Passive Network Fingerprinting

Luca Deri<deri@ntop.org>
@lucaderi

Who am I

- ntop founder (http://www.ntop.org): company that develops open-source network security and visibility tools:
    - ntopng: web-based traffic monitoring and security
    - nDPI: deep packet inspection toolkit
    - PF_RING: High-Speed Packet Capture
- Author and contributor to various open source software tools.
- Lecturer at the CS Dept, University of Pisa, Italy.

nDPI in a nutshell

- C-based 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, LGPL v3

Agenda

- What We'll Cover in This Talk
  - Fingerprints tutorial
  - Overview of nDPI supported fingerprints
  - Initial flow fingerprint (this talk)

- What We'll NOT Cover in This Talk
  - Post-connection behavioural fingerprint (not this talk)

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

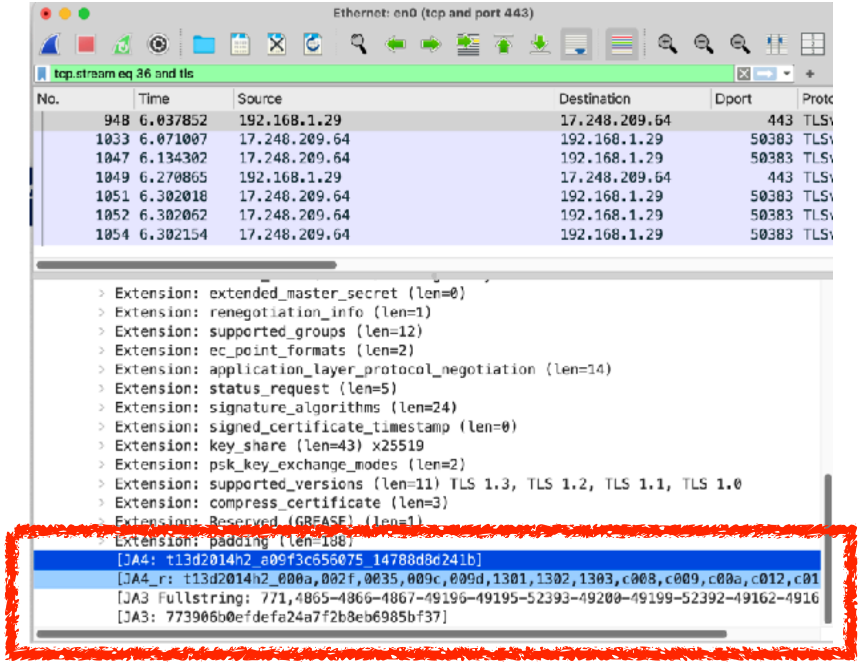
How can I Use a Fingerprint?

- 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)

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 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.

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.

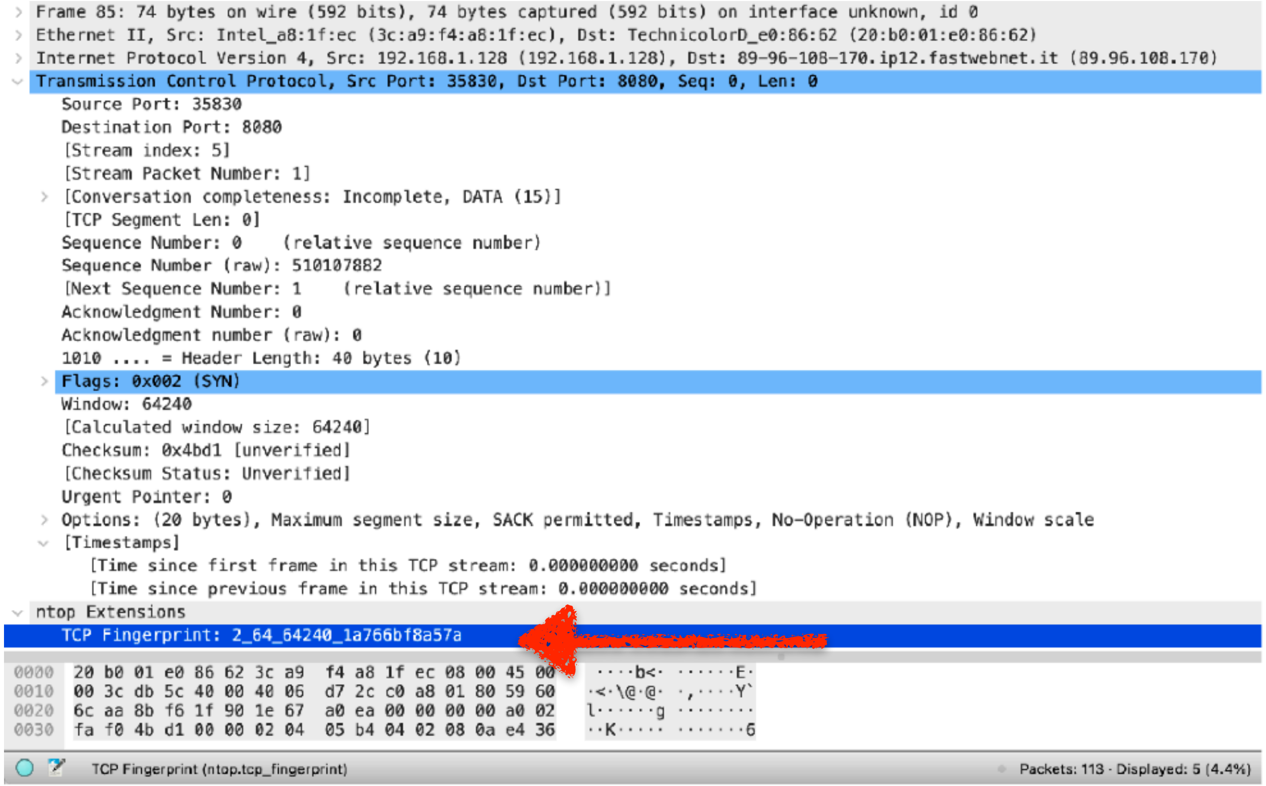
TCP/IP Stack Fingerprinting [1/2]

- As discussed earlier, TCP/IP stack fingerprinting is one of the most popular methods for detecting the OS from network traffic.
- Unfortunately there is no single standard/representation hence there are various formats produced by the many available fingerprint tools.
- 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 [2/2]



```
> 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%)
```

Some TCP/IP Stack Fingerprinting Findings

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.

```
v 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 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]
```

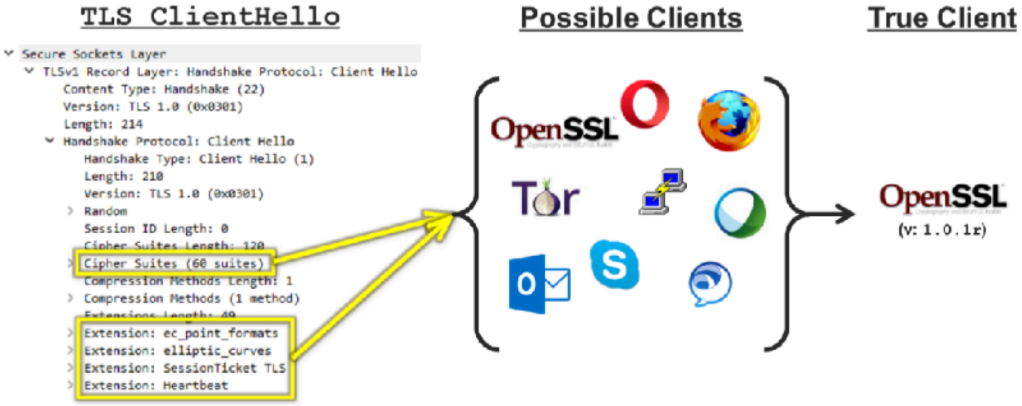
https://zmap.io/

```
> 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

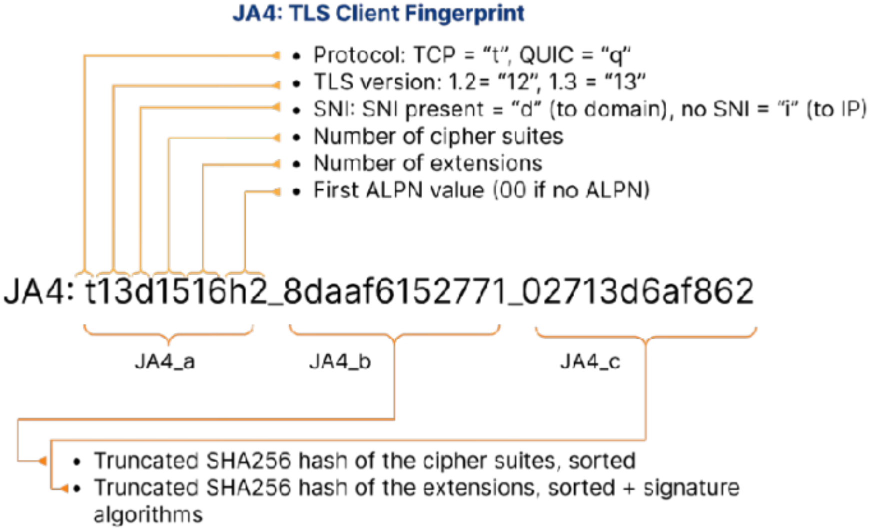
TLS/QUIC Fingerprinting [1/2]

- 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 [2/2]

- 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. nDPI implements only JA4.



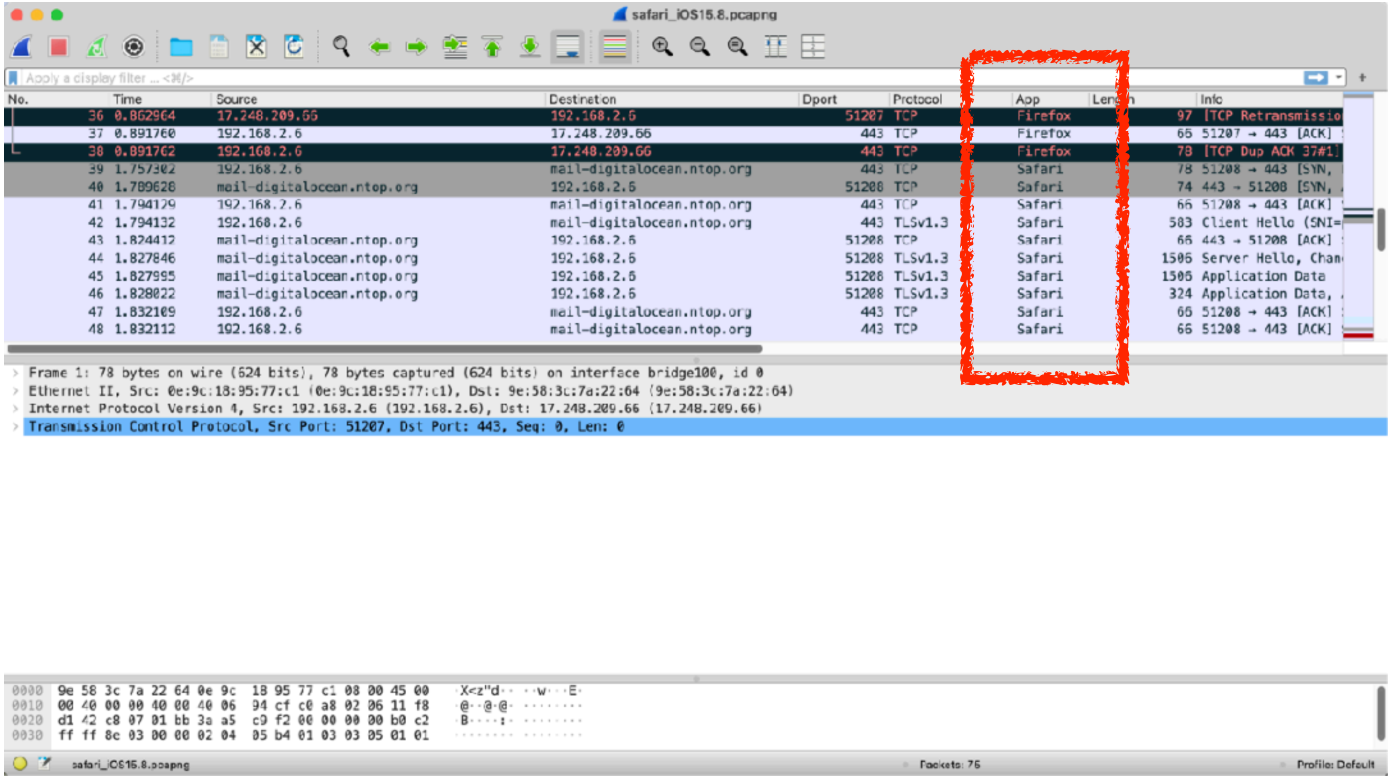
Browser Fingerprints [1/2]

Missing JA4_a

```
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',
```

......

Browser Fingerprints [2/2]



Additional nDPI Fingerprints

- RDP (Remote Desktop Protocol)
- SSH (Secure Shell)
- DHCP (Dynamic Host Configuration Protocol)
- OpenVPNs (and dialects)
- Obfuscated TLS (encrypted tunnels based on a TLS dialect)
- Fully Encrypted Protocols (ShadowSocks, VMess, Trojan,...)

Thank You, and See you at PacketFest

May 7-9, Zürich, Switzerland

https://www.packetfest.ch