SAND2025-02246

SANDIA REPORT

Printed Feb 2025

A Taxonomy and Feature set for Server-Side Identification of Proxies

Charles Smutz

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

ABSTRACT

Malicious actors frequently use proxies and VPNs to evade detection and hide their origin. Current challenges to information security include the use of residential proxies to blend in with normal traffic and Man-in-the-Middle phishing proxies that are used to compromise accounts protected with mult-factor authentication. We advance a taxonomy and feature set for the identification of proxied traffic based on the network layer where proxying occurs. We describe how these features apply to common proxy types and how to use these features in the classification of the proxied traffic. Collection of these additional features is feasible using existing network sensors and web servers, while only adding about 30% volume to commonly deployed network sensor logs.

This page intentionally left blank.

ACKNOWLEDGEMENT

Thanks to the attackers who have shown us that proxies are a detection opportunity, our team members who have helped develop and implement these methodologies, and our management for their encouragement to publish this information.

This page intentionally left blank.

CONTENTS

LIST OF FIGURES

This page intentionally left blank.

LIST OF TABLES

This page intentionally left blank.

1. INTRODUCTION

Proxies and Virtual Private Networks (VPNs) are network traffic intermediaries that can be used to route internet traffic indirectly and can cause traffic to appear to originate from a network that is different from the actual traffic origin. Proxies are used for many reasons including privacy of individual internet users, and enhanced authentication and confidentiality of organizational traffic. Unfortunately, they can also be utilized to mask the origin of malicious activity.

The malicious use of proxies is enabling new attack vectors and evasion of existing defensive mechanisms in internet services. For example, Proofpoint recently reported a "surge of over 100% in successful cloud account takeover incidents" attributed to Adversary-in-the-Middle phishing (also known as MitM) which is "based on a reverse proxy architecture, which allows attackers to steal MFA-protected credentials and session cookies".[1] According to Microsoft, this increase in business email compromise is fueled by the use of residential IP proxies that "match the victim's location" and "which empower cybercriminals to mask their origin".[2] This allows attackers to circumvent "impossible travel" timings and other location-related indicators of account compromise. Microsoft also reported on an attack group that was seeking persistent access to critical infrastructure. They reported that this attacker "tries to blend into normal network activity by routing traffic through compromised small office and home office (SOHO) network equipment, including routers, firewalls, and VPN hardware."[3] The use of these types of adhoc proxied infrastructures are a growing part of emerging tactics that is challenging information security practitioners.

Current approaches to proxy detection predominately focus on a reputation model where a database of proxy exit node IP addresses is constructed using information derived from internet scanning, proxy network infiltration, or the collection of reported exit node IPs. In this model, a network defender queries a database to determine if traffic from a given IP is likely proxied based on previous findings. This approach has numerous limitations including the timely detection of new exit nodes and separating direct traffic from proxied traffic in the case that two types of traffic both originate from the same IP (as is common with residential proxies). Alternatively, we are focused on detecting the malicious use of proxies from the perspective of a web service provider using attributes of the traffic itself, instead of reliance on previous observations and classification.

There are two related goals in countering the malicious use of proxies. First, a network defender gains an advantage from being able to reliably detect the use of proxies whether or not the traffic coming from the proxy is malicious. Second, there is added benefit from being able to isolate

[1] `https://www.proofpoint.com/us/blog/email-and-cloud-threats/cloud-account-takeover-campaign-leveraging-evilproxy-targets-top-level`

[2] `https://www.microsoft.com/en-us/security/business/security-insider/reports/cyber-signals/shifting-tactics-fuel-surge-in-business-email-compromise/`

[3] `https://www.microsoft.com/en-us/security/blog/2023/05/24/volt-typhoon-targets-us-critical-infrastructure-with-living-off-the-land-techniques/`

specific proxy networks and patterns that identify traffic from a specific adversary. In the case of a persistent adversary using a custom proxy network or common proxies in a unique way, the use of proxy profiling serves as a way to counter subsequent malicious activity such as the use of an unknown exploit or the use of newly compromised authentication tokens. Our approach differs from most privacy research in that our primary goal is to identify the tooling and infrastructure of malicious actors and is not focused on determination of the real-world identities or geographic location of the actor.

We propose a taxonomy for categorizing proxies/VPNs based on the network layer where proxying occurs. This taxonomy employs features derived from network path artifacts and software fingerprinting that highlight discontinuities across the network stack. These specific features can be collected by network sensors or at the web server. We demonstrate that this taxonomy and feature set is applicable in identification of recent attack vectors utilizing well-studied and relevant examples. We have implemented the collection of these feature sets in a couple of open-source and widely deployed network sensors and web servers. We focus on HTTPS traffic because of the ubiquitous nature of these types of web services, but most of the approaches described here can also be applied to other protocols and use cases. We also demonstrate that it is feasible to collect the metadata necessary for proxy identification at the network edge. Lastly, we briefly address some potential evasion mechanisms and show that some forms of evasion open attackers to additional detection opportunities.

Our contributions include:

- A taxonomy of proxy types based on network artifacts
- A feature set for server side proxy identification
- Feasibility of these methods as demonstrated by a real-world deployment
- Examples showing that many evasion methods may result in additional detection opportunities

In section 3, we briefly outline our proxy taxonomy. In section 4, we expound the proposed features used in proxy identification with examples of common proxies used in current attacks. In section 5, we show this method is feasible for broad deployment. In section 6, we address evasion methods of countering evasion.

2. RELATED WORK

Anonymity networks, such as Tor [3], have been extensively studied, attacked, and improved [14, 9, 18]. Xue et al. demonstrated that OpenVPN can be fingerprinted from the point of view of an ISP using network payload fingerprinting and active probing of potential OpenVPN servers [25]. We build upon this privacy research but attempt to address a different problem. We advance the detection of malicious web clients from the perspective of a web service provider.

Habibi et al. developed a feature set that includes inter-packet and flow times, bytes transferred, and duration to determine the type of traffic (web browsing, video chat, etc) transferred through Tor [6]. These features have been used for numerous feature space studies, using various computational and machine learning methods [1, 17]. This approach and feature set is effective for determining the type of traffic transferred through a proxies whereas we focus on detecting the type of proxy used.

Residential IP (RESIP) proxy networks are utilized primarily as a way to defeat server-side blocks and are frequently used for criminal activities. Mi et al. describe how residential IP proxies operate through infiltration (acting as a client) and port scan fingerprinting both from the inside and outside of the proxy network [13]. Tosun et al. use flow analysis, especially matching of inbound to outbound flows at the ISP level to detect RESIP traffic [21]. Yang et al. perform an extensive enumeration of RESIPs in China, adding features derived from web crawling of RESIP provider web sites and passive DNS information [26]. Similarly, our taxonomy and features apply directly to RESIPs, but we add the ability to identify proxied traffic at the target web server.

Kondracki et al. study the emergence of MitM phishing toolkits and demonstrate that they can be detected using timing and TLS fingerprint features from the client perspective but limit their evaluation to comparing TLS fingerprints with HTTP User-Agent strings on the server side [11]. We enumerate complementary detection opportunities including identification of HTTP header changes made by the MitM proxy and the practical implementation of TLS RTT on the web server side.

Features based on timing analysis and network protocol fingerprinting have been studied extensively for a variety of purposes. In one of the earliest public reports of a widespread intrusion by an apparent espionage actor, Stoll demonstrated that timing analysis provided evidence that malicious activity was originating from a distant source [20]. Webb et al. further demonstrate server-side detection of some types of proxies by comparing TCP RTT and HTTP RTT [24]. Our taxonomy clarifies how specific timing analysis features apply in various proxy network types. Ramesh et al. advance a similar method for detecting proxies based on comparing differences in RTT at various layers of the TCP stack and employ a novel method for measuring IP RTT [16]. We advance a similar layer-based approach to server-side detection of proxies but provide a more granular and comprehensive taxonomy that incorporates additional timing and fingerprinting features.

Similarly, studies of network protocol fingerprinting (both active and passive fingerprinting of the TCP stack to identify operating systems) date back to before the turn of the century [19, 5]. Netalyzer by Kriebich et al. used a java applet on the client to gather information on path MTU, DNS resolution, HTTP caching, etc. for debugging various network issues [12]. These features were found to be useful for detecting some types of forward proxies [23]. TCP fingerprinting has been used to identify both vulnerable target and malicious origin systems for decades. For example, p0f[1] is a passive OS network fingerprinting tool that was written in 2000 and was widely used in edge security devices and network sensors but has since been abandoned, with the last revision of p0f being released in 2016. Unfortunately, many of the previously studied fingerprinting references and tools are no longer up to date with current operating systems and attack methods.

JA3[2] was released in 2017 and is a widely used implementation of TLS fingerprinting. The recently released JA4[3] is designed to address some of the weaknesses of JA3. JA4 is also incrementally adding features such as TCP and latency. Our work overlaps with JA4 in the type of data collected, but we focus on identifying proxy networks using cross-layer comparisons.

Fingerprinting of individual user devices can be used to both counter fraud and to weaken user privacy. Alaca et al. survey various fingerprinting methods including TCP fingerprinting, DNS side channels, and browser fingerprinting to improve authentication [2]. Iqbal et al. demonstrate that browser fingerprinting is widely deployed and offer improved countermeasures [8]. Kol et al. exposed a flaw in the Linux ephemeral port selection algorithm that allowed tracking of individual systems [10]. We complement these approaches by advancing features for proxy profiling without de-anonymizing individual users or devices.

[1]`http://lcamtuf.coredump.cx/p0f3/`

[2]`https://github.com/salesforce/ja3`

[3]`https://github.com/FoxIO-LLC/ja4`

3. TAXONOMY FOR IDENTIFYING PROXIES

In this section, we present a taxonomy for classifying proxies based on the layer of the network stack where the proxy operates. More precisely, we propose classifying proxies based on the fundamental unit of information that is transferred through the proxy network. This fundamental unit (packet, data stream, or application layer request) corresponds to the lowest layer of the network stack that is end-to-end across the communication. Figure 3-1 shows examples of the three major proxy types, highlighting the end-to-end nature of the unit of data that is transferred with each proxy type.

We propose classifying proxies by network stack layer because this organization helps highlight cross-layer discontinuities visible to web service providers. In this effort, we focus on two major classes of attributes at each layer: software fingerprints and path artifacts (including round-trip time (RTT) estimates). Table 3-1 enumerates the attributes at each layer of the network stack separated by the proxy types. Attributes above the layer of a given proxy type are artifacts of the original system, attributes below that layer are artifacts of the proxy network/exit node, and artifacts at the same layer are unique to that proxy variety.

Cross-layer comparisons of the artifacts from the original client against artifacts from the exit node are important because many of the artifacts at a given layer are common, but combinations of artifacts across the stack provide opportunities for discernment. For example, TCP/IP operating system fingerprinting can rarely be used as a sole indicator of proxy use. There are a relatively small number of common operating systems and these operating systems are used with many proxy types and can be found exhibiting both benign and malicious activity. However, if a TCP/IP fingerprint indicates the operating system is different from that indicated by TLS fingerprinting, then an L4 proxy may have been employed. Similarly, rarely is a path indicator such as a specific round trip time (RTT) alone a reliable indicator of malicious activity, and most observed RTTs fall within a common range of values. However, a discrepancy that exists between TCP and TLS RTT can provide evidence that an L4 proxy is being employed. Furthermore, the delay between the finish of the TCP handshake and the TLS client hello can help narrow the type of L4 proxy that is being employed.

This multi-layer approach to proxy network analysis is important not only for detecting the use of a proxy network, but also for profiling specific proxy networks and identification of specific threats. For example, a specific threat may be identified by cataloging unique exploitation patterns (such as specific URLs) even though the traffic appears to originate from many IP addresses because a proxy is used to front the traffic. If a unique proxy profile can be created for the threat activity, novel activity using unknown exploit URLs and new IP addresses can be detected by identifying similar traffic with overlapping proxy indicators. These types of proxy profiles are often possible to construct by combining many attributes across the network stack and side channels.

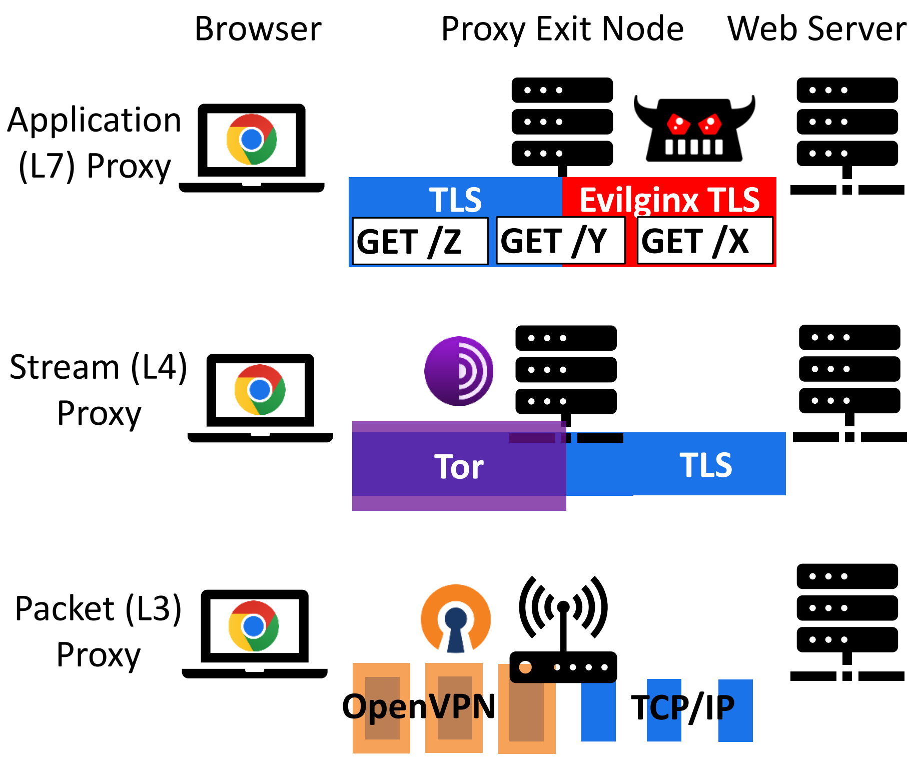

Figure 3-1. Examples of proxy types demonstrating end-to-end nature of network stack layer where proxy operates

Source	Path Artifact	Fingerprint Data
Browser	HTTP RTT	HTTP Headers
L7: App. Proxy		Header Changes
TLS Library	TLS RTT	TLS extensions
L4: Stream Proxy	Hello delay	
Operating Sys.	TCP RTT	TCP/IP options
L3: Packet Proxy	MTU (MSS)	
Internet Scan	IP RTT (Ping)	Port Scan, Banners
IP Reputation	Geolocation	Known Exit Nodes

Table 3-1. Attributes used for identifying proxies organized by network stack layer and separated by proxy types

This taxonomy focuses on artifacts of network traffic that can be used to identify proxies by service providers. Additionally, the taxonomy complements and enhances alternate approaches to counter the malicious use of proxies and can supplement the use of IP reputation databases, detection of side channels, and detection of overt malicious behavior.

This page intentionally left blank.

4. EXAMPLE PROXIES AND FEATURES

In this section, we will describe features that will be useful for proxy identification through examples that are representative of common proxy deployments at different layers of the network stack. These examples were chosen because they are easily repeatable but provide broad coverage across current threat vectors and proxy types.

4.1. L4: Tor Anonymity Network

Tor is an L4 or stream proxy whose anonymity guarantees have been studied extensively and that is used widely in practice. Tor provides clients a SOCKS interface, which is a common client interface for L4 proxies. Other examples of L4 proxies include web proxies that use the HTTP CONNECT method, various port forwarding tools, and malware such as the reGeorg[1] backdoor. While detecting Tor traffic is not challenging because exit nodes are publicly published, we demonstrate methods of detection and characterization applicable to L4 proxies generally that can be observed by web service providers.

4.1.1. TCP and TLS Timing

One of the strongest indicators of an L4 proxy is calculated as the difference in TCP RTT and TLS RTT timings. In an L4 proxy, TLS is end-to-end while TCP is terminated at the proxy exit node. Measurement at an endpoint, such as the web server, is straightforward since the kernel can be queried for the TCP RTT. The TLS handshake RTT measurement was just recently added to OpenSSL. Care must be taken, however, when measuring RTTs on an intermediate monitor that does not terminate the connection. Assuming that a monitor is "pretty close" to an endpoint makes a monitor susceptible to evasion by proxies designed to circumvent edge defenses (e.g., reGeorg). Network intrusion detection systems (NIDS) should measure the time from observation of the first transmission to the observation of the 3rd transmission in a synchronous conversation, such as the TCP 3-way handshake (i.e., 2-way is not enough). Figure 4-1 demonstrates timing measurements using passive NIDS sensor. Recent studies have stated that TCP sequence numbers prevent trivial TCP handshake-based RTT manipulation [16], but this notion overlooks the fact that ACKs can be easily spoofed by an intermediary. In fact, geosynchronous satellite internet providers routinely employ various TCP acceleration mechanisms to lower effective traffic latency, resulting in some artifacts of L4 proxies in almost all traffic from these service providers.

Measuring TLS RTT is similar to measuring TCP RTT. TLS handshake messages aren't guaranteed to fit in a single packet (e.g., large certificate chains) and there is more computational expense in

[1]`https://github.com/sensepost/reGeorg`

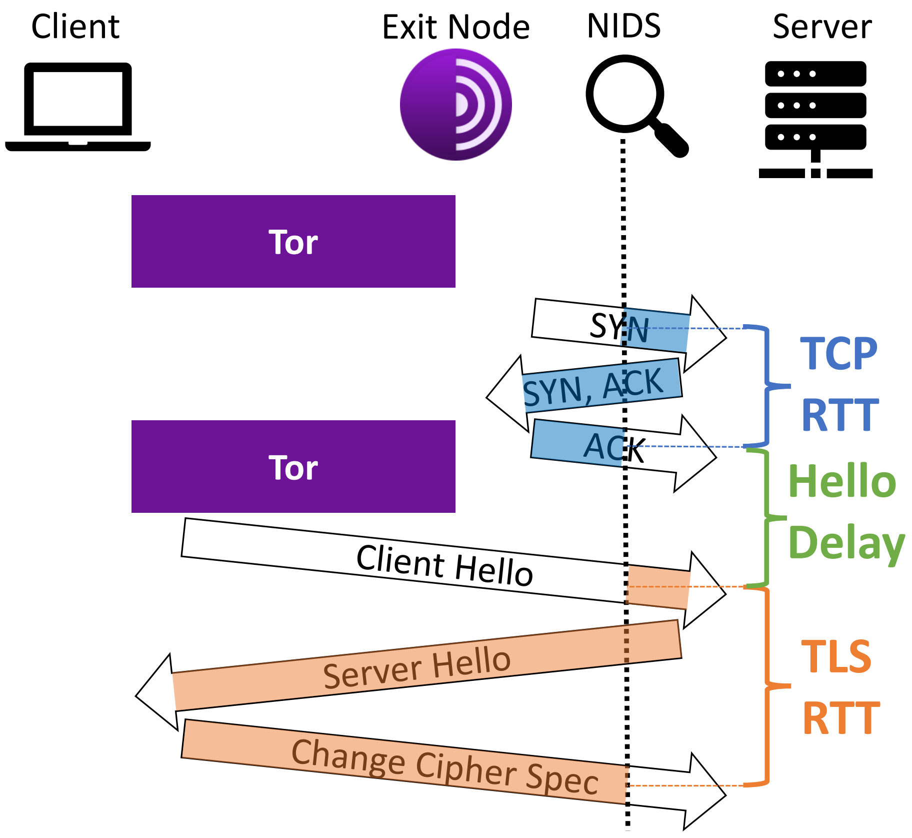

Figure 4-1. Measuring TCP RTT, TLS RTT, and TLS Client Hello delay using passive NIDS

TLS handshakes (e.g., PKI operations), but some points in TLS handshakes are guaranteed to be synchronous due to cryptographic operations used for key negotiation. TLS version 1.2 and older usually have an observable 4-way handshake in the case of a new connection and a 3-way handshake for some forms of session resumption. TLS 1.3 supports 0-RTT for resumed TLS sessions, but new sessions still have an observable 3-way handshake. Since TLS handshakes can involve potentially high latency tasks, such as certificate status checks or an additional TCP RTT due to TCP window exhaustion, the lowest observed TLS RTT for a client should be used as an estimation of the signal propagation delay taken by the TLS communication.

TLS hello delay is a metric that can be used to differentiate L4 proxies based on their buffering/connection setup strategy. In a direct web connection, the TLS client hello immediately follows the TCP handshake. However, in a SOCKS proxy system, the client signals to the SOCKS proxy to establish a connection and after the TCP connection is established and ready to use, the SOCKS proxy signals back to the browser that the connection is established. If the client waits for a signal from the proxy that the upstream TCP connection has been established before beginning the TLS connection, there will be a large apparent timing gap as observed by the server. Other L4 proxies establish a TCP connection with the client and then buffer the TLS Client Hello at the exit node resulting in no TLS client hello delay (or a smaller delay depending upon where buffering occurs relative to TCP termination). Hello delay represents the signal propagation time from the observed exit node to the SOCKS proxy or where TLS connection is blocked/buffered. In Tor and similar systems that use a SOCKS proxies local to the browser, this timing delay represents the distance from the exit node to the browser. In Tor, the sum of TCP RTT and TLS client hello delay is usually close to TLS RTT. Since Tor uses a SOCKS interface, and because the multihop nature of Tor adds significant propagation distance, TLS RTT and TLS client hello delays are very high on an absolute scale, frequently surpassing 500ms. We believe that we are the first to describe and implement TLS hello delay for proxy profiling. Since all major browsers have chosen to not implement TCP fast open (which would combine the TCP ACK and TLS client hello into a single packet), the hello delay applies consistently to web traffic.

We propose the use of synchronous RTT measurements at the connection handshake with care to separate out the latency of each network layer in proxy identification measurements. Inter-packet delay is commonly implemented to profile the traffic transferred through the proxy [6, 18] and is not relevant for our goal of fingerprinting proxies by network defenders. Focusing on handshake RTTs also helps minimize the impact of buffers at multiple layers and better supports taking action on malicious requests.

4.1.2. TCP and TLS Fingerprinting

Network stack fingerprinting relies on observable differences in protocol implementations to identify the software employed. This type of fingerprinting can be used to identify specific devices and software, but identification of a specific operating system or browser alone is rarely enough for a meaningful identification. However, proxies can be detected and profiled by comparing fingerprints at each layer of the network stack. In an L4 proxy, for instance, it is common for the exit node TCP fingerprint to be incompatible with the client TLS fingerprint.

Passive fingerprinting, such as that implemented by p0f, has largely fallen into disuse, but the same features are largely relevant in the context of proxy profiling. It is possible to separate major operating system families using TCP fingerprinting. In Appendix A, we document a more comprehensive description of the features useful for fingerprinting current operating systems.

ja3[2] is a widely used passive TLS fingerprinting tool, often used in NIDS (e.g., Zeek[3]) and internet scanners (e.g., Shodan[4]). ja3 hashes the types and order of various TLS parameters including the offered cipher suites and extensions. TLS fingerprints reflect the software and configuration of the client and usually maps to a range of versions of a given software. ja3 hashes can be impacted by the client's operating system if browser relies on libraries or configuration from the operating system.

Since ja3 uses a cryptographic hash for fingerprint digests, it is very brittle. A browser may exhibit different ja3 digest values simply based on the use of extensions for server name identification, session resumption, or even optional padding. To use ja3 to link related hashes, either the full pre-hash values must be used or a database mapping ja3 hash values to raw values must be maintained. We concur with other researchers and recommend retaining full pre-digest values [7] for fingerprinting. In Appendix B, we enumerate the attributes that can be useful for fingerprinting TLS software.

In the case of Tor, the TCP fingerprint as observed by the web server reflects the operating system of the exit node (in practice this is usually Linux or BSD). In fact, the lack of diversity in relay nodes is so extreme that the project has asked for more non-Linux relays. [5] The Tor browser reports the same User-Agent of Firefox on Windows 10 regardless of whether it is running on Windows or Linux. Hence, traffic from the Tor browser consistently exhibits a mismatch between the exit node (Linux or BSD) and the reported User-Agent (Windows 10), whether or not a mismatch actually exists. Additionally, the ja3 of the Tor browser is distinct from the default configuration of Firefox (on which it is based) because it offers a smaller list of cipher suites, making the ja3 singularly useful for profiling the Tor browser. An additional discerning artifact that we observed is that the default behavior of Firefox is to open two TCP/TLS connections to a target web server upon initial communication whereas the Tor browser only opens a single connection in similar situations. We observed the same behavior in a default install of Firefox: two initial connections when connecting directly and a single initial connection when connecting through an OpenSSH SOCKS proxy. Further, the Tor browser is easy to profile because it exhibits indicators that do not necessarily exist in other proxy networks.

Comparing TCP Fingerprints to TLS Fingerprints (or HTTP Fingerprints) can also be useful for generalized proxy detection. For example, if a client reports a ja3 associated with a graphical browser but matches the TCP fingerprint of a headless IoT device, this could indicate that the IoT devices is being used as an L4 proxy.

[2]`https://github.com/salesforce/ja3`
[3]`https://zeek.org`
[4]`https://www.shodan.io/`
[5]`https://community.torproject.org/relay/technical-considerations/`

4.1.3. Side Channels

When used correctly, Tor has strong anonymity guarantees at the network layers, but is also known to be susceptible to side channels via user behavior and browser artifacts [3]. These side channels all represent opportunities to profile specific malicious actors.

4.2. L3: Residential VPN

We use the generic example of a VPN server on a residential network, which is representative of a capability frequently provided by consumer network devices, regardless of the specific VPN protocol. Common proxy protocols include PPTP, IPSEC, OpenVPN, and WireGuard. While many residential proxies operate at L4, we choose this example configuration because it is easy for researchers to replicate. Most of the properties described here will apply to broader use of L3 VPNs, but because residential proxies represent a pressing security challenge regardless of the layer of operation, we will focus on this configuration in our examples.

4.2.1. MTU (MSS)

The Maximum transmission unit (MTU) of packets exiting a L3 proxy can be used to identify L3 proxies, and can often help identify the specific VPN type or client configuration of the VPN client. Because L3 proxies operate by encapsulating packets inside other packets, the maximum packet size on a L3 proxy client interface will be lower than normal. Typically, the maximum packet size in a proxied packet is reduced in size by at least 40-100 bytes, depending on the protocols used and other conventions. Maximum packet size is efficiently inferred by NIDS via the advertised maximum segment size (MSS) in the initial SYN packet which advertises the maximum TCP payload size that can be transmitted through the network. MTU can also be inferred from the largest packet observed, but this requires the endpoint to send data large enough to reliably fill a whole packet. The Linux kernel tracks the MSS advertised by a peer, the MSS determined by Path MTU discovery (ICMP), and the size of the largest segment received. Canonical values for IPv4 on ethernet are 1500 for MTU and 1460 for MSS. The IP header in IPv6 is 20 bytes larger, so 1440 is a common value for MSS for IPv6 on ethernet. Common default values for MSS for different VPNs include 1380 for Wiregaurd and 1360 for the builtin VPN client on Windows.

MTU can also be influenced by various network properties. Some residential ISPs utilize Point-to-Point Protocol over Ethernet (PPPoE) which decreases MTU and MSS by 8 bytes. If a double VPN is employed, where one VPN is tunneled through another VPN, MSS will be reduced even further. Most VPN clients allow customization of MTU so that an unnecessarily low or unique MSS value can be used to profile VPN specific software, VPN services, or individual client configurations.

4.2.2. IP RTT (ping)

In the case of a L3 proxy, packets are end-to-end from the client to the server. However, since the IP layer is responsible for internet routing, many IP-layer and IP-based artifacts are actually based on the exit node. For example, pinging the apparent source IP of traffic leaving a VPN exit node will often result in a response from the exit node instead of the original endpoint. This is especially true in the case of residential networks were NAT is the norm, but other factors can cause similar results. Care must be taken to discern if IP layer artifacts are from the VPN client or the exit node. It is possible for a server to observe IP headers from both the client and the exit node associated with the same connection. Considering the case of a VPN client that is not configured to advertise a reduced MSS, traffic in the main TCP channel will have IP headers that originate from the VPN client but the resulting ICMP messages that notify the server to reduce MSS will typically originate from the VPN exit node.

The simple timing comparison would be to compare exit node IP RTT to VPN client TCP RTT. This is difficult in practice because IP RTT is difficult to measure passively and some methods of IP RTT measurement are dependent upon VPN type or network policies. Ping and other ICMP methods are challenging to use because ICMP is inconsistently blocked and ICMP methodologies typically involve side-channels or other active changes to network traffic.

Regardless if the VPN client has a public IP address or not, traffic to the observed source IP is routed through the internet based on the path to the exit node. This means that most data based on the apparent client IP address such as geolocation and Internet scan data is a function of the network of the exit node.

4.2.3. Geolocation

Residential proxies are often used to defeat IP-based geolocation restrictions, such as ensuring that traffic is originating from a residential network or a certain geograpic region. Databases such as MaxMind GeoIP [6] are widely integrated into edge security tools including network sensors and web servers. Previous work [15] has shown that the relationship between geographic distances to network RTT is not linear and accurate to 100s of km depending upon circumstances.

An alternative to comparing RTT to geographic distance is to compare the TCP RTT of proxied traffic with the minimum RTT of traffic from the same ISP and geographic region. In practice, if legitimate users are known to be located in a small geographic region, then static thresholds can be used to identify attackers originating from distant locations, especially where traversing oceans helps create a gap between local and distant traffic.

4.2.4. Internet Scans

One other source of discrepancy that can be used to identify L3 proxies is comparing the software fingerprint of the network stack with internet scan data. Since packets are end-to-end in a L3 proxy,

[6] `https://www.maxmind.com/`

the TCP fingerprint of the traffic matches that of the original client. An important exception here is that since NAT is the norm in residential routers, the ephemeral port selection criteria of traffic usually matches that of a router that performs NAT instead of that of the original host.

Internet scan data (e.g., data provided by Shodan or Censys[7]) is widely available to network defenders and is frequently integrated with NIDS and security event analysis systems. The information in these internet scans can be used to identify internet accessible services including software vulnerabilities and to identify specific device types. In the context of proxy detection, inferred device type can be compared to the TCP fingerprint of traffic originating from the same IP for differences. Internet scan data can also be used to identify exposed VPN services or other related indicators (ex. TLS certificate subjects/issuers) [25]. Unsurprisingly, it is common for consumer devices like residential routers that are used in malicious proxy networks to have management consoles or other services exposed to the internet, sometimes with specific vulnerabilities or mis-configurations directly observable in the scan data (ex. outdated firmware version). Presumably, these exposed services are a common pathway for compromise of these devices and their integration in malicious proxy networks. However, care must be taken when using the systems identified by internet scan data because port forwarding often combines the services of multiple devices, especially on residential networks.

4.2.5. IP TTL

IP TTL or (hop limit for IPv6) is a counter that is decremented each time the packet is forwarded by a router. As such, it is a path artifact and can be used to compare relative distance if two packets share a common path. However, IP hop count alone is not a reliable predictor of geographic distance or latency. Since packets are encapsulated and encrypted in a L3 proxy, TTL is only decremented when outside of the VPN tunnel (i.e., it is not an end-to-end metric like TCP RTT). Inferred default IP TTL is more commonly used to fingerprint broad operating system families and is often combined with TCP attributes. Relying on IP TTL alone can expose a discrepancy in operating system if IP headers from the client and ICMP from the exit node indicate a different default TTL.

4.3. L7: Evilginx MitM

To demonstrate the properties of L7 proxies, we use Evilginx[8] as an example of an application layer proxy. Evilginx is a man-in-the-middle (MitM) phishing toolkit that facilitates collection of credentials and session tokens by covertly acting as an intermediary between legitimate clients and web sites. Countering this type of MitM phishing, which can defeat some types of multifactor authentication, is a current challenge. When a MitM proxy is used, the end client is usually an unwitting victim. Despite the differences in attack roles, most of the indicators for a L7 proxy are the same for malicious clients using a L7 proxy to hide their origin and MitM proxies used to compromise the accounts of benign clients.

[7] `https://censys.io/`
[8] `https://github.com/kgretzky/evilginx2`

4.3.1. Browser Fingerprinting

Any discrepancy identified in TLS and browser fingerprints can be a strong indicator of the use of a L7 proxy. Evilginx is written in Golang, resulting in TLS fingerprints (ex. ja3) that overlap with other applications written in Golang. These TLS fingerprint values are both predictable and distinct from commonly used browsers such as Firefox, Safari, or Chrome.

The TLS fingerprint of Golang can be compared with either the User-Agent HTTP header or other browser fingerprinting methods. We provide no unique contributions to the myriad of browser fingerprinting methodologies. Advanced methods of browser fingerprinting can be used to identify specific users [10], however, individual user resolution is not necessary to identify L7 proxies, and reliable identification of the source browser software is sufficient. Comparing TLS fingerprints with User-Agents was described in prior work [11], but scripting languages like Golang are frequently observed with spoofed User-Agents for various reasons. Therefore, additional features will be necessary to reliably identify MitM phishing attacks.

4.3.2. HTTP RTT

In an L7 proxy, HTTP requests originate at the client and are replayed to the server, usually with minor modifications. TLS is necessarily terminated at the MitM proxy. Hence, the timing discrepancy in a L7 proxy is between the HTTP and TLS layers. Calculating HTTP RTT is so well established and so ubiquitous that many browsers have implemented a method where the server can ask the browser to provide the HTTP RTT as measured by the browser. [9] We provide no additional contributions to the many known pathways for measuring HTTP RTT on the server side. We simply note that measuring HTTP RTT is not straightforward with passive network sensors when encrypted with TLS and that web server logs should contain at least millisecond resolution timestamps for this analysis.

4.3.3. HTTP Header Modifications

Commonly, L7 proxies pass HTTP requests from the client to the server with minimal modifications. However, the modifications that do occur are an opportunity to detect specific proxy implementations. For example, Evilginx modifications include adding and removing headers, alphabetizing headers and request query parameter order, and changing the case of headers. Many of these changes are not necessary for MitM proxy operation, but do allow identification of Evilginx-specific behavior. Other L7 proxies used by malicious clients to hide their origin might add headers such as "Via" or "X-Forwarded-For". These examples show the value of analyzing all HTTP headers including details such as header order and capitalization. If only select headers are analyzed or if headers are normalized, then the ability to detect these anomalies is curtailed.

[9] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/RTT`

4.3.4. Side Channels

In a phishing MitM L7 proxy scenario, a small number of domains are proxied and often they are domains used for authentication. The boundary between the proxied domains and other domains is the location of many useful indicators. For example, web resources and subsequent pages that are not intercepted by the phishing proxy will exhibit a referrer of the phishing domain instead of the expected authentication domain. It is common for requests to different domains to originate from both the original client IP and the IP of MitM proxy contemporaneously, which differs from a client that is simply moving from one IP to another. When the stolen authentication cookie (or other secret) is used in another browser, there are numerous potential side channels for detection including inconsistencies in browser fingerprint, cookies, web cache, etc.

This page intentionally left blank.

5. EVALUATION

This section will evaluate the overhead that collecting the values needed for proxy identification will add to a realistic network sensor deployment. For this evaluation, we measured data taken from the webserver and network monitor of a mid-sized organization (about 10,000 people) for a single day. This data set includes over 1.5 million web requests and over 500,000 TLS connections from over 28,000 unique client IPv4 addresses. In this collection, there is no known bias (i.e., it represents a typical day).

5.1. Performance

We measure the additional computational complexity and increase in log volume that proxy identification features add to a typical network monitor by comparing resources used by a default Zeek installation to the additional resources required for the Zeek extensions that collect TCP and TLS proxy features. We replay an approximately 12 minute packet capture taken from the internet boundary of a network having an average bandwidth of about 1.5 Gbps. Hence the packet capture is over 100 GB in size and includes over 1 million packets. Testing against all edge traffic in lieu of only traffic to the web server represents a more realistic network monitor deployment for many organizations. We replayed the same capture 5 times with each set of extensions, taking the average of CPU time and reporting uncompressed log sizes.

Table 5-1 shows the increase in CPU time and logs sizes for additional Zeek exentions. The CONN log contains metadata for every unique Layer 4 (UDP, TCP, ICMP) connection and the SSL log contains metadata for every TLS connection. The ja3 extension adds TLS fingerprints to the SSL log. The gait[1] software package is used to add TCP fingerprint and timing metadata to the CONN log and TLS timing information to the SSL log. Adding both ja3 and gait results in about 19% additional computational workload and about a 30% increase to log sizes. The additional resources reported here will vary based on factors such as composition of traffic, but this shows that collecting these features is feasible as the resources required would represent a fraction of that already being used in many environments.

[1]`https://github.com/sandialabs/gait`

Zeek Extensions	CPU Time Time (s)	Delta (%)	CONN log Size (MB)	Delta (%)	SSL log Size (MB)	Delta (%)
none	1536	0	723	0	123	0
ja3	1564	2	723	0	147	20
gait	1788	16	947	31	135	10
ja3 + gait	1831	19	947	31	160	30

Table 5-1. Additional CPU time and log sizes for Zeek with ja3 and gait extensions

6. EVASION AND COUNTERING EVASION

All of the proxy identification features and detection mechanisms outlined in this paper are subject to evasion through spoofing, obfuscation, and layering of evasive capabilities. Therefore, it is not possible to provide a robust evaluation of all possible evasion techniques. However, in this section, we will provide some representative examples of evasion methods and possible countermeasures to these methods. Our goal is to show that most evasion techniques actually open the attacker up to additional, more targeted detection and profiling. Some fundamental constraints in proxy methodologies are hard to disguise or require trade-offs in performance/usability.

6.1. Known Trade-Offs in Tor

Tor has long balanced well known trade-offs in anonymity and performance [3]. It is known that mechanisms such as padding and traffic shaping will provide better anonymity, limiting the ability to perform attacks such as traffic profiling, but these mechanisms come with performance costs. Latency itself is an example of a fundamental tension between anonymity (which is enhanced by additional hops and higher average latency) and usability (which benefits from fewer hops and lower latency). Tor itself does not implement protocol cleansing which makes traffic going over the network susceptible to protocol layer de-anonymization attacks, but also allows for a wide range of client tools and features. The free nature of Tor improves anonymity and simplifies use, but also frequently results in more demand than available capacity in the network [4]. No anonymity tools can fully mask truly unique user behaviors, such as exploiting previously unknown vulnerabilities or very specific targeting.

6.2. MTU/MSS in L3 Proxies

Decreases to MTU as reflected by the advertised MSS are a well-known artifact of L3 proxies. However, the correct advertisement of MSS is important for proper network operation. If MSS is not advertised correctly, it can result in connections stalling due to large packets being dropped. Even if path MTU discovery allows connections to function when MSS is misconfigured or spoofed, the discrepancy between discovered path MTU and advertised MSS can be detected.

Some commercial VPNs provide a L3 VPN interface to clients but function as L4 proxies at the exit node, translating traffic from the L3 interface to an L4 proxy interface. Since the exit node terminates the TCP connection in this scenario, the advertised MSS of the connection appears normal. However, depending on how much buffering occurs in the translation of L3 packets to the L4 stream occurs, the reduced MTU of the client connection may still be expressed as an effective maximum packet size observed in client uploads. It should be noted that common TCP stack implementations, such as the Linux kernel, can track and provide attributes such as the advertised MSS (tcpi_advmss), MTU derived from path MTU discovery (tcpi_pmtu), and largest observed segment (tcpi_rcv_mss).[1]

It may not be practical to disguise the diminished MTU in the case of a residential proxy where the malicious client is utilizing functionality built into the device. Of course, a compromised device can always have additional software installed or the kernel parameters modified, but these discrepancies also provide an opportunity for more specific profiling.

Individual users and service providers can change the MTU from the default for most VPN software, but departing from default values can make the traffic stand out even more significantly. Even if not directly observable through advertised MSS, the use of a L3 VPN is often still detectable through observing maximum packets sizes or ICMP messages used for path MTU discovery. Again, this highlights the trade-off inherent in disguising fundamental properties of proxied traffic and maintaining usability/performance.

6.3. Chrome TLS Extension Randomization

Chrome recently introduced TLS extension permutation where the order of TLS extensions is randomized.[2] This is specifically designed to ensure future implementation flexibility. While this does change the value of ja3 hashes, it really does not prohibit identification of Chrome as the browser since only a portion of the fingerprint source is changed and only the order is randomized (not the values). The successor to ja3, ja4[3], deals with extension order randomization by making the fingerprint extension order independent. This approach decreases the number of digests that map to Chrome, but abandons the discriminatory power of extension order.

As mentioned in Section 4.1.2, TLS extensions change frequently in normal conditions requiring the use of the full fingerprint or a lookup database. Prior research has shown that randomization itself can be fingerprinted and serve as a detection mechanism [22]. In this case, the randomization implemented by Chrome makes it stand out due to high entropy and actually permits differentiation from Chrome derived browsers that do not implement this features. Of course, a client could actually change the TLS negotiation values, instead of just extension order, but that would have real impacts on security and compatibility as well as serving as an additional detection opportunity.

6.4. Evilginx Internet Scan Cloaking

As described in prior work [11], Evilginx implements various cloaking mechanisms to prevent phishing content-based detection via web scans. If a client connects without providing the correct phishing domain, then the TLS connection is dropped and the source IP is blocked. In Evilginx versions prior to 3.0, port 80 is open by default for Let's Encrypt domain validation. When a web request is made to this port, it redirects to HTTPS which is immediately blocked if the phishing

[1]`https://github.com/torvalds/linux/blob/master/include/uapi/linux/tcp.h`
[2]`https://chromestatus.com/feature/5124606246518784`
[3]`https://github.com/FoxIO-LLC/ja4`

domain is not requested. Furthermore, when domain based scanning is implemented, Evilginx will block and ban source IPs that do not request a specific phishing URL. While cloaking does prevent trivial content-based phishing detection of Evilginx, the cloaking behavior itself is observable and abnormal, providing detection opportunities. Cloaking to evade content-based detection opens Evilginx to detection of cloaking behavior.

6.5. Compound Proxies

In practice, multiple proxy types can be combined. Proxy combinations are used to evade multiple types of filters or to provide presumed higher anonymity. As an example of evading multiple filters, Evilginx supports SOCKS proxies for connections to the phished website. Evilginx could be hosted on a typical web hosting provider but route upstream traffic through a residential proxy such that clients connect to the web hosting provider and traffic to the phished website appears to originate from a residential network. However, this combination further increases the detection opportunities because the targeted server can observe indicators of both L4 and L7 proxies.

Some users combine multiple proxy types with the assumption that layered proxies provide higher anonymity. For example, users may use a double VPN or two layers of VPN. Path indicators, such as timing analysis, compound in these scenarios, resulting in the potential to identify users who employ combinations of proxies and separate these users from normal proxy users.

This page intentionally left blank.

7. AVAILABILITY

In our efforts to implement and research methods for conducting proxy identification, we have authored a tool, gait[1], as an open source offering that can be utilized to generate the features outlined in this paper. Gait is a Zeek extension that collects metadata relevant for proxy detection at the IP, TCP, and TLS layers. In addition to the development of the gait tool, we have also mentored undergraduate research projects that have implemented the collection of relevant metadata in OpenSSL and the NGINX web server. For example, TLS RTT calculation was added to OpenSSL version 3.2[2] as a part of these efforts.

[1]`https://github.com/sandialabs/gait`

[2]`https://github.com/openssl/openssl/blob/master/doc/man3/SSL_get_handshake_rtt.pod`

This page intentionally left blank.

8. FUTURE WORK

While software fingerprinting has been studied extensively in the past, there is a need to research current problems and systems such as identifying traffic originating from IoT devices. Updated tools and databases would be welcomed by the network defender community.

It is expected that broad adoption of proxy profiling features and techniques will lead to improvements including more efficient feature collection or new features. Availability of these feature sets will facilitate research in machine learning techniques to enable automated classification of proxied traffic and identification of anomalies.

A related area of research is improving fingerprinting principles generally, especially as it relates to usability of fingerprint representations. For example, we have highlighted the trade-off of using cascading hash functions to represent fingerprints which make it easy to identify changes but make comparing related fingerprints difficult. The community would benefit from standards and tools for sharing and effectively applying specific proxy profiles.

This page intentionally left blank.

9. CONCLUSIONS

Proxies can be effectively categorized by the layer which separates artifacts from the original client and the exit node. Features based on path artifacts and fingerprinting discontinuities improve detection of proxies at the network edge. Collection of this feature set requires modest increases in CPU and storage and is implemented as open source extensions to widely deployed systems. While proxies can frustrate current defenses, they also present an additional detection opportunity.

This page intentionally left blank.

REFERENCES

[1] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150, 2021. https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.4150.

[2] Furkan Alaca and P. C. van Oorschot. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, pages 289–301, New York, NY, USA, December 2016. Association for Computing Machinery.

[3] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium (USENIX Security 04)*, San Diego, CA, August 2004. USENIX Association.

[4] Roger Dingledine and S. Murdoch. Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it. 2009. https://blog.torproject.org/why-tor-slow-and-what-were-going-do-about-it/.

[5] Lloyd G. Greenwald and Tavaris J. Thomas. Toward Undetected Operating System Fingerprinting. In *First USENIX Workshop on Offensive Technologies (WOOT 07)*, Boston, MA, August 2007. USENIX Association.

[6] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of Tor Traffic using Time based Features:. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, pages 253–262, Porto, Portugal, 2017. SCITEPRESS - Science and Technology Publications.

[7] Jenny Heino, Ayush Gupta, Antti Hakkala, and Seppo Virtanen. On Usability of Hash Fingerprinting for Endpoint Application Identification. In *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 38–43, Rhodes, Greece, July 2022. IEEE.

[8] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1143–1161, May 2021. ISSN: 2375-1207.

[9] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. In *Proceedings 2014 Network and Distributed System Security Symposium*, San Diego, CA, 2014. Internet Society.

[10] Moshe Kol, Amit Klein, and Yossi Gilad. Device Tracking via Linux's New TCP Source Port Selection Algorithm. pages 6167–6183, 2023.

[11] Brian Kondracki, Babak Amin Azad, Oleksii Starov, and Nick Nikiforakis. Catching Transparent Phish: Analyzing and Detecting MITM Phishing Toolkits. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 36–50, New York, NY, USA, November 2021. Association for Computing Machinery.

[12] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 246–259, New York, NY, USA, November 2010. Association for Computing Machinery.

[13] Xianghang Mi, Xuan Feng, Xiaojing Liao, Baojun Liu, XiaoFeng Wang, Feng Qian, Zhou Li, Sumayah Alrwais, Limin Sun, and Ying Liu. Resident Evil: Understanding Residential IP Proxy as a Dark Service. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1185–1201, May 2019. ISSN: 2375-1207.

[14] S.J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 183–195, May 2005. ISSN: 2375-1207.

[15] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for internet hosts. *ACM SIGCOMM Computer Communication Review*, 31(4):173–185, August 2001.

[16] Reethika Ramesh, Philipp Winter, Sam Korman, and Roya Ensafi. CalcuLatency: Leveraging Cross-Layer network latency measurements to detect Proxy-Enabled abuse. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2263–2280, Philadelphia, PA, August 2024. USENIX Association.

[17] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, pages 108–116, Funchal, Madeira, Portugal, 2018. SCITEPRESS - Science and Technology Publications.

[18] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. Subverting Website Fingerprinting Defenses with Robust Traffic Representation. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 607–624, Anaheim, CA, August 2023. USENIX Association.

[19] Matthew Smart, G. Robert Malan, and Farnam Jahanian. Defeating TCP/IP Stack Fingerprinting. In *9th USENIX Security Symposium (USENIX Security 00)*, 2000.

[20] Clifford Stoll. Stalking the wily hacker. *Communications of the ACM*, 31(5):484–497, May 1988.

[21] Altug Tosun, Michele De Donno, Nicola Dragoni, and Xenofon Fafoutis. RESIP Host Detection: Identification of Malicious Residential IP Proxy Flows. In *2021 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6, January 2021. ISSN: 2158-4001.

[22] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through Network-Protocol Obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 57–69, New York, NY, USA, October 2015. Association for Computing Machinery.

[23] Nicholas Weaver, Christian Kreibich, Martin Dam, and Vern Paxson. Here Be Web Proxies. In Michalis Faloutsos and Aleksandar Kuzmanovic, editors, *Passive and Active Measurement*, Lecture Notes in Computer Science, pages 183–192, Cham, 2014. Springer International Publishing.

[24] Allen T. Webb and A. L. Narasima Reddy. Finding proxy users at the service using anomaly detection. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 82–90, October 2016.

[25] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J. Alex Halderman, Jedidiah R. Crandall, and Roya Ensafi. OpenVPN is Open to VPN Fingerprinting. pages 483–500, 2022.

[26] Mingshuo Yang, Yunnan Yu, Xianghang Mi, Shujun Tang, Shanqing Guo, Yilin Li, Xiaofeng Zheng, and Haixin Duan. An Extensive Study of Residential Proxies in China. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 3049–3062, New York, NY, USA, November 2022. Association for Computing Machinery.

This page intentionally left blank.

APPENDIX A. TCP Attributes for Fingerprinting

TCP fingerprinting relies on identifying attributes that are artifacts of operating system defaults or software such as network scanners that override system defaults. Typically, attributes are taken from the SYN packets during connection establishment.

A.1. Most Important TCP Attributes

The following TCP/IP attributes are useful for identifying the software used to generate network traffic from initial SYN packets in approximate order of importance and consistency. The most important attributes vary based on the operating systems compared. This list is compiled based on our own analysis but it largely overlaps with other TCP fingerprinting tools such as p0f.

A.1.1. IP TTL

IPv4 Time-to-live (TTL)/IPv6 hop limit is effective for identifying some operating system families. It is especially helpful because it is present in all IP traffic including UDP/ICMP traffic. Typically, the default TTL is not directly observed but is inferred from a small number of common values (64, 128, or 255).

A.1.2. TCP Options

The list of TCP option identifiers are useful for discriminating operating systems. Order is important for separating some operating systems. Options 0 (end of list) and 1 (no-op) should be included in the fingerprint to ensure fingerprints account for the full length of the options field, allowing correlation to SYN packet size.

A.1.3. TCP Window Scale

TCP window scale is an option where the value is useful for fingerprinting operating systems and device configurations. For some operating systems, the default window scale varies based on available system memory.

A.1.4. TCP Window Size

TCP window size is a field where the value is useful for fingerprinting operating systems but can also depend on network configuration. The default value for this field is hardcoded for some operating system but for some operating systems, this value is a multiple of MSS.

A.2. Other TCP Attributes

A.2.1. TCP MSS

For most modern operating systems, the default MSS TCP option value is not a function of the operating system, but is most commonly a function of the network configuration (based on MTU).

A.2.2. TCP Timestamps

In the past, the timestamp option could be used to identify uptime for certain hosts, but in modern operating systems where these values are randomized, the value is not known to be useful for fingerprinting (the existence of the timestamp option is already reflected in the options list).

A.2.3. IP DF Flag

The IP Don't Fragment (DF) bit is enabled for most major operating systems and path MTU discovery has been universally adopted. However, this bit may not be set for some scanner software that overrides system defaults.

A.2.4. IP and TCP ECN Flags

Use of Explicit Congestion Notification (ECN) can help strengthen prediction of the relatively few operating systems that use ECN by default.

A.3. Ephemeral Port Selection

In addition to attributes visible in individual SYN packets, the patterns of source port selection (ex. range of values and incrementing vs. random values) across multiple connections are an artifact of the operating system. Understanding operating system defaults can be used to identify modifications made during the traffic path (NAT changes to source port) or source host traffic patterns.

APPENDIX B. TLS Attributes for Fingerprinting

TLS fingerprinting relies on attributes that are typically determined by the client software including web browsers. Attributes are taken from the Client Hello (and Server Hello) message. Similar to TCP, there are both primary fields and extensions (similar to options). TLS is different from TCP in that the Client Hello message is usually 100s of bytes, versus a TCP header which is 10s of bytes. As a result, the data available for fingerprinting of TLS is much larger and there is larger variance in TLS fingerprints than TCP fingerprints. It is possible to create compact fingerprints using counts of values in fields/extensions, abstracting specific values and ordering, and allowing for efficient similarity comparisons.

B.1. Most Important TLS Attributes

The following TLS attributes are the most useful for identifying client software based on our analysis. These attributes are used in either ja3 or ja4–most are reflected in both. These are roughly ordered by classification value and consistency but this varies depending upon the software profiled.

- Cipher Suites
- Supported Groups
- Signature Algorithms
- Application Layer Protocol Negotiation
- Supported Versions
- Extensions (list of extension numbers)

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

This page intentionally left blank.