



Fingerprinting the Shadows: Unmasking Malicious Servers with Machine Learning-Powered TLS Analysis

Andreas Theofanous

Foundation for Research and Technology - Hellas
Heraklion, Greece
theofanous@ics.forth.gr

Alexander Shevtsov

Technical University of Crete
Chania, Greece
asevtsov@tuc.gr

Eva Papadogiannaki

Technical University of Crete
Chania, Greece
epapadogiannaki@tuc.gr

Sotiris Ioannidis

Technical University of Crete
Chania, Greece
sotiris@ece.tuc.gr

ABSTRACT

Over the last few years, the adoption of encryption in network traffic has been constantly increasing. The percentage of encrypted communications worldwide is estimated to exceed 90%. Although network encryption protocols mainly aim to secure and protect users' online activities and communications, they have been exploited by malicious entities that hide their presence in the network. It was estimated that in 2022, more than 85% of the malware used encrypted communication channels.

In this work, we examine state-of-the-art fingerprinting techniques and extend a machine learning pipeline for effective and practical server classification. Specifically, we actively contact servers to initiate communication over the TLS protocol and through exhaustive requests, we extract communication metadata. We investigate which features favor an effective classification, following state-of-the-art approaches. Our extended pipeline can indicate whether a server is malicious or not with 91% precision and 95% recall, while it can specify the botnet family with 99% precision and 99% recall.

CCS CONCEPTS

• Security and privacy → Network security; • Computing methodologies → Machine learning; • Networks;

KEYWORDS

TLS; TLS Fingerprinting; Active Probing; Botnet; Command and Control; Server Characterization; Machine Learning

ACM Reference Format:

Andreas Theofanous, Eva Papadogiannaki, Alexander Shevtsov, and Sotiris Ioannidis. 2024. Fingerprinting the Shadows: Unmasking Malicious Servers with Machine Learning-Powered TLS Analysis. In *Proceedings of the ACM Web Conference 2024 (WWW '24)*, May 13–17, 2024, Singapore, Singapore. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3589334.3645719>



This work is licensed under a Creative Commons Attribution International 4.0 License.

WWW '24, May 13–17, 2024, Singapore, Singapore
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0171-9/24/05.
<https://doi.org/10.1145/3589334.3645719>

1 INTRODUCTION

As of October 2020, more than 90% of Internet traffic is communicated over TLS [4]. The adoption rate of TLS 1.3 surpassed the adoption rate of previous versions of TLS with remarkable speed. Based on the TLS Telemetry Report in 2021 [8], TLS 1.3 has become the preferred protocol for 63% of the top 1 million web servers on the Internet [3]. Yet, the increasing popularity and simplified usage of TLS led malware to exploit encryption to hide its presence and communications [12]. WatchGuard observed in 2021 that 91.5% of malware is delivered through encrypted channels [11]. Furthermore, the TLS Telemetry Report revealed that the proportion of phishing sites using HTTPS and valid certificates had risen to 83% in the same year [8]. To establish a secure channel using the TLS protocol, a crucial step involves the exchange of metadata between the client and the server. This metadata is shared through unencrypted *Client Hello* and *Server Hello* messages.

Several works [2, 7, 29, 40] and studies [49] have acknowledged the significance of metadata exchanged during the TLS handshake and attempted to leverage this information to enhance Internet security [46]. The majority of these works focus on the passive analysis of the messages exchanged during the TLS handshake. By employing active approaches, like active TLS fingerprinting, researchers and organizations [15, 21] can collect relevant TLS parameters that when properly processed, they enable the extraction of valuable information for the network status and the participating devices and applications [6, 45].

In this work, we leverage a range of active TLS fingerprinting techniques to examine server behavior. First, we initiate our interaction with servers by sending the first request from a set of predefined requests, and then, we start re-sending this request by progressively forcing the server to select an alternative cipher suite. The resulting TLS parameters are then utilized from the enhanced version of a publicly available semi-automatic machine learning pipeline [20]. To ensure a robust evaluation, we employ three distinct classification models. Building upon this methodology, we compile a database with traffic from benign and malicious entities. We create a binary classification system capable of accurately labeling servers as either benign or malicious. Additionally, we implement a multi-class classification model to further indicate specific botnet families. Finally, we implement a TLS fingerprinting technique and evaluate the performance of 4 different feature

categories. Through this comparison, we explore the strengths and limitations of each method. The contributions of this work follow:

- We examine state-of-the-art approaches for server classification through active probing and integrate them into a publicly available machine learning pipeline.
- We present an analysis of these approaches using a comparative evaluation between different (i) features categories based on fingerprinting and (ii) ML models for classification.
- We present the most extensive dataset known to date for actively categorizing servers. The already processed data is available and ready-to-use in [25].

2 BACKGROUND

To establish a secure channel via TLS, a secure TLS parameters exchange is essential immediately following the TCP handshake, allowing both sides to share their capabilities and preferences.

The TLS handshake begins with the client sending a *Client Hello* message to the server. This message contains information, such as the supported TLS versions, cipher suites, and extensions. It serves as a way for the client to communicate its capabilities and preferences to the server. Upon receiving the *Client Hello* message, the server responds with a *Server Hello* message. This message includes the selected TLS version, cipher suite, and other parameters chosen by the server. It allows the server to inform the client of its preferences and capabilities, enabling both parties to negotiate and agree upon the most secure and compatible settings for their communication. A common technique is to leverage these parameters during the initial stages of establishing the secure channel for different purposes (e.g., OS identification [45], server-side libraries [44], client identification [38, 52], or in censorship circumvention tools [37]), since the messages exchange happens in plaintext.

3 RELATED WORK

Although we follow an active probing approach, in this paragraph, we briefly discuss related works that focus on passive analysis. In 2009, Ristic's work [54] made TLS fingerprinting gain popularity [27], leading to more research [1, 2, 29, 33, 40, 55]. Over time, TLS fingerprinting was applied to diverse applications. It has been employed for longitudinal studies [44], identification of android apps [52], characterization of malware [28], and the detection of IoT components[51].

In the domain of active probing-based fingerprinting techniques, there are two popular methodologies, ATSF [60] and JARM [6], which send a fixed number of *Client Hello* messages to a server. Both techniques generate fingerprints according to the *Server Hello* responses they receive. Papadogiannaki et al. [50] use JARM to generate fingerprints for servers that are known to participate in botnets and they examine their evolution over time. They also show that the percentage of fingerprint overlapping with benign servers progressively rises. In ATSF, authors propose a list of alternative *Client Hello* messages, which seem to result in more expressive and optimized fingerprints when compared to JARM. Nonetheless, a notable aspect of these approaches is the initial generation of the first 10 *Client Hello* messages. This initial step, could benefit from heightened explainability, thus facilitating potential optimization strategies and refining the generation process. DissecTLS [59]

introduces an enhanced functionality of a recursive process of systematically excluding selected server parameters or preferences from the interaction sequence. By iteratively eliminating these values and subsequently re-sending the requests, the DissecTLS methodology strives to generate more expressive fingerprints. Notably, DissecTLS concentrates on the comprehensive extraction of server configurations for server classification. This emphasis on configuration extraction sets DissecTLS apart, offering insights into server behavior outperforming similar tools [23, 24].

To the best of our knowledge, there is no related work that uses machine learning to classify server activity with information collected after active probing. Although several studies have explored the utilization of machine learning techniques in conjunction with TLS parameters and fingerprinting [32, 39, 41, 43, 45, 48, 53, 57, 62], they focus on different use cases (e.g., OS identification, website fingerprinting, detection of privacy leaks). To accurately compare works that do not share analogous techniques is challenging. The combination of active probing and machine learning adds an extra layer of complexity, as active probing generates data in real-time, while machine learning depends on static data for training. This requires continuous processing, which is challenging for large and dynamic datasets due to the computational resources requirements. Also, combining active probing with machine learning, dictates the mutual reinforcement of each other. Kim et al. [42] focused on analyzing TLS traffic based on enhanced neural networks combined with TLS fingerprinting methods. Yet, its reliance on passive techniques contrasts with our emphasis on active ones.

In this work, we aim to explore every facet of existing approaches, striving to identify the optimal outcome in terms of classification. That's why we employ active techniques to gather a broad range of features and subsequently feed them into machine learning models, which are then utilized in our fingerprinting methods. In the upcoming sections, we'll explore the benefits of fingerprinting methods used in active probing and combine the different techniques used in ATSF, JARM and DissecTLS. Finally, we use a modified version of a publicly available machine learning pipeline tailored for our experiments. By testing different fingerprinting techniques and adapting the machine learning models, we aim to examine how they perform in active scenarios.

4 METHODOLOGY

In this section, we present an overview of the methodology used. Our approach centers on optimizing and comparing the current active approaches of collecting unique messages from servers during TLS parameter negotiation.

An approach to attain this is by employing a predefined number of particular requests to a server and receiving the corresponding responses in return. For this purpose, instead of generating new initial *Client Hello* message, we leverage existing ones from previous works. We select the JARM tool, renowned for its scalability and efficiency [5], widely adopted by major Internet scanners, such as *Censys* [16] and *Shodan* [22]. JARM sends 10 customized TLS *Client Hello* messages to a target TLS server, enabling us to identify a distinct set of responses.

Each of these *Client Hello* messages triggers a response from the server in the form of a *Server Hello* message, agreeing on the

selected parameters. During this process, it is possible for a server to abort the handshake or attempt to renegotiate. DissecTLS [59] exploits this opportunity by conducting intensive scans on servers. This process continues until the complete TLS configuration can be successfully reconstructed. Based on this approach, we implement an extension of JARM, integrating this additional functionality. Our tool executes each of its initial requests by progressively removing previous preferences. Our choice is to concentrate on a specific parameter for reduced complexity. The key is to iteratively remove previous cipher suites selected by servers. This process continues until an error occurs or a timeout elapses. This proves to be the best case due to its wide range of options and the presence of a preference order. More details can be found in §4.1. Finally, we extend an existing pipeline to enhance and fine-tune it for the purpose of server classification through active probing.

4.1 Cipher Suite Selection

To enhance the method of active probing and achieve a more streamlined process, we focus on a single parameter during TLS parameter negotiation – the cipher suite. The cipher suite plays a crucial role in establishing a secure channel between the client and server. Cipher suites encompass a wide variety of cryptographic algorithms and key exchange methods, and their order of preference can significantly impact the server's behavior.

We perform an iterative process, where we interact with the server by initiating TLS connections and start removing the selected cipher suites one by one. We continue this process until an error occurs or a timeout is reached. The primary objective is to identify the server's preferred cipher suite by observing its behavior when specific cipher suites are eliminated. By concentrating on a single parameter, we aim to extract all possible preferences of the server related to the cipher suite, leading to more unique responses. This approach not only increases the precision of our model, but also reduces the extra overhead associated with probing multiple parameters simultaneously, while also delaying the occurrence of errors. We monitor the server's responses and gather data on each interaction. This data serves the crucial purpose of identifying the server's most preferred cipher suite and comprehending how its behavior evolves when different cipher suites are removed.

4.2 Data Collection

Our data collection methodology is designed to include a wide spectrum of servers and network configurations. The process involves two main sources: the "Top 10K domains" from the Tranco list and various "Blocklists" containing potentially malicious IP addresses.

4.2.1 Top 10K Domains. To initiate our data collection process, we select the "Top 10K domains" from the Tranco list, which is a reliable source of Internet-wide domain information [14]. Considering the time and resources required to scan the entire Tranco list on a daily basis, we scan a representative sample; the top 10K domains. To maintain the relevance and timeliness of our data, we schedule nightly downloads. This regular update process allows us to capture evolving trends within the TLS ecosystem.

4.2.2 Blocklists. In addition to the top 10K domains, we incorporate the Feodo Blocklist into our data collection [9]; a blocklist that

contains IP addresses associated from five different botnet families. To expand our dataset, we also include unlabeled blocklists that contain potential malicious IP addresses (i.e., Blocklists.de [13], Ci-Badguys [17], SSLBL [10], and Darklist.de [18]). In the paper, the term *Blocklists* refers to these 4 unlabeled lists. The daily number of unique IP addresses contained in these blocklists ranges from 30 to 20K. This integration enables us to gain insights into existing TLS behaviors across a broader range of servers, encompassing both benign and potentially malicious activities. To handle the large size of blocklists, we employ weekly Censys's active TLS scans [34], which filter the lists and provide us with an up-to-date and manageable set of active IP addresses along with a list of open ports.

4.2.3 Database. After this preparation phase, we employ our tool, starting to send its 10 consecutive *Client Hello* messages. For each one of them, we proceed with an iterative process, where we remove the cipher suite selected by the server. Initially, we send the first *Client Hello* message to the server and wait for its response. Upon receiving the server's selection of a cipher suite, we proceed with removing this specific option from the list of supported cipher suites for the subsequent interaction. We then resend the same *Client Hello* message to the server, this time without the eliminated cipher suite. This iterative approach continues until the server either refuses to respond or stops acknowledging our queries. Subsequently, we repeat this entire procedure for each of the remaining 9 *Client Hello* messages in a methodical manner.

Through this process, we acquire valuable insights into TLS parameter negotiation, allowing us to assess how servers dynamically adapt their responses to different cipher suite options. Throughout this interaction, we monitor and collect the network packets exchanged using the tcpdump tool. In contrast to real-time fingerprint generation approaches, we store the collected traffic for subsequent analysis. This decision allows us to focus on the efficiency of feature selection when creating fingerprints. The stored traffic forms a rich dataset that enables deeper investigations into TLS parameter negotiation and server behavior.

Over a five-month period (01/2023 – 05/2023), we accumulate approximately 1.8M samples, resulting in unprocessed data of 278 GB in total. This extensive repository empowers us to observe evolutionary trends within the TLS ecosystem, and monitor how malicious botnets adapt their activities on benign servers. Furthermore, this dataset provides fertile ground for exploring novel capabilities and potential in TLS parameter negotiation and secure channel establishment research.

4.3 Filtering

To ensure the integrity and reliability of our dataset, we implement a filtering process to retain only the successful samples while reducing potential anomalies. First, we remove servers that did not respond to any of the initial 10 *Client Hello* messages, resulting in empty packet capture files. Next, we identify scenarios where servers initially respond successfully to our requests but subsequently stop acknowledging our messages during the TLS renegotiation, leading to a timeout. Such occurrences are flagged as "Incomplete", as they do not provide valuable insights into server behavior in cases that communication is unexpectedly disrupted. Furthermore, we encounter instances that servers respond with no

Table 1: Number of data samples processed

Source	Filtered Samples
Tranco	763,443
Blocklists	84,354
QakBot (Feodo)	3,890
Dridex (Feodo)	1,369
BumbleBee (Feodo)	931
Emotet (Feodo)	863
BazarLoader (Feodo)	75
Total	854,925

TLS packets or repeatedly provide the same *Server Hello* for each request. These anomalies are marked as “Disrupted” and discarded from processing alongside with the “Incomplete” samples (§ A.1).

To minimize these anomalies to the maximum extent possible, as a final step, we perform a flow checksum on the collected traffic. We carefully check and confirm that the way each communication happens follows the usual patterns and rules expected. For example, we make sure that the TLS information is contained within the TCP packet. We also look at cases where packets arrive too late (after the timeout lapsed) or when ACK numbers do not match the SEQ ones. If things don’t match up or follow the rules, we remove that data from our collection. Table 1 provides an overview of the final data samples resulting from our filtering process.

4.4 Data Transformation and Parameters Selection

After filtering our dataset and retaining only the *Completed* files, the next crucial step is to address the challenge posed by the size of packet capture files, since processing and feeding them directly into a classification model for training proved impractical. Therefore, we have created a lighter format that would still retain the essential features needed for our analysis and evaluation. To achieve this, a subset of TLS parameters, extensions, and certificates is carefully selected (§ A.2). These parameters include TLS versions, cipher suites, ALPNs, Elliptic Curves, certificates (x509) and other relevant information. During the data transformation phase, we aim for a balance between information richness and efficiency. Thus, we create a list large enough to enable the utilization of different parameters subsets and perform comparisons effectively. Finally, we unify the original data samples into a single CSV file, containing all the extracted fields and reaching the size of 55 GB. Its simplicity and ease of use make it an ideal choice for representing the curated list of parameters and TLS negotiation data.

To ensure consistent data presentation, each packet capture file in our database corresponds to a single row into a CSV file. This row contains all the parameters extracted from each server’s response, providing a total view of the TLS negotiations that occur. Each column holds a particular parameter’s value that the server chose from the corresponding request at that specific time. Since we perform exhaustive requests to servers, the exact number of their total responses for each *Client Hello* varies, resulting in a dataset that includes rows with features ranging from 200 to 20K. The calculation yielding 20K features is determined by the following

factors: we extract 46 distinct parameters from each *Server Hello* message, the maximum observed number of iterations is 45, and there are 10 initial handshake procedures. Consequently, if a server responds 45 times recursively for each of the 10 initial handshakes, the resultant dataset comprises 20,700 features (46 parameters x 45 iterations x 10 handshakes), in addition to columns indicating the corresponding date of the sample, category, botnet family (if applicable), IP/domain, port, and 10 columns indicating the total number of responses for each of the 10 initial handshakes until the server encountered an error or fails to respond.

Finally, in the last step, we transform string-type values into an arithmetic-like format representation. This conversion is necessary for the selected classification machine learning models, as they can only process numeric features. We implement a technique to represent strings as integers while ensuring consistency for the same input. By leveraging the MD5 hash function, we effectively convert these string-based parameters into a standardized format that is suitable. Consequently, we successfully transform the string-based parameters into fixed-size representations and then convert the hexadecimal output into integers. In cases where the input is empty, for example, ‘None’, our implementation returned ‘-1’ to maintain data integrity and signify the absence of applicable data.

5 ANALYSIS

Before proceeding to the design of our models, we conduct a preliminary analysis on the final transformed and extracted data to gain insights and understand the variation in server responses when performing exhaustive requests. In this analysis phase, we calculate the average number of responses for each data source to examine the server’s behavior during TLS parameter negotiation and cipher suite selection. Additionally, we apply the Borda Count rule [35, 36, 56] to identify the top cipher suites among the servers in our dataset, based on their min-max normalized scores [26].

To gain a deeper understanding of the data, we visualize the distribution of server responses. Following figures represent the average responses of the sources and the distribution of them through box plots, providing an overview of the data central tendency, dispersion, and outliers. By analyzing these plots, we examine the variation in server responses and determine if exhaustive techniques can lead to better results. The insights gained from this preliminary analysis help us identify patterns, trends, and potential variations in server behavior during TLS parameter negotiation. Understanding these aspects is critical to move on with the evaluation phase and compare our models. The box plots, in particular, allow us to visually compare the distribution of server responses across different sources, highlighting any variations and enable us to draw important conclusions about the impact of uniqueness in server classification.

5.1 Number of Responses

Figure 1 presents the mean number of responses for each source (i.e., Tranco, Blocklists, Feodo) during the 10 exhaustively iterative handshakes until the server either responds with an error message or ignores the request. We observe that at the last handshakes the majority of sources exhibit a lower number of responses compared to earlier ones. This could open up intriguing possibilities for future

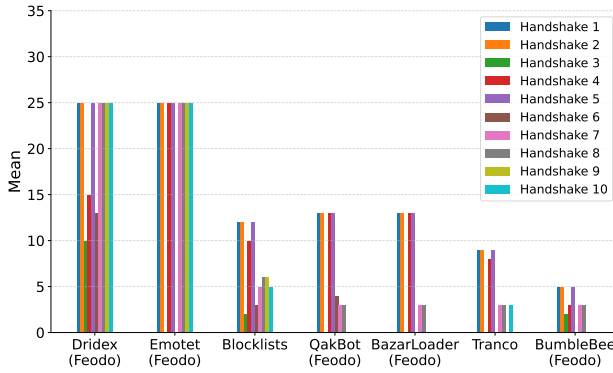


Figure 1: Mean number of successful handshakes per iteration and data source (i.e., Tranco, Blocklists, Feodo)

investigations into the specific *Client Hello* parameters responsible for this reduction and why they influence server behavior. Dridex and Emotet botnets exhibit the highest average number of responses among all sources. Notably, the Dridex botnet demonstrates a high response rate across all handshakes, indicating its reluctance to refuse a TLS connection regardless of the utilization of any outdated parameters. On the other hand, the Emotet botnet seems to respond in a standardized manner, consistently producing specific patterns in its responses following a specific configuration.

Figure 2 illustrates the distribution and variability of the total response counts across the different sources. The extent of spread within the boxes and the length of the whiskers directly indicate the degree of variation. *Tranco*, which represents benign server activity, exhibits a high number of outliers. This outcome is in line with expectations considering the extensive diversity among domains and their unique configurations. In contrast, the *Feodo* source has a reduced number of outliers (resulting from BumbleBee, Emotet and QakBot botnet families). It consists of traffic from 5 different botnet families, each consistently following its distinct behavior. For this source, an outlier might indicate a misclassification. Lastly, interpreting the list namely *Blocklists* is challenging, since the family of the IP addresses contained is unknown. Unlike the *Feodo* list, the list *Blocklists* contains raw IP addresses with no other information.

5.2 Variances in Cipher Suite Selection

In the next phase of our analysis, we evaluate the cipher suite preferences exhibited by various sources within the dataset. We extract the cipher suite selections for each *Client Hello* message and aggregate them based on the specific handshakes employed. To benchmark the preferences, we harness the Borda Count rating technique [36]. This method assigns points to candidates according to their rankings on each ballot. Lower-ranked candidates are allocated fewer points, while higher-ranked ones garner more. By summing up the scores granted to each cipher suite (divided per source list), we derive server preferences. To ensure uniformity, we apply min-max normalization. This allows us to identify the cipher suite selections per source list, based on the initial handshakes and to observe variations across sources. Coupled with the analysis of response counts, these findings could potentially reveal botnet

Table 2: Top 10 SSL/TLS cipher suites of Tranco ranked by Borda count (presented in descending order) and the corresponding values of the C2 lists

Cipher	Score		
	Tranco	Blocklists	Feodo
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	1.000	0.998	0.975
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	0.965	1.000	0.964
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	0.742	0.730	0.856
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	0.730	0.834	1.000
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	0.711	0.756	0.843
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	0.703	0.846	0.979
TLS_RSA_WITH_AES_128_GCM_SHA256	0.586	0.650	0.751
TLS_RSA_WITH_AES_128_CBC_SHA	0.563	0.940	0.873
TLS_RSA_WITH_AES_256_CBC_SHA	0.531	0.937	0.852
TLS_RSA_WITH_AES_256_GCM_SHA384	0.497	0.658	0.737
Unique cipher suites	69	68	49

attempts to imitate benign TLS configurations. Table 2 presents the top 10 Tranco cipher suites selection. For each cipher suite, we display its corresponding score as occurred during the examination of the Blocklists and Feodo source lists.

6 MODEL DESIGN

In this section, we present the machine learning classification approach and an implementation of a fingerprinting method with 4 different feature categories, each one derived from a different strategy. Our aim is to explore the potential of these methods for server classification. Notably, existing methods are based on the similarity between new and already known fingerprints. Such an approach is limited concerning the detection of newly observed samples. Also, the computation of exact matches without proper reduction of non relevant parameters generates constraints towards the generalizability of the method. Our research seeks to address this limitation by investigating the effectiveness of machine learning models in this context, combined with the benefits of active probing.

6.1 Machine Learning Model

In this study, one of our main objectives is to investigate the efficacy and accuracy of machine learning classification approaches. We examine the characterization of servers into benign and malicious (binary classification), as well as the categorization into different types of malicious sources (multi-class classification). Machine learning classification tasks entail several key steps, including data pre-processing, feature selection, defining multiple models along with their configurations, fine-tuning during cross-validation, and ultimately selecting the best model. It is crucial to execute each step properly to avoid common pitfalls such as information leakage between training and testing sets, class imbalance, overfitting/underfitting, and decision threshold optimization.

To ensure a robust implementation, we have chosen a publicly available semi-automatic machine learning pipeline that addresses common issues encountered in classification tasks [20, 58]. Furthermore, we have selected 3 distinct classification models to facilitate a comprehensive comparison of different classification approaches: Gaussian Naive Bayes [61], Random Forest [30], and the state-of-the-art XGBoost [31]. These selected methods are well-equipped to handle class imbalance and perform effectively on imbalanced datasets, a crucial consideration for our dataset.

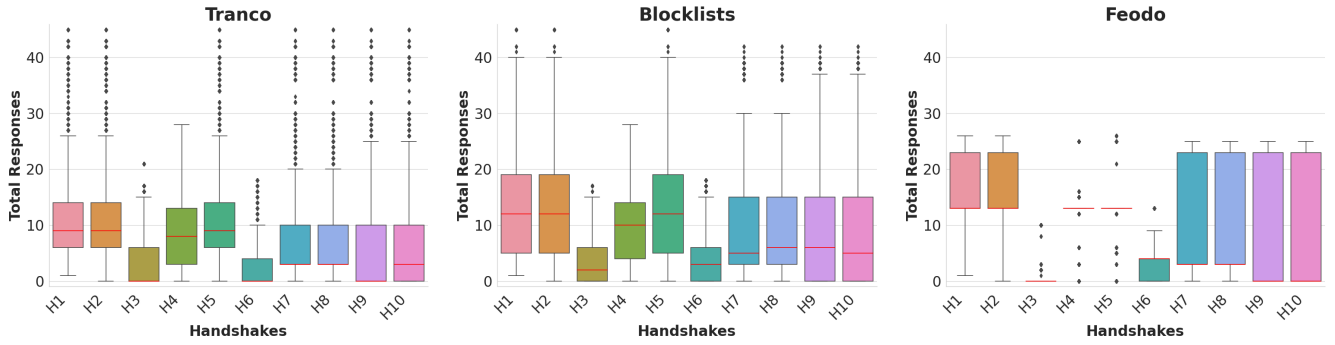


Figure 2: The distribution of total responses across the sources (i.e. Tranco, Blocklists, Feodo)

The chosen pipeline includes data division into 80%/20% proportions for training-validation and testing (also known as hold-out). However, this random split based solely on the sample class category (also known as a target) is not suitable for our dataset due to the presence of different malicious or benign machines that have been sampled multiple times (daily measurements). In some cases, a machine may or may not change its configuration during the monitoring period, and the initial splitting based solely on the target value could result in nearly identical samples being present in both the training and testing data portions. This type of data separation could lead to overly optimistic model performance.

To address this issue, we have taken measures to ensure the appropriateness of our data splitting. Specifically, we have retained only unique samples for each machine in our dataset. Furthermore, we have divided the dataset into 80% for train/validation and 20% for the testing portion. This division is based on the combination of the target (benign or malicious) and the machine’s IP address, ensuring that the testing set contains 20% of the unique machines, avoiding the presence of similar samples between training and hold-out set.

6.2 Fingerprinting

Fingerprinting techniques consist of two fundamental steps. The first step focuses on feature selection, which is the most significant aspect of this methodology. During this phase, most informative and discriminative parameters from TLS metadata are carefully curated and identified. The selected features play a pivotal role in characterizing the server behavior and for effective classification. The second step involves the fingerprint generation by concatenating these features and then by utilizing hash functions, converting them into standardized format outputs.

In a dataset with more than 20K possible features, selecting the optimal ones is challenging. Based on current approaches we create 4 distinct combinations of features. Each category offers valuable insights into the trade-off between the selected features and the model’s precision. Table 3 presents the different categories. *Exhaustive* category, contains features from only 1 parameter, the chosen cipher suite extracted from each *Server Hello* during the handshakes. Although changing cipher suites could impact other parameters, we concentrate on the most relevant one during our step-by-step removal process. Conversely, in *Predefined* category, 21 out of the 46

Table 3: Maximum number of features per category

Exhaustive	Predefined	ML-Selected	All-Possible
450	210	84	20,710

parameters are included as features as we choose to exclude parameters relevant to certificates and concentrate solely on behavioral patterns. These features are exclusively extracted from the initial response of each handshake, ignoring the previous exhaustive approach completely. The last 2 categories are designed with the first containing the features selected from the machine learning pipeline outlined in § 6.1 and the second with all features possible, with an additional 10 features that indicate the total responses per handshake. All in all, each category is inspired by the methodologies in the state-of-the-art (i.e., “Exhaustive”/“All-Possible” categories are inspired by [59] and the “Predefined” category comes from [6] and [60]), whereas the “ML-Selected” category is the newly evaluated feature set that is introduced in our work.

To generate the fingerprints, we follow a simple approach. We concatenate the selected features and pass directly through the SHA256 hash function, resulting to a 32-byte output. This process should provide a single fingerprint for each unique server behavior.

JARM’s 62-character fingerprints are segmented, with the initial 30 characters representing the server’s chosen TLS version and ciphers for each of the 10 client hello messages, and the subsequent 32 characters forming a truncated SHA256 hash of the cumulative server extensions, excluding x509 certificate data [19]. Unlike JARM, we’re not emphasizing in a fingerprint generation process that relies on partial fingerprint matching for similarities. Using the methodology described above, our goal is to observe the variations in results across the different feature categories, each representing a different feature selection approach.

Finally, we use a different approach for data splitting compared to the machine learning process. Given that fingerprinting techniques depend on periodically updated databases we divide the dataset into 80%/20% based on the dates of collection. We retain the 80% of daily traffic for training, generating fingerprints that are stored in our database, while the remaining 20% is allocated for testing.

Table 4: Total and unique fingerprints per feature category (i.e., Exhaustive, Predefined, ML-Selected, All-Possible)

Source	Total FPs	Unique FPs			
		Ex/ve	Pred/ed	ML	All
Tranco	763K	5K	123K	207K	329K
Blocklists	84K	2K	19K	27K	35K
QaKBot (Feodo)	3K	4	109	354	1855
Emotet (Feodo)	863	2	37	72	158
Dridex (Feodo)	1369	3	42	62	330
BumbleBee (Feodo)	931	5	16	35	215
BazarLoader (Feodo)	75	3	28	3	33
Total	854K	7K	143K	235K	367K

7 EVALUATION

In this section, we outline the steps we followed to evaluate our models. The entire process was performed on an AMD EPYC 7543 32-Core processor with 512GiB of system memory, equipped with an NVIDIA A30 GPU. This configuration was essential for efficiently handling and processing the substantial amount of data, given the extensive number of extracted features and the fine-tuning of machine learning models. We evaluate our models based on two distinct classification techniques. First, we perform binary classification to specify whether a server is malicious (C&C) or not. Then, we perform multi-class classification, to identify the botnet family of each server. To compare the performance of our models, we use the precision, recall and F1 metrics.

7.1 Pre-Assessment

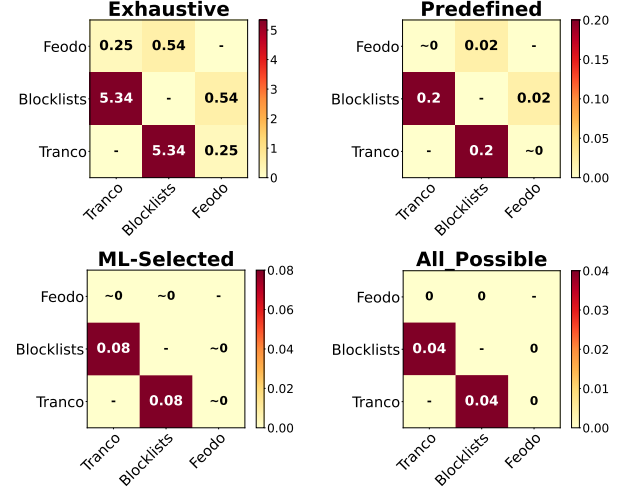
Before training, we perform some checks for possible IP address overlaps over the whole dataset. We encounter only 3 IP addresses overlapping in different dates between the *Emotet* and *Dridex* botnets of Feodo. For realistic reasons, none of them was excluded.

Subsequently, we extract all possible fingerprints from our different feature selections. Table 4 provides an overview of the total and unique fingerprints per category. Interestingly, even though *ML-Selected* category contains the fewest features, produces more unique fingerprints than the *Exhaustive* and *Predefined*.

Next, we aim to determine the number of overlaps between the fingerprints. Figure 3 illustrates the ratio of the unique overlaps across sources. We observe that, once again, the *ML-Selected* category, despite having fewer features, exhibits fewer overlaps compared to the *Exhaustive* and *Predefined*. Notably, several servers from the *Blocklists* list share identical fingerprints with the *Tranco*, possibly indicating an effort to imitate legitimate behavior and duplicate configurations.

7.2 Malicious vs benign

7.2.1 Machine Learning. The binary classification pipeline effectively reduces the dataset’s dimensionality by selecting only the top 84 most important features from the pool of maximum 20K extracted features. We identify the significant features using the Lasso regression model based on non-zero coefficients. Following the feature selection process, the pipeline proceeds to fine-tune each predefined model, retaining only the best configuration for

**Figure 3: The ratio (%) of unique overlaps**

each. The average performance of these configurations during K-Fold cross-validation is summarized in Table 5. The results indicate that XGBoost is the optimal classification configuration, achieving an average F1 score of 0.973 over the validation set.

To comprehensively evaluate the performance of the selected model, we utilize the remaining of the hold-out dataset, which contains 120,374 unseen data samples. The chosen configuration achieves an F1 score of 0.931, a ROC-AUC score of 0.971, a recall score of 0.95, and a precision score of 0.911 (§ A.3). Additionally, to further evaluate and stress the selected model, we repeat the testing procedure with newly added data, originating from IP addresses never processed before¹. Specifically, we collect different sets of unseen benign and malicious machines for 2 separate time periods (06/2023 and 07/2023). We process unseen data from 18,883 samples collected in June and 21,782 samples collected in July. We utilize these data samples in order to evaluate the model performance in a realistic case scenario. According to the results presented in Table 6, our model (XGBoost) manages to achieve more than 0.95 F1 score and more than 0.986 precision in both datasets (06/2023 and 07/2023). We have extracted the explanation of the captured model patterns using the SHAP model explainability technique (Fig. 4).

Table 5: Performance of selected binary and multi-class classification models during the K-Fold cross validation (F1 score)

Model	Multi-class			Binary		
	Training	Validation	Testing	Training	Validation	Testing
XGBoost	0.9976	0.9811	-	0.9865	0.9737	0.9311
Random Forest	0.9973	0.9819	0.9907	0.9481	0.9395	-
Gaussian NB	0.9300	0.9257	-	0.8636	0.8636	-

7.2.2 Fingerprinting. In our next step, we want to examine how the selection of features based on different approaches affects the results. Any sample within our testing data, as discussed in section §6.2, which generates a fingerprint observed at least once in

¹Same sources used: 10K Tranco domains [14], Feodo [9] and blocklists [10, 13, 17, 18]

Table 6: Performance of the selected binary classification model over three different datasets: (i) the hold-out dataset, (ii) the dataset collected in June 2023, and (iii) the dataset collected in July 2023

Dataset	F1-Score	ROC-AUC	Precision	Recall
Testing	0.931	0.971	0.911	0.950
06/2023 samples	0.955	0.976	0.925	0.986
07/2023 samples	0.953	0.943	0.919	0.990

Blocklists-Feodo traffic, is classified as malicious. The calculation of the precision, recall and F1 score is calculated using the *classification_report* function from the python library *metrics*.

Surprisingly, the results reveal a significant deviation. The categories *Exhaustive*, *Predefined*, *ML-Selected* and *All-Possible* achieve precision/recall scores of 0.38/0.94, 0.51/0.69, 0.71/0.59, and 0.64/0.47 respectively. This outcome suggests that while the *All-Possible* category has fewer overlaps compared to *ML-Selected*, the overall overlap count was actually higher leading to more miss-classifications.

The scores highlight the essential role of feature selection in determining the quality of predictions. ATSF [60] uses an empirical strategy of randomly generating *Client Hellos* in order to find the best feature set based on the specific kind of classification each time. Their binary classifier which decides whether a server is a C2 server from a blocklist achieves 99% precision score, with a 35% recall due to the nature of these techniques which rely on exact matching.

7.3 Malicious separation

7.3.1 Machine Learning. Similarly to binary classification, the multi-class pipeline also succeeds in significantly reducing the feature space of the extracted dataset, from 20K down to a 108 best features, determined by the Lasso regression coefficients. Moreover, the model fine-tuning process accurately identifies the optimal configuration for each of the predefined classification models, as documented in Table 5. In contrast to the binary classification, the pipeline selects the Random Forest model as the most suitable choice, primarily due to its higher average validation F1 score.

Upon selecting the appropriate model, its performance is further evaluated using the hold-out dataset. The chosen configuration for the Random Forest model yields impressive results, achieving a ROC-AUC score of 0.999, F1 score of 0.990, precision of 0.990 and recall of 0.990 (§ A.4).

7.3.2 Fingerprinting. Simultaneously, we use the fingerprinting technique similarly to the binary classification to explore how feature categories behave in a multi-class problem. After updating the new features to the *ML-Selected* category and considering only labeled traffic, we extract the following scores. The categories *Exhaustive*, *Predefined*, *ML-Selected* and *All-Possible* achieve precision/recall scores of 0.70/0.79, 0.68/0.71, 0.68/0.70, and 0.62/0.28 respectively. Remarkably, this time, the highest scores are achieved by the *Exhaustive* category, which exclusively leverages the cipher suites selected from the botnet families. Given that each server within this experiment is associated with a distinct botnet family, it is supposed to consistently respond in the same manner. On

the other hand, the *All-Possible* category demonstrated lower performance due to the potential issue of overfitting caused by the inclusion of numerous features.

Compared with the binary classification based on fingerprinting in §7.2.2, we see that these techniques work better when servers consistently follow a regular pattern, which makes them particularly suitable for categorizing familiar behaviors. Moreover, since they rely on exact matches with known patterns before, they are optimal for configuration replication discovery. Finally, it is necessary to mention that we do not directly compare them to our machine learning experiments, since we split the dataset differently in order to reflect how each technique would be used in real-life situations.

8 ETHICAL CONSIDERATIONS

We contact IP addresses that are advertised as malicious in public blocklists and we do not perform any port scanning. In addition, the communication initiated by our machines did not provoke any reaction from system administrators (e.g., email warnings [47]).

9 LIMITATIONS

We utilize two distinct IP addresses for the data collection process. We have not performed any analysis to detect any potential blacklisting from the servers side. If an attacker is able to fully mimic the TLS configurations of common, benign machines, it will cause a misclassification. In the future, we will investigate if servers detect our activity and respond using fixed TLS Server Hello messages to bypass the analysis.

10 CONCLUSION

In this paper, we utilize active TLS fingerprinting techniques in conjunction with a machine learning pipeline to examine whether a server is part of a botnet network and to identify in which specific botnet the server participates. We evaluate 3 machine learning models and 4 distinct feature categories selected with different approaches based on fingerprinting. Our results demonstrate that both machine learning and fingerprinting techniques mutually enhance one another, resulting in higher precision. Finally, the dataset resulting from this work is publicly available and can be found in [25].

As future work, we aim to enrich our TLS fingerprints database with more and different botnets, explore approaches that could help us recognize the randomization of cipher suite vectors and try to recognize servers in the wild. We will perform a more in depth analysis of those server TLS responses specifically to uncommon “TLS Client Hello” configurations. Ultimately, we plan to optimize our fingerprinting techniques through machine learning by refining the utilization of the initial 10 handshakes.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers. This work was supported by the projects GREEN.DAT.AI, SENTINEL and SecOPERA, funded by the European Commission under Grant Agreements No. 101070416, No. 101021659, and No. 101070599, respectively. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which maybe made of the information contained therein.

REFERENCES

- [1] 2012. SSL Fingerprinting for p0f. <https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f/>. Accessed: 2023-08-11.
- [2] 2016. FingerprintinTLS. <https://github.com/LeeBrotherston/tls-fingerprinting>. Accessed: 2023-08-11.
- [3] 2019. <https://tranco-list.eu/>. Accessed: 2023-08-11.
- [4] 2019. HTTPS encryption on the web – Google Transparency Report. <https://transparencyreport.google.com/https/overview?hl=en>. Accessed on 2023-08-11.
- [5] 2020. JARM: A Solid Fingerprinting Tool for Detecting Malicious Servers. <https://securitytrails.com/blog/jarm-fingerprinting-tool>. Accessed on 2023-08-11.
- [6] 2020. JARM: An active Transport Layer Security (TLS) server fingerprinting tool. <https://github.com/salesforce/jarm>. Accessed: 2023-08-11.
- [7] 2020. Suricata Open Source IDS / IPS / NSM engine. <https://www.suricata-ids.org/>. Accessed: 2023-08-11.
- [8] 2021. The 2021 TLS Telemetry Report. <https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report>. Accessed: 2023-08-11.
- [9] 2021. The Feodo Tracker Browse Botnet C&Cs. <https://feodotracker.abuse.ch/browse/>. Accessed: 2023-08-11.
- [10] 2021. The SSL Blacklist (SSLBL) . <https://ssllbl.abuse.ch/blacklist/sslipblacklist.txt>. Accessed: 2023-08-11.
- [11] 2021. WatchGuard Threat Lab Reports 91.5Arrived over Encrypted Connections in Q2 2021. <https://www.watchguard.com/wgrd-news/press-releases/watchguard-threat-lab-reports-915-malware-arrived-over-encrypted>. Accessed: 2023-08-11.
- [12] 2022. Spoiler: New ThreatLabz Report Reveals Over 85Attacks Are Encrypted. <https://www.zscaler.com/blogs/security-research/2022-encrypted-attacks-report>. Accessed: 2023-08-11.
- [13] 2023. <https://lists.blocklist.de/lists/all.txt>. Accessed: 2023-08-11.
- [14] 2023. A Research-Oriented Top Sites Ranking Hardened Against Manipulation. <https://tranco-list.eu/list/JXP6Y/1000000>. Accessed: 2023-08-11.
- [15] 2023. Censys. <https://search.censys.io/search/definitions?resource=hosts>. Accessed: 2023-08-11.
- [16] 2023. Censys Search. <https://search.censys.io/>. Accessed: 2023-08-11.
- [17] 2023. The CINS Score CI-Badguys list. <https://cinsscore.com/list/ci-badguys.txt>. Accessed: 2023-08-11.
- [18] 2023. The Darklist IP blacklist. <https://darklist.de/raw.php>. Accessed: 2023-08-11.
- [19] 2023. Easily Identify Malicious Servers on the Internet with JARM. <https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a/>. Accessed: 2023-08-11.
- [20] 2023. Semi Automated Machine Learning Pipeline. <https://github.com/alexdrk14/SAMPLP>. Accessed: 23-06-23.
- [21] 2023. Shodan Facet Analysis. <https://beta.shodan.io/search/facet?query=http&facet=ssljarm>. Accessed: 2023-08-11.
- [22] 2023. Shodan Search Engine. <https://www.shodan.io/>. Accessed: 2023-08-11.
- [23] 2023. SSlyze: A fast and powerful SSL/TLS scanning tool. <https://github.com/nabla-c0d3/sslyze>. Accessed: 2023-08-11.
- [24] 2023. TestSSL. <https://testssl.sh/>. Accessed: 2023-08-11.
- [25] 2024. Dataset of paper "Fingerprinting the Shadows: Unmasking Malicious Servers with Machine Learning-Powered TLS Analysis". <https://doi.org/10.5281/zenodo.10655329>. Accessed: 2024-02-18.
- [26] Luai Al Shalabi, Ziyad Shaaban, and Basel Kasasbeh. 2006. Data mining: A preprocessing engine. *Journal of Computer Science* 2, 9 (2006), 735–739.
- [27] Blake Anderson and David McGrew. 2019. Tls beyond the browser: Combining end host and network data to understand application behavior. In *Proceedings of the Internet Measurement Conference*. 379–392.
- [28] Blake Anderson, Subharthi Paul, and David McGrew. 2018. Deciphering malware's use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques* 14 (2018), 195–211.
- [29] Brandon Enright Lucas Messenger Adam Weller Andrew Chi Shekhar Achary Blake Anderson, David McGrew. 2019. Mercury: A network metadata tool for capturing and analysis. <https://github.com/cisco/mercury>. Accessed: 2023-08-11.
- [30] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.
- [31] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. 2015. Xgboost: extreme gradient boosting. *R package version 0.4-2* 1, 4 (2015), 1–4.
- [32] Yige Chen, Tianning Zang, Yongzheng Zhang, Yuan Zhou, and Yipeng Wang. 2019. Rethinking encrypted traffic classification: A multi-attribute associated fingerprint approach. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 1–11.
- [33] Bill Hudson David McGrew, Blake Anderson and Philip Perricone. 2017. Joy. <https://github.com/cisco/joy>. Accessed: 2023-08-11.
- [34] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. 2015. A search engine backed by Internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 542–553.
- [35] Peter Emerson. 2013. The original Borda count and partial voting. *Social Choice and Welfare* 40 (2013), 353–358.
- [36] Jon Fraenkel and Bernard Grofman. 2014. The Borda Count and its real-world alternatives: Comparing scoring rules in Nauru and Slovenia. *Australian Journal of Political Science* 49, 2 (2014), 186–205.
- [37] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention.. In *NDSS*.
- [38] Ralph Holz, Jens Hiller, Johanna Amann, Abbas Razaghpanah, Thomas Jost, Narseo Vallina-Rodriguez, and Oliver Hohlfeld. 2020. Tracking the deployment of TLS 1.3 on the Web: A story of experimentation and centralization. *ACM SIGCOMM Computer Communication Review* 50, 3 (2020), 3–15.
- [39] Mahdi Jafari Siavoshani, Amirhossein Khajepour, Amir Mohammad Ziaei Bideh, Amirali Gattmiri, and Ali Taheri. 2023. Machine learning interpretability meets tls fingerprinting. *Soft Computing* 27, 11 (2023), 7191–7208.
- [40] Jeff Atkinson John B. Althouse and Josh Atkins. 2017. JA3. <https://github.com/salesforce/ja3>. Accessed: 2023-08-11.
- [41] Kinan Keshkeh, Aman Jantan, Kamal Aliyean, and Usman Mohammed Gana. 2021. A Review on TLS Encryption Malware Detection: TLS Features, Machine Learning Usage, and Future Directions. In *Advances in Cyber Security: Third International Conference, ACeS 2021, Penang, Malaysia, August 24–25, 2021, Revised Selected Papers 3*. Springer, 213–229.
- [42] Hyundo Kim, Minsu Kim, Joonseo Ha, and Heejun Roh. 2022. Revisiting TLS-Encrypted Traffic Fingerprinting Methods for Malware Family Classification. In *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 1273–1278.
- [43] Maciej Korczyński and Andrzej Duda. 2014. Markov chain fingerprinting to classify encrypted traffic. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 781–789. <https://doi.org/10.1109/INFOCOM.2014.6848005>
- [44] 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 *Proceedings of the Internet Measurement Conference 2018*. 415–428.
- [45] Martin Laštovička, Stanislav Špaček, Petr Velan, and Pavel Čeleda. 2020. Using TLS fingerprints for OS identification in encrypted traffic. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–6.
- [46] Xigao Li, Babak Amin Azad, Amir Rahmati, and Nick Nikiforakis. 2021. Good bot, bad bot: Characterizing automated browsing activity. In *2021 IEEE symposium on security and privacy (sp)*. IEEE, 1589–1605.
- [47] Antonio Nappa, Zhaoyan Xu, M Zubair Rafique, Juan Caballero, and Guofei Gu. 2014. Cyberprobe: Towards internet-scale active detection of malicious servers. In *In Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS 2014)*. The Internet Society, 1–15.
- [48] Chaeyeon Oh, Joonseo Ha, and Heejun Roh. 2021. A survey on TLS-encrypted malware network traffic analysis applicable to security operations centers. *Applied Sciences* 12, 1 (2021), 155.
- [49] Eva Papadogiannaki and Sotiris Ioannidis. 2021. A survey on encrypted network traffic analysis applications, techniques, and countermeasures. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–35.
- [50] Eva Papadogiannaki and Sotiris Ioannidis. 2023. Pump Up the JARM: Studying the Evolution of Botnets using Active TLS Fingerprinting. In *Proceedings of the 28th IEEE Symposium on Computers and Communications (ISCC)*.
- [51] Muhammad Talha Paracha, Daniel J Dubois, Narseo Vallina-Rodriguez, and David Choffnes. 2021. IoTLS: understanding TLS usage in consumer IoT devices. In *Proceedings of the 21st ACM Internet Measurement Conference*. 165–178.
- [52] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. 2017. Studying TLS usage in Android apps. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 350–362.
- [53] Shahbaz Rezaei, Bryce Kroencke, and Xin Liu. 2019. Large-scale mobile app identification using deep learning. *IEEE Access* 8 (2019), 348–362.
- [54] Ivan Ristic. 2009. HTTP Client Fingerprinting using SSL Handshake Analysis. <https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html>. (2009). Accessed: 2023-08-11.
- [55] Ivan Ristić. 2012. Sslhalf. <https://github.com/ssllabs/sslhalf>. Accessed: 2023-08-11.
- [56] Donald G Saari. 1985. *The optimal ranking method is the Borda Count*. Technical Report. Discussion paper.
- [57] Meng Shen, Zhenbo Gao, Liehuang Zhu, and Ke Xu. 2021. Efficient fine-grained website fingerprinting via encrypted traffic analysis with deep learning. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.
- [58] Alex Shevtsov, Despoina Antonakaki, Ioannis Lamprou, Polyvios Pratikakis, and Sotiris Ioannidis. 2023. BotArtist: Twitter bot detection Machine Learning model based on Twitter suspension. *arXiv preprint arXiv:2306.00037* (2023).
- [59] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, and Georg Carle. 2023. DissecTLS: A Scalable Active Scanner for TLS Server Configurations, Capabilities, and TLS Fingerprinting. In *International Conference on Passive and Active Network Measurement*. Springer, 110–126.

- [60] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, Georg Carle, Claas Grohnfeldt, Michele Russo, and Daniele Sgandurra. 2022. Active TLS stack fingerprinting: characterizing TLS server deployments at scale. *arXiv preprint arXiv:2206.13230* (2022).
- [61] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkilainen. 2010. Naïve Bayes. *Encyclopedia of machine learning* 15, 1 (2010), 713–714.
- [62] Ziqing Zhang, Cuicui Kang, Gang Xiong, and Zhen Li. 2019. Deep forest with LRRS feature for fine-grained website fingerprinting with encrypted SSL/TLS. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 851–860.

A APPENDIX

A.1 Data Collection

In this section, we present some details regarding the blocklists used as input in this work. As stated in the webpage of Darklist.de [18], “Darklist.de is an IP blacklist that uses multiple sensors to identify network attacks (e.g. SSH brute force) and spam incidents”.

The webpage of SSLBL [10] states that “The SSL Blacklist (SSLBL) is a project of abuse.ch with the goal of detecting malicious SSL connections, by identifying and blacklisting SSL certificates used by botnet C&C servers”. Concerning Blocklist.de, the following is stated in their webpage [13]: “In the Export / DNS list all IP addresses which have in the past 48 hours executed an attack on our systems and/or partners”. Finally, in the webpage of CINS Army [17], it is reported that the blacklist “consists of IP addresses that meet one of two basic criteria: 1) The IP’s recent Rogue Packet score factor is very poor, or 2) The IP has tripped a designated number of ‘trusted’ alerts across a given number of our Sentinels deployed around the world”. Concerning the Feodo blacklist, all the IP addresses are associated with botnets [9].

We consider a sample as “successful” (also referred to as “Completed”) in cases where the server properly terminates a connection. A sample is marked as “disrupted” (also referred to as “Disrupted”) when a server consistently insists with a specific Server Hello message (e.g., using a fixed cipher suite that is not provided in the cipher suite list in the Client Hello message, etc.). Finally, a sample is marked as “Incomplete” when there is a time-out in the connection. The numbers that we have calculated regarding the connection attempts that we performed, follow. “Completed” connections: (1) Tranco list 79.52%, (2) Blocklists 5.53%, (3) QakBot 79.40%, (4) BazarLoader 94.93%, (5) BumbleBee 97.08%, (6) Dridex 85.83%, (7) Emotet 43.94%. “Refused” connections that correspond to non-responsive servers in the lists: (1) Tranco list 8.26%, (2) Blocklists 42.85%, (3) QakBot 6.55%, (4) BazarLoader 0%, (5) BumbleBee 1.35%, (6) Dridex 7.71%, (7) Emotet 0.15%. “Incomplete”/“Disrupted” connections : (1) Tranco list 12.21%, (2) Blocklists 51.61%, (3) QakBot 14.04%, (4) BazarLoader 5.06%, (5) BumbleBee 1.56%, (6) Dridex 6.45%, (7) Emotet 55.90%. For the evaluation phase we opted to retain only the “Completed” files to ensure more robust results in our classification. “Incomplete”/“Disrupted” samples can be used for fingerprinting, if the features missing are eliminated.

A.2 Parameters Extraction

Table 7 contains a proportion of the parameters extracted from each packet capture file from our database. Each parameter is derived from each individual *Server Hello* and it is presented with the corresponding name from our source code alongside with a small description. We select a wide range of options available for

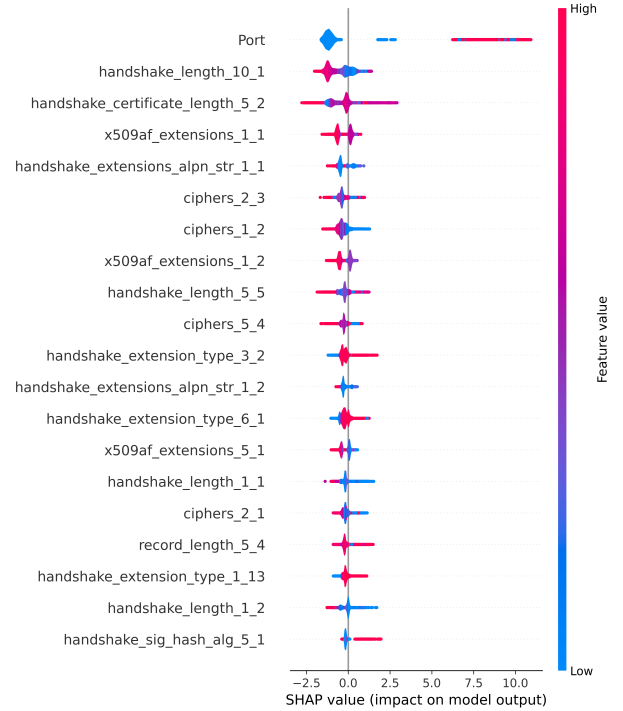


Figure 4: SHAP decision explanation of the binary classification model. Illustrates the variances among the top 20 significant features and their influence on the final model’s decision (i.e. `handshake_length_10_1` corresponds to the handshake length extracted from the 10th handshake and the 1st *Server Hello*).

feature selection. Together with the exhaustive approach, we are allowed to exponentially increase the total number of parameters. We aim to generate an extensive collection of 20K potential features and let machine learning choose the optimal ones. Since manually testing every possible combination is impractical, even impossible, we leveraged a machine learning pipeline for accomplishing this.

A.3 Binary Classification

A.3.1 Machine Learning. Figure 4 shows the insights about feature values and the effect of those feature values over the final model decision of the binary classification. More specifically, in the y-axis we can see the most significant features which have the biggest influence over the model decision. Each of the presented features have a range of values, which are represented between blue (low feature values) and red (high feature values) colors. Furthermore, the x-axis represents the impact of the particular feature value over the final model decision (SHAP value). The higher SHAP values push the model decision towards the positive decision, in our case represented as the malicious sample prediction, and lower values push the model towards the negative decision (benign class). For example, samples with low value of port number will push the model towards predicting a server as benign, and the higher port number samples will classify a server as malicious. Additionally, the

Table 7: A selection of descriptions for the TLS parameters extracted from each *Server Hello*

Parameter	Description
versions	TLS protocol version
ciphers	Cipher suite
record_length	Length of the TLS record
handshake_length	Total length of the handshake message
handshake_extensions_length	Length of the handshake extensions field in bytes
handshake_extension_type	Type of the handshake extension included in the negotiation
handshake_extensions_alpn_len	Length of the Application-Layer Protocol Negotiation
handshake_extensions_alpn_str_len	Length of the ALPN protocol string in the handshake extensions
handshake_certificate_length	Total length of the X.509 certificate chain
x509af_signedcertificate_element	Element containing the signed certificate in the X.509 certificate structure
x509af_version	Version number of the X.509 certificate
x509af_serialnumber	Serial number of the X.509 certificate
x509af_signature_element	Element containing the signature in the X.509 certificate
x509af_algorithm_id	Algorithm identifier used for the signature in the X.509 certificate
x509af_issuer	Issuer of the X.509 certificate
x509if_id	Identifier for the X.509 certificate
x509sat_utf8string	UTF-8 string in the X.509 Subject Attribute Type
x509af_notbefore	Certificate validity start date in the X.509 certificate
x509af_algorithm_element	Element containing the algorithm info in the X.509 certificate
x509af_extensions	Extensions included in the X.509 certificate
x509af_extension_id	Identifier for the extensions in the X.509 certificate
ber_bitstring_padding	Padding used for the BER-encoded bit strings
handshake_server_curve_type	Elliptic curve type used by the server
handshake_sig_hash_alg	Signature hash algorithm used

dot cluster thickness provides information about the distribution of tested values. In the case of the port number feature, it is clear that most of the samples have low values since our dataset is imbalanced and a higher percentage of tested samples belong to the benign class. Furthermore, as it is shown in Figure 4 the separation and the effect of feature values is not always binary since in some cases machine configuration may utilize low port number but additional features will lead to a classification decision towards malicious.

Table 8 visually presents a detailed breakdown of the number of accurately predicted samples through a confusion matrix of the binary classification.

Table 8: Confusion matrix of the selected binary model over the hold-out dataset

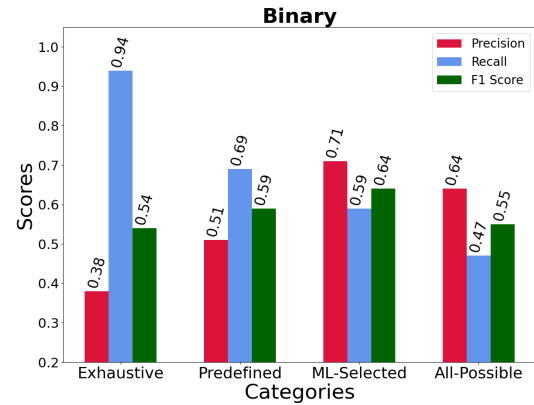
TN	FP	FN	TP
108,854	517	968	10,035

A.3.2 Fingerprinting. Figure 5 shows the precision, recall and F1 score that achieved each category during the binary classification process based on TLS Fingerprinting.

A.4 Multi-Class Classification

A.4.1 Machine Learning. Figure 6 shows the detailed predictions for each class during the multi-class classification.

Similarly to the binary classification, in the case of multi-class classification, we also utilize the SHAP explainability method to extract additional insights from our developed method concerning prediction decisions (Fig. 7). Unlike the previous explainability

**Figure 5: The performance of each category in binary classification based on fingerprinting**

method (binary classification), the current figure illustrates the mean impact of the most important features overall for class differentiation. In this representation, features are displayed on the y-axis, and the impact of each feature on a specific class can be discerned through the colored bars and their sizes. For instance, the feature *hs_len_5_1* has a significant impact on the QakBot category. The explainability presented offers general insights into the patterns captured by the developed classification model for each botnet category.

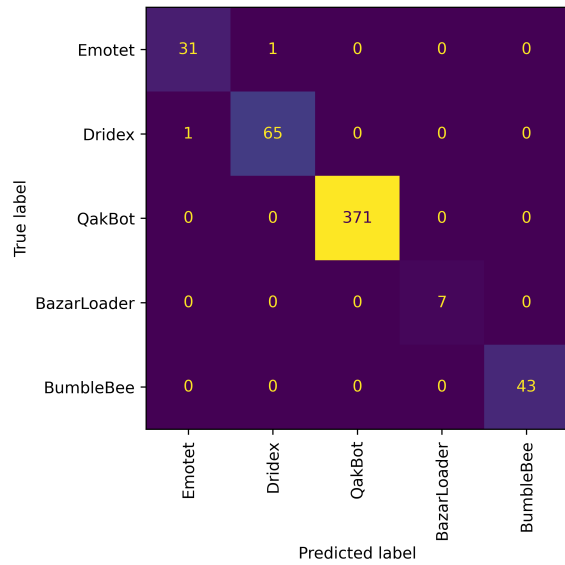


Figure 6: Confusion matrix of our multi-class classification model over the hold-out dataset portion.

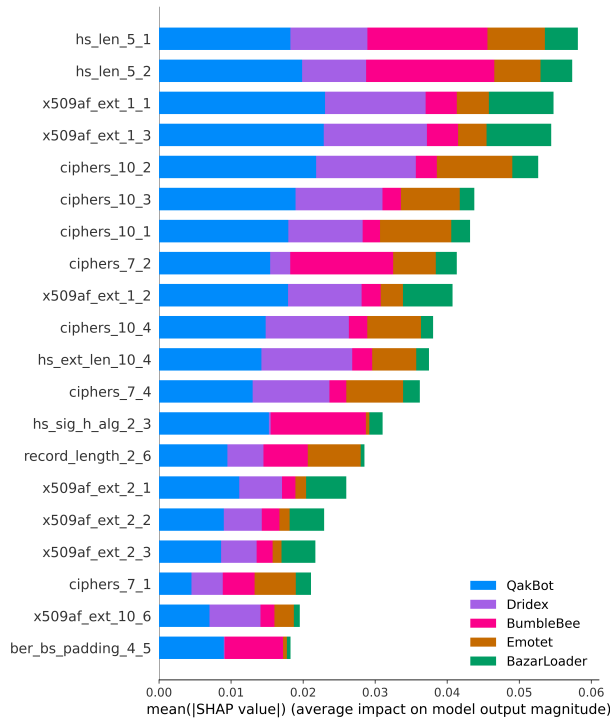


Figure 7: SHAP explanation of the secondary model based on multi-class classification. This figure illustrates the average impact of the top 20 features over the final model's decision.

Moreover, Figure 8 shows the SHAP multi-class model explainability for the QakBot botnet. The presented figure illustrates the top 20 features along with their decision distribution.

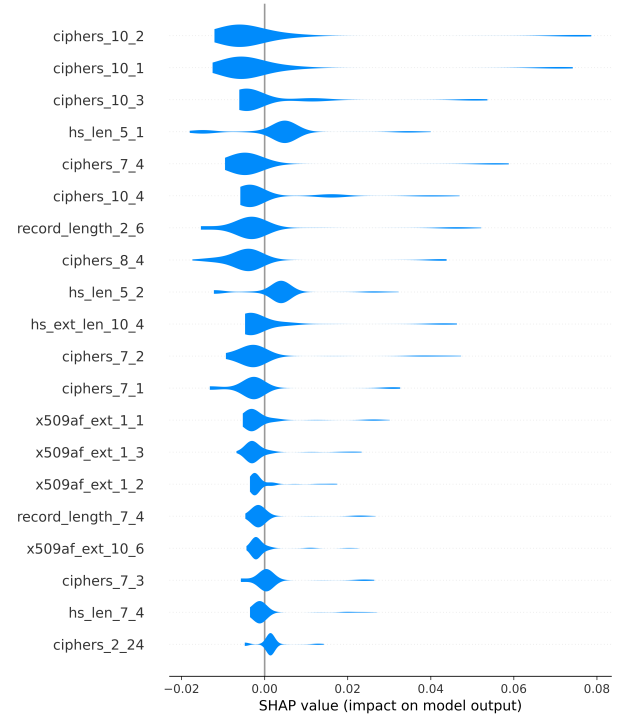


Figure 8: SHAP decision explanation (abbreviations are used) of the multi-class classification model of the QakBot botnet (i.e. ciphers_10_2 corresponds to the cipher suite extracted from the tenth handshake and the second Server Hello).

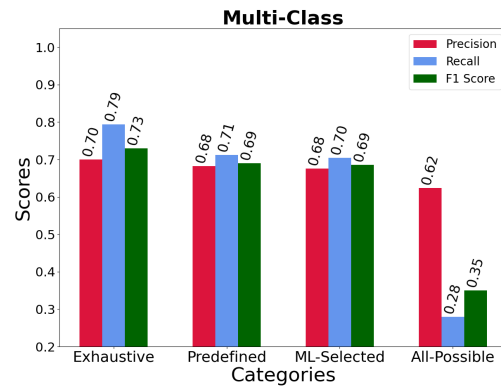


Figure 9: The performance of each category in multi-class classification based on fingerprinting

A.4.2 Fingerprinting. Figure 9 shows the precision, recall and F1 score that achieved each category during the multi-class classification process based on TLS Fingerprinting.