TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior

Blake Anderson
Cisco
blake.anderson@cisco.com

David McGrew
Cisco
mcgrew@cisco.com

ABSTRACT

The Transport Layer Security (TLS) protocol has evolved in response to different attacks and is increasingly relied on to secure Internet communications. Web browsers have led the adoption of newer and more secure cryptographic algorithms and protocol versions, and thus improved the security of the TLS ecosystem. Other application categories, however, are increasingly using TLS, but too often are relying on obsolete and insecure protocol options.

To understand in detail what applications are using TLS, and how they are using it, we developed a novel system for obtaining process information from end hosts and fusing it with network data to produce a TLS fingerprint knowledge base. This data has a rich set of context for each fingerprint, is representative of enterprise TLS deployments, and is automatically updated from ongoing data collection. Our dataset is based on 471 million endpoint-labeled and 8 billion unlabeled TLS sessions obtained from enterprise edge networks in five countries, plus millions of sessions from a malware analysis sandbox. We actively maintain an open source dataset that, at 4,500+ fingerprints and counting, is both the largest and most informative ever published. In this paper, we use the knowledge base to identify trends in enterprise TLS applications beyond the browser: application categories such as storage, communication, system, and email. We identify a rise in the use of TLS by non-browser applications and a corresponding decline in the fraction of sessions using version 1.3. Finally, we highlight the shortcomings of naïvely applying TLS fingerprinting to detect malware, and we present recent trends in malware's use of TLS such as the adoption of cipher suite randomization.

CCS CONCEPTS

• **Security and privacy → Security protocols**; *Web protocol security*; Malware and its mitigation.

ACM Reference Format:
Blake Anderson and David McGrew. 2019. TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. In *IMC '19: ACM Internet Measurement Conference, October 21–23, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3355369.3355601

1 INTRODUCTION

Transport Layer Security (TLS) fingerprinting assigns a process or TLS library name to set a cryptographic parameters observed in the TLS `ClientHello` [22, 40] and is successful due to TLS's large parameter space [31, 32] which applications use to optimize their performance and security. TLS fingerprinting has been examined in the academic literature [27, 28, 30, 34, 38] and utilized by the open source community [1, 14, 20, 42]. As a whole, this body of work is extremely valuable for tracking the evolution of TLS, for understanding how fingerprinting affects user privacy, and for developing reliable methods to detect vulnerable or malicious processes.

We extend this line of research by creating the first large-scale system that continuously generates a TLS fingerprint knowledge base by fusing together data from ongoing network and endpoint observations of the same sessions. The network data for this paper was collected using our open source tool Joy [35], and the endpoint data was generated by Cisco AnyConnect [7]. The resulting fingerprint dataset represents real network traffic generated by 24,000+ managed endpoints, 471+ million benign TLS connections, and millions of malware connections. As of this writing, the knowledge base has 5,000+ malware-generated and 3,500+ endpoint-labeled TLS fingerprints, with the endpoint-labeled data covering 33,000+ distinct values of the SHA-256 hash of the process executable and 4,300+ unique process names.

Each fingerprint is associated with a rich set of contextual data taken from real-world sessions, e.g., processes, operating systems, destination IP addresses, hostnames, and ports. We also generate a TLS fingerprint dataset for connections that lack endpoint ground truth that currently has ~21,000 entries based on ~8 billion TLS connections, where each entry tracks prevalence and destination information. We actively maintain an open source subset of the endpoint-enriched knowledge base that currently has 4,500+ fingerprints and is both the largest and most informative ever released [36]. The open source knowledge base is larger than the results reported in this paper because it includes data collected with mercury [36] from an additional network that only became available in late June 2019.

In this paper, we use the datasets generated by our system to study longitudinal trends in the use of TLS beyond the browser and the Web. We also investigate the efficacy of TLS fingerprinting, especially the number of unique processes that map to a distinct TLS fingerprint, and the identification of application categories that are more likely to use the operating system's default library.

Our endpoint enriched knowledge base is generated daily and summarizes real-world network traffic from both benign and malicious processes, which we use to report overall TLS client trends and trends broken out by application categories such as storage, communication, system, and email. Because browsers are responsible for most TLS connections and typically reflect TLS best practices, measurement studies not incorporating endpoint ground truth will miss insights with respect to the full TLS ecosystem due to the inherent class imbalance.

By understanding how different application categories use TLS, we were able to explain a counterintuitive finding: during the period around the ratification of version 1.3, the fraction of sessions using that version *decreased*. This relative decline occurred because non-browser applications increasingly used TLS during this period, but only supported earlier versions. Our analysis shows that 1.3 adoption was nearly nonexistent in most application categories until MacOS made 1.3 the default for 10.14.4 in March 2019. At the other extreme, RC4-based cipher suites have been removed from all modern browsers, but we are consistently observing new applications on modern operating systems offering them. These trends necessitate additional large-scale measurement studies that incorporate endpoint information to ensure all application categories are following TLS best practices.

Finally, by applying the data fusion system to the artifacts generated from a malware analysis sandbox, we were also able to analyze malware detection via TLS fingerprinting, highlight the shortcomings of naïve approaches, and demonstrate situations where fingerprinting is valuable. Malware has continued its trend of abusing privacy enhancing technologies like Tor. We review these trends and explore more recent developments such as malware's adoption of cipher suite offer vector randomization and how malware's use of randomization can be detected.

Our contributions include:

- A novel system for obtaining process information from end hosts and fusing it with network data to produce a TLS fingerprint knowledge base.
- The first large-scale study of enterprise TLS client behavior leveraging endpoint ground truth for a diverse set of applications.
- A long-term analysis of malware's abuse of privacy enhancing technologies.
- An actively maintained open source knowledge base that, at 4,500+ endpoint-enriched fingerprints and counting, is both the largest and most informative ever published.

This paper is organized as follows: Section 2 reviews background and related work, and Section 3 presents the generation methodology and a statistical overview of our knowledge base. Section 4 examines the inferences that can be made from TLS fingerprints, Section 5 details the results of our longitudinal study, and Section 6 investigates TLS fingerprinting with respect to malware. Section 7 discusses the biases our data collection strategy introduced and future directions. Section 8 concludes. Ethical issues are examined in Appendix A.

2 BACKGROUND

TLS fingerprinting extracts features from the first application data packet sent by the client, i.e., the `ClientHello`. The `ClientHello` message provides the server with a list of cipher suites and extensions that the client supports [22, 40]. The cipher suite list is ordered by preference of the client, and each cipher suite defines a set of cryptographic algorithms needed for TLS to operate. These cryptographic algorithms have real-world trade-offs, and this is reflected in different applications choosing to omit, include, or prefer cipher suites more aligned with their goals.

TLS extensions provide additional information to the server that facilitates extended functionality, e.g., the `application_layer_protocol_negotiation` extension [26] indicates the list of application layer protocols that the client supports, e.g., `h2` and `http/1.1`. Some data carried in the extensions are useful for identifying the client application, while other extension data is session specific, e.g., the `session_ticket` data.

The most recent version of the TLS protocol, TLS 1.3 [40], was released in August 2018, and added many security and performance-oriented improvements. Currently, TLS 1.3 only defines five cipher suites, but most clients are continuing to offer TLS 1.2-compatible cipher suites for backwards compatibility. There are several new extensions, such as `supported_versions`, that allows the client to advertise the official TLS and draft TLS 1.3 versions it supports.

2.1 Related Work

TLS fingerprinting gained popularity in 2009 based on Ristić's initial work [42] which spurred several open source initiatives [1, 14, 20, 35, 36, 43]. The academic community has leveraged these ideas to study the TLS ecosystem with respect to security and privacy [16, 25, 27, 28, 30, 34, 38].

Razaghpanah et al. developed the Lumen Privacy Monitor app for Android platforms which acts as a MiTM TLS proxy and forwards TLS configuration information for further analysis [38]. Their inclusion of endpoint data is similar to our approach, but this prior work was specific to Android platforms. Our work focuses on identifying trends specific to applications found on enterprise endpoints by fusing endpoint ground truth with network data, which offers a complimentary set of results to the detailed Android study [38].

Kotzias et al. [34] passively collected a large number of TLS fingerprints and used services such as BrowserStack [6], open source TLS fingerprint databases [20], and their own previous results [38] to supply ground truth. Our fingerprint generation method involves continuously fusing real-world endpoint data including the process name, SHA-256 of the process, and the operating system from 24,000 managed endpoints and a malware analysis sandbox. The fingerprint knowledge base we put forward in this paper offers detailed insights into a wide variety of applications observed within an enterprise network. Most importantly, because TLS fingerprint distributions change over time and our fingerprint knowledge base is automatically updated with the most recent ground truth, the data associated with this paper [35] will continue to offer the community fresh insights.

Kotzias et al. primarily used the ICSI SSL Notary dataset [15] to examine how the TLS ecosystem responded to high-profile attacks with a focus on popular browsers. Our dataset is yet another view into the TLS ecosystem, and several of the measurement studies presented in this paper, e.g., TLS's version usage in Section 5.2, provide a more recent picture from a new vantage point relative to prior work. On the other hand, our fingerprint knowledge base is derived from hundreds of millions of TLS connections from over 4,300 unique process names and 33,000 unique process SHA-256's, e.g., `chrome.exe` can map to many different SHA-256's depending on the version and operating system. This level of detail and the daily generation of TLS fingerprint knowledge bases has allowed us to develop efficient measurement techniques that can identify application TLS usage trends. These techniques do not rely on TLS fingerprinting, but are instead entirely based on ground truth sent from the endpoint. We use these techniques to better understand the TLS ecosystem and to extend prior work by, for example, identifying a lower limit to the number of processes mapping to the same fingerprint and to study TLS 1.3's adoption by individual application categories.

Frolov and Wustrow analyze traffic from an academic network to identify TLS `ClientHello` trends with the goal of developing a censorship resistant TLS library [27]. Similar to previous work, Frolov and Wustrow used BrowserStack to generate 270 unique fingerprints from ~535 user agents, and used this TLS fingerprint database to report top applications. In contrast, our application studies do not rely on TLS fingerprinting, i.e., they only rely on the ground truth provided by the daily and aggregated TLS fingerprint knowledge bases that are automatically generated from a large set of real-world network and endpoint data as mentioned above. Frolov and Wustrow put forward several measurement studies relative to their academic network, e.g., popular cipher suites, extensions, and weak cipher suites. The current paper builds on their prior work by leveraging our unique datasets to provide additional application context, which is based on ground truth and not an approximation via TLS fingerprinting, to these measurement studies.

Both Frolov and Wustrow and the current paper study popular censorship circumvention tools. Frolov and Wustrow studied these tools from the perspective of evading censorship and developed an open source library, uTLS [11], to enable `ClientHello` fingerprinting resistance through randomization and parroting [27]. To compliment this prior work, we approach this topic by studying malware's *abuse* of these technologies. In Section 6.2, we show that a non-trivial amount of malware relies on censorship circumvention tools such as Tor, Psiphon, and Ultrasurf and evasion techniques such as `ClientHello` randomization to evade detection in order to continue exploiting the end user.

3 THE TLS FINGERPRINT KNOWLEDGE BASE

In this section, we describe a novel system for obtaining process information from end hosts and fusing it with network data to produce a TLS fingerprint knowledge base. This data has a rich set of context for each fingerprint, is representative of enterprise TLS deployments, and is automatically updated from ongoing data collection. We primarily use 3 data sources to build our fingerprint knowledge base:

(1) `Passive`: enterprise connections which have no endpoint ground truth
(2) `Endpoint`: enterprise connections with endpoint ground truth
(3) `Malware`: connections taken from a malware analysis sandbox enriched with per-connection process information.

The `Passive` and `Endpoint` datasets were collected from five enterprise edge networks in geographically distinct countries between May 7th, 2018 and September 5th, 2019. Each edge network observes traffic from 500-2,000 unique managed endpoints per week, and we observed over 24,000 unique managed endpoints in the full dataset. The `Passive` dataset is generated from the connections that lack endpoint ground truth.

Data Source	# Fingerprints	# Connections
Passive	21,754	8.05e9
Endpoint	3,510	4.71e8
Malware	5,070	2.75e7
All	496	5.41e9
Endpoint/Passive	2,988	2.62e9
Passive	17,995	4.64e8
Malware/Passive	275	5.55e7
Malware	4,299	4.45e5
Endpoint	26	1.45e2
Total	26,079	8.55e9

Table 1: Unique fingerprint and connection counts for each data source considered individually (first set of rows), for overlapping data sources (second set of rows), and the total (final row).

All data sources include passively collected destination information such as the IP address, port, and hostname taken from the `server_name` extension. This information is represented as a tuple with counts that are associated with specific processes in the `Endpoint` and `Malware` datasets, and with the fingerprint in the `Passive` dataset. The open source database generalizes this information, e.g., IP addresses are generalized to autonomous systems.

3.1 Data Fusion

The managed endpoints at these sites reported the network 5-tuple, begin/end timestamps, endpoint identifier, operating system, and a process SHA-256/name for each connection. Our open source program [35] monitored all inbound and outbound TLS traffic, recording the network 5-tuple, begin/end timestamps, full `ClientHello`, and other fields not relevant for this discussion.

Each night, an Apache Spark job joined the two datasets collected during the previous day based on the network 5-tuple, choosing between duplicate matches based on the minimum difference between the two datasets' begin timestamp field. This operation created a set of enriched records correlating the `ClientHello` with the operating system and the name and SHA-256 hash of the executable of the process responsible for initiating the connection. While it was common for the endpoint's and network monitor's timestamps to not be aligned, the difference was constant in all manually inspected cases. Spurious matches were further reduced by identifying and discarding any host records that had a timestamp difference greater than five seconds away from the mean, which discarded less than one in ten thousand enriched records.

We also perform data fusion with packet captures and analysis files generated from a malware analysis sandbox that processes tens-of-thousands of new malware samples each day on Windows XP (currently deprecated), 7 and 10 virtual machines. We collected this dataset from May 17th, 2016 to September 5th, 2019. Similar to the edge network data, this enriched dataset correlates the

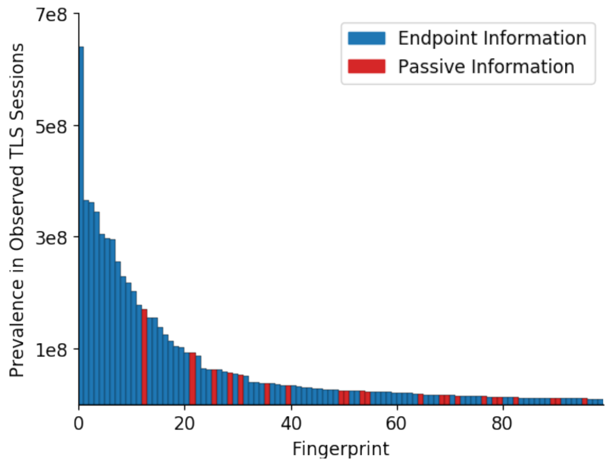


Figure 1: Fingerprint Histogram. Endpoint Information includes fingerprints with endpoint exported process information. Passive Information includes fingerprints with no endpoint ground truth, and only contains the associated servers' network-observable features and counts.

ClientHello with the operating system and SHA-256/name of the process responsible for initiating the TLS connection.

3.2 Fingerprint Generation

For each enriched record, we map the ClientHello to an octet string using a reversible encoding:

```
(version)(cipher suites)((extensions)...)
```

where each field is a hex representation of the bytes observed in the packet, all orderings are maintained, and version is the TLS handshake version. GREASE [17] cipher suites, extension types, and extension data values are left in place, but normalized to the value 0a0a to preserve ordering information. We found that while GREASE values were selected at random, their positions were deterministic. The current GREASE specification states that implementations should balance diversity with determinism, "A client which positions GREASE values deterministically over a period of time (such as a single software release) stresses fewer cases but is more likely to detect bugs from those cases" [17].

extensions includes the data for 21 extensions. The data associated with session specific parameters is omitted, e.g., server_name and key_share, and data associated with client specific parameters is kept, e.g., supported_groups, supported_versions, and compress_certificate. Please refer to our open source package [35] for the full list of extracted extension data. The fingerprint generation process will be refined as necessary if new extensions are introduced or the scope of GREASE is expanded, e.g., if the order of extensions was randomized.

Each days' records are grouped by their fingerprint strings, generating a list of all processes associated with a given fingerprint on a specific day. For each process, we maintain the name, SHA-256, total number of observations, counts of associated operating systems, and destination IP address/port/hostname tuple counts.

Operating System	# Connections	% Connections
Windows 10 (10.0.17134)	204.0 million	45.0%
MacOS 10.14.5	62.9 million	13.9%
MacOS 10.14.6	55.5 million	12.3%
MacOS 10.13.6	28.6 million	6.3%
MacOS 10.14.4	24.7 million	5.5%
Windows 7 (6.1.7601)	13.8 million	3.0%
MacOS 10.14.3	12.2 million	2.7%
Windows 10 (10.0.15063)	10.5 million	2.3%
MacOS 10.12.6	9.8 million	2.2%
Windows 10 (10.0.17763)	6.1 million	1.3%

Table 2: Most prevalent operating systems in the Endpoint dataset. The remaining 6.3% of operating systems included many older versions of Windows and MacOS, as well as several flavors of Linux.

In addition to the process-enriched fingerprints, we also track fingerprints for which there is no endpoint ground truth. For this set, we record the total number of observations and destination IP address/port/hostname tuple counts. The full knowledge base is generated by merging all daily datasets.

3.3 Knowledge Base Statistics

As of this writing, the knowledge base has 26,079 TLS fingerprints. These fingerprints were generated from the Passive, Endpoint, and Malware datasets as described above. Table 1 provides statistics on the number of fingerprints and connections collected from each data source and highlights the amount of overlap between the various sources. 496 fingerprints appeared in all three datasets and were responsible for ~63% of the total connections. As reported in previous studies [27, 34], the majority of connections in our knowledge base are represented by a relatively small set of fingerprints. Figure 1 presents the prevalence of the top-100 fingerprints in our combined dataset; we note that the top 10 fingerprints account for 40% of the connections. We explain this bias in Section 5.4 by examining the behavior of SSL/TLS scanners and in Section 6.2 by examining malware's abuse of privacy enhancing technologies.

The top ten fingerprints were generated from the following set of applications: Microsoft Office on Windows and MacOS, Chrome versions 72-76, Webex Teams, Firefox versions 64-68, Python, and OSquery. Additional information on TLS configurations and popular applications can be extracted directly from our open source database [36]. The "Passive Information" fingerprints in Figure 1 were never observed in the Endpoint dataset; the top-3 of these fingerprints are related to the Android mobile platform. The remaining "Passive Information" fingerprints are related to Linux and mobile platforms. There was not an obvious IoT presence, which is most likely due to biases introduced by our data collection.

Table 2 lists the ten most prevalent operating systems for the endpoint-labeled connections. The remaining 5.5% of connections were generated from a long tail of older versions of Windows, MacOS, and various flavors of Linux including Ubuntu, Linux Mint, Fedora, elementary OS, and CentOS. Ubuntu 18.04.3 LTS was the most prevalent Linux endpoint operating system, but only had ~56,000 connections or ~.01% of the total number of endpoint-labeled connections. Aside from a lack of deeper Linux visibility,

Category	# Connections	% Connections
browser	173.3 million	37.1%
email	90.0 million	19.3%
communication	80.1 million	17.2%
system	42.2 million	9.0%
productivity	31.7 million	6.8%
security	28.9 million	6.2%
storage	12.5 million	4.6%
other	7.8 million	1.7%

Table 3: Application category prevalence in the `Endpoint` dataset.

Cipher Suite	% Fingerprints
`RSA_WITH_AES_128_CBC_SHA`	66.27%
`RSA_WITH_AES_256_CBC_SHA`	65.37%
`ECDHE_RSA_WITH_AES_128_CBC_SHA`	63.34%
`ECDHE_RSA_WITH_AES_256_CBC_SHA`	62.65%
*`EMPTY_RENEGOTIATION_INFO_SCSV`	56.82%
`ECDHE_RSA_WITH_AES_128_GCM_SHA256`	54.93%
`ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`	54.12%
`ECDHE_ECDSA_WITH_AES_256_CBC_SHA`	53.31%
`ECDHE_RSA_WITH_AES_256_GCM_SHA384`	50.76%
`RSA_WITH_AES_128_GCM_SHA256`	49.88%

Table 4: The 10 most common offered cipher suites in the full knowledge base. * indicates a pseudo-cipher suite.

one notable omission included mobile platforms. We further discuss dataset biases in Section 7.

The `Endpoint` dataset contained 33,000+ distinct values of the SHA-256 hash of the process executable and 4,300+ unique process names. 14% of the processes were responsible for 99% of the TLS connections, and we grouped this set into 8 categories (Table 3). The most popular examples include Chrome for `browser`, Outlook for `email`, Webex Teams for `communication`, SearchProtocolHost for `system`, Cisco AMP for Endpoints for `security`, PowerPoint for `productivity`, and nsurlsessiond for `storage`. nsurlsessiond was labeled as storage primarily due to the overwhelming majority of its connections contacting icloud.com. The `other` category contained virtual machine, custom, and the remaining low prevalence applications. Popular examples from the `other` category include iTunes, Hearthstone, Microsoft Photos, VirtualBox, and Python. Additional information on application categorizations can be extracted directly from our open source database [36].

62% of the fingerprints were TLS 1.2, 9% were TLS 1.1, 18% were TLS 1.0, and 3% were SSL 3.0. 7% of the fingerprints were TLS 1.3, which we determined by analyzing the cipher suites and extensions. There were 484 unique cipher suites in the full dataset, where 81 had values that were reserved for private use "`0xFF,0x00-FF`" [32]. We also observed thousands of connections preferring a post-quantum key exchange-based cipher suite, `CECPQ1_ECDSA` [13], but these were all confirmed to be scanners. Table 4 lists the 10 most common cipher suites in our knowledge base. The two most common, `RSA-AES128-CBC-SHA` and `RSA-AES256-CBC-SHA`, do not provide perfect forward secrecy [22] and introduce the potential for advanced attacks [18]; they are also the most preferred cipher suites in ~4.7% of the fingerprints.

Extension	% Fingerprints
`supported_groups`	85.44%
`ec_point_formats`	84.14%
`signature_algorithms`	70.29%
`server_name`	62.94%
`session_ticket`	43.73%
`extended_master_secret`	40.54%
`renegotiation_info`	39.27%
`status_request`	35.63%
`app_layer_protocol_negotiation`	27.30%
`heartbeat`	26.50%

Table 5: The 10 most common advertised extensions in the full knowledge base. Data for bolded extensions are a part of the fingerprint definition.

There were 45 unique extensions type codes in the full dataset, which includes multiple `channel_id` type codes, a `padding` type code used specifically with NSS 3.15.5, and the non-standardized `encrypted_server_name` extension type code, which was seen in 16 fingerprints. Table 5 lists the 10 most common extensions in the full dataset, where data from the bolded extensions are incorporated into the fingerprint definition.

4 VISIBILITY STUDY

In this section, we discuss visibility on two fronts, where visibility is defined as information that could be used to provide process or operating system level information on a per-connections basis. First, we analyze the visibility provided by endpoint monitoring and how many TLS connections match fingerprints generated from endpoint telemetry. This information is either exact ground truth in the case of exported endpoint data or approximate ground truth in the case of a fingerprint match or approximate match.

Second, assuming a fingerprint match, we analyze how useful that match is in practice. Because a diverse set of processes can map to a single fingerprint, a straightforward application of TLS fingerprinting may not provide the assumed benefits.

4.1 Visibility from Endpoint Telemetry

While telemetry from managed endpoints within an enterprise network offers the most informative view into the behavior of those endpoints, solely relying on this approach has several drawbacks. Managed endpoints typically comprise a small percentage of the total number of Internet-capable devices, which in our study was ~10%. The monitoring software often makes design decisions to improve performance that results in lower visibility, e.g., devices might have a limited buffer to store data while connecting via VPN that would later be transferred when the endpoint is physically on the network. Finally, the monitoring software may not support all operating systems and architectures.

Figure 2 highlights the gap between TLS visibility offered by endpoint agents and the total number of TLS connections observed on the monitored networks. Over the course of the year, process

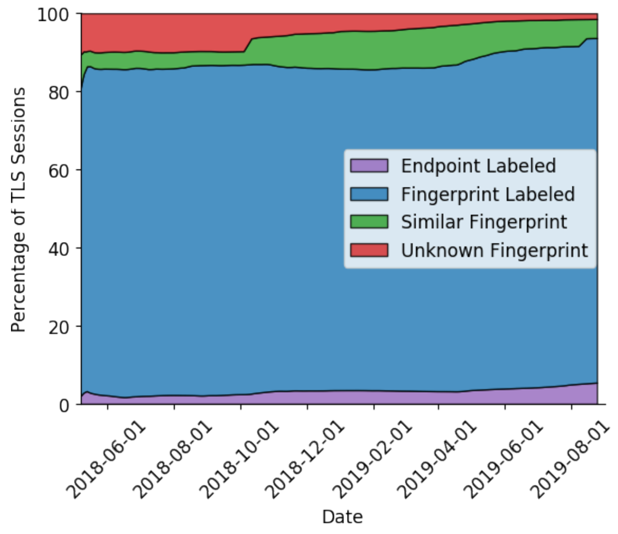


Figure 2: Endpoint visibility gap.

Category	Processes per Fingerprint	Fingerprints per Process
browser	22.19 (50.79)	3.18 (3.12)
email	65.73 (118.84)	2.67 (2.76)
communication	64.00 (108.78)	1.87 (1.51)
system	41.07 (94.41)	1.63 (1.30)
productivity	56.42 (108.72)	1.86 (1.47)
security	76.77 (138.60)	2.05 (2.64)
storage	48.51 (110.38)	1.59 (1.40)
other	52.99 (113.56)	1.53 (0.99)

Table 6: The mean and standard deviation for the number of process SHA-256 hashes per fingerprint and fingerprints per process SHA-256 hash.

information was available for only ~5% of the outbound TLS connections. On the other hand, we were able to consistently find matching fingerprints for ~85-90% of the TLS connections using the knowledge base generated from the previous days' data. These numbers illustrate the value of our system: for most TLS sessions, fingerprinting is needed and is informative.

Similar to previous work [27], we also implemented a fingerprint similarity metric based on the Levenshtein distance. Figure 2 shows that under this metric, we were able to identify "close" fingerprints for nearly all TLS connections after a year of data collection, where we define "close" as a Levenshtein distance less than or equal to 10% of the number of cipher suites, extension types, and extensions values contained within the fingerprint definition. The noticeable jump in similar fingerprints during October 2018 was due to the collection of several examples of OpenSSL running on MacOS. We record the closest fingerprint and all destination information for unmatched connections.

4.2 Visibility from TLS Fingerprinting

Assuming a TLS `ClientHello` matches a fingerprint with well-labeled data, what actionable intelligence can either a network administrator or a censor infer? We examine this visibility with respect to exact matching, approximate matching, operating system, and process correlation in this section, and defer this discussion with respect to malware visibility to Section 6.1. TLS fingerprints can provide process and operating system insight, but it is often difficult to differentiate processes using system-provided TLS libraries with default settings. Incorporating destination information into the process inference system does help to disambiguate the candidate processes provided by our knowledge base, but this analysis is outside the scope of this paper. Finally, as we explain below, censorship circumvention software may be better served by emulating system libraries or common configurations of OpenSSL.

Exactly matching a passively observed session's TLS fingerprint to an entry in our knowledge base provides insight into the class of applications and libraries that have been previously observed using that fingerprint. However, an exact match does not guarantee that the process which initiated the session is represented among the list of possible processes. To approximate the likelihood of a process's, or a directly related process's, representation in a fingerprint entry given an exact match, we measured the percentage of destination IP address, port, and hostname tuples observed in the `Passive` dataset which also appeared in the `Endpoint` dataset. ~85% of the destination tuples in the `Passive` dataset were associated with processes in the `Endpoint` dataset. The remaining ~15% were either associated with fingerprints correlated with generic processes, e.g., Python3, or had destinations related to mobile devices or Linux-based infrastructure.

The Levenshtein distance is an intuitive method for identifying "close" fingerprints, especially considering TLS libraries often make minor adjustments to default cipher suites or extensions between minor version releases and more drastic changes between major version releases [8]. Some TLS libraries will also change their default parameters to better suit the platform on which they are running. To validate the Levenshtein distance's ability to provide useful information, we performed leave-one-out cross-validation on the top-100 fingerprints in the `Endpoint` dataset. For each cross-validation fold, we examined whether the process names associated with the target fingerprint were represented in the approximate fingerprint's knowledge base entry. ~73.5% of the target processes and ~80% of the target sessions appeared in the approximate fingerprint's process list. The processes that were absent in the approximate fingerprint's process list overwhelmingly shared the same or a similar TLS library, e.g., a MacOS Microsoft Office and Safari fingerprint sent the `session_ticket` extension, but the approximate fingerprint differed in only this extension and was only observed from com.apple.geod. While the Levenshtein distance is an appropriate method of last resort, exact and approximate fingerprint matching should still be applied with some caution as explain below and in Section 6.1.

For the operating system analysis, we consider fingerprints that are strong indicators of a general operating system platform (WinNT or MacOS) and strong indicators of an operating system major or minor version. Schannel and BoringSSL [5] are responsible for almost all strong operating system inferences and are the default TLS libraries for Windows and MacOS, respectively. To reduce the chance of spurious matches, we only consider the set of fingerprints

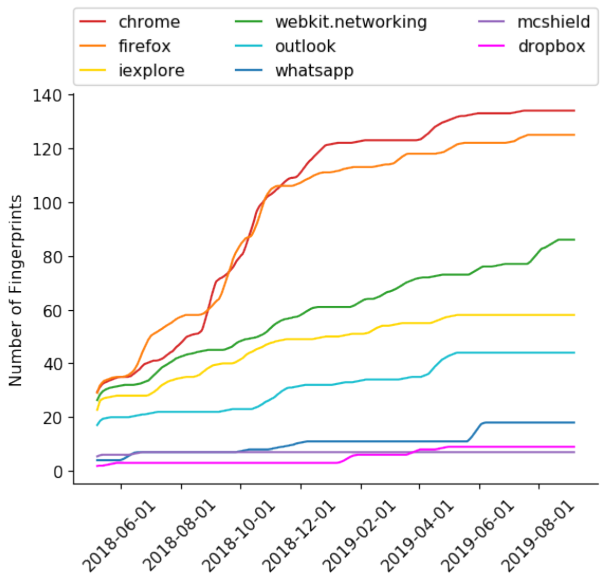


Figure 3: Cumulative fingerprint total for various process names.

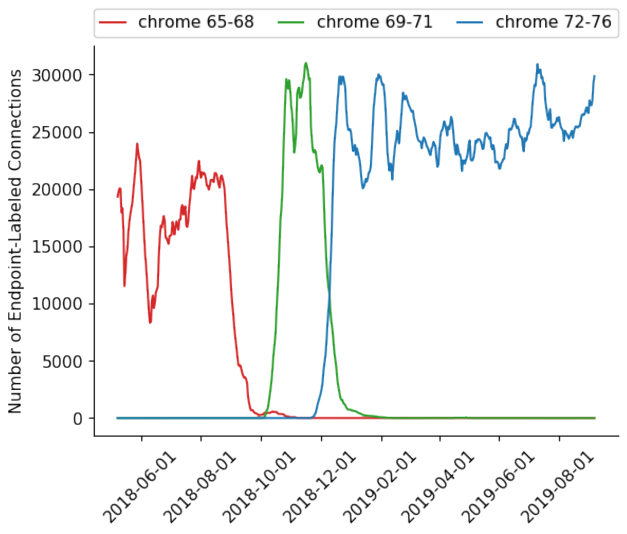


Figure 4: Succession of the most popular fingerprint for Chrome 65-69, 69-71, and 72-76.

from the Endpoint dataset that were generated from at least 100 unique hosts, of which there were 1,630 out of 3,510, which we will label as strong. 446 of the strong fingerprints were only observed on WinNT operating systems, 323 were only observed on specific major versions of Windows such as Windows 10 or Windows 7, and 190 were only observed on specific minor versions of Windows such as Windows 10.0.15063. 718 of the strong fingerprints were only observed on MacOS operating systems, 275 were only observed on specific major versions of MacOS such as 10.14, and 141 were only observed on specific minor versions of MacOS such as 10.14.6. The remaining 466 strong fingerprints were observed on multiple operating systems and were mostly related to NSS, BoringSSL, or OpenSSL.

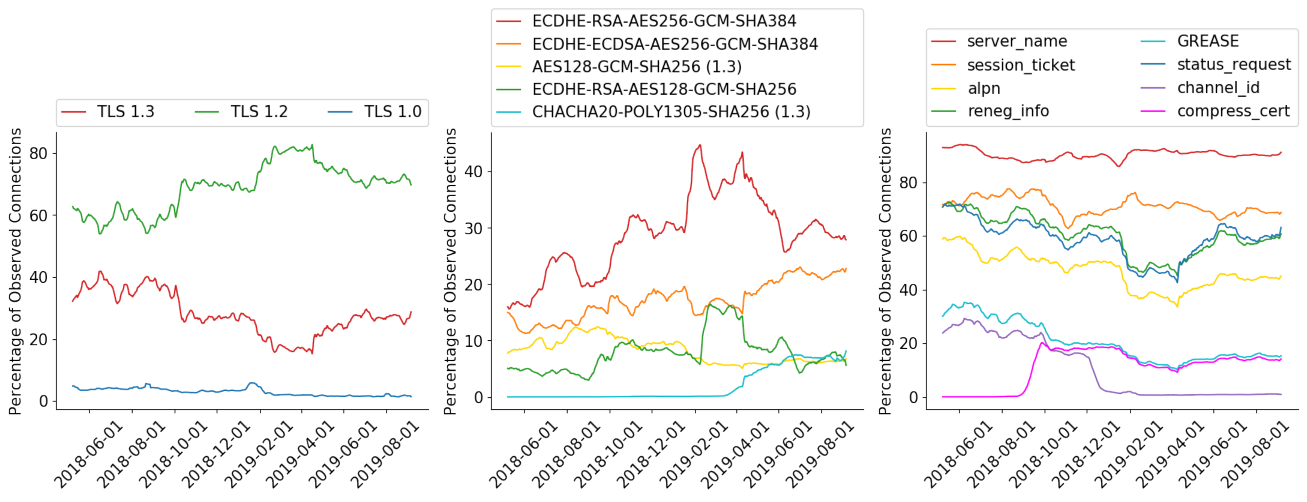
To look at the process visibility provided by a fingerprint, we analyzed the endpoint-labeled fingerprint knowledge base in order to understand the uniqueness of process-fingerprint mappings. Table 6 lists the mean and standard deviation for the number of unique process SHA-256 hashes assigned to each fingerprint (Process per Fingerprint) and the number of fingerprints used by each unique process SHA-256 hash (Fingerprints per Process). This table separates the results by application category, where "Processes per Fingerprint" considers all processes for a fingerprint if at least one process belongs to the target application category.

Processes in the browser category had the fewest processes per fingerprint with a mean of 22.19 and a standard deviation of 50.79. This is due to Firefox and Chrome using non-system TLS libraries, support for TLS 1.3 as explained in Section 5.2, and browsers in general using more distinctive TLS extensions such as compress_certificate, channel_id, and application_layer_protocol_negotiation offering support for HTTP/2 and HTTP/1.1. The use of system libraries and standard configurations of OpenSSL resulted in non-browser application categories having between 40 and 75 processes per fingerprint. When considering unique process names as opposed to SHA-256 hashes, browsers had 4.48 unique names per fingerprint with a standard deviation of 7.13. Other categories had mean values ranging from 8.79 for system applications to 16.13 for security applications.

When looking at the number of fingerprints generated by each process, we find that browsers produce the most unique fingerprints per SHA-256 hash. Browsers' TLS versions and offered cipher suites remain constant when considering popular fingerprints, with the diversity stemming from the inclusion or exclusion of TLS extensions such as padding and pre_shared_key. Other application categories produced between 1.53 and 2.67 fingerprints per SHA-256 hash, exhibiting less diversity in TLS extensions.

Figure 3 illustrates the cumulative number of fingerprints associated with each process name from May 2018 to September 2019. Only WinNT-based processes are considered, except for com.apple.webkit.networking which is specific to MacOS. Chrome and Firefox had the largest number of fingerprints during our data collection, rising significantly after the publication of TLS 1.3. Browsers using system libraries, iexplore and com.apple.webkit.networking, had the next largest number of fingerprints, where the separation after April 2019 is due to MacOS adopting TLS 1.3. Other processes typically introduced fingerprints less frequently.

There have been several examples of censors using TLS parameters to block Tor [44]. Many censorship circumvention applications mimic common browsers in order to evade TLS fingerprint-based detection, such as Tor/Firefox or Psiphon/Chrome. Unfortunately, these profiles become detectable once browsers deprecate or add new TLS parameters to their stable versions. Figure 4 shows the popularity of the most prevalent fingerprint for Chrome 68, Chrome 70, and Chrome 72, where each fingerprint additionally provided coverage for versions 65-68, 69-71, and 72-76, respectively. Similar patterns held for Firefox. Once a Chrome or Firefox fingerprint



(a) Version (b) Preferred Cipher Suite (c) Extensions

Figure 5: General TLS usage as observed in all edge network data.

begins to be replaced by an updated profile, it has nearly 0 observations after one month. Because major browsers have an infrastructure that supports the rapid deployment of updates, censorship circumvention software will need to similarly deploy updates or mimic more stable fingerprint distributions such as those generated by system-level libraries (e.g., Outlook) or OpenSSL (e.g., DropBox).

5 LONGITUDINAL STUDY

This section is devoted to a longitudinal study of desktop TLS usage with an emphasis on differences observed between categories of applications. The main thesis of this section is that visibility into application categories is essential in order to understand the security posture of the entire TLS ecosystem.

We start by highlighting some general trends observed in the combination of the Passive and Endpoint datasets such as the most popular offered TLS versions over the past year. We then use the Endpoint dataset to highlight the declining prevalence of browser-based TLS connections in favor of broader support from more application categories. Sections 5.2 and 5.3 examine the implications of TLS's adoption by more applications on the measurement community. Specifically, we highlight trends that can be overlooked or poorly explained in measurement studies without endpoint ground truth, but become clear with the addition of this data. Finally, we analyze the longevity of TLS fingerprints, and use an additional dataset collected from TLS scanning tools to explain the majority of short-lived fingerprints.

5.1 General TLS Trends

Figure 5 presents general TLS trends observed on the edge networks. Figure 5a shows the counter-intuitive decrease in popularity of TLS 1.3 over the past year despite its publication in August 2018. During our measurement study, SSL 3.0 and TLS 1.1 clients initiated less than 0.1% of the connections on any given

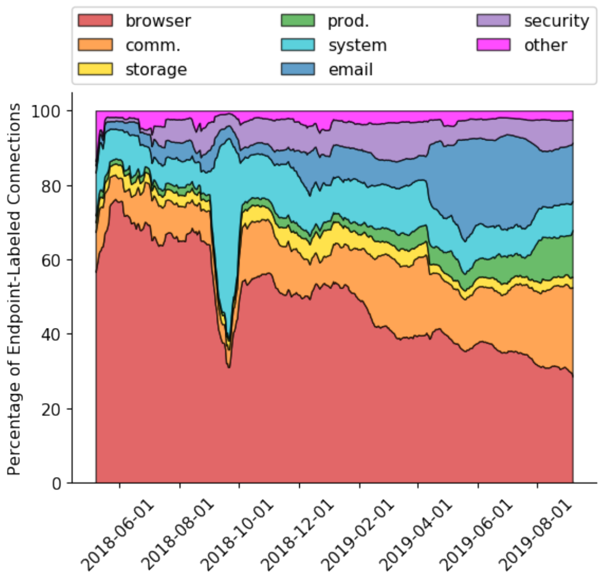


Figure 6: TLS usage by application category.

day, and were omitted from the figure. Figure 5b shows the prevalence of the four most client-preferred cipher suites, along with CHACHA20-POLY1305-SHA256 which significantly increased its preferred status during April 2019. ECDHE-RSA-AES256-GCM-SHA384's popularity more than doubled throughout the year as TLS 1.2 clients shifted to preferring a stronger cipher suite.

Figure 5c shows several trends in extension usage, e.g., server_name was offered in ∼90% of the connections throughout the year (or rather not offered in 10% of connections), Chrome's inclusion of compress_certificate, and the deprecation of channel_id by most TLS clients. The prevalence of GREASE aligns with the prevalence of Chrome in our dataset. Extensions such as

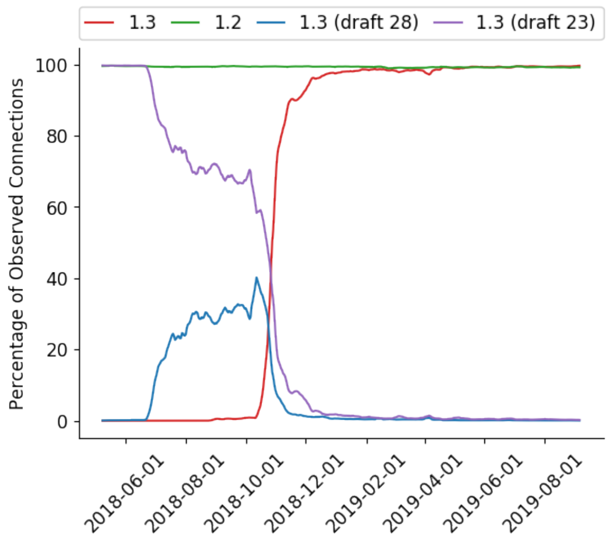


Figure 7: TLS 1.3 Supported Versions. TLS 1.1 and TLS 1.0 were offered with nearly the same frequency as TLS 1.2 throughout the study.

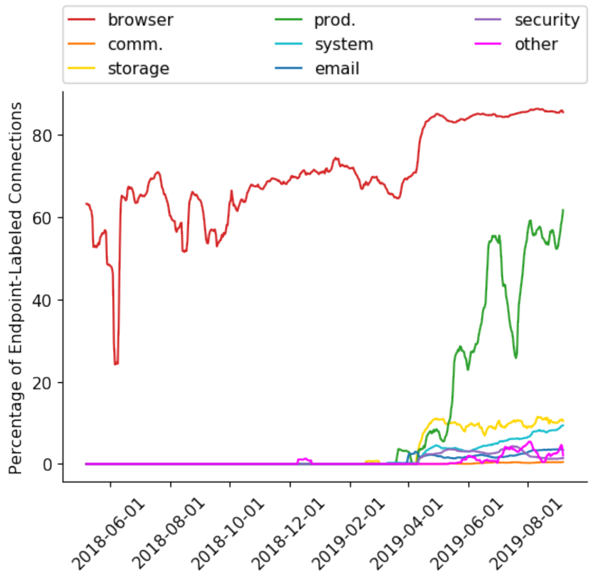


Figure 8: TLS 1.3 adoption within application categories.

supported_groups and ec_point_formats appear in almost all TLS connections; other extensions such as max_fragment_length and encrypted_server_name appeared in less than 0.1% of the total connections.

When not segmenting measurement studies by application, important trends can often be overlooked. With endpoint-labeled data that provides process-level visibility from 24,000+ hosts, we are able to identify and explain trends that in the past would be skewed towards browser behavior. As Figure 6 demonstrates, we have noticed a shift in the dominance of browser-initiated TLS connections, further motivating the need for large-scale measurement studies that incorporate this information. The brief increase in system initiated TLS connections in Figure 6 was due to MacOS 10.14 updates. In the remaining subsections, we use process ground truth to investigate TLS 1.3 adoption, RC4 usage, and fingerprint prevalence. Figures use a 14-day moving average to reduce artifacts due to weekly usage patterns. The measurements related to application categories are restricted to the set of TLS fingerprints derived from the endpoint-labeled data.

5.2 TLS 1.3

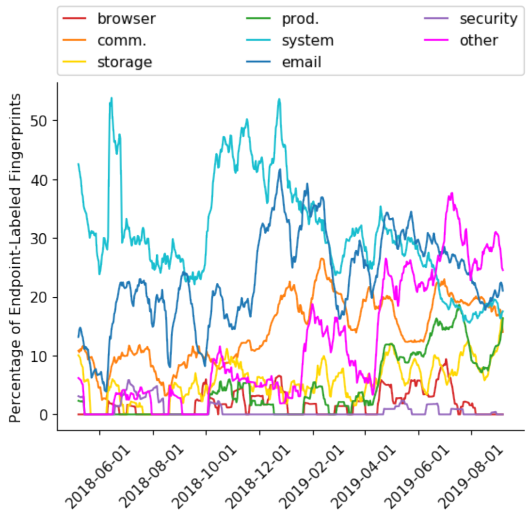
TLS 1.3 [40] added many security and performance oriented improvements. As the drafts progressed, clients supporting a specific draft implementation of the protocol would advertise this in the new supported_versions extension. We observed 22 unique values within this extension over the past year: SSL 3.0 to TLS 1.2, all draft versions between 18 and 28, two experimental draft implementations (7e01 and 7e02), and four experimental Facebook implementations (fb28, fb1a, fb17, and fb16). We also observed several TLS 1.2 fingerprints advertising the supported_versions extension, i.e., these fingerprints did not offer any TLS 1.3 cipher suites or advertise any other TLS 1.3 extensions such as key_share. The most common application advertising the supported_versions extension without 1.3 support was Terraform, an application to manage cloud infrastructure.

Shortly after TLS 1.3 was published, we observed a rapid transition from clients that supported draft versions to clients that supported the final release version. Figure 7 shows the TLS 1.3 official version replacing the two primary drafts, 23 and 28, within 3 months. This again speaks to the ability of Chrome and Firefox to quickly push out updates on a massive scale.

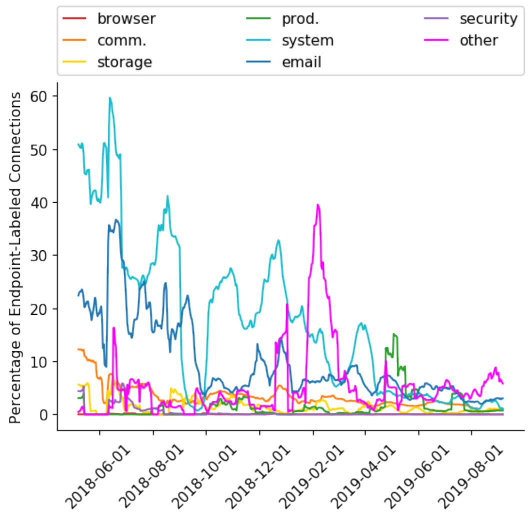
There were 1,851 TLS 1.3 fingerprints in the full knowledge base, derived from 2.2 billion connections (or ∼25% of connections). In the endpoint-labeled knowledge base, there were 693 TLS 1.3 fingerprints derived from 176 million connections (or ∼37% of endpoint-labeled connections), and these 693 fingerprints were associated with 2.17 billion connections in the full knowledge base.

Chromium-based and Firefox-based browsers accounted for over 85% of the TLS 1.3 connections in the endpoint-labeled dataset over the past year. Figure 8 shows the percentage of connections within each application category that supported TLS 1.3. Throughout the year, 60% to eventually 85% of browser-initiated connections offered some version of TLS 1.3, while other categories had near-zero support during the first half of our study.

There has recently been a significant increase in TLS 1.3 usage by other application categories. In March 2019, MacOS 10.14.4 made TLS 1.3 support the default. Figure 8 shows the impact of this release on our application measurements; specifically, we begin to see more diversity in TLS 1.3 usage during March and April of 2019. Beginning in May 2019, Microsoft Office applications leveraging MacOS's default TLS library contributes to the rise of productivity applications' TLS 1.3 usage. Because a large number of system and custom applications use the operating system's default TLS library, ensuring these libraries follow best practices and respond to new security developments as quickly as browsers is critically important.



(a) Fingerprints by Application Category



(b) Connections by Application Category

Figure 9: RC4 client-enabled TLS connections/fingerprints within each application category.

While the overall number of TLS 1.3 connections increased during the past year, we did observe a drop in TLS 1.3's relative prevalence as shown in Figure 5a. The positive message is that a larger and more diverse set of applications running on end hosts are choosing to encrypt their network communication with TLS 1.2 and are preferring strong cipher suites as shown in Figure 5b. On the other hand, this new wave of TLS-capable applications is lacking important security and privacy benefits introduced in 1.3 such as the deprecation of many weak parameters and certificate encryption.

5.3 RC4

In response to several high profile attacks against RC4 [12, 19], most browsers removed RC4 from their cipher suite offer vectors in May 2015 [34]. Other libraries and applications were slower to adapt; OpenSSL removed default RC4 support with version 1.1.0 in August 2016 [3], MacOS's Secure Transport in September 2016 [2], and Microsoft's Schannel in October 2017 [4]. Despite most major TLS libraries removing RC4 from their default configurations 1-2 years before we began our measurement study, 33% of the observed fingerprints and 9% of the observed connections offered RC4. Between November 2018 and September 2019, we consistently observed 100-200 new fingerprints, unrelated to scanning activity, each month offering RC4.

Figure 9 shows the percentage of fingerprints and connections offering RC4 cipher suites within each application category. Obsolete operating systems were the primary driver for RC4 usage. ~60% of the connections offering RC4 originated from MacOS < 10.12 and WinNT < 10.0.17134, and browsers using RC4 were limited to Internet Explorer and Safari on these obsolete operating systems.

The remaining set of RC4-enabled clients use obsolete TLS libraries on up-to-date operating systems. Mail on MacOS 10.13.6 offered four RC4 cipher suites, but they were the lowest priority. In some cases, applications would use the default version of OpenSSL installed on the operating system, e.g., we observed Microsoft OneNote using OpenSSL 1.0.2 on MacOS 10.14.6 offering RC4 as recently as September 2019. Python applications also contribute to the set of RC4-enabled connections by using obsolete version of OpenSSL, and more generally, we found that other programming languages such as Java and Ruby shared the trend of using obsolete TLS libraries. Viber is a more concerning case also observed in September 2019: it shares its TLS fingerprint across platforms, it only offers 3 cipher suites, none of which provide forward secrecy (this has also been observed on Android platforms [38]), and the two most preferred cipher suites use RC4.

Although our analysis and Figure 9a shows that the number of unique fingerprints offering RC4 remained relatively constant during our measurement study, Figure 9b shows that several application categories have made significant progress towards deprecating RC4. As more obsolete operating systems are upgraded, the system, productivity, and email categories will continue to improve. Python implementations caused the sudden spike in February 2019, which shows that there is still room to improve with respect to overall hygiene.

5.4 Fingerprint Prevalence and Longevity

Given Figure 1 and the fact that the top 10 fingerprints in our knowledge base account for 40% of the connections, it is clear that not all TLS fingerprints are created equal. The ten most prevalent fingerprints belong to either browsers, video conferencing tools, Python development applications, or security monitoring tools like osquery. Of this set, the browser fingerprints were the only ones to have a lifespan of less than 6 months.

We define a fingerprint's longevity as the difference between the first and last observation recorded in our knowledge base. For simplicity, this metric does not differentiate between fingerprints only observed on the first and last day and fingerprints that were also observed on each or most days within the interval, and in

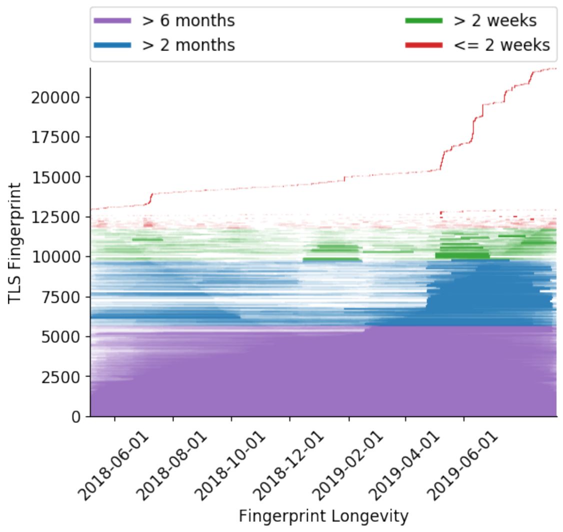


Figure 10: Fingerprints longevity as seen within the edge network collected data.

practice, this differentiation made little difference in our dataset. Figure 10 presents fingerprint longevity for all fingerprints collected from the monitored edge networks. Approximately one quarter of the fingerprints had a longevity of 6 months or greater. The subset of these fingerprints appearing in the endpoint-labeled dataset were associated with applications that used popular system libraries, tools like osquery and Dropbox, and browsers. The next quarter had a longevity between 2 weeks and 6 months and was predominantly composed of browsers and less popular applications in the endpoint-labeled dataset.

The final half of fingerprints in Figure 10 had a longevity of less than 2 weeks. This set included fingerprints associated with unpopular applications, fingerprints that were only first observed near the end of our measurement study, and a large set of fingerprints that were only observed on a single day. The observation that there tends to be a long tail of fingerprints only observed on a single day has been previously reported [34]. We add the following to the discussion of these singletons: while these fingerprints rarely showed up in our endpoint-labeled dataset, we were able to attribute nearly ~95% of the singleton fingerprints to TLS scanning behavior. We used the destination IP addresses, destination ports, and source IP tracking to make this determination. The remaining 5% may have been related to implementation errors or evasive software as discussed in the context of malware in Section 6.2.4.

To further investigate the TLS scanner issue, we installed SSLyze [24], CipherScan [37], and sslscan [39]. We ran these three scanners using their default configurations against the Alexa top-100 websites and extracted all fingerprints. This process generated 2,365 unique fingerprints. 295 of the scanner-generated fingerprints were in our knowledge base, 283 of which mapped to short-lived fingerprints previously marked as scanners. Given the size and evolution of the TLS `ClientHello` parameter space [31, 32], it is not unreasonable to assume that scanners could continue to generate a large number of rarely observed fingerprints.

6 MALWARE'S EVOLVING USE OF TLS

There has been previous work analyzing malware's general use of TLS [16]. In contrast to the prior work, we analyze malware specifically in the context of TLS fingerprinting. We first test a freely available malware TLS fingerprint blacklist to determine the feasibility of directly using fingerprinting to identify malware. We test against this additional dataset for reproducibility reasons due to our current inability to release our `Malware` dataset and the popularity of the freely available dataset.

We then analyze malware's abuse of censorship circumvention tools. To estimate the efficacy of malware's evasion, we define an acceptable false alarm rate to be less than 1 per one million TLS connections throughout this section. The results of Section 6.2 are taken from the `Malware` dataset as explained in Section 3.

6.1 TLS Fingerprinting for Malware Detection

JA3 is a method for creating TLS fingerprints that can be disseminated as JA3 hashes [14], and SSLBL generated a list of JA3 hashes for public consumption from packet captures generated from a malware analysis sandbox [45]. As a caveat to our analysis, the SSLBL website clearly states:

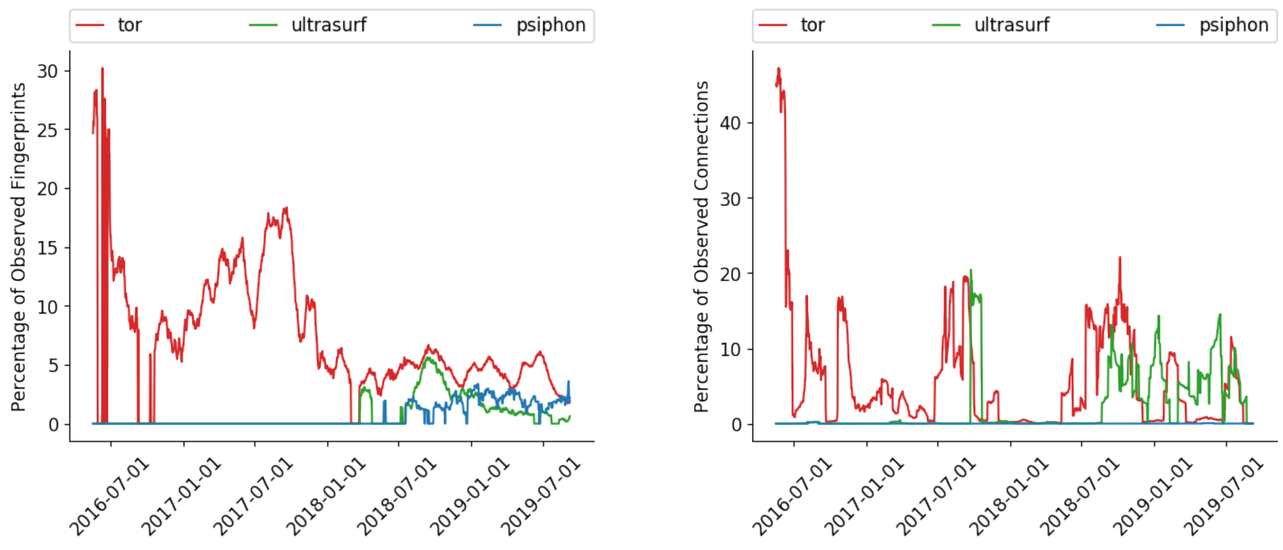
> *These fingerprints have not been tested against known good traffic yet and may cause a significant amount of FPs!*

As of this writing, there were 67 JA3 malware hashes on the SSLBL watch list. The JA3 fingerprint computation is irreversible, i.e., it destroys all information about the TLS parameters, making its direct analysis impossible. However, we worked around this limitation by computing the JA3 fingerprint for each entry in our knowledge base. As our fingerprint format has more data features than JA3, there were about 3,000 fewer distinct fingerprints with the latter format. We were then able to match 64 of the 67 SSLBL JA3 hashes to entries in our knowledge base but were unable to get any information of the remaining three. 55 of the matched hashes were fingerprints strongly associated with benign applications, of which 30 would generate false alarms exceeding the 1 per one million false alarm threshold on the networks we observed. We were only able to confirm 9 hashes as being strong indicators of malware relative to our knowledge base.

The results from Section 4.2 highlight the number of unique processes associated with a single fingerprint, and these results hold for malware. While directly applying fingerprint-based detection to the malware domain may not be an effective solution, it does offer valuable context with respect to underlying TLS libraries or applications that can be further combined with additional indicators such as destination information. Finally, if the TLS fingerprint is reversible, malware's abuse of evasive technology becomes more clear when analyzing many connections as explained in Section 6.2.4

6.2 Malware's Abuse of Censorship Circumvention Tools

In this section, we examine malware's abuse of three popular censorship circumvention tools and the general strategy of randomizing `ClientHello` parameters. In many cases, these tools attempt to



(a) Fingerprints **(b) Connections**

Figure 11: Malware's Abuse of Censorship Circumvention Tools.

mimic popular TLS profiles, which has previously been shown to be difficult to achieve in practice [29]. Figure 11 presents an overview of malware's abuse of Tor, Psiphon, and UltraSurf both in terms of percentage of fingerprints and connections observed in the malware analysis sandbox. It is important to note that malware has more operational constraints than end users, and this section should not be used as an indication of these tools' general behavior in the wild.

6.2.1 Tor. Tor anonymously transmits application layer data using a combination of TLS, its own encryption protocol, and an overlay network [23]. Tor is open source, and is widely used for beneficent and benign purposes, e.g., allowing users to evade government censorship, but this technology has also been misappropriated by malware to evade detection. Similar to observations in Section 5, while Figure 11 indicates malware's usage of Tor is decreasing, this is only true in a relative sense. The absolute usage of Tor has steadily increased in our malicious knowledge base, but this is overshadowed by more of the malware ecosystem adopting TLS.

16 fingerprints in our knowledge base were generated by malware's abuse of Tor. All samples used OpenSSL, and no fingerprints collided with a top-1,000 fingerprint. Blocking two of the Tor fingerprints would have generated false positives that exceeded the 1 per one million false alarm threshold, but only because the data collected on the edge network included Tor connections. 13 fingerprints were never observed in the edge network data.

6.2.2 UltraSurf. UltraSurf is a closed source program and was created by Chinese dissidents in order to evade the Great Firewall of China [10]. UltraSurf clients proxy connections directly to the UltraReach network. Malware's use of UltraSurf is more recent than that of Tor but has seen more use over the past 12 months. We identified two classes of fingerprints that were similar to Chrome and OpenSSL fingerprints.

Malware's use of UltraSurf generated 23 unique fingerprints. Blocking six of the UltraSurf fingerprints would have generated false positives that exceeded the 1 per one million false alarm threshold. These fingerprints were all similar to Chrome, but never aligned to the most popular Chrome fingerprints, e.g., blocking these fingerprints would have resulted in ~3-7 false alarms per one million connections. 11 UltraSurf fingerprints were never observed in the edge network data. These fingerprints would have matched known Chrome fingerprints but had unseen orderings of TLS extensions or omitted the `server_name` extension.

6.2.3 Psiphon. Psiphon is an open source censorship circumvention tool that chooses between multiple protocols to proxy connections [9]. Our analysis will be limited to malware's abuse of Psiphon's TLS functionality. Psiphon has seen comparatively little adoption among malware families relative to Tor and UltraSurf in our datasets, but Psiphon's evasion strategies made it one of the most prolific generators of new TLS fingerprints during the past year as shown in Figure 11a. Psiphon randomly selects a TLS `ClientHello` profile from a list of popular clients, e.g., recent versions of Firefox or Chrome, or generates a randomized profile.

Malicious Psiphon use was responsible for a total of 104 fingerprints. Blocking nine of the Psiphon fingerprints would have generated false positives that exceeded the 1 per one million false alarm threshold, and malware's abuse of Psiphon is the only successful instance of malware abusing a censorship circumvention tool that mimicked top-1,000 fingerprints. Malware's abuse of Psiphon failed when choosing to omit the `server_name` extension, which allowed an otherwise common fingerprint to be blockable. Psiphon also generated iOS fingerprints that were never observed on WinNT platforms. Using TCP fingerprints to identify operating systems in addition to TLS fingerprints would enable the detection of Psiphon on WinNT platforms.

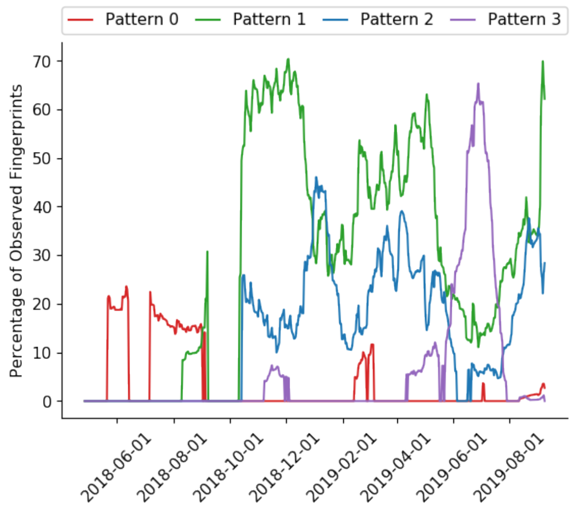


Figure 12: Malware's use of randomized cipher suite offer vectors. These implementations can currently be identified by stability in the advertised extensions and destination information.

Beginning on June 17th, 2018, we began observing randomized TLS profiles generated from malware that leveraged Psiphon. 76 Psiphon fingerprints utilized a profile that randomized the version, cipher suite inclusion and offer order, and extension inclusion, type order, and data. This trend began approximately one month after Psiphon updated its uTLS [11] support on May 16th, 2018. To avoid alarming on the introduction of legitimate fingerprints, retrospective detection by only considering an additional hour of traffic would be enough to block randomized fingerprints without exceeding the 1 per one million false alarm threshold.

6.2.4 Randomized Cipher Suite Offer Vectors. 3,974 out of the 5,070 fingerprints generated from the malware analysis sandbox were only observed in a single connection. 2,907 of the singleton fingerprints were associated with the Tofsee malware family [21] and shared one of nine static extension profiles belonging to versions of OpenSSL 1.0.x.

Beginning on June 5th, 2018, we observed a large increase in the number of TLS fingerprints generated from the malware analysis sandbox data. Figure 12 shows the prevalence of singleton fingerprints matching one of the top-4 static extension profiles. The destination hostnames were also static across all connections for a given extension pattern. Identifying and clustering the static components of the singleton fingerprint strings and associated destination information provides a means of identifying malicious use of randomized TLS `ClientHello` parameters.

7 DISCUSSION

7.1 Dataset Bias

The full set of edge network data was collected from enterprise campuses in 5 geographically distinct countries, which provides a global view of enterprise TLS usage. The 5 sites all belonged to a single company, which may have introduced biases by weighting the analysis towards applications preinstalled on managed endpoints, but users were allowed to freely install custom applications. The endpoint-labeled data was primarily collected from managed endpoints running WinNT or MacOS operating systems, and we lacked a significant number of labels from mobile and Linux platforms. There exists previous work complementing our lack of mobile visibility [38].

7.2 Future Directions

The fingerprint definition in our knowledge base could be improved by including both inner and outer TLS versions, compression methods, and a value that indicates if the `client_random` includes a timestamp. These omissions were due to deficiencies in the data collection tools. We are able to easily extend the fingerprint definition to accommodate additional extension data fields as they are introduced into the ecosystem.

The underlying data fusion process is not TLS specific, i.e., it generates process-labeled network data for all protocols. Extending the current analysis to DTLS [41], SSH [46], and QUIC [33] would all be interesting directions.

8 CONCLUSIONS

By studying TLS use across all application categories, we identified some important trends, including a temporary decrease in the fraction of TLS sessions using 1.3 caused by the adoption of (earlier versions of) TLS by non-browser application categories. This is partly good news in that earlier versions of TLS are still far more secure than the absence of that protocol, but there is a clear need for applications in these categories to modernize their use of TLS.

We leveraged detailed information obtained from a fraction of the endpoints on a network to make measurement inferences that covered the majority of its TLS sessions. Our use of endpoint knowledge enabled us to identify the prevalence and longevity of fingerprints, to illustrate the succession of fingerprints across application versions, to refine the analysis of malware fingerprints, and to analyze malware's abuse of censorship evasion technologies. This knowledge was produced by a novel automated system that continuously generates fingerprints with endpoint context from ongoing collection and fusion of data from endpoint and network observations of the same session. The number of fingerprints, and the complexity of their relationship to processes, suggest that this type of automation will be valuable going forward. We actively maintain an open source knowledge base that, at 4,500+ fingerprints and counting, is both the largest and most informative ever published [36].

ACKNOWLEDGMENTS

We thank Keith Schomburg for his support in developing and maintaining the data fusion pipeline, Brian Long and Cisco CSIRT for maintaining and providing access to the data capture infrastructure, and Bill Hudson and the Cisco Firefly team for their Joy development efforts. We also thank and acknowledge Ed Paradise and the Cisco Technology Fund for their ongoing support of this work. Finally, we thank Johanna Amann, Chris Wood, and the anonymous reviewers for their detailed comments and highly constructive criticisms.

REFERENCES

[1] 2012. SSL Fingerprinting for p0f. https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f/.

[2] 2018. macOS Security: Overview for IT. https://www.apple.com/business/resources/docs/macOS_Security_Overview.pdf.

[3] 2018. OpenSSL 1.1.0 Series Release Notes. https://www.openssl.org/news/openssl-1.1.0-notes.html.

[4] 2018. TLS Cipher Suites in Windows 10 v1703. https://docs.microsoft.com/en-us/windows/desktop/secauthn/tls-cipher-suites-in-windows-10-v1709.

[5] 2019. Apple Developer: Network Framework Documentation. https://developer.apple.com/documentation/network?language=objc.

[6] 2019. BrowserStack. https://www.browserstack.com/.

[7] 2019. Cisco AnyConnect Secure Mobility Client. http://www.cisco.com/go/anyconnect.

[8] 2019. OpenSSL Changelog. https://www.openssl.org/news/changelog.html.

[9] 2019. Psiphon. https://www.psiphon3.com.

[10] 2019. Ultrasurf. https://ultrasurf.us.

[11] 2019. uTLS. https://github.com/refraction-networking/utls.

[12] Nadhem AlFardan, Daniel J Bernstein, Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. 2013. On the Security of RC4 in TLS. In *USENIX Security Symposium.* 305–320.

[13] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. 2016. Post-quantum Key Exchange-A New Hope. In *USENIX Security Symposium.* 327–343.

[14] John B. Althouse, Jeff Atkinson, and Josh Atkins. 2017. JA3. https://github.com/salesforce/ja3.

[15] Bernhard Amann, Matthias Vallentin, Seth Hall, and Robin Sommer. 2012. Extracting Certificates from Live Traffic: A Near Real-Time SSL Notary Service. *Technical Report TR-12-014* (2012).

[16] Blake Anderson, Subharthi Paul, and David McGrew. 2018. Deciphering Malware's Use of TLS (without Decryption). *Journal of Computer Virology and Hacking Techniques* 14, 3 (2018), 195–211.

[17] David Benjamin. 2019. Applying GREASE to TLS Extensibility. Internet-Draft (Informational). https://tools.ietf.org/html/draft-ietf-tls-grease-04.

[18] Hanno Böck, Juraj Somorovsky, and Craig Young. 2018. Return of Bleichenbacher's Oracle Threat (ROBOT). In *USENIX Security Symposium.* 817–849.

[19] Remi Bricout, Sean Murphy, Kenneth G Paterson, and Thyla Van der Merwe. 2018. Analysing and exploiting the Mantin biases in RC4. *Designs, Codes and Cryptography* 86, 4, 743–770.

[20] Lee Brotherston. 2015. FingerprinTLS. https://github.com/synackpse/tls-fingerprinting.

[21] Edmund Brumaghin. 2016. Want Tofsee My Pictures? A Botnet Gets Aggressive. https://blog.talosintelligence.com/2016/09/tofsee-spam.html.

[22] Tim Dierks and Eric Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard). http://www.ietf.org/rfc/rfc5246.txt.

[23] Roger Dingledine and Nick Mathewson. 2017. Tor Protocol Specification. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt.

[24] Alban Diquet. 2019. SSLyze. https://github.com/nabla-c0d3/sslyze.

[25] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception. In *Network and Distributed System Security Symposium (NDSS).*

[26] Stephan Friedl, Andrei Popov, Adam Langley, and Emile Stephan. 2014. Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension. RFC 7301 (Proposed Standard). http://www.ietf.org/rfc/rfc7301.txt.

[27] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention. In *Network and Distributed System Security Symposium (NDSS).*

[28] Ralph Holz, Johanna Amann, Olivier Mehani, Matthias Wachs, and Mohamed Ali Kaafar. 2016. TLS in the Wild: An Internet-wide Analysis of TLS-based Protocols for Electronic Communication. In *Network and Distributed System Security Symposium (NDSS).*

[29] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. 2013. The Parrot is Dead: Observing Unobservable Network Communications. In *IEEE Symposium on Security and Privacy (S&P).* 65–79.

[30] Martin Husák, Milan Cermák, Tomá Jirsík, and Pavel Celeda. 2015. Network-Based HTTPS Client Identification using SSL/TLS Fingerprinting. In *Availability, Reliability and Security (ARES).* 389–396.

[31] IANA. 2019. Transport Layer Security (TLS) Extensions. https://www.iana.org/assignments/tls-extensiontype-values/.

[32] IANA. 2019. Transport Layer Security (TLS) Parameters. https://www.iana.org/assignments/tls-parameters/.

[33] Jana Iyengar and Martin Thomson. 2019. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet Draft. https://tools.ietf.org/html/draft-ietf-quic-transport-23.

[34] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G. Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. 2018. Coming of Age: A Longitudinal Study of TLS Deployment. In *ACM SIGCOMM Internet Measurement Conference (IMC).* 415–428.

[35] David McGrew, Blake Anderson, Bill Hudson, and Philip Perricone. 2017. Joy. https://github.com/cisco/joy.

[36] David McGrew, Brandon Enright, Blake Anderson, and Shekhar Acharya. 2019. Mercury: Fast TLS, TCP, and IP Fingerprinting. https://github.com/cisco/mercury.

[37] Mozilla. 2018. CipherScan. https://github.com/mozilla/cipherscan.

[38] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. 2017. Studying TLS Usage in Android Apps. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT).* 350–362.

[39] ioerror rbsec. 2019. sslscan. https://github.com/rbsec/sslscan.

[40] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard). http://www.ietf.org/rfc/rfc8446.txt.

[41] Eric Rescorla and Nagendra Modadugu. 2012. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard). http://www.ietf.org/rfc/rfc6347.txt.

[42] Ivan Ristic. 2009. HTTP Client Fingerprinting using SSL Handshake Analysis. https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html.

[43] Ivan Ristić. 2012. sslhaf. https://github.com/ssllabs/sslhaf.

[44] runa. 2012. UAE uses DPI to block Tor. https://trac.torproject.org/projects/tor/ticket/6246.

[45] SSLBL. 2019. SSL Blacklist: JA3 Fingerprints. https://sslbl.abuse.ch/ja3-fingerprints/.

[46] Tatu Ylonen and Chris Lonvick. 2006. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard). 4253 (2006). http://www.ietf.org/rfc/rfc4253.txt.

A ETHICS

When collecting and building our datasets, we followed all institutional procedures and obtained the appropriate authorizations. This included signing institutional documents declaring that we would not violate the institution's Code of Ethics. Specifically, these guidelines state that you will minimize personally identifiable information, maintain the confidentiality of all raw and processed data, receive written consent from your direct management chain before releasing any data, and pledge to not follow any practices that could be deemed discriminatory.

We leveraged the institution's existing endpoint and network monitoring infrastructure at all sites, which forwarded the raw data to an environment that has strict access control policies. The post-processed databases were stripped of all user identifiers such as source IP addresses, MAC addresses, and endpoint agent identifiers.

We carefully considered the level of detail regarding the amount of information to present with respect to the malicious data. We did not open source any fingerprints generated by the malware analysis sandbox in order to respect the data owner's privacy and to limit the amount of information disclosed to the malware authors.