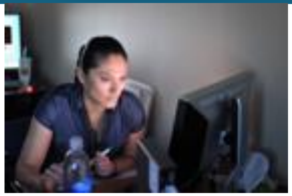
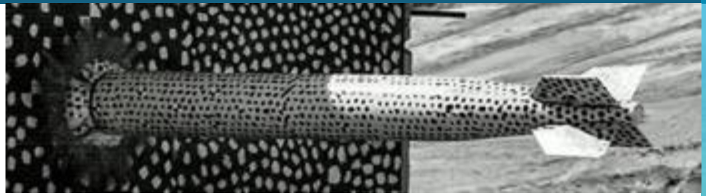
Profiling Malicious Web Clients

PRESENTED BY

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Unlimited Release/TLP:WHITE

SAND2022-12702 PE

Outline

Motivation

Approach

- TCP (and IP)
- TLS
- HTTP
- Internet Scan
- Side Channels

Examples

Future Work

Conclusions

Motivation

Infrastructure is an Important Part of Countering Attacker Technology

Security community shares threat information for common defense

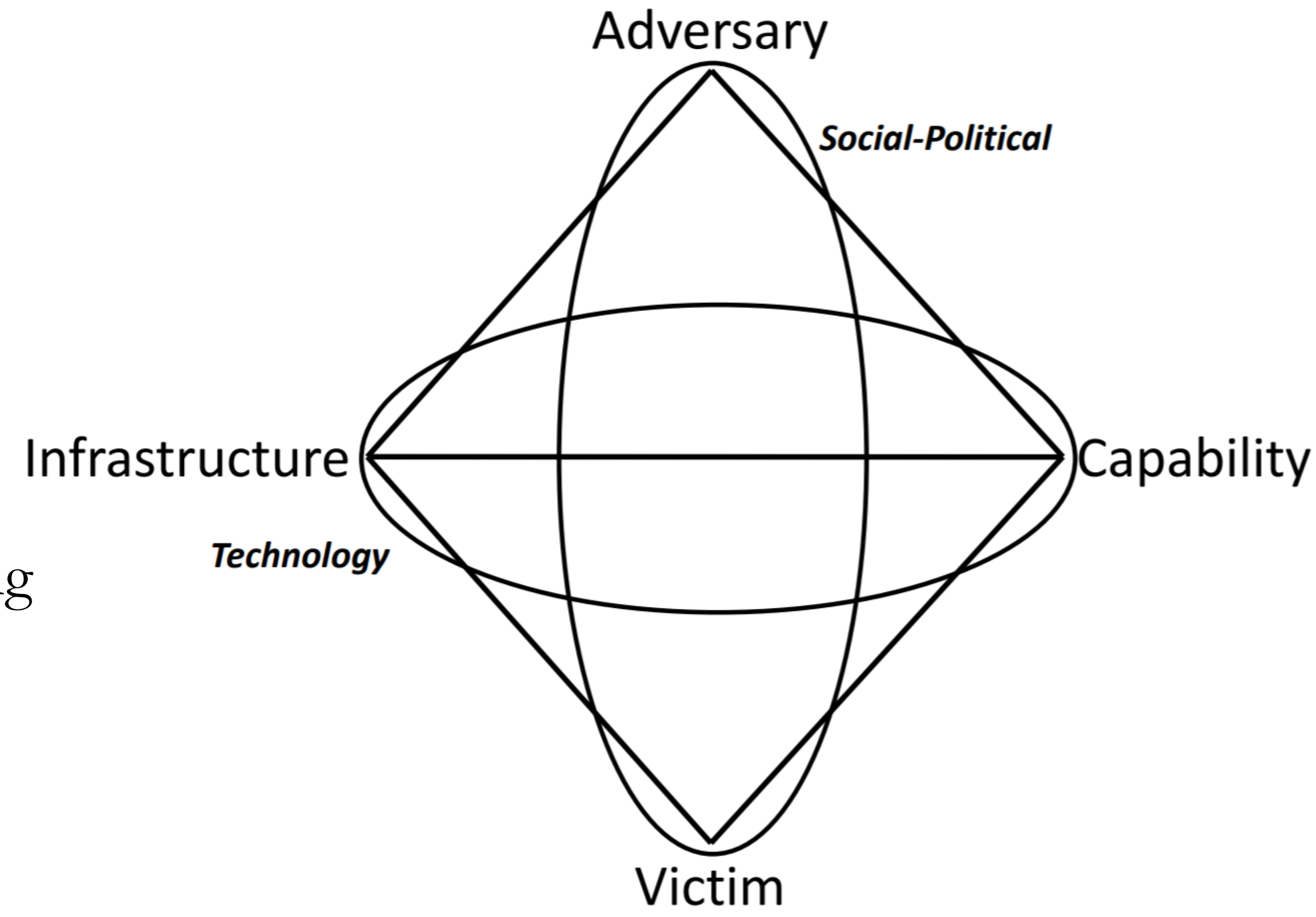
Capabilities (Malware)

- Streamlined signature distribution

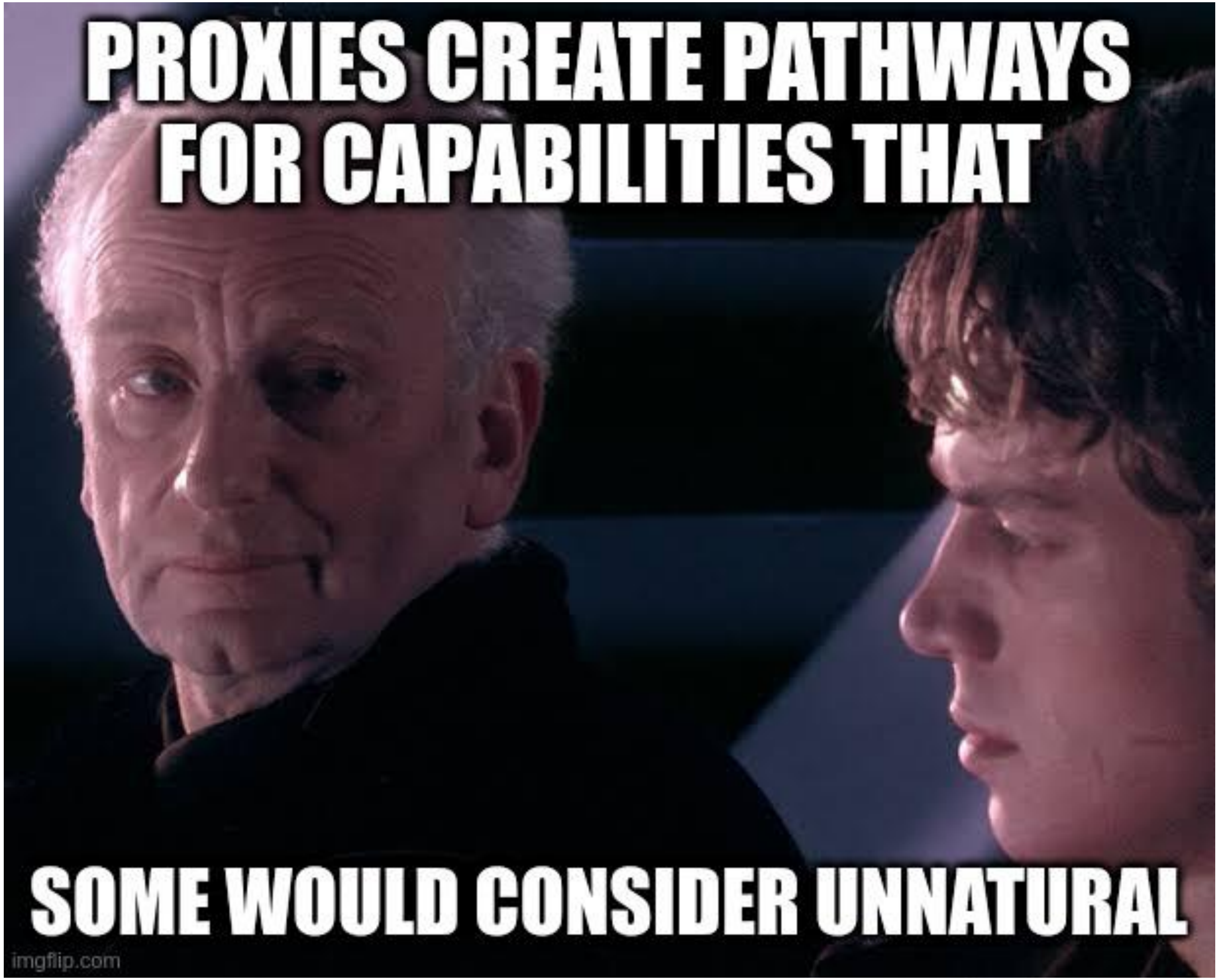
Infrastructure

- IP/domain reputation distribution

VPN/Proxies frustrate tracking/sharing Threat IPs



https://apps.dtic.mil/sti/pdfs/ADA586960.pdf


PROXIES CREATE PATHWAYS
FOR CAPABILITIES THAT
SOME WOULD CONSIDER UNNATURAL
imgflip.com

Focus on Web Service Threat Vector

We will address web service threat vector (malicious web client)

- Presented from view of ~~victim~~ target network defender, monitoring web server

Ubiquity: Trend for technology to be implemented as web services

- Cloud architectures mean more misuse originates from malicious web clients (vs. traditional backdoor malware)

For simplicity: Web services are well understood; many concepts apply to other protocols/architectures/situations

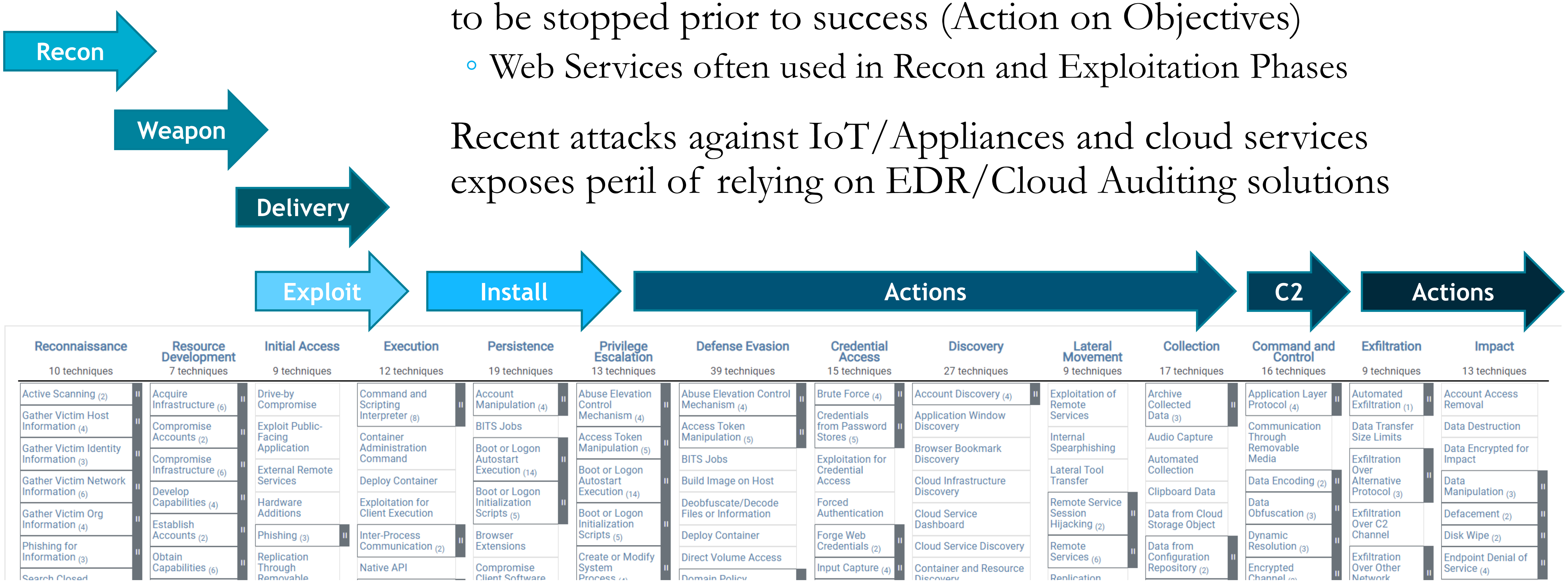
Why Focus on Pre-Exploitation Activity?

Defender's goal is to limit adversary activity

Detection earlier in Kill Chain/ATT&CK model allows attacks to be stopped prior to success (Action on Objectives)

- Web Services often used in Recon and Exploitation Phases

Recent attacks against IoT/Appliances and cloud services exposes peril of relying on EDR/Cloud Auditing solutions



Reconnaissance (10 techniques)	Resource Development (7 techniques)	Initial Access (9 techniques)	Execution (12 techniques)	Persistence (19 techniques)	Privilege Escalation (13 techniques)	Defense Evasion (39 techniques)	Credential Access (15 techniques)	Discovery (27 techniques)	Lateral Movement (9 techniques)	Collection (17 techniques)	Command and Control (16 techniques)	Exfiltration (9 techniques)	Impact (13 techniques)
Active Scanning (2)	Acquire Infrastructure (6)	Drive-by Compromise	Command and Scripting Interpreter (8)	Account Manipulation (4)	Abuse Elevation Control Mechanism (4)	Abuse Elevation Control Mechanism (4)	Brute Force (4)	Account Discovery (4)	Exploitation of Remote Services	Archive Collected Data (3)	Application Layer Protocol (4)	Automated Exfiltration (1)	Account Access Removal
Gather Victim Host Information (4)	Compromise Accounts (2)	Exploit Public-Facing Application	Container Administration Command	BITS Jobs	Access Token Manipulation (5)	Access Token Manipulation (5)	Credentials from Password Stores (5)	Application Window Discovery	Internal Spearphishing	Audio Capture	Communication Through Removable Media	Data Transfer Size Limits	Data Destruction
Gather Victim Identity Information (3)	Compromise Infrastructure (6)	External Remote Services	Deploy Container	Boot or Logon Autostart Execution (14)	Boot or Logon Autostart Execution (14)	BITS Jobs	Exploitation for Credential Access	Browser Bookmark Discovery	Lateral Tool Transfer	Automated Collection	Data Encoding (2)	Exfiltration Over Alternative Protocol (3)	Data Encrypted for Impact
Gather Victim Network Information (6)	Develop Capabilities (4)	Hardware Additions	Exploitation for Client Execution	Boot or Logon Initialization Scripts (5)	Boot or Logon Initialization Scripts (5)	Build Image on Host	Forced Authentication	Cloud Infrastructure Discovery	Remote Service Session Hijacking (2)	Clipboard Data	Data Obfuscation (3)	Exfiltration Over C2 Channel	Data Manipulation (3)
Gather Victim Org Information (4)	Establish Accounts (2)	Phishing (3)	Inter-Process Communication (2)	Browser Extensions	Create or Modify System Process	Deobfuscate/Decode Files or Information	Forge Web Credentials (2)	Cloud Service Dashboard	Remote Services (6)	Data from Cloud Storage Object	Dynamic Resolution (3)	Exfiltration Over Other Network	Defacement (2)
Phishing for Information (3)	Obtain Capabilities (6)	Replication Through Removable	Native API	Compromise Client Software		Deploy Container	Input Capture (4)	Cloud Service Discovery	Replication	Data from Configuration Repository (2)	Encrypted Channel		Disk Wipe (2)
Search Closed						Direct Volume Access		Container and Resource Discovery					Endpoint Denial of Service (4)
						Domain Policy							

NSA, CISA, & FBI | Chinese State-Sponsored Cyber Operations: Observed TTPs

Encrypted Multi-Hop Proxies. Chinese state-sponsored cyber actors have been routinely observed using a VPS as an encrypted proxy. The cyber actors use the VPS as well as small office and home office (SOHO) devices as operational nodes to evade detection.

https://media.defense.gov/2021/Jul/19/2002805003/-1/-1/1/CSA_CHINESE_STATE-SPONSORED_CYBER_TTPS.PDF

NSA, CISA, FBI, & NCSC | Russian GRU Conducting Global Brute Force Campaign

In an attempt to obfuscate its true origin and to provide a degree of anonymity, the Kubernetes cluster normally routes brute force authentication attempts through TOR and commercial VPN services, including CactusVPN, IPVanish®, NordVPN®, ProtonVPN®, Surfshark®, and WorldVPN. Authentication attempts that did not use TOR or a VPN service were also occasionally delivered directly to targets from nodes in the Kubernetes cluster.

https://media.defense.gov/2021/Jul/01/2002753896/-1/-1/1/CSA_GRU_GLOBAL_BRUTE_FORCE_CAMPAIGN_UOO158036-21.PDF

KrebsonSecurity

In-depth security news and investigation

HOME ABOUT THE AUTHOR ADVERTISING/SPEAKING

A Deep Dive Into the Residential Proxy Service '911'

July 18, 2022 25 Comments

From a website's perspective, the IP traffic of a residential proxy network user appears to originate from the rented residential IP address, not from the proxy service customer. These services can be used in a legitimate manner for several business purposes — such as price comparisons or sales intelligence — but they are massively abused for hiding cybercrime activity because they can make it difficult to trace malicious traffic to its original source.

https://krebsonsecurity.com/2022/07/a-deep-dive-into-the-residential-proxy-service-911/

Profiling Malicious Web Proxies

Various types of proxies often used by various threats
- "evade detection"
- "obfuscate its true origin"
- "difficult to trace malicious traffic to it's original source"

Superficial indicators, ex. IPs, easy to change

Highly specific consistencies in proxy networks often exist

Many proxy networks represent investment by adversary
- Build own proxy network
- Compromise infrastructure

Many users have habits that allow De-anonymization
- Use commodity proxy network in unique way

Use of proxies as detection opportunity!

Approach

Unlimited Release/TLP:WHITE

Related Work

Privacy Research

- Focus is typically ensuring client anonymity including malicious clients

Reputation/Geolocation Databases/Services

- Categorically opaque to web service operators/defenders
- Focus is generic use of proxy/anonymization technology
- Focus is not specific threat actor or activity patterns

Our Approach:

- Understand proxy infrastructure based on network defender observable artifacts
- Enable network defenders to profile malicious clients

Profiling Malicious Web Clients

We will focus on two major methods of profiling proxy infrastructure

- Inconsistencies, discontinuities in software fingerprints
- Path artifacts, especially timing analysis

Individual indicators are usually not unique

- Ex. specific operating system or timing artifact

Combining indicators across the network stack can result in highly specific profiles

- "Extra layer of indirection" is not your friend here!

We will focus less on methods we consider well understood/consistently employed

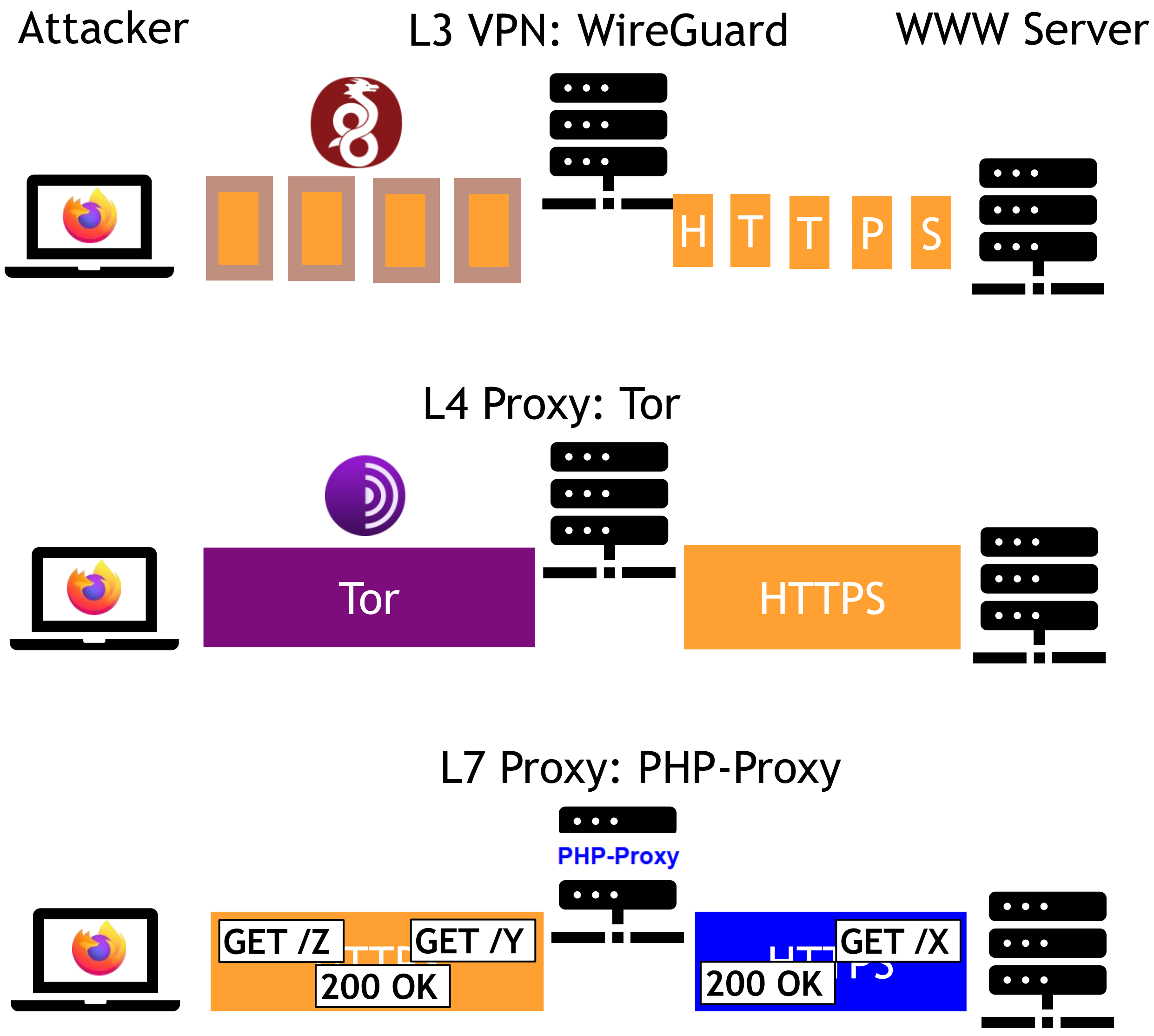
- Cookies/web analytics
- TLS and Browser Fingerprinting
- Identifying Malicious Services via Internet Scans

Proxy Taxonomy: Network Layer

Layer 3 (ex. WireGuard):
- Unit: Packet
- Lower than normal MTU (TCP MSS)
- low ping time (IP iRTT), high TCP iRTT
- TCP fingerprint mismatch IP, Internet Scan

Layer 4 (ex. Tor):
- Unit: TCP Stream
- Low TCP iRTT, high TLS iRTT
- TCP Handshake/TLS Client Hello Delay
- TCP fingerprint mismatch TLS

Layer 7 (ex. PHP-Proxy):
- Unit: HTTP request/response
- Low TLS iRTT, high HTTP iRTT
- TCP, TLS fingerprint mismatch Browser



Gait: Adding Fingerprinting/Timing Attributes to Zeek

https://github.com/sandialabs/gait

Extensions to zeek to add attributes for profiling endpoints

- IP attributes
- TCP attributes and RTT
- TLS attributes and RTT (compliment full ja3)
- SSH timing coming soon!

```
redef record connection +=
{
    tcp_handshake_duration: interval &optional;
};

event connection_first_ACK(c: connection)
{
    #check for "normal" tcp handshake (weeds out some connections where RTT can't be calculated accurately):
    if (c$orig$num_pkts == 1 && c$resp$num_pkts == 1 && c$history == "ShA")
    {
        c$tcp_handshake_duration = network_time() - c$start_time;
    }
}
```

TCP (and IP) Profiling

TCP (and IP) Profiling

Operating System of TCP endpoint

Path: Tunneling and Distance

Warning: Most TCP fingerprinting tools and references out of date

TCP Operating System Fingerprinting: IP TTL

IP field: Counter decreasing with each hop

Default TTL	Operating Systems
64	Linux, Android, most Unix, iOS
128	Windows
255	Others

IP header (TTL in IPv4, Hop Limit in IPv6): UDP and ICMP too!

TCP Operating System: Ephemeral Port Range Selection

Algorithm for client port selection

Platform	Ephemeral Port Range	Ephemeral Port Order	Global/Local
iOS	1024 - 65535	incrementing	global
Android (version 10)	37000 - 49999	incrementing (n+2)	local (same server ip, port)
Ubuntu 18 LTS	32768 - 60999	incrementing (n+2)	local (same server ip, port)
Windows 7	49152 - 65535	incrementing	global
Windows 10	49152 - 65535	random	-

Useful when there are multiple connection in small time range

- Ex. estimate density of port scan

TCP Operating System Fingerprinting: TCP options

Variable length options in TCP header

Operating System	TCP Options Kind List	TCP Options Length
Linux	2,4,8,1,3	20
Windows	2,1,3,1,1,4	12

Order Matters: NOP for byte aligned padding

Values that are useful:

- 2: MSS
- 3: Window Scale

Value that are not useful for fingerprinting:

- SACK Permitted (flag)
- Timestamps (now random, only useful for endpoint flow control)

Kind	Length	Description
0	1	End of Option List
1	1	No-Operation
2	4	Maximum Segment Size
3	3	Window Scale
4	2	SACK Permitted
8	10	Timestamps

TCP Operating System Fingerprinting: Window Size and Scale

Window size used for flow control (amount of un-ACK'd data allowed)

Size is TCP field

Scale is a TCP optional field (to support larger windows): size * 2 ^ (scale)

Platform	Windows Size	Window Scale
iOS	65535	5
Android (version 10)	65535	12
Ubuntu 18 LTS	64240	7, 11
Windows 7	8192	2
Windows 10	64240	8

Can change :

- based on congestion (likely to be default in both SYN packets)
- modified by routers in path

Path Fingerprinting: Maximum Packet Size

Maximum Segment Size is related to maximum packet size

- 1460 is default value on ethernet (1500 MTU) for IPv4

Lower MSS often caused by Tunneling/VPNs

While IP fragmentation is possible, in practice, MTU config or detection is used

Many consumer Privacy focused VPNs spoof/fake MTU and other attributes

- Present packet-level interface to client, operate at stream level in proxy network
- If packets aren't end-to-end, functionally a layer 4 proxy, from defender perspective

Detection of MTU through upload segment size?

- Possible metric: Max packet size seen in flow

Path Fingerprinting: Propagation Time (~Distance)

TCP can be used to infer Round Trip Time (iRTT)

- Ex. time between packet and ACK on that packet
- Necessarily measures host response time as well

Consistent measurement: handshake duration

- Simplifies in vs. out issues by combining

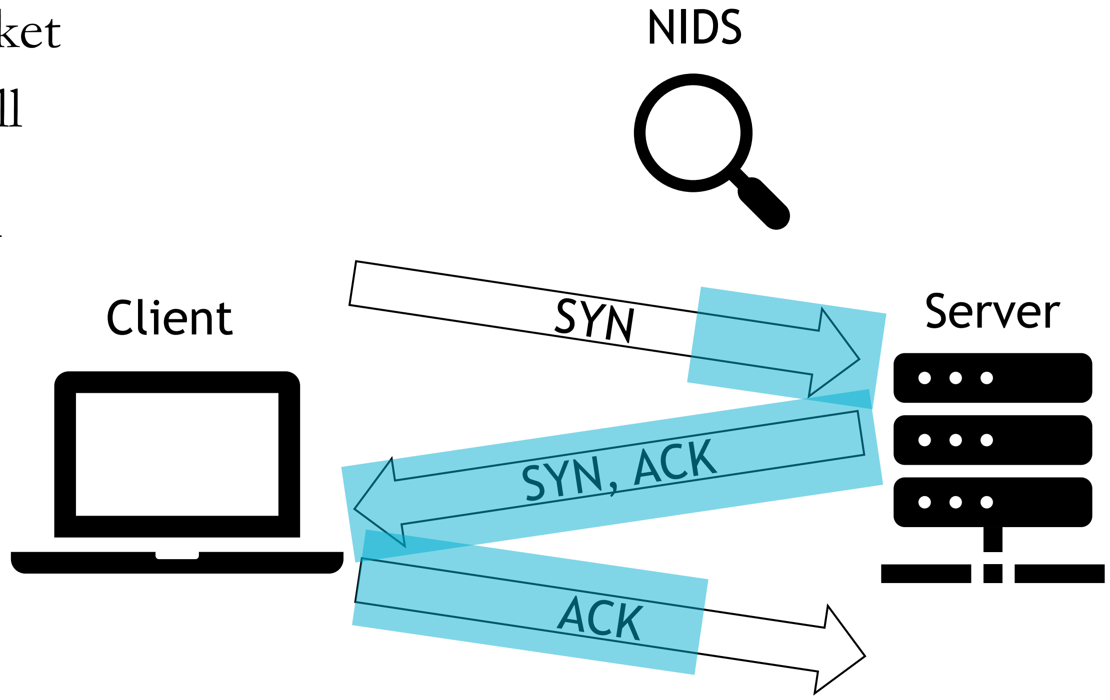
Is traffic really coming from endpoint?

- helps confirm use of proxy, VPN, etc.

Compare to TLS RTT, HTTP RTT

Precise geolocation can be hard

- What if multiple proxies are used?



Recommendations for TCP metadata for fingerprinting

Value	Name	tshark field	gait/zeek field	Purpose
High	TCP port	tcp.srcport	id.orig_p	Ephemeral port selection
	IP TTL	ip.ttl	orig_ttl	IP default and hop count
	Inferred RTT	tcp.analysis.initial_rtt	tcp_handshake_duration	inferred round-trip time
	TCP MSS	tcp.options.mss_val	orig_mss	max packet size
Med	TCP Options Kinds	tcp.option_kind	orig_tcp_options	TCP default settings
	TCP Window Size	tcp.window_size_value	orig_win_size	TCP default settings
	TCP Window Scale	tcp.options.wscale.shift	orig_win_scale	TCP default settings
Low	TCP flags (DF)	tcp.flags	orig_df	TCP default settings
	TCP timestamp	tcp.options.timestamp.tsval	-	deprecated: Host Uptime

TTL applies to all IP traffic

Initial (SYN) packet size denotes TCP options size (weak substitute for options kinds)

Many other possibilities: IP flags, other TCP options

TLS Profiling

TLS Fingerprinting

Very similar to TCP Options kind list, but more fields and options

ja3, ja3s

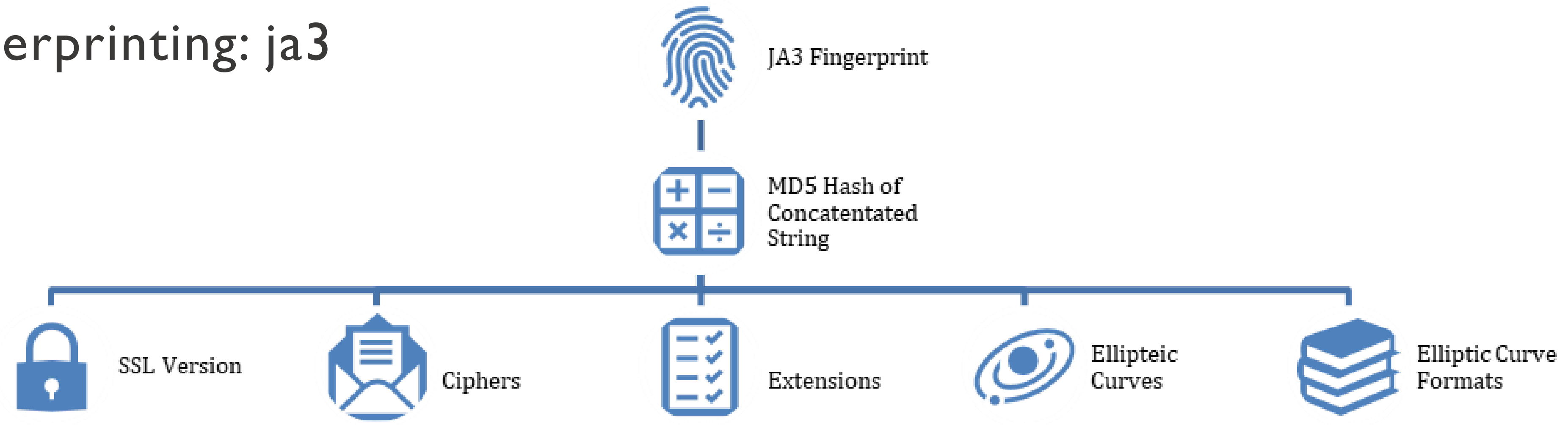
- Passive

jarm

- Server only
- Active: probes edge cases
- Fuzzy hash digest

https://github.com/salesforce/ja3
https://github.com/salesforce/jarm

TLS Fingerprinting: ja3



System	md5 digest	Version
Python-requests, python 2.7, Win10	86cb13d6bbb3ac96b78b408bcfc18794	771
Python-requests, python 2.7, Linux	af26ba5e85475b634275141e6ed3dc54	771

Cipher
49200-49196-49199-49195-159-158-49202-49198-49201-49197-165-161-164-160-49192-49188-49172-49162-49194-49190-49167-49157-49191-49187-49171-49161-49193-49189-49166-49156-107-105-104-57-55-54-103-63-62-51-49-48-157-156-61-53-60-47-255
4866-4867-4865-49196-49200-49195-49199-52393-52392-159-158-52394-49327-49325-49326-49324-49188-49192-49187-49191-49162-49172-49161-49171-49315-49311-49314-49310-107-103-57-51-157-156-49313-49309-49312-49308-61-60-53-47-255

Extensions	EC Curves	EC Formats
0-11-10-13-15-16-21	23-25-28-27-24-26-22-14-13-11-12-9-10	0-1-2
11-10-35-22-23-13-43-45-51-21	29-23-30-25-24	0-1-2

TLS Fingerprinting: ja3 metadata

Many already collecting ja3 digests via passive NIDS, other tools

ja3 digests ostensibly simple

- Significant overlap in ja3: based on TLS libraries and application

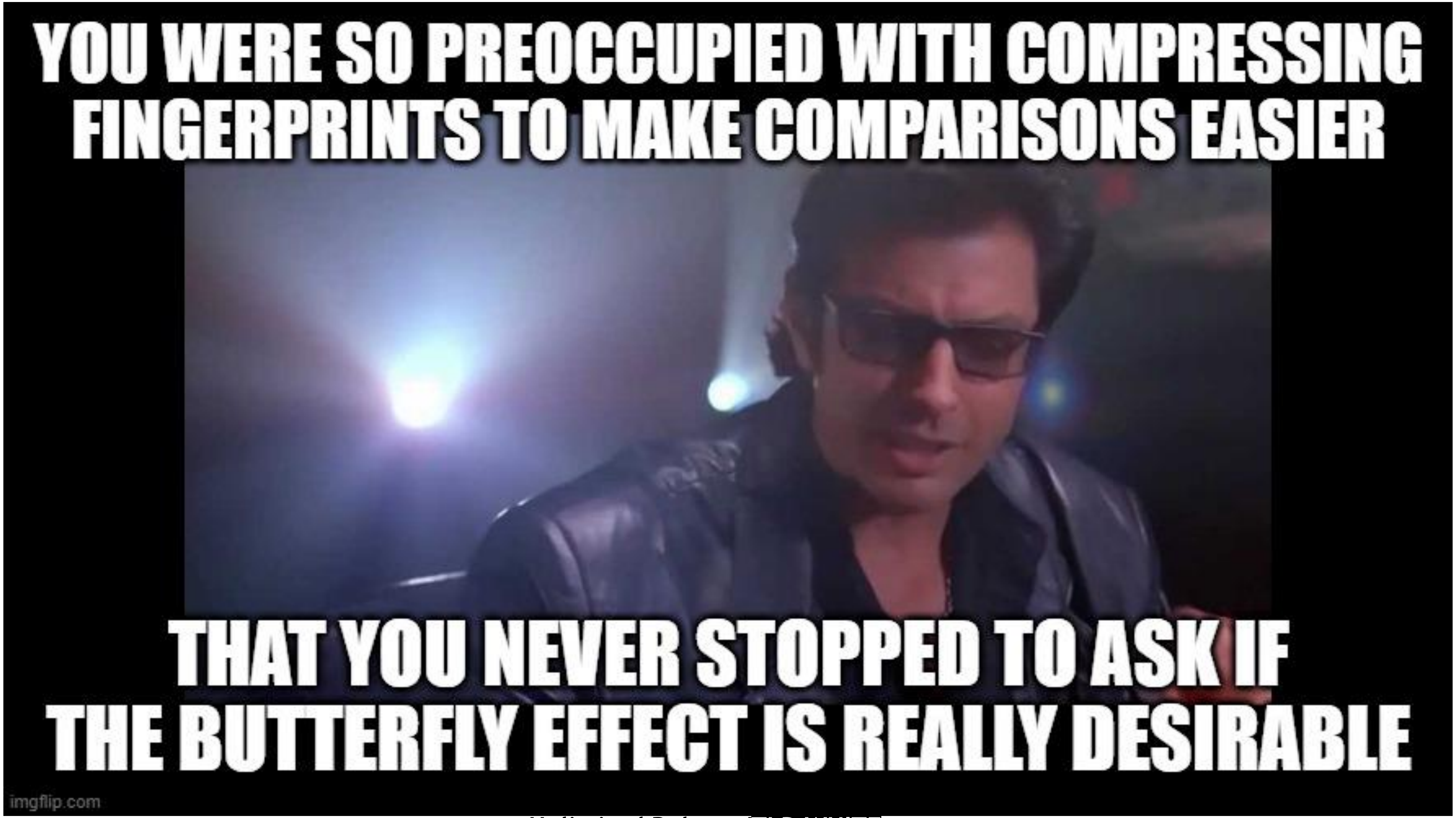
ja3 alone is rarely a strong indicator of malicious activity

- Often can identify application(s) on a specific OS type
- Usually not enough to infer intent—need additional indicators from other layers

ja3 digest limitations

- cryptographic hash is bad choice for digests when ability to recognize small differences is desirable—need a better digest method that support local comparisons/clustering
- Are there no other values that are useful for fingerprinting?
  - Extension 16: APLN (next protocol)
  - Extension 28: record_size_limit

TLS Fingerprinting



TLS Fingerprinting: ja3

Advanced use requires knowing, understanding full ja3

- Related digests: ja3 with minor differences
  - Padding extension
  - Extensions for TLS session resumption
  - SNI extensions missing when connection direct to IP

Potential technical aids:

- Database of ja3 digest and full ja3
- Database of ja3 -> User-Agent header, ja3s -> Server header mappings

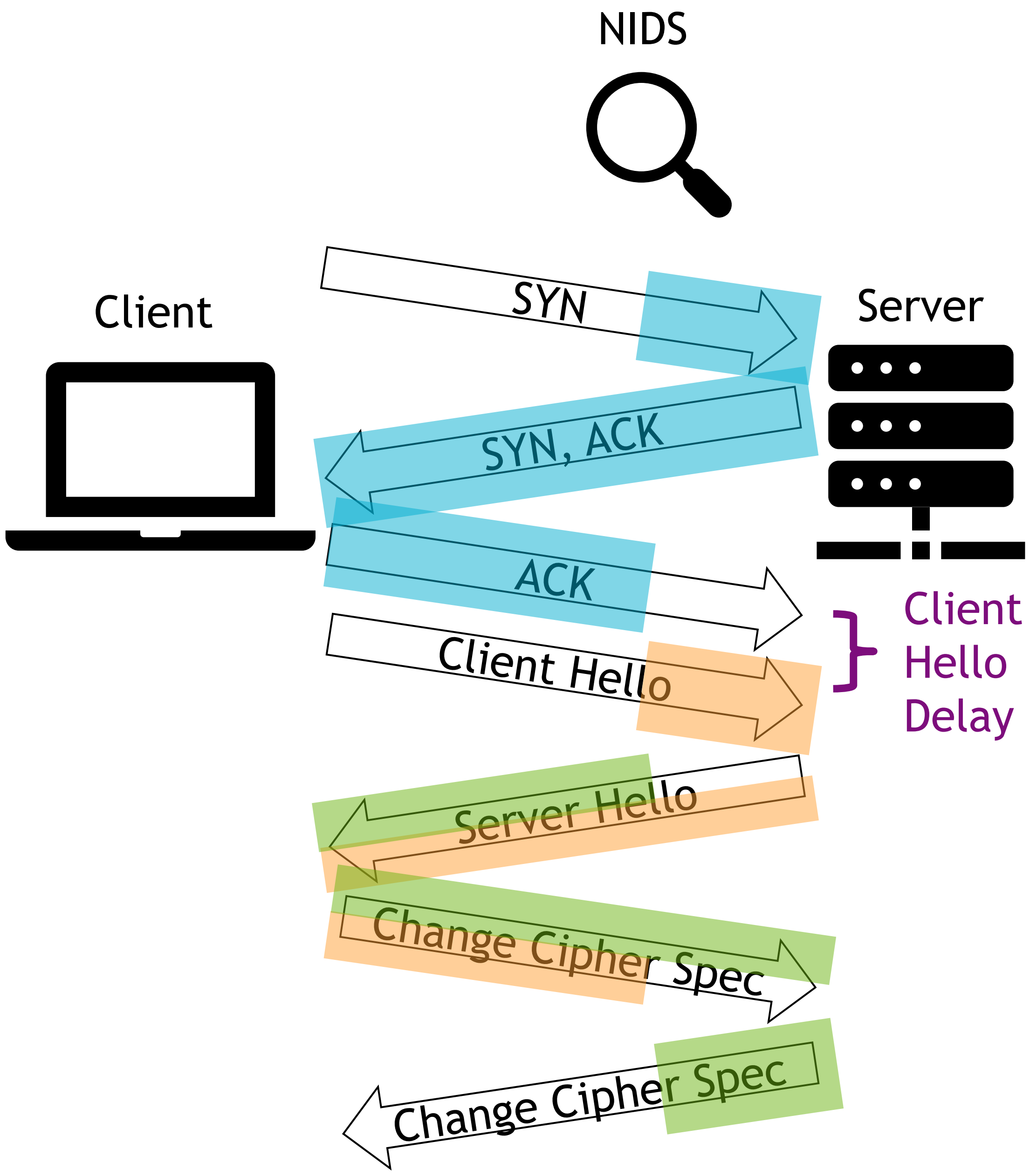
TLS Timing Analysis

TLS RTT can be simply measured by NIDS for most handshakes

- Some handshakes have more than one RTT
- Some resumed TLS 1.3 sessions are 0-RTT
  - The initial connection has full RTT

Gap between TCP ACK and Client Hello

- Telltale of SOCKS/Tor proxy
- ~0 in normal case
- Artifact of queueing/connection blocking



HTTP Profiling


AND THEN HE SET
HIS USER-AGENT TO CHROME
imgflip.com

HTTP Profiling

Profiling using existence or absence of headers and header values (and order)

Full headers best, small additions to web logs put you ahead of pack

Usually Collected	High Value	Medium Value
Request/URI	Cookie/Session ID	Connection
Host	Accept-Language	Accept
User-Agent	Accept-Encoding	Raw Auth
Referer	X-Forwarded-For	Host Header
Bytes Transferred	App/Attacker Specific	TCP Ports
	High Resolution Time	

```
LogFormat "%v %p %h %{remote}p %l %u %t %{sec}t.%{usec_frac}t %D \"%r\" %>s %I %O \"%{Host}i\"
\"%{Referer}i\" \"%{User-Agent}i\" \"%{Connection}i\" \"%{Accept-Language}i\" \"%{Accept-
Encoding}i\" \"%{Accept}i\" \"%{X-Forwarded-For}i\" \"%{Cookie}i\" \"%{Authorization}i\"" extended
```

HTTP Timing Analysis

HTTP RTT Can be measured actively through Javascript, etc

Passively measured by focusing on HTTP content that drives response with as little browser delay as possible

- HTTP redirect
- HTML resource (JS, CSS, etc)

Referer helps correlate events

Requires high resolution timestamps

Internet Scan Profiling

Internet Port/Banner Scans

Internet Port/Banner Scans: shodan, censys, etc.

Well known, strong indicators:

- Often (C2) server focused
- Host verification and hostnames: Ex. TLS certificate subject or key
- Rare banners/responses/typos: Ex. cobalt strike "extraneous space"

Weaker indicators useful with correlation:

- Often (recon, exploit) client focused: ex. VPS, VPN, proxy
- Can overlap with, sometimes compliment to reputation services
- Open port
- Specific software, appliance, IoT device type/version
- CVE/vulnerability

History often necessary

Side Channel Analysis

Unlimited Release/TLP:WHITE

Side Channels: DNS Fingerprinting

VPN, proxy, etc handle web connections

- DNS is often handled separately, different path
- DNS is often overlooked

DNS request almost always comes through recursive, caching resolver

- Low level attributes, specifics rarely interesting
- Really hard for attackers to observe end requests
- Separate path: different timing characteristics

Most common indicators

- Geolocation
  - Sometimes DNS request originate closer to end node than Proxy/VPN service
- ISP/ASN
  - DNS service used can differentiate activity from same Proxy/VPN service

Side Channels: DNS Fingerprinting

Associating Web activity to originating DNS request is a correlation problem

DNS requests lead Web connections by ~1 RTT

Much easier if focus on rare domains, low activity times

- Most common domains often cached

Triangulation of requests can increase confidence

obscuredomain1.company.com
Residential ISP C, Country 1
DNS service J, Country 2
VPS A, Country 1

obscuredomain2.company.com
DNS service J, Country 2
Wireless ISP V, Country 1
Web Spider B, Country 5

obscuredomain3.company.com
University Y, Country 3
VPS X, Country 4
DNS service J, Country 2

Side Channels: Web Analytics

Tracking Cookies, etc.

Web analytics (screen resolution)

Search engine queries

Systematically collect and search web analytics data

Tracking Threats == Privacy Invasion

- Service provider hypocrisy: against tracking by everyone except themselves

Attributes that are sometime useful for identifying activity in analytics/data

- Time
- Specific URLs
- Geolocation

Side Channels: Web Resources

Using external resources is a proven pathway to visibility in web services

What would a Tech Giant do?

- External image, javascript, etc
- Federated authentication or n-factors

Especially useful for appliances/cloud services with limited visibility

Other Side Channels

Web Cache Leak

Various “Cookie” methods

Unique browsing habits

- Specific page view sequence
- Unique, repeated misspellings, invalid URLs, etc

Knowledge of Target/Specific social engineering themes

Examples

Examples

Layer 3 Proxy: Wireguard

Layer 4 Proxy: Tor

Related ja3 digest

User-Agent Spoofing: Python

NAT and Virtual Machines: VirtualBox

Layer 3 Proxy (WireGuard VPN): Client OS Fingerprint

Ground Truth: Windows 10

TTL: 105 (default 128)

- Default for Windows

TCP Options and Order

- Default for Win10

Window Scale: 8

- Default for Win10

Window Size: 64860

- Multiple of MSS

MSS: 1380

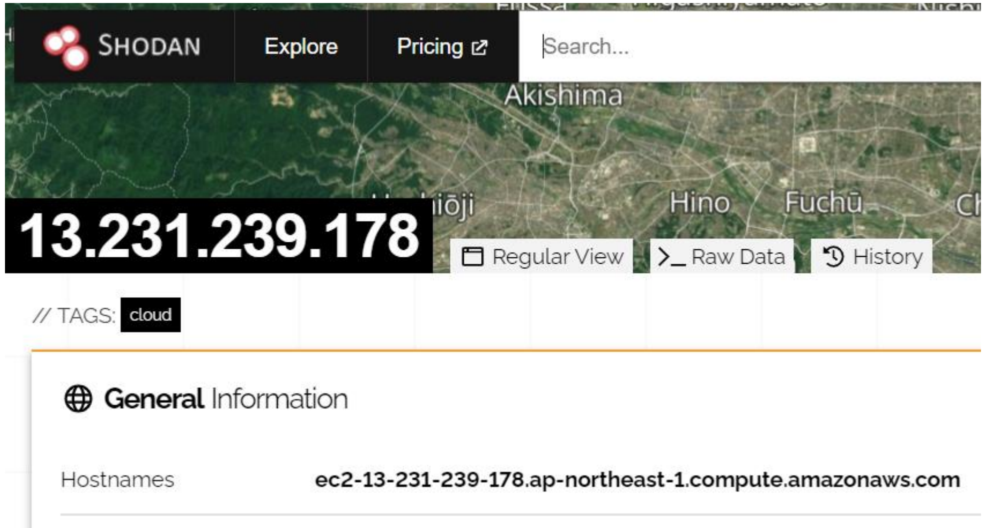
- WireGuard default MTU 1420

```
Internet Protocol Version 4, Src: 13.231.239.178, Dst: 172.31.15.248
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0xf7f1 (63473)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 105
    Protocol: TCP (6)
    Header Checksum: 0x6021 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 13.231.239.178
    Destination Address: 172.31.15.248
Transmission Control Protocol, Src Port: 55469, Dst Port: 443, Seq: 0, Len: 0
    Source Port: 55469
    Destination Port: 443
    Window: 64860
    Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale,
      > TCP Option - Maximum segment size: 1380 bytes
      > TCP Option - No-Operation (NOP)
      > TCP Option - Window scale: 8 (multiply by 256)
      > TCP Option - No-Operation (NOP)
      > TCP Option - No-Operation (NOP)
      > TCP Option - SACK permitted
```

Layer 3 Proxy (WireGuard VPN): Proxy OS Fingerprint

Ground Truth: Amazon Linux 2

Internet Scan: Running Apache 2.4.54



Layer 3 Proxy (WireGuard VPN): Proxy OS Fingerprint

Shodan | Maps | Images | Monitor | Developer | More...

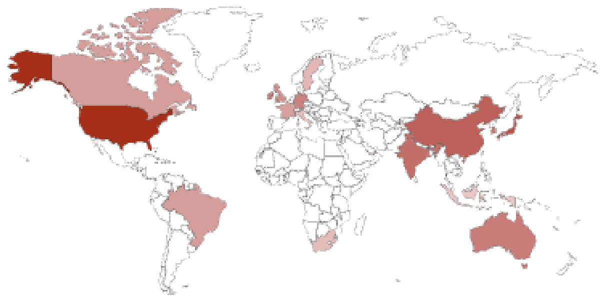
SHODAN | Explore | Pricing

5e2a830765dc0

TOTAL RESULTS

12,948

TOP COUNTRIES



United States	**6,155**
Japan	**1,798**

View Report | View on Map

New Service: Keep track of what you have connected to the Internet. Che

Test Page for the Apache HTTP Server

54.150.95.175
ec2-54-150-95-175.ap-northeast-1.compute.amazonaws.com
Amazon Data Services Japan
Japan, Tokyo

cloud

```
HTTP/1.1 403 Forbidden
Date: Thu, 15 Sep 2022 14:39:55 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Thu, 30 Jun 2022 11:01:19 GMT
ETag: "e2e-5e2a830765dc0"
Accept-Ranges: bytes
Content-Length: 3630
Content-Type: text/html; charset=UTF-8
```

Layer 3 Proxy (WireGuard VPN): Proxy OS Fingerprint

Further Investigation would determine this is default page for Apache on Amazon Linux

Things to look for:

- Exposed router management
- Exposed IoT devices
- proxy/VPN services
- Application/OS/Service identifiers

TOP ORGANIZATIONS	
Amazon Technologies Inc.	3,759
Amazon Data Services NoVa	1,581
Amazon Data Services Japan	1,509
Amazon.com, Inc.	1,431
Ningxia West Cloud Data Technology C...	1,212
More...	

TOP PRODUCTS	
Apache httpd	12,434
nginx	19
DrayTek Vigor Router	1

Layer 3 Proxy (WireGuard VPN): Timing Analysis

GeoIP2 City Plus Web Service Results

IP Address	Country Code	Location	Network	Postal Code	Approximate Coordinates*	Accuracy Radius (km)	ISP	Organization	Domain	Metro Code
54.153.106.102	US	San Jose, California, United States, North America	54.153.104.0/21	95141	37.1835, -121.7714	20	Amazon.com	Amazon.com	amazonaws.com	807
13.231.239.178	JP	Tokyo, Tokyo, Japan, Asia	13.231.224.0/19	151-0053	35.6893, 139.6899	1000	Amazon.com	Amazon.com	amazonaws.com	

https://www.maxmind.com/en/geoip2-precision-demo
https://wondernetwork.com/pings

Server: us-west-1 (N. California)

Exit Node: ap-northeast-1 (Tokyo)

Client: CONUS (Mountain West Region)

	Tokyo
Denver ✖	141.3ms
San Jose ✖	117.23ms

Layer 3 Proxy (WireGuard VPN): Timing Analysis

```
src_p  ts                        src_ip          dst_ip        dst_p  tcp_rtt   tls_rtt   hello_delay
55469  2022-09-13T22:36:20-0600  13.231.239.178  172.31.15.248 443    0.243513  0.251595  0.006553

13.231.239.178 55469 - - [14/Sep/2022:04:36:20 +0000] 1663130180.767923 268 "GET / HTTP/1.1"
13.231.239.178 55469 - - [14/Sep/2022:04:36:21 +0000] 1663130181.069146 317 "GET /mvp.css HTTP/1.1"

[ec2-user@ip-172-31-15-248 ~]$ ping 13.231.239.178
PING 13.231.239.178 (13.231.239.178) 56(84) bytes of data.
64 bytes from 13.231.239.178: icmp_seq=1 ttl=229 time=106 ms
64 bytes from 13.231.239.178: icmp_seq=2 ttl=229 time=106 ms
```

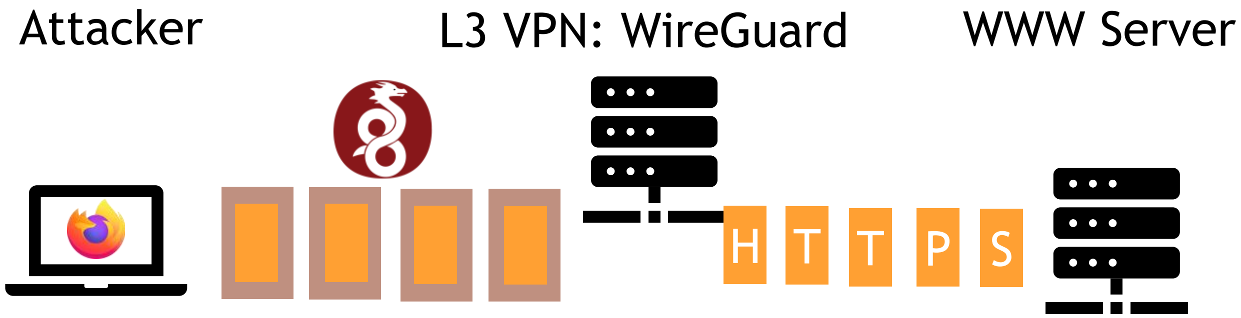
HTTP RTT: .301223

Metric	Exit Node	Delta	Browser
IP RTT (ping)	106 ms		
-		~140ms	
TCP RTT			243ms
TLS RTT			251ms
HTTP RTT			301ms

Layer 3 Proxy (WireGuard VPN): Recap

Layer 3 (ex. WireGuard):

◦ Packets are end-to-end	
◦ Lower than normal MTU (TCP MSS)	1380 MSS (default for WireGuard)
◦ low ping time (IP iRTT), high TCP iRTT	106ms ping, 243ms TCP RTT, 251ms TLS RTT
◦ TCP fingerprint mismatch Internet Scan	Windows TCP/TLS vs. Amazon Linux Web/IP



Layer 4 Proxy (Tor): Timing Analysis

TCP RTT 171ms

Client Hello Delay 287ms

TLS RTT 453ms

No.	Time	Source	Destination	Protocol	Length	Info
15	1663131417.492756	89.236.112.100	172.31.15.248	TCP	74	32830 → 443 [SYN] Seq=0 Win=64240 Len=
16	1663131417.492786	172.31.15.248	89.236.112.100	TCP	74	443 → 32830 [SYN, ACK] Seq=0 Ack=1 Win
17	1663131417.663424	89.236.112.100	172.31.15.248	TCP	66	32830 → 443 [ACK] Seq=1 Ack=1 Win=6425
18	1663131417.950568	89.236.112.100	172.31.15.248	TLSv1.2	583	Client Hello
19	1663131417.950608	172.31.15.248	89.236.112.100	TCP	66	443 → 32830 [ACK] Seq=1 Ack=518 Win=64
20	1663131417.951961	172.31.15.248	89.236.112.100	TLSv1.2	1514	Server Hello, Certificate
21	1663131417.951971	172.31.15.248	89.236.112.100	TLSv1.2	192	Server Key Exchange, Server Hello Done
22	1663131418.122630	89.236.112.100	172.31.15.248	TCP	66	32830 → 443 [ACK] Seq=518 Ack=1575 Win
23	1663131418.404320	89.236.112.100	172.31.15.248	TLSv1.2	192	Client Key Exchange, Change Cipher Spe
24	1663131418.404347	172.31.15.248	89.236.112.100	TCP	66	443 → 32830 [ACK] Seq=1575 Ack=644 Win
25	1663131418.404619	172.31.15.248	89.236.112.100	TLSv1.2	117	Change Cipher Spec, Encrypted Handshak
26	1663131418.575051	89.236.112.100	172.31.15.248	TLSv1.2	516	Application Data

```
src_p  ts                        src_ip          dst_ip         dst_p  tcp_rtt   tls_rtt   hello_delay
32830  2022-09-13T22:56:57-0600  89.236.112.100  172.31.15.248  443    0.170668  0.452648  0.287144
```

Metric	Exit Node	Delta	Browser
TCP RTT	171ms		
Hello Delay		287ms	
TLS RTT			453ms

Layer 4 Proxy (Tor): Client Browser Fingerprint

Ground Truth: Tor browser on Win 10

```
"GET / HTTP/1.1" 200 1093 6686 "54.153.106.102" "-" "Mozilla/5.0 (Windows NT 10.0; rv:91.0)
Gecko/20100101 Firefox/91.0" "keep-alive" "en-US,en;q=0.5" "gzip, deflate, br"
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8" "-" "-" "-"

"GET /mvp.css HTTP/1.1" 200 389 9030 "54.153.106.102" "https://54.153.106.102/" "Mozilla/5.0
(Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0" "keep-alive" "en-US,en;q=0.5" "gzip,
deflate, br" "text/css,*/*;q=0.1" "-" "-" "-"


ja3
2a6c83d6c97c17cdba17c3c10e60525c
```

Layer 4 Proxy (Tor): Exit Node TCP Fingerprint

Ground Truth: Varies

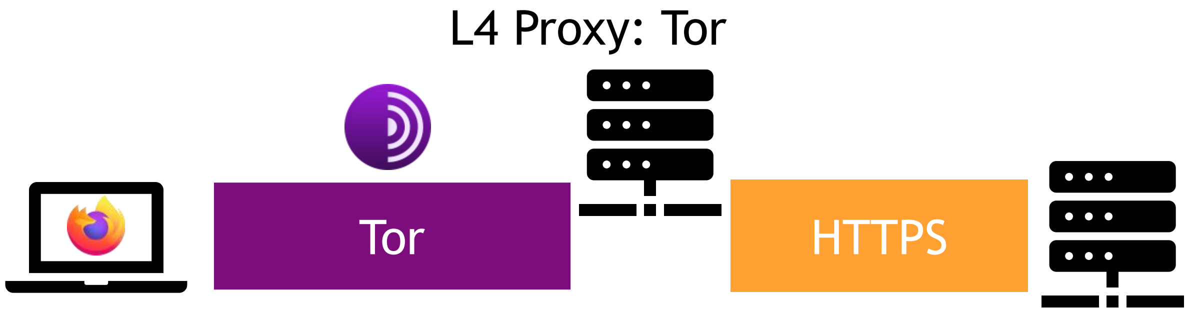
Apparently Linux (consistent with recent Ubuntu)

src_p	src_ip	dst_ip	dst_p	state	size	scale	mss	ttl	df	options
32830	89.236.112.100	172.31.15.248	443	S3	64240	7	1460	42	T	2,4,8,1,3

Layer 4 Proxy (Tor): Recap

Layer 4 (ex. Tor):

- TLS is end-to-end
- Low TCP iRTT, high TLS iRTT — 171ms TCP RTT, 453ms TLS RTT
- TCP Handshake/TLS Client Hello Delay — 287ms Client Hello Delay
- TCP fingerprint mismatch TLS: — Windows TLS/Browser vs Linux TCP



ja3 Related Digests

System	md5 digest	Version
Tor browser, Win 10, to domain	c834494f5948ae026d160656c93c8871	771
Tor browser, Win 10, to IP	2a6c83d6c97c17cdba17c3c10e60525c	771

Cipher
4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-156-157-47-53-10
4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-156-157-47-53-10

Extensions	EC Curves	EC Formats
0-23-65281-10-11-16-5-34-51-43-13-28-21	29-23-24-25-256-257	0
23-65281-10-11-16-5-34-51-43-13-28-21	29-23-24-25-256-257	0

Only One difference

0 is Server Name Indication

SNI informs server of connection domain so server can provide correct certificate, etc.

ja3 md5 digest is completely different for same browser

- Two digests? One for extensions (likely to change) and one for everything else (static)?
- Digest tied to meaning (compress but don't have cascading change)?

HTTP Header Analysis: Spoofed User-Agent

HTTP Header	Python	Python Spoofing Firefox	Actual Firefox
User-Agent	`python-requests/2.22.0`	`Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:104.0) Gecko/20100101 Firefox/104.0`	`Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:104.0) Gecko/20100101 Firefox/104.0`
Connection	`keep-alive`	`keep-alive`	`keep-alive`
Accept-Language	-	-	`en-GB,en;q=0.5`
Accept-Encoding	`gzip, deflate`	`gzip, deflate`	`gzip, deflate, br`
Accept	`*/*`	`*/*`	[varies based on requested content-type] `"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8"`

Impact of NAT, Virtual Machines Containers

Ubuntu 18 LTS guest in VirtualBox on Windows 10 host

Difference between standard "NAT" and "bridged" network config

mode	src_p	dst_ip	dst_p	state	size	scale	mss	ttl	df	options	tcp_rtt	tls_rtt
bridged	58636	172.31.15.248	443	S1	64240	7	1460	39	T	2,4,8,1,3	0.039816	0.052614
NAT	59172	172.31.15.248	443	S1	64240	8	1460	103	T	2,1,3,1,1,4	0.042208	0.056231

Bridged TCP Fingerprint is typical of Ubuntu Linux

NAT TCP Fingerprint is typical of Windows 10

Ephemeral Port Selection: Should match port selection of last NAT router

Hop Count is the Same: 64 – 39 = 128 – 103 = 25

Latency is ~2ms higher with NAT (didn't measure variance)

Future Work and Conclusions

Advancing Malicious Web Client Profiling: Analysis Techniques

Manual Analysis/Domain Knowledge

Metadata Collection (gait)

Information Retrieval/Correlation
- Pivot on fingerprints
- Map ja3 to full attributes, related hashes
- Codify Attacker TTPs (ex. browsing habits)
- Capture observations about specific fingerprints

Semi/Un-supervised Learning
- Mapping of indicators to meaning
- Low FP anomaly detection?

Advancing Malicious Web Client Profiling: Community Cooperation

Begin collecting, using fingerprinting/timing data

Develop language and methods for sharing malicious client profiles

Demand visibility/customization from vendors

Improve data collection and analytic techniques

Conclusions

Analytic Methods

- Software fingerprinting
- Path artifacts (RTT)

Most indicators at a given layer are not unique

- Combination of multiple layers can be high fidelity

Detect malicious web clients

- Earlier in ATT&CK/Kill Chain
- Effective opaque devices/services without EDR, pre-actions on objectives visibility

Turn adversary deception against them!

Thanks, Questions?

Backup

Exercise: Set up test web server

Create Test Web Server (instruction assume EC2 Amazon Linux 2)

Enable TLS with self-signed cert

- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/SSL-on-amazon-linux-2.html

Make simple web page with HTML resources

- cd /var/www/html
- wget https://andybrewer.github.io/mvp/mvp.html
- wget https://andybrewer.github.io/mvp/mvp.css
- mv mvp.html index.html

Enable more verbose apache logs:

- Add following to virtualhost section of /etc/httpd/conf.d/ssl.conf before </VirtualHost> tag:

```
LogFormat "%v %p %h %{remote}p %l %u %t %{sec}t.%{usec_frac}t %D \"%r\" %>s %I %O \"%{Host}i\"
\"%{Referer}i\" \"%{User-Agent}i\" \"%{Connection}i\" \"%{Accept-Language}i\" \"%{Accept-Encoding}i\"
\"%{Accept}i\" \"%{X-Forwarded-For}i\" \"%{Cookie}i\" \"%{Authorization}i\"" extended
CustomLog logs/ssl_extended_log extended
```

Restart apache:

- systemctl restart httpd

Exercise: Capture Web Traffic on Web Server

Capture Web Traffic on Web Server

Make Changes to network config so packet capture better reflect what is transferred over internet (resets at reboot)

- sudo ip link set dev eth0 mtu 1500
- sudo ethtool -K eth0 tso off
- sudo ethtool -K eth0 gro off
- sudo ethtool -K eth0 gso off

Capture Traffic for each scenario

- sudo tcpdump -nn -i eth0 -s0 "tcp port 443" -w /tmp/example.pcap

Exercise: Install zeek and use ja3 and gait extensions for profiling

Install zeek

- https://docs.zeek.org/en/master/install.html
- https://docs.zeek.org/en/master/quickstart.html

Install ja3 and gait extensions (or download and simply include on command line for zeek execution)

- https://github.com/salesforce/ja3
  - Uncomment sections to generate full ja3, not just digest
- https://github.com/sandialabs/gait

Use following commands to generate concise fingerprint data for web clients:

```
#!/bin/bash
# profile web clients using zeek

rm conn.log ssl.log

/usr/local/zeek/bin/zeek -C -r "$1" local | grep -v -F "WARNING: No Site::local_nets have been
defined."

( echo "src_p ts src_ip dst_ip dst_p state size scale mss ttl df options tcp_rtt tls_rtt dur
trips mung hello_delay ja3";
join -1 3 -2 1 <( cat conn.log | /usr/local/zeek/bin/zeek-cut -d ts id.orig_h id.orig_p id.resp_h
id.resp_p conn_state orig_win_size orig_win_scale orig_mss orig_ttl orig_df orig_tcp_options
tcp_handshake_duration | sort -g -k3) <(cat ssl.log | /usr/local/zeek/bin/zeek-cut id.orig_p
min_rtt ssl_handshake_duration ssl_handshake_trips time_munging orig_hello_delay ja3 | sort -g) |
sort -k2 ) | column -t -s" "
```

Exercise: Create and Connect Using Wireguard VPN Server

Create WireGuard VPN server in location geographically far from test web server

Example Instructions:

- https://www.freecodecamp.org/news/how-to-set-up-a-vpn-server-at-home/
- https://www.cyberciti.biz/faq/install-set-up-wireguard-on-amazon-linux-2/

If needed, set MTU to better match normal scenario:

- sudo ip link set dev eth0 mtu 1500
- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/network_mtu.html

Connect to VPN, set up packet capture on web server, and browse web server

Exercise: Tor, related ja3 hashes

Download and install Tor browser

Demonstrate related ja3 digests

- Browse to test server IP and domain
- Browse to test server multiple times (subsequent connections will resume TLS session)

Exercise: HTTP User-Agent Spoofing

Demonstrate naïve HTTP User-Agent Spoofing using python

```
python3
import requests
requests.get('https://testserver', verify=False)
requests.get('https://testserver', headers={'Spoofed UA'}, verify=False)
```

Compare extended web logs with standard web logs

Exercise: NAT and Virtual Machine Containers

Use a host and guest VM with different Operating System

Toggle between NAT and bridge networking mode, comparing TCP fingerprints

Exercise: Additional Ideas

Use SSH to create remote SOCKS proxy

- Ex. https://linuxize.com/post/how-to-setup-ssh-socks-tunnel-for-private-browsing/

Use privacy focused commercial VPN (many options)

- What layer does client interface operate on?
- From web server perspective, what layer does proxy operate on?
- Some VPN provider support multiple protocols
  - How does client interface differ?
  - How does traffic received at server differ?
- In event MTU/MSS is spoofed, can you detect internal MSS used by proxy network?
  - Hint: look at large uploads to web server

Web Client Profiling Resources

https://incolumitas.com

- Excellent proxy fingerprinting ideas

https://github.com/salesforce/ja3

- Passive TLS fingerprinting

https://github.com/salesforce/hassh

- SSH fingerprinting attributes

[Intentionally left blank]

- TCP fingerprinting reference

Path Fingerprinting: TCP timestamps (TCP option)

Timestamp is a TCP option

TCP timestamps used by endpoints for flow control (calculate RTT, prevent seq wrapping)

Monotonically increasing counter, echoed by other endpoint

- Often increases by milliseconds

Older Linux systems leak uptime

Fixed in 2016/kernel 4.10/ubuntu 15

- https://github.com/torvalds/linux/commit/95a22caee396cef0bb2ca8fafdd82966a49367bb

Random offset for each socket (server_ip, server_port)

TCP Operating System Fingerprinting: Linux Ephemeral Port Selection

Modern Linux is interesting example

Update in 2016/kernel 4.2 (~ubuntu 15) to increase efficiency, decrease contention

- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=1580ab63fc9a03593072cc5656167a75c4f1d173

Hash function assumes lower port range is even, upper port range is odd

connect() uses incrementing (n+2) even ports

bind() uses incrementing (n+2) odd ports

port selection is local, but not process specific

- Incrementing ports only observed with same server ip, port