Active TLS Stack Fingerprinting: Characterizing TLS Server Deployments at Scale

Markus Sosnowski*, Johannes Zirngibl*, Patrick Sattler*, Georg Carle*,
Claas Grohnfeldt†, Michele Russo†, and Daniele Sgandurra†
*Chair of Network Architectures and Services, Technical University of Munich, Germany
{sosnowski, zirngibl, sattler, carle}@net.in.tum.de
†AI4Sec, Huawei Technologies Munich, Germany
{claas.grohnfeldt, michele.russo1, daniele.sgandurra}@huawei.com

***Abstract*—Active measurements can be used to collect server characteristics on a large scale. This kind of metadata can help discovering hidden relations and commonalities among server deployments offering new possibilities to cluster and classify them. As an example, identifying a previously-unknown cybercriminal infrastructures can be a valuable source for cyber-threat intelligence. We propose herein an active measurement-based methodology for acquiring Transport Layer Security (TLS) metadata from servers and leverage it for their fingerprinting. Our fingerprints capture the characteristic behavior of the TLS stack primarily caused by the implementation, configuration, and hardware support of the underlying server. Using an empirical optimization strategy that maximizes information gain from every handshake to minimize measurement costs, we generated 10 general-purpose Client Hellos used as scanning probes to create a large database of TLS configurations used for classifying servers. We fingerprinted 28 million servers from the Alexa and Majestic toplists and two Command and Control (C2) blocklists over a period of 30 weeks with weekly snapshots as foundation for two long-term case studies: classification of Content Delivery Network and C2 servers. The proposed methodology shows a precision of more than 99 % and enables a stable identification of new servers over time. This study describes a new opportunity for active measurements to provide valuable insights into the Internet that can be used in security-relevant use cases.**

***Index Terms*—Active Scanning, TLS, Fingerprinting, Server Classification, Command and Control Servers**

I. Introduction

The Internet is characterized by a high level of complexity and heterogeneity. The arising metadata can enable novel and promising data mining use cases. One way to handle the complex data is to represent servers in a simple form, that is, by devising server fingerprints that succinctly represent their main characteristics, such as their implementation or configuration options. The Transport Layer Security (TLS) is currently the *de facto* standard for encrypted communication on the Internet [1] and has grown to a complex ecosystem due to continuous development and required backward compatibility [2]. Due to this, the protocol inherently provides a variety of meta-information related to client and server capabilities that are exchanged during the initial TLS handshake and that can be used to characterize a server. In previous work, these meta-data have been exploited by multiple passive approaches [3–5]; whereas, in this work we advocate the usage of active measurements that allow engaging with any responsive server on a large scale, considering encrypted data, and building a comprehensive data set from a single vantage point.

978-3-903176-47-8 ©2022 IFIP

Fingerprinting TLS servers can help to understand, model, and secure the Internet. If TLS fingerprints are able to indicate a level of trust of an infrastructure, they can be valuable cyber-threat intelligence, especially because the TLS is increasingly utilized by cybercriminals [6]. Possible use-cases are: *(i)* Intrusion Detection Systems fingerprint servers seen in network flows on-demand and compare results with known malicious fingerprints; *(ii)* security researchers use fingerprints from Internet-wide measurements to find unknown threats; or *(iii)* regularly monitoring own servers where deviations from a fingerprints baseline can indicate an unintended software change or a malware infection. A need for this kind of information can be derived from the fact that Internet scanning companies like *censys.io* have started to incorporate JARM [7] into their portfolio (according to their host data definition [8]). JARM is an open-source TLS server fingerprinting tool that uses similar data to this work and has just recently emerged.

However, no long-term systematic study has yet been conducted to show the applicability and the performance of detection use cases with actively collected TLS fingerprints nor has the effectiveness of their collection been analyzed.

In this work, we will investigate *(i)* how to construct a similarity relation among TLS server deployments, *(ii)* how effective scanning configurations can be found while minimizing the measurement costs, and *(iii)* how applications built with these fingerprints perform on a large scale.

We propose Active TLS Stack Fingerprinting as an effective method for Internet measurements and provide the following contributions:

- *(i)* reasoned selection of TLS handshake features for fingerprinting the TLS stack and their encoding in an extendable and shareable format;
- *(ii)* methodology for finding or tailoring effective TLS Client Hellos (CHs) for fingerprinting use cases and 10 general-purpose CHs that maximize complementary information extraction from servers;
- *(iii)* long running measurement study covering seven months to validate the methodology, show improvements to the related work JARM, and demonstrate the potential of

TLS fingerprinting based on two case studies to detect Content Delivery Network (CDN) and Command and Control (C2) servers; and

(iv) published data and open-source scanner to enable reproducible results and to support the community.[1]

II. Related Work

The large amount of metadata from TLS handshakes has been used in multiple passive traffic classification and fingerprinting related works [3–5]. In the context of the Transmission Control Protocol (TCP), fingerprinting with active scans has been successfully used by Refs. [9, 10] and [11] to detect the Operating System (OS) on a remote server. Similar to our CH selection, Greenwald *et al.* [9] used the Entropy from the information theory as metric to minimize the number of probes needed for classification.

To the best of our knowledge, the only related work we can directly compare ourselves to is the JARM tool developed by Althouse *et al.* [7]. It is a popular open-source tool for TLS server fingerprinting. Compared to this work, our tool differs in the selection of CHs and the extracted features from the TLS handshake. They use 10 custom-defined CHs for fingerprinting that "have been specially crafted to pull out unique responses in TLS servers" [7]. In contrast to this work, they do not complete the TLS handshake, only use unencrypted data, and do not consider TLS alert nor extension data besides the Application Layer Protocol Negotiation (ALPN) protocol. We will show in section V-E that JARM can be used for C2 server detection; however, the effectiveness can be improved with the data suggested by this paper.

Tools like `testssl.sh` [12] can collect fingerprintable information about TLS servers, but need a very large number of requests to collect this information. We observed `testssl.sh` to make $100-200$ requests to the same server. This makes it time expensive and ethically questionable to conduct Internet wide scans. Additionally, their focus on the configurable part of the TLS on servers (*e.g.*, supported cipher suites) results in neglecting fingerprintable implementation specific features like the extension order.

Chung *et al.* [13] investigated the usage of the Online Certificate Status Protocol (OCSP) stapling from different webservers and observed Nginx servers, which did not return OCSP responses to the first client connecting, appending the information only to consecutive requests. These implementations did not pre-fetch the information nor wait until the Certificate Authority (CA) returned the necessary OCSP response they could forward to the client. This means, from the clients point of view, that the presence of the OCSP stapling response is non-deterministic. This aligns with our observations of the non-deterministic presence of the Status Request extension because this TLS extension is currently only used to announce an OCSP stapling [14] response in the handshake. Gigis *et al.* [15] investigated Hypergiants, including CDNs over a period of seven years. They showed the increasing role of servers deployed in Autonomous Systems (ASs) not managed by the CDN to influence and localize CDN traffic to the user. Their results align with ours because we were also able to find server deployments outside the networks managed by the CDN and they found indicators of reverse proxies that influence the measurement results. Their results rely on servers correctly offering identification material in the certificates, while this work is more subtle and can identify deployments where this information is deliberately hidden (*e.g.*, to detect C2 servers).

[1] https://active-tls-fingerprinting.github.io

III. Methodology

The TLS protocol family "is the backbone of secure communication over the Internet" (as introduced in more detail by Holz *et al.* [16]) and currently the *de facto* standard for encrypted communication [1]. This work exploits the TLS to discover similarity relations among servers by fingerprinting their Server Behavior. A *Server Behavior* is defined as the totality of the capabilities, the interpretation (deviations from the standard or implementation of undefined parts, such as the order of extensions) and the configuration of the TLS on a server, which can influence the outcome of the TLS handshake. Our work is based on the assumption that every TLS server has a specific Server Behavior that depends on the implementation, capabilities, and configuration of the TLS library, hardware, and application utilizing the TLS. Identifying these behaviors allows to characterize server deployments either directly or in conjunction with additional data (*e.g.*, obtained on the Hypertext Transfer Protocol (HTTP) level).

TLS handshakes are initiated by clients, and servers only need to react to the initial handshake request (*e.g.*, a server chooses one cipher from a list that was previously proposed by the client). Therefore, the Server Behavior we want to fingerprint is not directly revealed by the server; only the reaction to different requests (*i.e.*, CHs) can be observed. Using multiple CH increases the acquirable knowledge and coverage of the Server Behavior. For each CH, we collect the TLS version, cipher, and TLS extension data from different types of TLS messages to construct the fingerprint. We only initiate handshakes with TLS versions 1.0 to 1.3 but also store a fingerprint if the server answers with an older version.

This work proposes a methodology for capturing a part of this Server Behavior by sending a fixed number of specifically crafted CHs to a server, extract features as string encoded information for each CH, as detailed in section III-A, and combine these features to a fingerprint according to section III-B.

A. Features Extracted from TLS Handshakes

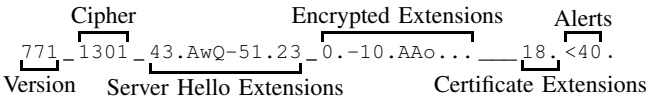
Given a CH, we extract features from a single handshake in a textually encoded format to be used for fingerprinting.

Features are the selected version, cipher suite, received alerts and extensions data. Extensions are extracted as an ordered list of key-value pairs each from the *Server Hello*, *Encrypted Extensions*, *Certificate Request*, *Hello Retry Request* and *Certificate* TLS messages. The value is only included for

TABLE I
TLS EXTENSIONS WHERE THE DATA CONTAINED IN THE EXTENSIONS ARE USED FOR ACTIVE TLS STACK FINGERPRINTING.

ID	Name	ID	Name
1	Max Fragment Length	19	Client Certificate Type
7	Client Authentication	20	Server Certificate Type
8	Server Authentication	24	Token Binding
9	Cert Type	27	Compress Certificate
10	Supported Groups	28	Record Size Limit
11	EC Point Formats	43	Supported Versions
13	Signature Algorithms	47	Certificate Authorities
15	Heartbeat	50	Signature Algorithms Cert
16	ALPN	51	Key Share (only selected group)

several hand-picked extensions (Table I) or else left empty. An example of a textually represented fingerprint is



In the subsequent paragraphs, we will discuss the reasons for the inclusion of each feature. The version, selected cipher suite, and used extensions directly depend on the capabilities and the configuration of the TLS library used by the server and are, therefore, part of the fingerprint. The content of each individual extension is more diverse, and its fingerprinting value depends on the actual extension. Information depending on the current TLS session, specific server, or TLS certificate are excluded. From an analysis of different TLS extensions [14, 17] and from the observations in our scans, we infer the content of the extensions listed in Table A.I as a feature. An exception to this schema is the Key Share extension, where we remove the session specific part and only keep the selected group used for the Diffie–Hellman key exchange. The fingerprints are defined in a format that can easily be adapted (*e.g.*, to include values specified in future extensions). Error handling is implementation specific; hence, the TLS alerts sent by the server are also included. We do not create a fingerprint if the error was caused by the TCP layer. The current approach cannot differentiate whether the error is part of the Server Behavior or if it was a nondeterministic failure of the TCP stack. We consider the order of extensions as an important and implementation-specific information that we include in our fingerprints. The presence of the Status Request extension was nondeterministic in our measurements (section V-B); therefore, we removed this extension from the fingerprints trading the information about the OCSP stapling support of a server for results consistency.

B. Fingerprinting with Multiple Requests

In this work, data from multiple server responses are combined to construct the TLS fingerprint of the Server Behavior because a single response was not precise enough to provide good results in our experiments.

While a single CH reveals only a potentially small request-dependent subset of the information about the target server, multiple request–response pairs allow the collection of complementary information and, thus, a more complete picture about the Server Behavior. Increasing the number of CHs is a trade–off between learned information and measurement costs. However, the benefit of sending multiple CHs decreases with every additional CHs one sends because of the limit to which extent a Server Behavior can influence the TLS handshake. Moreover, the number of CHs should be limited based on time, resources, and ethical factors. Hence, the input set CH of CHs is an optimizable parameter influencing the effectiveness of fingerprinting. Let $f(s, c)$ return the features from a server s given a specific CH c in the format described in Section III-A, then the server fingerprint is defined as

$$fp(s) = \bigcup_{c \in CH} (c, fp(s, c)).$$

The features obtained with a single CH are only comparable in the context of the same CH; hence, the CH used to generate each part of the fingerprint must be stored along the combined fingerprints. We never compared information obtained with different CHs because we do not know what combination of parameters in the CH has caused the particular response.

In conclusion, the number of requests and the design of different CHs are crucial parameters that can be optimized to maximize the amount of collectible information while minimizing measurement costs and respecting ethical aspects. This will be experimentally done in section IV.

C. Active Measurements under Ethical Considerations

We have used an active measurement pipeline based on established tools and by following basic ethical principles.

The pipeline takes a list of IP addresses, domains, and CHs as input. The domains are resolved according to the IPv4 and IPv6 addresses with MassDNS[2] and a local Unbound[3] server, resulting in a set of IP address and domain tuples we call targets. We fingerprint with multiple CHs; thus, the final input is a randomly ordered cross-product of the targets and the CHs. This list is fed to the TLS scanner based on an implementation of Amann *et al.* [18], a scanner designed for Internet wide usage. If a domain name is available for an IP address, we used it as the Server Name Indication (SNI). We designed a custom TLS library based on the Golang standard library that allows the definition of arbitrary CHs as input for each TLS connection and to extract required TLS handshake meta-data. Both scanner and library are open-sourced [19].

We reduce the impact on third parties by following the best practices, as described by Durumeric *et al.* [20]. Our work does not harm individuals or reveal private data as covered by Refs. [21] and [22] and focuses on publicly reachable services. We use rate limiting, maintain a blocklist, use dedicated scan servers with abuse contacts, informative rDNS entries, and websites that inform about our research, and provide contact information for further details or scan exclusion. Additionally, because we scan the same target with multiple requests, we

[2]https://github.com/blechschmidt/massdns
[3]https://www.nlnetlabs.nl/projects/unbound

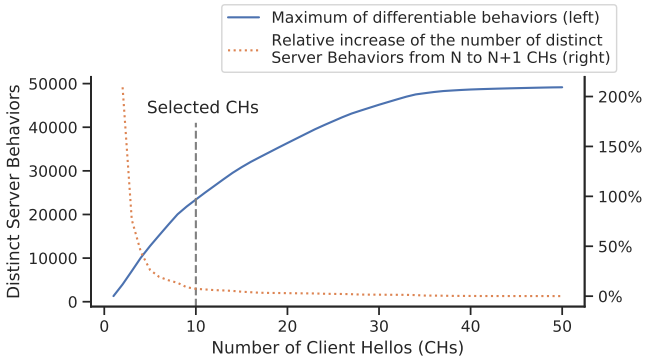


Fig. 1. Experimentally determining the trade-off to increase the number of CHs versus the ability to distinguish Server Behaviors (measured in distinct fingerprints). We selected a set of 10 CHs for our following analyses.

limit the interference by spreading the requests over a large time frame (*i.e.*, two days in the longitudinal study).

IV. Systematic Design of Client Hellos

The internal mechanism of TLS servers is a black-box for active scanners. Without knowledge about the implementation of every TLS server, it is impossible to find the best method for fingerprinting. However, more effective fingerprints can be developed by optimizing their distinctiveness. We propose herein an empiric design of CHs by analyzing a large pool of randomly generated candidates to find an optimal subset maximizing a given metric. We choose the total number of distinguishable servers as metric to find general-purpose CHs usable for a wide range of use-cases. If the use-case is known (*e.g.*, detecting CDN or C2 servers), a different strategy could be to minimize the necessary probes needed for a classification. We will later revisit this use-case driven design regarding C2 servers in section VI. We have open-sourced the general-purpose CHs and their generation code as part of our scanner [23].

The experiment is designed as follows: *(i)* randomly generate 5 000 CHs each from the feature space the used TLS scanner supports and the complete feature space as defined by IANA [24]; *(ii)* distribute a measurement with these CHs over the Alexa [25] and Majestic [26] toplists with a maximum of 13 CHs per server to gain a first impression of good-performing CHs; and *(iii)* select the best-performing CHs from the previous measurement and conduct a second measurement of the same targets and fingerprint each target with 50 CHs. We choose the prime-number 13 together with a round-robin algorithm to increase the variation of the different sets of CHs sent to a single server. The decision to scan with 50 CHs per server was a pure trade-off between development speed and data quality (the scan took more than 4 days).

Figure 1 shows the number of behaviors that can be distinguished with subsets of the 50 CHs. The sets were constructed by iteratively selecting CHs that increase the number of distinguishable Server Behaviors most. To remove the potential bias from nondeterministic TCP errors, we consider only servers for which every CH produced a fingerprint. We can see that every added CH enables to differentiate additional Server Behaviors; however, the relative increase of behaviors decreases the more CHs we use. We could not reach the theoretical upper limit of distinguishable behaviors. Based on this analysis, we selected 10 general-purpose CHs with a good performance in distinguishing Server Behaviors. We thought that this number was adequate for our use cases because according to Fig. 1 the relative increase of information was quite low when using more than 10 CHs, we can directly compare related work, and this number seemed to be acceptable for Internet scanners like `censys.io` already fingerprinting with JARM [8]. However, in section VI, we will illustrate that the number can be lower for specific use cases (*e.g.*, detecting C2 servers). We only manually adapted some cryptographic parameters of these CHs that were too CPU-expensive, such as the 512-bit version (`secp521r1` [27]) of the elliptic curve domain parameters for the precomputed TLS 1.3 Key Share. This curve would have more than doubled our scanning time.

Through the experiment described in this section, we gained a set of 10 general-purpose CHs that we will use in the following section to fingerprint servers on the Internet. They are a good trade-off between limiting the number of requests and the resulting impact on the scanned infrastructure and providing a high distinctiveness of the Server Behaviors.

V. Longitudinal Study of Toplists and Blocklists

To investigate the applicability of TLS fingerprinting on the Internet, we performed measurements of two toplists and two C2 blocklists over 7 months. The following sections analyze the stability of the fingerprints, apply the methodology to detect CDN and C2 servers, and compare to related work. The two case studies were selected to have one with a big sample size and where the ground truth can be verified and one where the value of the study is high, but the sample size is low.

A. Data

We scanned servers from the two toplists and two blocklists over a period of 30 weeks using 25 weekly snapshots starting on July 19, 2021. Five scans failed and the data for these weeks were skipped.

Table II presents the number of scanned servers. A *target* is the scanned combination of IP addresses, TCP port, and domain name. The targets were aggregated over a period of seven days from Alexa [25] and Majestic [26] to cover the weekend effect and the weekly patterns observed by Scheitle *et al.* [28]. The last 30 days were used for the SSLBL [29], while the current list was utilized for the Feodo Tracker [30]. We took a larger time frame for the SSLBL, because in addition to the IP addresses, we used the provided certificate hashes to reduce false positives. This list of around 4M weekly targets was taken as input to the scanning pipeline, as described in section III-C. The scanning probes are both the 10 CHs designed in section IV and the 10 CHs modeled after *JARM* [7] to compare both approaches. However, unless stated otherwise, the following analyses are only based on the 10 CHs designed

TABLE II
TOTAL NUMBER OF COLLECTED DATA SAMPLES OVER 30 WEEKS FOR THE LONGITUDINAL ANALYSES. ALSO SHOWS THE NUMBER OF DISTINCT TARGETS AND DOMAIN NAMES THE DATA COVERS ($M = 10^6$).

Source	Scanned			Successful		
	Total	Targets	Domains	Total	Targets	Domains
Alexa	80.88M	26.48M	11.69M	68.48M	22.21M	9.87M
Majestic	37.26M	4.72M	1.58M	31.23M	3.59M	1.36M
SSLBL	951	250		558	127	
Feodo	8.14k	1.06k		7.07k	883	
Total	103.77M	27.69M	12.01M	87.87M	23.12M	10.16M

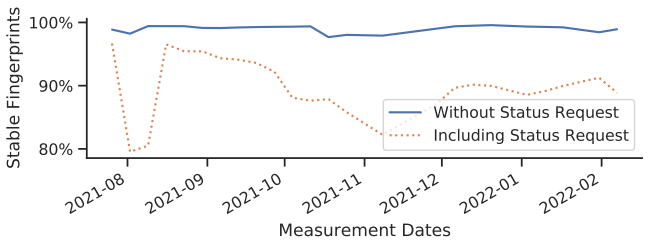


Fig. 2. Percentage of targets with the same TLS fingerprint on the $n-1$ and nth measurement in relation to the total targets fingerprinted on both weeks. The Status Request extensions are responsible for the most unstable fingerprints with drops under 90% mostly caused by Cloudflare.

for this study. Targets are only considered to be successfully fingerprinted if a fingerprint for each CH was collected. The total number of targets was less than the sum of each list due to an overlap between the lists.

B. Consistency of TLS Fingerprints

The fingerprints only provide value for identification purposes if they can be unambiguously assigned to a server, and this assignment does not change, in other words, is stable.

For each measurement, a large number of servers was already seen in the last measurement ($\approx 48\%$ each week); hence, their fingerprints can be compared over time. Figure 2 shows the relative number of targets remaining stable during each measurement. On average the targets remained stable 99% of the time. The stability drops on average to 90% if the Status Request extension is considered for fingerprinting. In these cases, the presence of the extension is nondeterministic. This is especially visible during stability drops under 90%, where only $\approx \frac{3}{4}$ of the TLS handshakes to Cloudflare contained a Status Request extension compared to the other weeks.

This analysis concludes that Status Request extensions should not be considered for obtaining useful fingerprints. However, they are, without the extension, a very stable and consistent feature to identify servers. The subsequent sections analyze how differences in fingerprints can be used to identify whole deployments of similar servers.

C. Case study: Detecting CDN server deployments

A core assumption about TLS fingerprinting is that it reveals groups of similar server deployments. We tested this assumption, by analyzing the fingerprints of four major CDNs.

TABLE III
SERVERS SEEN WITH A TLS FINGERPRINT FROM THE RESPECTIVE CDN. A STRONG CORRELATION BETWEEN BOTH CAN BE SEEN.

	Akamai	Alibaba	Cloudflare	Fastly
Targets with a TLS fingerprint attributable to a CDN				
Total	98 701	405	7 021 087	572 294
IP addresses	24 332	267	222 050	7 238
Targets observed from ASs owned by the CDN				
Total	97 022	403	6 991 829	559 860
IP addresses	23 564	265	214 436	1 752
Targets observed from different ASs, but have access to CDN content				
Total	664		2 493	
IP addresses	212		139	
ASs	20		27	

On the one hand, these are TLS-enabled servers deployed by a single actor on a large scale. On the other hand, ground truth can be used because it is possible to verify if a server is a part of the CDN. With this analysis, we found servers outside of the ASs operated by the CDN that served the CDN content.

The analyzed CDNs have in common that they use their own ASs to deploy servers and CDN caches. Hence, we can identify them through the AS, determined through the Border Gateway Protocol (BGP) dumps downloaded from Routeviews[4] and Pyasn[5]. The content served by a CDN is independent of the actual server or the IP address. Therefore, we call servers serving this content a CDN cache. The CDN decides on the criteria like the SNI which content should be returned. In the same manner, the proper TLS certificate for the requested domain is selected by the CDN. Hence, we can evaluate whether or not a server is a valid CDN cache with our TLS scanner. The server is verified as CDN cache if it successfully completes a TLS handshake for a domain we have manually observed to be cached by the CDN and, therefore, proves possession of a valid certificate and the respective private key.

This analysis focused on the CDNs of Cloudflare, Fastly, Akamai and Alibaba. The CDN fingerprints were collected based on the scanned IP addresses located within their respective AS. The HTTP Server header is used to enhance the mapping, as described by Gigis *et al.* [15]. Note that for Fastly, the AS was sufficient to detect their CDN servers. Table III lists an overview of the results showing a strong correlation between the TLS fingerprint and the CDNs. Almost all targets with a CDN fingerprint were located within a CDN AS. The rate was higher than 99% for Cloudflare and Fastly. In addition, 7% of the targets we falsely assigned to one of the CDNs turned out to be valid caches of the CDNs from Akamai and Cloudflare outside of their respective networks. We will discuss these targets more detailed in section VI.

A simple multi-label classifier can be built using these data, by identifying a CDN server purely from their TLS

[4]https://routeviews.org/
[5]https://pypi.org/project/pyasn/

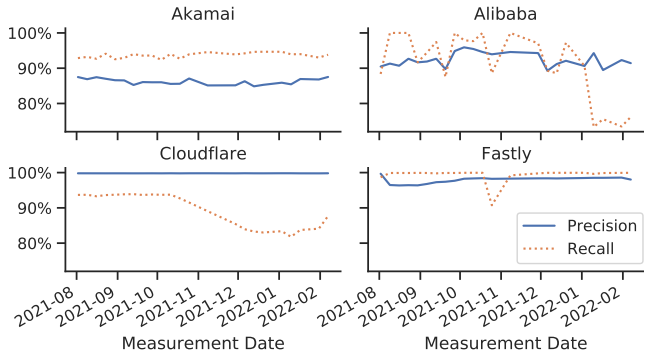


Fig. 3. Evaluation of a CDN server classifier. It was trained with TLS fingerprints from weeks $[1..n]$ and evaluated on week n.

fingerprints. The classifier was trained with the data of weeks $[1..n]$. It classifies a target in the $n+1$th week as a CDN server if the fingerprint was observed at least 10 times from the CDN, and the certificate was valid in these cases. This decision was always unambiguous because the fingerprints did not overlap. We evaluated the metrics precision and recall per CDN for each week. The precision and the recall are defined as $\frac{TP}{TP+FP}$ and $\frac{TP}{PP}$, respectively. The number of true observations PP is the sum of targets either *(i)* located within the CDN AS with the appropriate HTTP Server header values (according to Gigis *et al.* [15]) or *(ii)* validated as a CDN cache with the TLS scanner. Figure 3 illustrates the results. The precision was high with more than 85% for Alibaba and Akamai and more than 99% for Cloudflare and Fastly. The networks of the latter were much more uniform and easily clusterable compared to those of Alibaba and Akamai.

An interesting drop of the recall was observed for Fastly in October 2021. This drop was caused by a Server Behavior change, where new behaviors not covered by fingerprints from the previous measurement dates were observed. We particularly noticed fewer handshakes to complete with `http/1.1` as an ALPN. The new behavior was seen in the later measurements stabilizing the recall. Therefore, a potential fingerprint database must be regularly updated to provide the best performance. We introduced a threshold of 10 valid observations because we had a few cases, in which an Alibaba server responded with a fingerprint we could attribute in most other cases to Tengine webservers (inferred from the HTTP header). This webserver is not only deployed by Alibaba; thus, it would have produced many false positives. Not considering these fingerprints caused the recall drop for Alibaba in 2022. Additionally, Alibaba has a more diverse cloud portfolio on offer compared to the other examined companies that could be the reason for the fluctuations in the precision and recall.

Our fingerprints were able to detect minor differences among the deployments of the same CDN. This means, sometimes, the approach was too specific for the general use case to detect just the CDN. We mitigated this problem by mapping multiple fingerprints to each CDN covering all of these variations. To detect Akamai, Alibaba, Cloudflare, and Fastly we have used 86, 1, 801, and 87 fingerprints, respectively.

TABLE IV
FINGERPRINTING RESULTS FOR THE C2 SERVERS. COMBINING OUR FINGERPRINT (FP) WITH HTTP DATA RESULTS IN MORE FPS AND DISTINCT TARGETS (TAR.) UNIQUE TO A C2 LABEL.

C2 Label	Total		Unique		Unique (+HTTP)[1]		New Obs.	
	Tar.	FPs	Tar.	FPs	Tar.	FPs	Total	Known FP
Dridex	311	9	1	2	137	4	193	193
TrickBot	276	28	31	8	106	16	192	175
QakBot	127	6	0	0	122	3	86	72
BazarLoader	116	16	27	4	28	5	87	40
Emotet	92	3	1	1	1	1	83	62
Ransomware	12	2	0	0	0	0	11	10
CobaltStrike	11	9	1	6	5	7	4	4
AsyncRAT	11	3	0	0	0	0	9	7
Other	28	18	15	10	16	13	13	6

[1] Combining TLS fingerprints with the HTTP Server header.

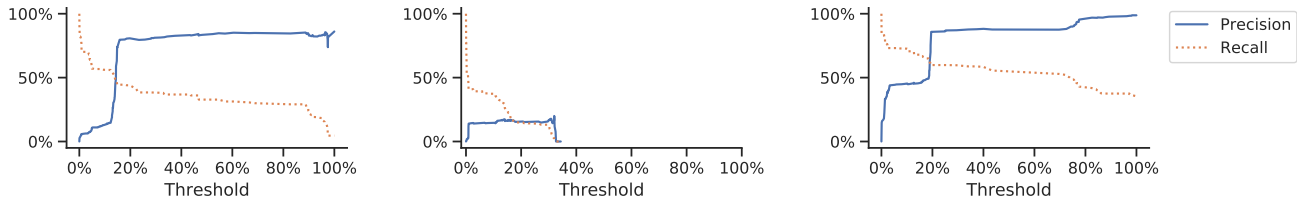
In summary, with TLS fingerprinting, large CDN deployments can be identified because they share a common TLS behavior. The precision was above 99% for some CDNs, and we have found several CDN caches in unexpected ASs. After showing that the approach works with major known deployments, we will now apply it to a much smaller sample size identifying potentially malicious C2 servers.

D. Case study: Identifying C2 servers

Aside from identifying CDN deployments, TLS fingerprinting can also be used to identify and track potentially malicious targets like C2 servers.

Blocklists containing C2 servers are used as an indicator of malicious behavior. Table IV presents the measurement results for each C2 label. The Ransomware, AsyncRAT, and CobaltStrike labels are from the SSLBL [29]. The remaining labels are from the Feodo Tracker [30]. Several fingerprints are unique to a certain type of C2 server. However, these unique fingerprints cover only a small part of all observations. The number of unique fingerprints gradually increases by combining fingerprints with additional HTTP data (*i.e.*, the HTTP Server header containing values like `nginx` or `Apache/2.4.18`). New servers added to the blocklists repeatedly had the same fingerprint as past servers, that is, 95% of the targets added during week $n+1$ have fingerprints already observed for this label during weeks $[1..n]$. In other words, these servers could be identified through fingerprinting.

This work uses a binary classifier to decide whether or not a server is a C2 server from a blocklist. The classifier takes a threshold t as a parameter and predicts a C2 server if the C2 rate of the observed fingerprint is higher than t. This rate is calculated for each fingerprint by dividing the number of times observed from a C2 server by the total number of times seen in the training data (*e.g.*, a threshold of 80% means a fingerprint must be observed more than 80% of the times from blocklists such that servers with this fingerprint are classified as a C2 server). The classifier is evaluated on every new target



(a) TLS Fingerprint (10 CHs) (b) HTTP Data (Server Header) (c) Combined (10 CHs + HTTP Data)

Fig. 4. Precision and recall to identify new observations on our input lists as C2 servers using the data described in Figs. 4a to 4c as input for the classification.

added to a toplist or a blocklist during week $n+1$ based on the training data from weeks $[1..n]$. The precision and the recall are defined as $\frac{TP}{TP+FP}$ and $\frac{TP}{PP}$, respectively. The true observations PP are the responsive targets found on a blocklist. Figure 4 shows results for different thresholds for three data sources to construct the input for the classification. The classifier performance will greatly increase if the fingerprints are combined with additional data from the HTTP Server headers. This HTTP datum alone is unsuited for a classifier, but when combined, it achieves a maximum precision of 99% for 35% of the added C2 servers. In contrast to the CDN detection, the low recall is an indicator that our fingerprinting was not fine-grained enough to detect the differences in the deployments needed to identify all C2 servers. Augmenting the fingerprints with HTTP data was our solution to improve this granularity for more effective fingerprinting.

This analysis demonstrates that many C2 servers have a very unique TLS behavior that can be used to identify them. We also presented how a classification of these servers can be implemented on a large scale, and that such an approach can achieve a high precision. Additionally, a potential fingerprint database for C2 servers would live much longer than the entries on the used blocklists, which means that they can provide valuable information about newly deployed C2 servers until their IP addresses get publicly known.

E. Comparison With JARM

After analyzing the applicability of TLS fingerprintings on a large scale, the subsequent paragraphs have a look at the performance of JARM [7] which can also be used to fingerprint TLS servers. While JARM uses similar data to our approach, we will show that both the data extracted from the TLS handshakes and the CH selection of this work provide an improved base for fingerprinting.

We scanned every target with the CHs used by JARM, as well as the empirically optimized ones from this work. Thus, we can compare both approaches. We did not use the open-source JARM script directly because it was not able to scan our number of targets on our hardware fast enough. Instead, we have used our own scanner with the JARM CHs and extracted the subset of features that JARM uses to construct its fingerprints from our data. In particular, fingerprints were stripped from alerts, any TLS message besides the Server Hello, and any extension data besides the ALPN (*i.e.*, the IDs and the order of the extensions remained intact, and only the data contained in these extensions were removed). Table V presents a comparison of how the selection of features and CHs affects the fingerprinting results. The improvements proposed herein consistently provide better results while maintaining the number of requests necessary for fingerprinting the same (*i.e.*, 10 CHs). In total, this work can differentiate 55% more Server Behaviors. Considering the C2 servers, this improved differentiation results in 16 additional unique C2 behaviors and four times more C2 servers identifiable with these unique behaviors. In this case, "unique" means no overlap was observed with any server found on a toplist.

TABLE V
COMPARING RELATED WORK JARM WITH ACTIVE TLS STACK FINGERPRINTING (ATSF) CONSIDERING BOTH THE DIMENSIONS OF FEATURE SELECTION AND USED CHS.

Feature selection approach	JARM		ATSF	
Used CHs	JARM	ATSF	JARM	ATSF
Unique fingerprints	52 316	63 037	65 111	81 180
Unique C2 fingerprints	7	12	16	23
Unique C2 targets	15	43	22	64

In conclusion, TLS fingerprinting tools like JARM can benefit from the advanced feature extraction and the systematic design of the CHs proposed in this work to improve the effectiveness of the approach.

VI. DISCUSSION

With our fingerprinting approach we gained new insights into the Internet and found interesting relations among TLS servers. Some of them, we discuss in the following paragraphs.

a) Advanced Similarity Comparison: This work explicitly does not obfuscate any information as done by Refs. [5] and [7] to keep the syntactic information of each part of the fingerprint intact. This supports explainability, allows to relate not only equal but similar behaviors in the future, and to adapt the fingerprints afterwards (*e.g.*, removing the Status Request extensions). Similar fingerprints can indicate deployments from an actor who has done just minor configuration changes.

b) The Success of Random CHs: In the beginning we have used the standard CH from the Go library and CHs mimicking popular browsers for fingerprinting. However, they could not extract enough information from servers to be effective in use cases because the responses were similar focusing on few popular TLS configurations. In contrast, the Random

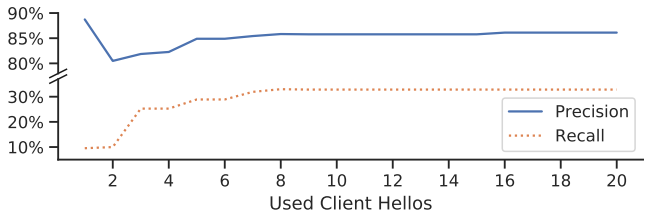


Fig. 5. Influence of the CHs on the C2 detection with an 80% threshold.

CHs were empirically optimized to distinguish servers and have unusual combinations and order of parameters. They vary in the combination of TLS versions, ciphers, ALPNs, and supported groups and, sometimes, are not realistic (*e.g.*, offer ALPNs unsuitable for webservers). Interestingly, two CHs use TLS 1.3 and none use TLS 1.2 as `legacy_version`, both is not conform to the RFC [17]. In contrast, JARM uses more realistic CHs with fewer variations in the parameters. They defined them through systematic subsets (*e.g.*, the top half) or a reversed order from a fixed input. In conclusion, the Random CHs are very successful for fingerprinting because they have a fuzzy character triggering more distinctive responses.

c) Adequate number of CHs for C2 detection: We designed the 10 CHs as a general-purpose configuration to provide a good base for a classification. However, for specific use cases they can be different. We scanned every server with 10 optimized and 10 JARM CHs; hence, the classification performance shown in Fig. 4a can be recomputed using up to 20 CHs as input for an 80% threshold. We used the same strategy to add CHs to the classification input as in section IV. While the precision was high for almost all sets of CHs, scanning with multiple ones mainly increased the number of classified servers (visible in the recall). The maximum precision and recall was achieved with 16 CHs, but the gain was minimal after eight CHs. For this detection use case, multiple CHs are necessary, but eight CHs would have been sufficient. Additionally, an adaptive scanning approach could be implemented in the future work, where additional requests are only sent to a server if the precision of its current classification is not high enough.

d) CDN Caches in a Foreign AS: We were not able to correctly identify all reasons why some domains resolved to CDN caches in a foreign AS. In particular, 80% of them were located in an AS from VTC Digicom, Amazon, or Render. Interestingly, the whole Render AS was proxied through Cloudflare and its IP address prefix functioned as CDN cache. It is possible that we have observed the effect of operators that try to remain in control of their traffic flow (*e.g.*, by deploying a Meta-CDN [31]) because 68% of the domains pointing to a CDN cache in a foreign AS used a nameserver unrelated to the CDN. To our knowledge, Cloudflare does not operate CDN caches outside of their ASs. At least six of these caches turned out to be reverse proxies set up by third parties; investigated with a tracing endpoint[6] suggested to us by Cloudflare. Additionally, our data contained 1 094 outlier domain names resolving to the public Domain Name System (DNS) resolver `1.1.1.1`, which was apparently also a cache of the Cloudflare network. 76% of these domain names used a Cloudflare nameserver. In contrast to Cloudflare, Akamai has deployed CDN caches in more than 1k foreign ASs to localize their traffic [32]. However, we detected just 20 ASs because we did not scan the full IPv4 address space, but IP addresses resolved from the toplists. Akamai uses the DNS to distribute the load on their servers [32]. We assume, we saw these ASs because our scan traffic was not always directed to the closest CDN cache but distributed across servers in multiple ASs.

e) CDN Inconsistencies: Some CDN caches were inconsistent in their responses because not every IP address successfully responded to every domain name requested. To the end, multiple domain names were necessary to validate caches. For Cloudflare, this was only a single IP address located in China. For Akamai, bigger clusters were visible and multiple domain names were needed to verify them.

f) Unstable Fingerprints: Some targets had inconsistent fingerprints that could be caused by the server or by more complex setups. Sometimes, we saw indicators of load balancers in the HTTP Server header indicating that the actual fingerprinted server could change during the scan process. This is also the main limitation of our approach because it relies on multiple TLS connections to connect to the same Server Behavior. However, this was rarely an issue (section V-B).

[6]https://cloudflare.com/cdn-cgi/trace

VII. Conclusion

This work proposed a methodology for acquiring and leveraging TLS metadata with the help of large scale active measurements. The value of the approach is backed by two measurement studies on the Alexa and Majestic toplists and two C2 blocklists over half a year to detect CDN and C2 servers. New C2 servers added to the blocklists were classified with a precision higher than 99%. Depending on the CDN and their infrastructure, the detection precision ranged from 85% (Akamai and Alibaba) to more than 99% (Cloudflare and Fastly). Additionally, 351 IP addresses were identified serving CDN content outside of the AS operated by the CDN.

The results were obtained with a reasoned selection of the features extracted from TLS handshakes and with the use of multiple scanning probes to construct fingerprints of the TLS stack on servers. These 10 probes were empirically optimized to provide as much information as possible while minimizing the measurement time and the ethical impact on targets.

This paper describes in detail how TLS stack fingerprinting can be efficiently conducted and proves that these data can be applied on real-world classification problems like C2 detection to provide valuable security-related insights.

Moreover, the extended feature extraction and improved CH design can improve existing TLS fingerprinting tools while maintaining the active scanning effort. Given the approach is independent of the actual CHs, we expect future works to tune their CHs to specific use cases, or individually adapt them on a per-server level.

References

[1] C. Labovitz, "Internet traffic 2009-2019," in *Proc. Asia Pacific Regional Internet Conf. Operational Technologies*, 2019.

[2] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, "Coming of Age: A Longitudinal Study of TLS Deployment," in *Proc. ACM Int. Measurement Conference (IMC)*, 2018.

[3] B. Anderson and D. A. McGrew, "Accurate TLS fingerprinting using destination context and knowledge bases," *CoRR*, vol. abs/2009.01939, 2020.

[4] M. Husák, M. Cermák, T. Jirsík, and P. Celeda, "Network-Based HTTPS Client Identification Using SSL/TLS Fingerprinting," in *2015 10th International Conference on Availability, Reliability and Security*, 2015.

[5] J. Althouse, J. Atkinson, and J. Atkins, "TLS Fingerprinting with JA3 and JA3S," 2019. [Online]. Available: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967

[6] Sean Gallagher, "Nearly half of malware now use TLS to conceal communications," 2021. [Online]. Available: https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications/

[7] J. Althouse, A. Smart, R. Nunnally, Jr., and M. Brady, "Easily Identify Malicious Servers on the Internet with JARM," 2020. [Online]. Available: https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a

[8] Censys.io. Data Definitions. Last accessed Feb. 24, 2022. [Online]. Available: https://search.censys.io/search/definitions

[9] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting," in *Proceedings of the First USENIX Workshop on Offensive Technologies*. USENIX Association, 2007.

[10] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov, "Hershel: Single-Packet OS Fingerprinting," *IEEE/ACM Transactions on Networking*, 2016.

[11] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[12] Dirk Wetter. Testing TLS/SSL encryption. Last accessed March 1, 2022. [Online]. Available: https://testssl.sh/

[13] T. Chung, J. Lok, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, J. Rula, N. Sullivan, and C. Wilson, "Is the Web Ready for OCSP Must-Staple?" in *Proc. ACM Int. Measurement Conference (IMC)*, 2018.

[14] D. E. Eastlake, "Transport Layer Security (TLS) Extensions: Extension Definitions," RFC 6066, 2011.

[15] P. Gigis, M. Calder, L. Manassakis, G. Nomikos, V. Kotronis, X. Dimitropoulos, E. Katz-Bassett, and G. Smaragdakis, "Seven Years in the Life of Hypergiants' off-Nets," in *Proc. ACM SIGCOMM*, 2021.

[16] R. Holz, J. Hiller, J. Amann, A. Razaghpanah, T. Jost, N. Vallina-Rodriguez, and O. Hohlfeld, "Tracking the Deployment of TLS 1.3 on the Web: A Story of Experimentation and Centralization," *ACM SIGCOMM Computer Communication Review*, 2020.

[17] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018.

[18] J. Amann, O. Gasser, Q. Scheitle, L. Brent, G. Carle, and R. Holz, "Mission Accomplished? HTTPS Security after Diginotar," in *Proc. ACM Int. Measurement Conference (IMC)*, 2017.

[19] O. Gasser, M. Sosnowski, P. Sattler, and J. Zirngibl. Goscanner. https://github.com/tumi8/goscanner.

[20] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," in *Proc. USENIX Security Symposium*, 2013.

[21] D. Dittrich and E. Kenneally, "The Menlo Report: Ethical principles guiding information and communication technology research," *US Department of Homeland Security*, 2012.

[22] C. Partridge and M. Allman, "Addressing Ethical Considerations in Network Measurement Papers," *Communications of the ACM*, 2016.

[23] "Active TLS Stack Fingerprinting: Additional Material"," 2022. [Online]. Available: https://active-tls-fingerprinting.github.io/

[24] IANA, "Transport Layer Security (TLS) Parameters," last accessed March 1, 2021. [Online]. Available: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

[25] Alexa, "Top 1M sites," last accessed 28 Feb. 2022. [Online]. Available: http://s3.dualstack.us-east-1.amazonaws.com/alexa-static/top-1m.csv.zip

[26] Majestic, "The Majestic Million," last accessed 28 Feb. 2022. [Online]. Available: https://majestic.com/reports/majestic-million/

[27] D. R. L. Brown, "SEC 2: Recommended Elliptic Curve Domain Parameters," 2010. [Online]. Available: http://www.secg.org/sec2-v2.pdf

[28] Q. Scheitle, O. Hohlfeld, J. Gamba, J. Jelten, T. Zimmermann, S. D. Strowes, and N. Vallina-Rodriguez, "A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists," in *Proc. ACM Int. Measurement Conference (IMC)*, 2018.

[29] abuse.ch, "SSL Certificate Blacklist," last accessed Feb. 28, 2022. [Online]. Available: https://sslbl.abuse.ch/

[30] ——, "Feodo Tracker," last accessed Feb. 28, 2022. [Online]. Available: https://feodotracker.abuse.ch/

[31] O. Hohlfeld, J. Rüth, K. Wolsing, and T. Zimmermann, "Characterizing a Meta-CDN," in *Proc. Passive and Active Measurement (PAM)*, 2018.

[32] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-Performance Internet Applications," *ACM SIGOPS Oper. Syst. Rev.*, 2010.