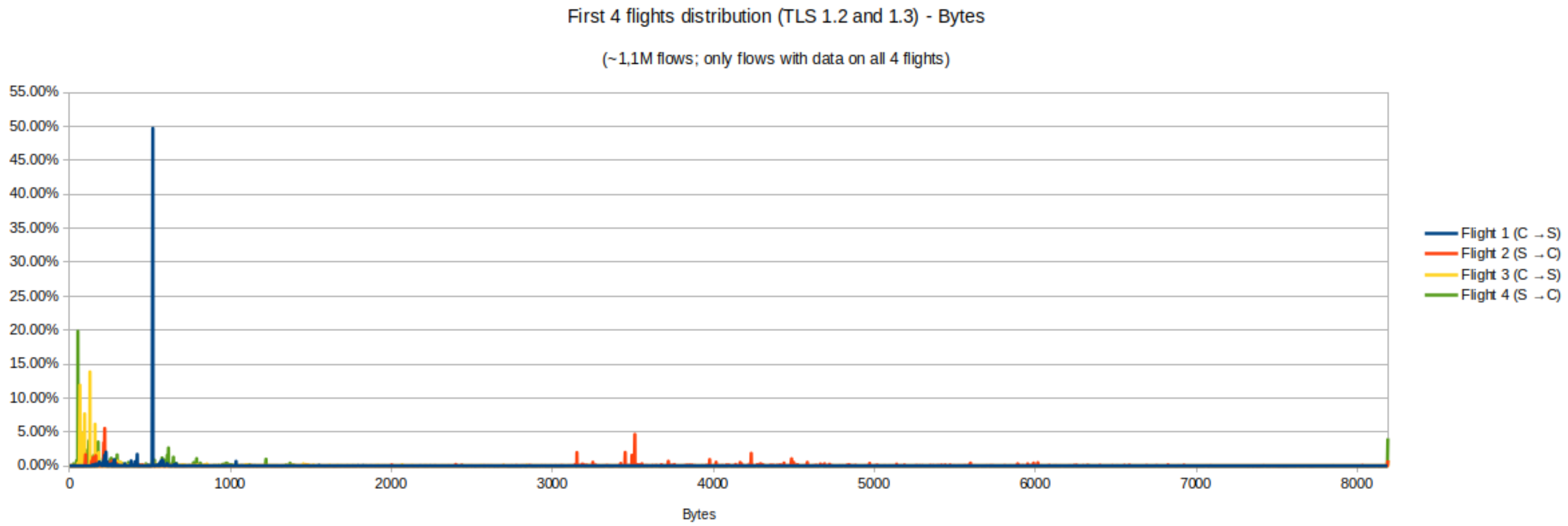
- The fact that ECH is being used is still visible
  - Greasing: clients might send dummy/fake ECH extension that is ignored by the server but it might help deployment ("don't stick out") and avoid ossification.
- For the purposes of this talk: CH with ECH is a little bit bigger (~150 more bytes)
- Wireshark and TLS libraries don't decrypt ECH, yet. Preliminary patches from Yaroslav Rosomakho, @ZScaler
- Example: 130_ech.pcap (with standard and wireshark-echkeylog)
- Example: 131_firefox_ech_pq_all_combinations.pcap

TLS: statistics

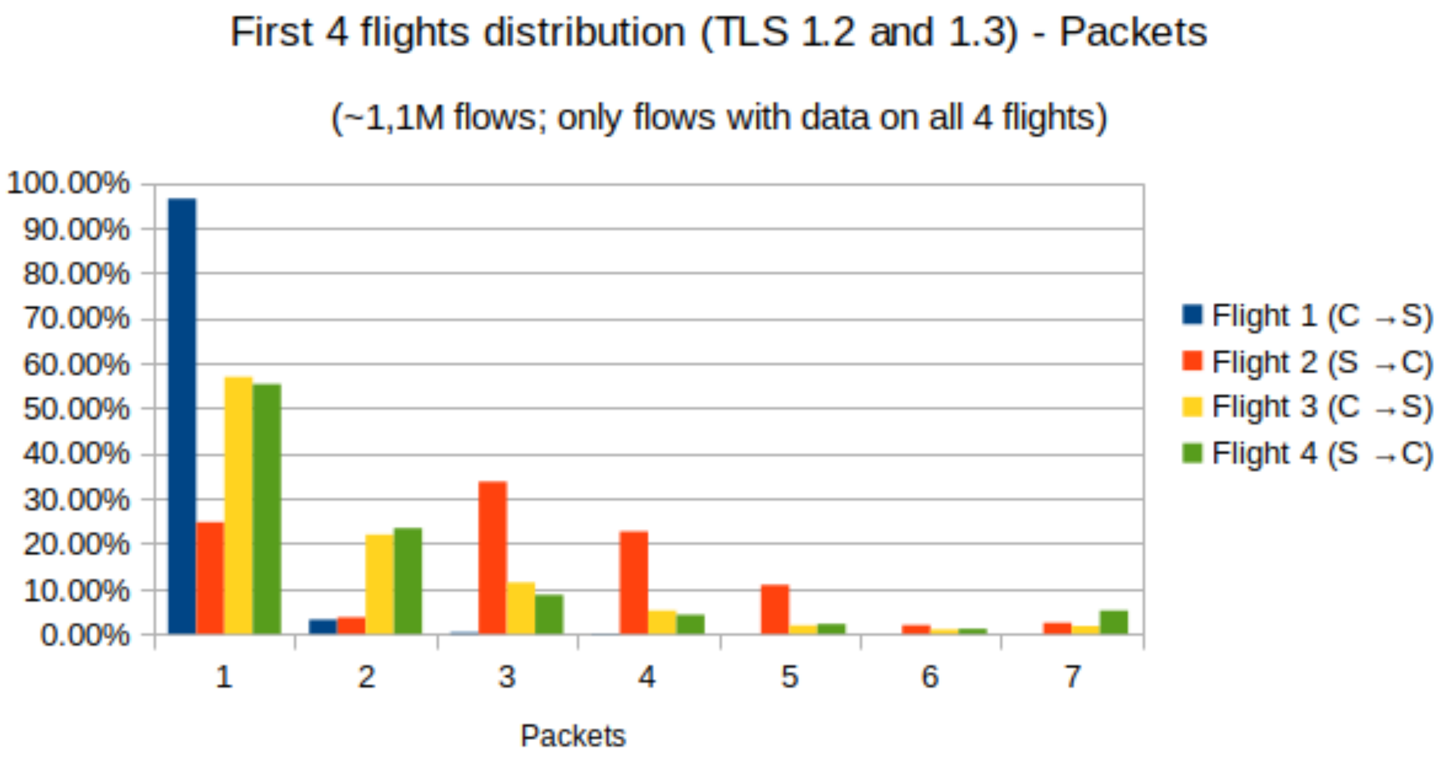




TLS: statistics



TLS: statistics



Obfuscated TLS

- TLS flows have a quite unique fingerprint about the packets/bytes distributions of their handshake
- The common idea underpinning all forms of proxying and tunneling is that of nested protocol stacks, where one protocol stack is encapsulated within the payload of another protocol
- This fingerprint is still detectable even if the TLS handshake is encrypted/ obfuscated/encapsulated
    - Core reason: tunneling/encryption doesn't change packet timing/ direction and size (too much, at least); it doesn't usually add/remove packets
- This fingerprint is detectable regardless of the specific obfuscation technic: it is protocol agnostic

Fingerprint before/after obfuscation

- Example: ShadowSocks (111_general_shadowsocks.pcapng)

tcp.len > 0 && tcp.stream != 3

	No.	Time	Source	Source Port	Protocol	Destination	Length	TCP Segment Len	Info
	4	2024-08-31 10:31…	127.0.0.1	44424	Socks	127.0.0.1	72	4	Version: 5 Co
	6	2024-08-31 10:31…	127.0.0.1	1080	Socks	127.0.0.1	70	2	Version: 5 Co
	8	2024-08-31 10:31…	127.0.0.1	44424	Socks	127.0.0.1	90	22	Version: 5 Co
	9	2024-08-31 10:31…	127.0.0.1	1080	Socks	127.0.0.1	78	10	Version: 5 Co
A	14	2024-08-31 10:31…	127.0.0.1	44424	TLSv1.3	127.0.0.1	585	517	Client Hello
	15	2024-08-31 10:31…	127.0.0.1	40164	TCP	127.0.0.1	704	636	40164 → 1234
	24	2024-08-31 10:31…	127.0.0.1	1234	TCP	127.0.0.1	1342	1274	1234 → 40164
B	30	2024-08-31 10:31…	127.0.0.1	1080	TLSv1.3	127.0.0.1	1276	1208	Server Hello,
	32	2024-08-31 10:31…	127.0.0.1	1234	TCP	127.0.0.1	5561	5493	1234 → 40164
B	34	2024-08-31 10:31…	127.0.0.1	1080	TLSv1.3	127.0.0.1	5459	5391	Application D
C	36	2024-08-31 10:31…	127.0.0.1	44424	TLSv1.3	127.0.0.1	148	80	Change Cipher
	37	2024-08-31 10:31…	127.0.0.1	40164	TCP	127.0.0.1	182	114	40164 → 1234
C	38	2024-08-31 10:31…	127.0.0.1	44424	TLSv1.3	127.0.0.1	114	46	Application D
C	39	2024-08-31 10:31…	127.0.0.1	44424	TLSv1.3	127.0.0.1	117	49	Application D
C	40	2024-08-31 10:31…	127.0.0.1	44424	TLSv1.3	127.0.0.1	103	35	Application D
C	42	2024-08-31 10:31…	127.0.0.1	44424	TLSv1.3	127.0.0.1	131	63	Application D
	44	2024-08-31 10:31…	127.0.0.1	40164	TCP	127.0.0.1	197	129	40164 → 1234
	45	2024-08-31 10:31…	127.0.0.1	40164	TCP	127.0.0.1	200	132	40164 → 1234
	49	2024-08-31 10:31…	127.0.0.1	1234	TCP	127.0.0.1	750	682	1234 → 40164
D	50	2024-08-31 10:31…	127.0.0.1	1080	TLSv1.3	127.0.0.1	716	648	Application D
	51	2024-08-31 10:31…	127.0.0.1	44424	TLSv1.3	127.0.0.1	99	31	Application D

- Original flow bursts (bytes): {517, 6599, 273, 648}
- ShadowSocks flow bursts (bytes): {636, 6767, 375, 682}

Fingerprint before/after obfuscation

- Example: TLS over TLS (132_vmess-tcp-tls_curl.pcapng)

tcp.len > 0 && tcp.stream in {0,1}

	No.	Time	Source	Source Port	Protocol	Destination	Length	TCP Segment Len	Info
	4	2024-08-31 19:20:50,807747990	127.0.0.1	40136	Socks	127.0.0.1	72	4	Version: 5 C
	6	2024-08-31 19:20:50,808089120	127.0.0.1	1080	Socks	127.0.0.1	70	2	Version: 5 C
	16	2024-08-31 19:20:50,812187491	127.0.0.1	40136	Socks	127.0.0.1	78	10	Version: 5 C
	17	2024-08-31 19:20:50,812298989	127.0.0.1	1080	Socks	127.0.0.1	78	10	Version: 5 C
	29	2024-08-31 19:20:50,847484167	127.0.0.1	57874	TLSv1.3	127.0.0.1	346	278	Client Hello
	31	2024-08-31 19:20:50,848915840	127.0.0.1	1234	TLSv1.3	127.0.0.1	1188	1120	Server Hello,
	34	2024-08-31 19:20:50,855146658	127.0.0.1	57874	TLSv1.3	127.0.0.1	132	64	Change Ciphe
A	35	2024-08-31 19:20:50,871252644	127.0.0.1	40136	TLSv1.3	127.0.0.1	585	517	Client Hello
	36	2024-08-31 19:20:50,871953776	127.0.0.1	57874	TLSv1.3	127.0.0.1	731	663	Application
	45	2024-08-31 19:20:50,896375696	127.0.0.1	1234	TLSv1.3	127.0.0.1	1276	1208	Application
	46	2024-08-31 19:20:50,896417817	127.0.0.1	1234	TLSv1.3	127.0.0.1	952	884	Application
	48	2024-08-31 19:20:50,896465640	127.0.0.1	1234	TLSv1.3	127.0.0.1	2138	2070	Application
	50	2024-08-31 19:20:50,896496160	127.0.0.1	1234	TLSv1.3	127.0.0.1	2138	2070	Application
	52	2024-08-31 19:20:50,896539877	127.0.0.1	1234	TLSv1.3	127.0.0.1	586	518	Application
B	53	2024-08-31 19:20:50,896744442	127.0.0.1	1080	TLSv1.3	127.0.0.1	2076	2008	Server Hello,
B	55	2024-08-31 19:20:50,896793685	127.0.0.1	1080	Socks	127.0.0.1	2116	2048	Version: 5 [
B	57	2024-08-31 19:20:50,896827248	127.0.0.1	1080	TLSv1.3	127.0.0.1	2612	2544	Application
C	59	2024-08-31 19:20:50,899922692	127.0.0.1	40136	TLSv1.3	127.0.0.1	148	80	Change Ciphe
C	60	2024-08-31 19:20:50,900172065	127.0.0.1	40136	TLSv1.3	127.0.0.1	114	46	Application
	61	2024-08-31 19:20:50,900184401	127.0.0.1	57874	TLSv1.3	127.0.0.1	172	104	Application
C	62	2024-08-31 19:20:50,900200441	127.0.0.1	40136	TLSv1.3	127.0.0.1	117	49	Application
C	63	2024-08-31 19:20:50,900217797	127.0.0.1	40136	TLSv1.3	127.0.0.1	103	35	Application
C	65	2024-08-31 19:20:50,900273987	127.0.0.1	40136	TLSv1.3	127.0.0.1	131	63	Application
	68	2024-08-31 19:20:50,900392807	127.0.0.1	57874	TLSv1.3	127.0.0.1	222	154	Application
	69	2024-08-31 19:20:50,900440909	127.0.0.1	57874	TLSv1.3	127.0.0.1	155	87	Application
	73	2024-08-31 19:20:50,903315384	127.0.0.1	1234	TLSv1.3	127.0.0.1	740	672	Application
D	74	2024-08-31 19:20:50,903449925	127.0.0.1	1080	TLSv1.3	127.0.0.1	716	648	Application

- Original flow bursts (bytes) : {517, 6600, 273, 648}
- VMess flow bursts (bytes): {663, 6750, 345, 672}

Implementation

- We create some models with "standard" TLS flows (web/browser traffic) as reference
- With real traffic, we evaluate the burst bytes/pkts distribution (in a sliding window) through the initial portion of the flow: if it "looks like" the distribution of "standard"/reference TLS traffic, then it is likely that we found an obfuscated TLS handshake
- From a mathematical point of view, "looks like" means "the distance between this specific distribution and the reference model is less than a threshold"
    - Threshold value is choose as tradeoff between TPR and FPR
- Example: 133_trojan-tcp-tls.pcapng (with extcap)
- Example: 134_vmess-websocket.pcapng (with extcap)
- Example: 135_shadowsocks-tcp.pcapng (with extcap)

TLS: results

Controlled traffic with only obfuscated TLS flows	Matches/Total flows
Firefox + ShadowSocks	303/426
Firefox + ShadowSocks(2)	1600/2176
Chrome + VMess over TLS	1692/2366
Chrome + VMess over Websocket	2428/3638
Firefox + Trojan over TLS	971/1317

- TPR = ~70%(similar to paper results)

TLS: results

Controlled traffic without obfuscated TLS flows	Matches/Total flows	Real traffic from an ISP	Matches/Total flows
Firefox (random sites)	3/4053	TCP	318/544066
Chrome (random sites)	4/6746	UDP	311/559791
Android (random apps, web, games, calls)	0/3315	TCP (443 only)	559/3429006
Edge (random sites)	1/7372	UDP (no 53, 443, 2152, 4500)	1739/298849
iPhone (random apps, web)	0/3224	UDP(443 only)	1769/988079
Office span port (Win, Linux, VM, Phones)	2/3968	STUN	0/106488
		IPv6 (no 53, 443)	226/879107
		RTP and DTLS	34/8452

- FPR (worst case) = ~0.7*10^-3

TLS: results

- Is FPR ~1*10^-3 good enough?
  - Usually, no, it isn't. Due to the huge volume of traffic passing through a real network and the low base rate of obfuscated traffic in the wild, this fingerprinting logic would likely label more legitimate connections as proxied than actual proxied connections
  - This fingerprint can be used anyway as a foundation upon which build further application logic, for example moving from "obfuscated flow" to "obfuscated server"
    - active probing of suspected obfuscated servers
    - statistical analysis of the overall traffic of suspected obfuscated servers
- Example: 138_obfuscated_servers_analysis_1.pcapng (via extcap)
- Example: 139_obfuscated_servers_analysis_2.pcapng (via extcap)

Obfuscated TLS fingerprinting

- Limitations and future work:
  - add support for session resumption/0RTT
  - take a look at UDP/QUIC (MASQUE)
  - track ECH/PQ development/deployment
    - Chrome 131 (06/12) will move from Kyber to ML_KEM
    - After one year, Cloudflare is re-enabling ECH
  - what about no-web traffic (i.e. IoT, VPN)?
    - TLS stack on IoT devices and VPN apps is usually not fully-featured as in major browsers
    - OpenVPN may use an obfuscated-like TLS handshake and most VPN apps provided a configuration option to encapsulate OpenVPN traffic on a TLS tunnel: this fingerprint might work with VPN also

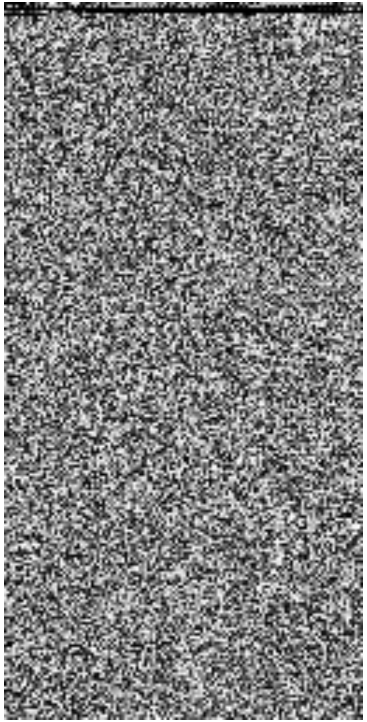
- How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic, Mingshi Wu et al., USENIX Security 32, 2023

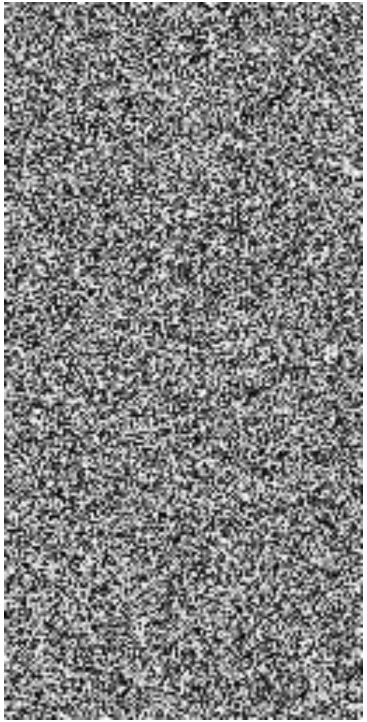
Fully Encrypted Protocols

- Fully Encrypted Protocols (FEPs): every byte in a connection appears independently and uniformly random.
    - TLS, HTTP, QUIC are NOT FEP
    - Examples: ShadowSocks, OBFS, VMess

FEP example

- Protocol bytes mapped to grayscale pixel: TLS and OBFS4.
- See: "How cryptography relates to Internet censorship circumvention", David Fifield, 2024





FEPs fingerprinting

- Basic idea:
  - most common protocols are not FEP
  - measure the entropy/randomness of the flow: if it is "too much", then it is likely a FEP
- The fingerprint is the flow entropy
- The concept seems simple enough: the hard part is how to measure the entropy to avoid too many false positives
- Example: 140_shadowsocks.pcap (with extcap)
- Results: TPR = ~100% with ShadowSocks. What about false positives?

FEPs: results

Controlled traffic without FEPs	Matches/Total flows	Real traffic from an ISP	Matches/Total flows
Firefox (random sites)	0/4053	TCP	8926/544066
Chrome (random sites)	0/6746	UDP	0/559791
Android (random apps, web, games, calls)	12/3315	TCP (443 only)	0/3429006
Edge (random sites)	1/7372	UDP (no 53, 443, 2152, 4500)	0/298849
iPhone (random apps, web)	0/3224	UDP(443 only)	0/988079
Office span port (Win, Linux, VM, Phones)	16/3968	STUN	0/106488
		IPv6 (no 53, 443)	0/879107
		RTP and DTLS	0/8452

- FPR (worst case) = ~1*10^-3

Part V
Obfuscation pitfalls

Don't roll your own ~~crypto~~ obfuscation code

- Avoiding fingerprints and deploy a working obfuscation technique is hard
  - We have just seen a few examples where with some basic statistic algorithms we can fingerprint/identify obfuscated flows
- There are good working programs/libraries that you should use if you want to obfuscate some traffic: V2Ray, uTLS, Tor...
- Don't try to roll your own obfuscation code: you will do it wrong and it will fail in a catastrophic/hilarious way!

Don't roll your own ~~crypto~~ obfuscation code

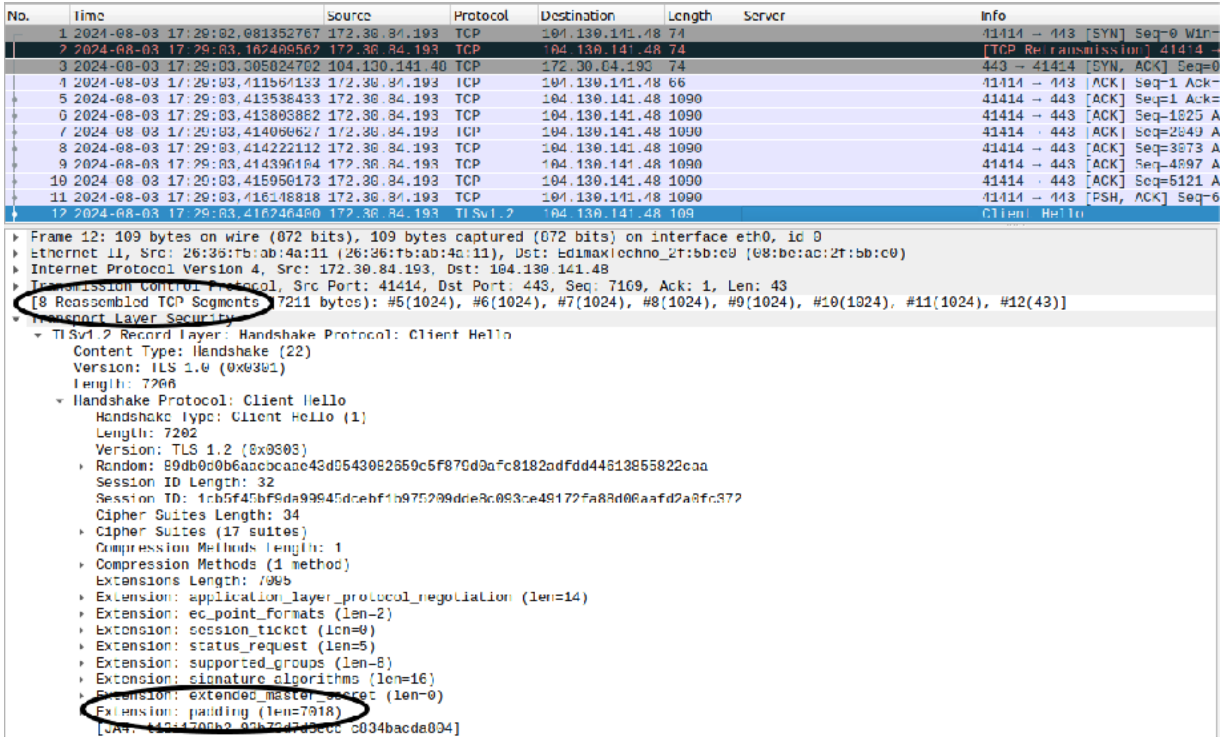
- "[it] provides extended online freedom. It [...] get[s] past any VPN blocks"

No.	Time	Source	Protocol	Destination	Length	SNI	Info
1	2024-09-12 08:15:50,741602224	10.57.153.127	TCP	152.47.116.178	74		45190 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=146
2	2024-09-12 08:15:50,991578912	152.47.116.178	TCP	10.57.153.127	74		443 → 45190 [SYN, ACK] Seq=0 Ack=1 Win=43440 Le
3	2024-09-12 08:15:50,997737849	10.57.153.127	TCP	152.47.116.178	66		45190 → 443 [ACK] Seq=1 Ack=1 Win=88064 Len=0 T
4	2024-09-12 08:15:50,998672017	10.57.153.127	TLSv1.3	152.47.116.178	345	[redacted]	Client Hello (SNI=[redacted]
5	2024-09-12 08:15:51,247878011	152.47.116.178	TCP	10.57.153.127	66		443 → 45190 [ACK] Seq=1 Ack=280 Win=45056 Len=0
6	2024-09-12 08:15:51,256155184	152.47.116.178	TLSv1.3	10.57.153.127	1506		Server Hello, Change Cipher Spec, Application D
7	2024-09-12 08:15:51,256507582	152.47.116.178	TLSv1.3	10.57.153.127	780		Application Data, Application Data, Application

- "Circumvent Censorship Feature: [...] protocol-level anti-censorship"

No.	Time	Source	Protocol	Destination	Length	Certificate Issuer	Info
1	2024-07-23 14:30:32,772331274	172.30.84.193	TCP	17.5.27.63	74		42192 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=14
2	2024-07-23 14:30:32,911061726	17.5.27.63	TCP	172.30.84.193	74		443 → 42192 [SYN, ACK] Seq=0 Ack=1 Win=43440
3	2024-07-23 14:30:32,997668634	172.30.84.193	TCP	17.5.27.63	66		42192 → 443 [ACK] Seq=1 Ack=1 Win=88064 Len=0
4	2024-07-23 14:30:33,016839106	172.30.84.193	TLSv1.2	17.5.27.63	248		Client Hello
5	2024-07-23 14:30:33,155514905	17.5.27.63	TCP	172.30.84.193	66		443 → 42192 [ACK] Seq=1 Ack=183 Win=43520 Len=
6	2024-07-23 14:30:33,155535541	17.5.27.63	TLSv1.2	172.30.84.193	1102	[redacted]	Server Hello, Certificate, Server Hello Done
7	2024-07-23 14:30:33,204957625	172.30.84.193	TCP	17.5.27.63	66		42192 → 443 [ACK] Seq=183 Ack=1037 Win=90112
8	2024-07-23 14:30:33,209874268	172.30.84.193	TLSv1.2	17.5.27.63	384		Client Key Exchange, Change Cipher Spec, Encry
9	2024-07-23 14:30:33,348237532	17.5.27.63	TCP	172.30.84.193	66		443 → 42192 [ACK] Seq=1037 Ack=501 Win=43520
10	2024-07-23 14:30:33,348263461	17.5.27.63	TLSv1.2	172.30.84.193	308		New Session Ticket, Change Cipher Spec, Encryp

- "XXX is an OpenVPN tunnel masked to look like HTTPS traffic. This protocol is very helpful on restrictive networks"
- "Don't stick out" problem: if your feature is visible at the network level, and you are the only one using it, you can trivially be detected



Don't roll your own ~~crypto~~ obfuscation code

SharkFest'24 EUROPE
Vienna, Austria • #sf24eu

- Adding some unexpected or carefully crafted data/ packets at the beginning of the flow might be a good idea
- Standard ports 443/53 are problematic: unknown traffic on 443/53 might be more suspicious than on random ports

No.	Time	Source	Protocol	Destination	Length	Server	Info
1	2024-08-03 17:28:43,293721491	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
2	2024-08-03 17:28:43,293778687	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
3	2024-08-03 17:28:43,344580812	172.30.84.193	UDP	82.221.172.193	50		37282 → 443 Len=8
4	2024-08-03 17:28:43,344600178	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
5	2024-08-03 17:28:43,344609225	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
6	2024-08-03 17:28:43,344632453	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
7	2024-08-03 17:28:43,344630795	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
8	2024-08-03 17:28:43,344848440	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
9	2024-08-03 17:28:43,344863205	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
10	2024-08-03 17:28:43,344868684	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
11	2024-08-03 17:28:43,344878786	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
12	2024-08-03 17:28:43,344879217	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
13	2024-08-03 17:28:43,344009127	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
14	2024-08-03 17:28:43,345035324	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
15	2024-08-03 17:28:43,345057976	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
16	2024-08-03 17:28:43,345060775	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
17	2024-08-03 17:28:43,345074082	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
18	2024-08-03 17:28:43,345080622	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
19	2024-08-03 17:28:43,345090086	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
20	2024-08-03 17:28:43,345102008	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
21	2024-08-03 17:28:43,345291060	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
22	2024-08-03 17:28:43,345314065	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
23	2024-08-03 17:28:43,345320671	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
24	2024-08-03 17:28:43,346614073	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
25	2024-08-03 17:28:43,346663631	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
26	2024-08-03 17:28:43,346676641	172.30.84.193	UDP	82.221.172.193	90		37282 → 443 Len=48
27	2024-08-03 17:28:43,347774627	172.30.84.193	WireGuard	82.221.172.193	190		Handshake Initiation, sender=0xBE2FE42C
28	2024-08-03 17:28:43,588280124	82.221.172.193	WireGuard	172.30.84.193	134		Handshake Response, sender 0x005D3401, receiver 0xBE2FE42C
29	2024-08-03 17:28:43,591879158	172.30.84.193	WireGuard	82.221.172.193	74		Keepalive, receiver 0x005D3401, counter 0
30	2024-08-03 17:28:43,592209119	172.30.84.193	WireGuard	82.221.172.193	154		Transport Data, receiver=0x005D3401, counter=1, datalen=80
31	2024-08-03 17:28:43,592351039	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver=0x005D3401, counter=2, datalen=64
32	2024-08-03 17:28:43,592477377	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver=0x005D3401, counter=3, datalen=64
33	2024-08-03 17:28:43,731521185	82.221.172.193	WireGuard	172.30.84.193	264		Transport Data, receiver=0xBE2FE42C, counter=0, datalen=190
34	2024-08-03 17:28:43,731608681	82.221.172.193	WireGuard	172.30.84.193	232		Transport Data, receiver=0xBE2FE42C, counter=1, datalen=158
35	2024-08-03 17:28:43,742867488	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver=0x005D3401, counter=4, datalen=64
36	2024-08-03 17:28:43,851465379	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver 0x005D3401, counter 5, datalen 64
37	2024-08-03 17:28:43,860030840	82.221.172.193	WireGuard	172.30.84.193	131		Transport Data, receiver 0xBE2FE42C, counter 2, datalen 68
38	2024-08-03 17:28:43,862500700	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver=0x005D3401, counter=6, datalen=64
39	2024-08-03 17:28:43,867623392	172.30.84.193	WireGuard	82.221.172.193	298		Transport Data, receiver=0x005D3401, counter=7, datalen=224
40	2024-08-03 17:28:43,886621100	82.221.172.193	WireGuard	172.30.84.193	134		Transport Data, receiver=0xBE2FE42C, counter=3, datalen=60
41	2024-08-03 17:28:43,893093998	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver=0x005D3401, counter=8, datalen=64
42	2024-08-03 17:28:43,897557832	172.30.84.193	WireGuard	82.221.172.193	608		Transport Data, receiver=0x005D3401, counter=9, datalen=592
43	2024-08-03 17:28:43,991984558	82.221.172.193	WireGuard	172.30.84.193	134		Transport Data, receiver=0xBE2FE42C, counter=4, datalen=60
44	2024-08-03 17:28:44,000014322	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver 0x005D3401, counter 10, datalen 64
45	2024-08-03 17:28:44,007033307	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver 0x005D3401, counter 11, datalen 1088
46	2024-08-03 17:28:44,009400055	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver=0x005D3401, counter=12, datalen=1088
47	2024-08-03 17:28:44,009530012	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver=0x005D3401, counter=13, datalen=1088
48	2024-08-03 17:28:44,009609320	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver=0x005D3401, counter=14, datalen=1088
49	2024-08-03 17:28:44,009865005	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver=0x005D3401, counter=15, datalen=1088
50	2024-08-03 17:28:44,010552776	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver=0x005D3401, counter=16, datalen=1088
51	2024-08-03 17:28:44,010612478	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver=0x005D3401, counter=17, datalen=1088
52	2024-08-03 17:28:44,010980438	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver 0x005D3401, counter 18, datalen 1088
53	2024-08-03 17:28:44,020300630	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver 0x005D3401, counter 19, datalen 1088
54	2024-08-03 17:28:44,022220557	172.30.84.193	WireGuard	82.221.172.193	1162		Transport Data, receiver=0x005D3401, counter=20, datalen=1088
55	2024-08-03 17:28:44,039529470	82.221.172.193	WireGuard	172.30.84.193	126		Transport Data, receiver=0xBE2FE42C, counter=5, datalen=52
56	2024-08-03 17:28:44,040045008	82.221.172.193	WireGuard	172.30.84.193	1354		Transport Data, receiver=0xBE2FE42C, counter=6, datalen=1280
57	2024-08-03 17:28:44,040224042	82.221.172.193	WireGuard	172.30.84.193	1354		Transport Data, receiver=0xBE2FE42C, counter=7, datalen=1280
58	2024-08-03 17:28:44,040263567	82.221.172.193	WireGuard	172.30.84.193	1354		Transport Data, receiver=0xBE2FE42C, counter=8, datalen=1280
59	2024-08-03 17:28:44,040279808	82.221.172.193	WireGuard	172.30.84.193	1354		Transport Data, receiver=0xBE2FE42C, counter=9, datalen=1280
60	2024-08-03 17:28:44,040299449	82.221.172.193	WireGuard	172.30.84.193	1354		Transport Data, receiver 0xBE2FE42C, counter 10, datalen 1280
61	2024-08-03 17:28:44,040319370	82.221.172.193	WireGuard	172.30.84.193	563		Transport Data, receiver 0xBE2FE42C, counter 11, datalen 489
62	2024-08-03 17:28:44,042471777	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver=0x005D3401, counter=21, datalen=64
63	2024-08-03 17:28:44,042520195	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver=0x005D3401, counter=22, datalen=64
64	2024-08-03 17:28:44,042025609	172.30.84.193	WireGuard	82.221.172.193	138		Transport Data, receiver=0x005D3401, counter=23, datalen=64

Avoiding fingerprint is hard

- In 10/2022 the bandwidth on a Snowflake bridge suddenly dropped: *some* Android devices were not able to connect to the bridge anymore
- Culprit: JA3C fingerprint!
- "Go crypto/tls ciphersuite ordering does not directly have to do with mobile versus desktop; it instead hinges on whether the platform has hardware AES-GCM acceleration or not. Some mobile platforms do not have such acceleration, while most desktop platforms do, which is why the ciphersuite order and hence the TLS fingerprint tends to differ across mobile and desktop". Details here
- Solution: use uTLS on Tor browser. It "provides ClientHello fingerprinting resistance, low-level access to handshake, fake session tickets ..."
- Bottom line: the fingerprint of your SW might depend on the HW!

See you at ntopConf 2025

https://www.ntop.org/ntopconf25/