Intrusion Detection on QUIC Traffic: A Machine Learning Approach

Lama Al-Bakhat and Sultan Almuhammadi
Information and Computer Science Department
King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
Email: g201901830@kfupm.edu.sa, sultan@almuhammadi.com

***Abstract*—Since the introduction of QUIC protocol, a major change has affected the Internet transport layer, which improves user experience with some security threats. Developed by Google in 2012, QUIC provides a low latency, connection-oriented and encrypted transport. In addition to the encryption capability of QUIC, it overcomes many issues found in the current transport protocols, such as the high-latency connection establishment in TCP. On the other hand, studies on the security analysis of QUIC's key establishment showed several drawbacks. Moreover, the encryption mechanism of the protocol allows adversarial Command & Control (C2) packets to blind with regular QUIC traffic without raising any alarms. Therefore, in this study, we develop a machine learning approach based on fingerprinting that can be used in intrusion detection systems to detect malicious C2 QUIC traffic. To demonstrate the effectiveness of this approach, we conducted an experiment and tested the performance of six machine learning classifiers. The results show that by utilizing the fingerprint, most of the classifiers recognised malicious C2 traffic with an average accuracy of 98%.**

***Keywords*—QUIC Protocol, Fingerprinting, Intrusion Detection, Command & Control (C2) packet.**

I. Introduction

rapid development of mobile and web applications shed the light on several limitations in the current transport protocols, such as the latency overhead in TLS/TCP protocol [1]. As a result, many efforts were made to build new transport protocols that reduce latency and provide security mechanisms [2], [3], [4], [5]. One of the most potential protocols is QUIC, a new protocol built by Jim Roskind at Google in 2012 [2]. Unlike TLS, QUIC protocol runs on top of UDP and requires a single UDP round trip to establish a connection. Thus, reducing the three-way handshake latency of TCP. Besides, QUIC provides encryption by default using TLS 1.3 [2].

Although QUIC succeeds in achieving low latency performance while maintaining essential security goals such as confidentiality, it has several drawbacks [6]. For instance, a simple bit flipping or replay attack during the handshake process prevents the completion of the connection establishment, leading the process to fall back to TCP [6]. In addition to its security flaws, most open-source intrusion detection and prevention tools fail to inspect useful details about QUIC packets necessary to detect malicious behaviour [7]. Thus, hampering the ability for security professionals to identify illegitimate QUIC traffic. The aim of this research is to propose a network-based detection approach that utilizes a statistical fingerprinting technique to identify malicious Command & Control (C2) flows over QUIC protocol, and distinguish between the type of commands executed within the C2 traffic.

In this paper, legitimate QUIC traffic is captured by surfing web applications that support QUIC protocol, such as Google, cloudflare, and YouTube. In addition, malicious traffic is captured while performing attacks using a C2 Server with proper QUIC configuration. Next, statistical characteristics are learned from the captured network traffic to construct a network fingerprint. In order to classify flows, six machine learning models are applied, namely: Random Forest, Decision Trees, K-Nearest Neighbors, Support Vector Machines, Adaptive Boosting classifier, and Multilayer Perceptron. The results show that by utilizing the fingerprint, the classifiers recognise malicious C2 traffic with an average accuracy of 98%.

The remainder of this paper is structured as follows. Section II provides an overview on QUIC protocol, and Command & Control Servers. Section III highlights the related works on statistical fingerprinting for network intrusion detection and defines the scope of this paper. Subsequently, in section IV, the proposed fingerprint is outlined and the features vector is described. The experiment setup and traffic capturing are explained in Section V. The pre-processing and model training are summarized in Section VI. The classification results and performance analysis are detailed in Section VII. Finally, the strengths and limitations of the proposed approach are discussed in Section VIII, while the conclusion in Section IX summarizes the contribution of this work and highlights its potential extensions.

II. Background

From user experience point of view, before the implementation of QUIC, Quality of Experience (QoE) of web applications had some limitations. For instance, in HTTP adaptive streaming (HAS), delays in media segments can be caused by TCP head-of-line blocking and subsequently, degrade user experience [8]. Another factor is that the high rate of rebuffer events when streaming videos over TCP decreases viewer engagement. A study by [9] showed that increasing rebuffer rate by 0.3% caused viewers to stop watching the video after only 30% of the video content was viewed. In addition, the TCP three-way handshake for connection establishment introduce latency. As a result, QUIC protocol was developed to overcome these issues and several more [10], [2].

Connection establishment in QUIC consists of two cases summarized as follows:

- If the client is attempting to connect to a QUIC server for the first time, connection establishment will require one round trip time (1-RTT).

- If the client is attempting to connect to a QUIC server that has already established at least one connection with the client within a specific period of time, the connection will require zero round trip time (0-RTT).

The main idea of using 0-RTT is for the client and server to use previously cached connection parameters to create a new session without having to negotiate these parameters from the beginning. The following sections explain these two cases in more details.

A. 1-RTT Connection Establishment

When a client is connecting to a server for the very first time, the client sends an empty *hello* message containing a random connection ID (CID) which will be used later for encryption.

The server then sends a *Reject* message encompassing information to the client such as the Source Address Token (*STK*) and Server Certificate (SCFG). When the client receives the *Reject* message, it verifies the server's certificate, and sends another *hello* message containing a *nonce*, and the client's DH values (i.e. Diffie-Hillman parameters used in the key establishment process). Once the connection is successfully established, the encrypted communication begins, and a transport parameter *idle_timeout* is set to specify the allowed idle time period before a connection closes. If the connection remains idle for longer than *idle_timeout*, the timeout expires and the connection closes. The client will have to establishment a new connection to the server from the beginning. Otherwise, 0-RTT connection is established as described in the following Section.

B. 0-RTT Connection Establishment

If a client wishes to connect to a server that it has previously established a connection with, within the specified time period in *idle_timeout*, and the timeout is not expired, the client does not need to send an empty *hello* message to the server. Instead, it sends a hello message consisting of the previously acquired *STK* and *SCFG* values, along with a new CID, nonce, and DH values.

When the server receives the hello message, it verifies the provided information. If the verification succeeds, the client and server can start an encrypted connection. Otherwise, the server goes back to 1-RTT and generates a Reject message with new *STK* value. The rest of the connection establishment continues as in 1-RTT case. Figure 1 illustrates the connection establishment steps in QUIC for these two cases.

C. Command & Control (C2) Servers

The *command and control* (C2) is a malicious activity where an attacker exploits a compromised host and uses it as a server to store stolen data and commands or download them [11]. C2 servers can operate as a controlling facility in botnets to which targeted malware reports [11]. The support for QUIC was added to the Merlin C2 framework in 2018 to assist security practitioners and penetration testers [12]. This server is utilized in our work to perform malicious QUIC traffic.

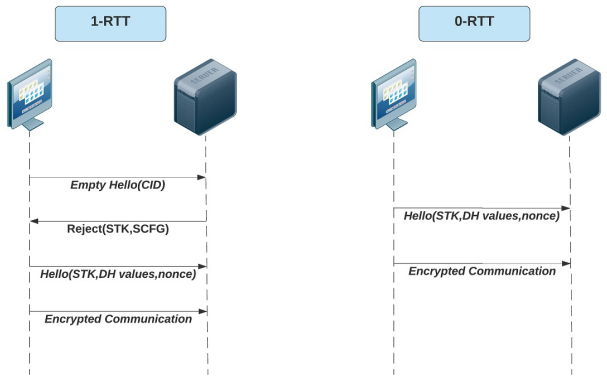


Figure 1: QUIC Connection Establishment in The Cases of 1-RTT and 0-RTT

III. Related Work

Intrusion detection and prevention includes many approaches, such as signature-based, behaviour-based, and anomaly-based detection. According to Muzammil et al. [13], the most accurate approach is signature-based intrusion detection. However, although this approach provides relatively high accuracy detection, it becomes ineffective if the attack signature is unknown. Statistical-based approaches overcome this issue with a slight compromise to accuracy [13]. These approaches leverage statistical information that can be learned from network traffic and form the basis of the detection process. Many research contributions are done in the development of statistical-based intrusion detection techniques. This section discusses related work on statistical-based techniques for intrusion detection.

Protocol fingerprinting was first introduced by Crotti et al. [14] in 2007. The researchers performed statistical fingerprinting of network traffic using three features: *packet size*, *inter-arrival time*, and *packet arrival order*. Although this method succeeded in identifying each protocol with a small error rate, it is limited to only three protocols: HTTP, SMTP, and POP3.

Dusi et al. [15] proposed an improved method from Crotti et al. protocol fingerprinting called Tunnel Hunter. This method used the same features in Crotti et al. and efficiently detected nearly 90% of encrypted tunnelling traffic in. However, these two methods suffer from the limitation of inadequate number of features. Moreover, the selected features are vulnerable to manipulation and could be altered to evade detection.

Some researchers utilize machine learning algorithms on statistical data [16], [17], [18], [19]. For instance, in 2009, Huang et al. [16] developed a clustering approach-based classification model for the purpose of identifying applications. They calculated 10 statistical features from network flows of four different applications, and applied k-nearest-neighbor (KNN) classifier on the datasets. Their classification results reached approximately 90% accuracy. However, the feature selection method used does not consider the correlation between application flows. As a result, the recall of MAIL flow, which shares similar behaviour to Instant Messaging (IM) flow, dropped

from 100% to 50% when IM flow was added to the datasets for evaluation.

Similarly, in 2011, Zhang et al. [17] proposed a classification method based on flow-clustering analysis to detect stealthy P2P traffic. This method clusters network traffic by exploiting statistical fingerprints of behavioural information corresponding to each host in the network. This method identifies P2P sets as hosts that remain active throughout the entire time an application is running. Thus, it does not take into consideration detection evading techniques that P2P botnets can use. For instance, if a botnet exchanges data periodically, or injects noise into traffic, the detection accuracy will be negatively affected.

On the other hand, Boero et al. [18] proposed a model which consists of a flow analyzer, and a machine learning-based filter. The flow analyzer captures network flows and extracts 14 features for each flow. While the filter applies machine learning classifiers on the extracted features to detect malware in each flow.

In 2019, Ahmed et al. [19] proposed a simple application fingerprinting approach that utilizes histogram-based approach. The approach utilizes both packet-level and flow-level statistical information. The fingerprints are used in a Dirichlet processing mixture model DPMM-based clustering algorithm to identify applications and detect DDoS attacks with a peak accuracy of 97%.

In order to evaluate machine learning classifiers' performance on statistical data, Muzzamil et al. [13] conducted a comparative study on five classification algorithms for statistical intrusion detection. The researchers used a public dataset published by Cambridge University, performed statistical analysis to extract 13 features from it, and applied machine learning classifiers on the extracted features. The most accurate classifiers according to their results are C4.5, Decision table, and OneR classifiers. Recently, Kamal and Almuhammadi [20] conducted a comparative analysis to assess the vulnerability of several VPN services to website fingerprinting attack by Cai et el [21]. This attack was proved to be successful when tested against multiple encrypted traffic such as Tor and SSH, as well as some defence mechanisms like traffic morphing and HTTPOS. In the attack, an adversary classifies web pages by collecting packet traces from page loads performed by the victim and modeling them using Hidden Markov Model (HMM). The results of Kamal and Almuhammdi's study showed that approximately 30-70% of VPN traffic is traceable by fingerprinting attacks.

The above mentioned approaches are capable of detecting encrypted traffic and protocols with high accuracy results. Since it is very challenging to detect encrypted traffic, especially malicious traffic, utilizing a similar technique to these approaches can assist in identifying malicious QUIC traffic. Therefore, we propose a statistical fingerprint that is detailed in the following section.

IV. Proposed Fingerprint

A network flow is defined as a bi-directional communication specified by the source and destination IP addresses, source and destination port numbers, and the type of protocol used. Specifically, a flow is denoted by the following: $(ip_{src}, ip_{dst}, port_{src}, port_{dst}, protocol)$. Therefore, a flow can be described as the quantity:

$$f_k = \{m_1, m_2, ..., m_{n_k}\},$$

where m_i is the i^{th} packet in the k^{th} flow. The total number of packets belonging to the k^{th} flow is denoted by n_k.

To build the flow fingerprint we proceed as follows. For the k^{th} flow $f_k \in F$, for a client A a server B, we consider the following features to represent the statistical fingerprint:

1) The number of pushed packets from A to B, denoted by Pkt_k^{AB}.
2) The number of pushed packets from B to A, denoted by Pk_k^{BA}.
3) Total number of packets in the flow, $n_k = Pkt_k^{AB} + Pkt_k^{BA}$.
4) The minimum packet length, defined as

$$\wedge L_k = \min\left(\{L_{k,i} \mid i = 1, \ldots, n_k\}\right),$$

where $L_{k,i}$ is the length of the packet.
5) The maximum packet length $\vee L_k$, given by:

$$\vee L_k = \max\left(\{L_{k,i} \mid i = 1, \ldots, n_k\}\right)$$

6) The average packet length $\bar{L}_k$, given by

$$\bar{L}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} L_i$$

7) The minimum inter-arrival time δ_k, defined by

$$\delta_k = \min\left(\{D_{k,i} : i = 1, \ldots, n_k\}\right)$$

8) The maximum inter-arrival time Δ_k, given by

$$\Delta_k = \max\left(\{D_{k,i} : i = 1, \ldots, n_k\}\right)$$

9) Thr standard Deviation of inter-arrival time σ_k, defined by

$$\sigma_k = \sqrt{\frac{1}{n_k - 1} \sum_{i=1}^{n_k} \left(D_i - \bar{D}_k\right)^2}$$

10) The minimum UDP segment Length $\wedge U_k$

$$\wedge U_k = \min\left(\{U_{k,i} : i = 1, \ldots, n_k\}\right)$$

11) The maximum UDP segment Length $\vee U_k$

$$\vee U_k = \max\left(\left\{U_{k,i} : i = 1, \ldots, N^k\right\}\right)$$

12) The average UDP segment Length $\bar{U}_k$

$$\bar{U}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} U_i \tag{1}$$

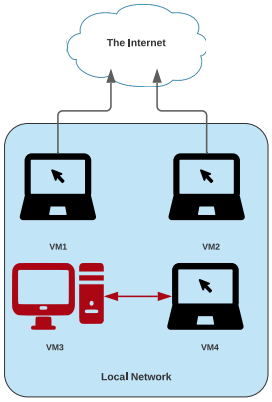


Figure 2: The network setup (VM1 and VM2 brwose the internet, while the C2 server VM3 is connected to the compromised host VM4)

V. Fingerprint Validation

The proposed fingerprinting approach is validated experimentally. The experiment is conducted in a virtual network comprising four hosts (labeled VM1, VM2, VM3, and VM4). First, VM1 and VM2 are running Ubnuntu version 20.04.1 and they are dedicated for browsing and capturing regular web traffic. VM3 is the attacker's machine running kali linux with merlin C2 server version 6.0 [22], while VM4 is the compromised host running a C2 agent process. Figure 2 explains the network setup.

Regular web browsing traffic is automated in VM1 and VM2 using Selenium WebDriver [23], along with subprocess python library [24]. QUIC protocol is enabled from the configuration settings in both Chrome and Firefox browsers. The automation process starts by running a wireshark instance to capture the traffic. While wireshark is running, a subprocess visits a list of websites that support QUIC. After the subprocess terminates, another subproccess runs that uses selenium webdriver to simulate user behaviour and browse YouTube, type a search query, and watch the first video that appears in the results for 20 seconds. At the end of these two subprocesses, the capturing process stops and the *pcap* file is saved for later analysis. The automated process is then repeated until sufficient dataset of traffic is obtained for model training.

To automate the capture of C2 traffic, pexpect Python library was utilized [25]. After the capturing by wireshark starts, pexpect process runs on the C2 server (VM3) on port 443 with QUIC protocol specified in the arguments. While the server is listening, a pexpect script on the compromised agent machine (VM4) connects the agent to the server through the server's URL. Once the connection is established, the server performs malicious activities such as downloading files from the agent, executing shell commands, or uploading files to the agent. After the C2 process is done, the connection closes and the captured *pcap* file is saved.

The obtained *pcap* files from web browsing and C2 traffic are used to extract the statistical fingerprint described in Section IV. The extracted fingerprint is then used to train the classification models discussed in Section VI-B. Figure 3 demonstrates an abstract diagram of the proposed approach.

VI. Evaluation

Dataset pre-processing is a vital first step before model training. In this work, the fingerprint dataset goes through a pre-processing phase summarized in the following Section.

A. Dataset Pre-processing

The pre-processing phase includes data cleaning, labeling, balancing and normalization. The details for each step are outlined in the following:

- **Data Cleaning**: while converting *pcap* files into *csv* files, it was found that, some of the captured files are damaged and contain corrupt data and thus they are removed. Null values and duplicates are then eliminated from the final *csv* files.
- **Labeling**: the datasets are labeled according to the type of traffic: *Normal*, or *Attack*. Within Attack data, malicious traffic is further labeled based on the type of activity that was performed: activities that involved executing shell commands are labeled as *cmd*, downloading files from the client are labeled as *download*, and uploading files to the client are labeled as *upload*.
- **Balancing**: in the original dataset, there are a total of 6980 instances, where 5968 are benign and 940 are malicious. Approximately 14% of the records are malicious. To overcome the class imbalance problem, we oversampled Attack instances through SMOTE, and randomly undersampled the majority class. The balanced dataset encompasses 2820 records for each class, with a total of 5640 instances.
- **Features Reduction**: the total number of packets, n_k, is not a useful feature since it can be directly derived by adding the first two features. Hence, it is removed from the fingerprint set. Moreover, some features are excluded for containing constant values (i.e. values with zero variance) across the samples. The reduced fingerprint contains the following features set,

$$\{Pkt_k^{AB}, Pkt_k^{BA}, \bar{L}_k, \Delta_k, \delta_k, \sigma_k, \bar{U}_k\} \quad (2)$$

- **Normalization**: The dataset is normalized such that all features are scaled to 0 and 1 to fit the training models.

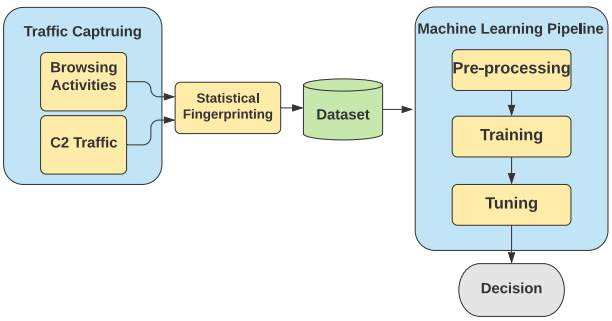


Figure 3: An abstract diagram of the proposed approach

B. Models Training

To evaluate the extracted fingerprint, we used 70% of the dataset obtained from the pre-processing phase for training, and the remaining 30% were used to testing. The following classifiers are used in the fingerprint validation experiment.

- Random Forest (RF): A class of ensemble learning algorithms where multiple de-correlated decision trees are trained and the results are averaged usually with bagging [26].
- Decision Tree (DT): A model that partitions the feature space into subspaces and perform classification using tree-like hierarchical decisions on those features [27].
- k-Nearest Neighbours (KNN): An instance-based classifier that groups similar instances by calculating the distance between the neighboring points.
- Support Vector Machine (SVM): A model represents instances in a space such that a hyperplane or multiple hyperplanes can be constructed to separate data points that belong to different classes from others. The hyperplane calculates the largest distance (i.e. margin) possible from each data point of any class [26].
- AdaBoost: A committee-based boosting algorithm that combines many *weak* classifiers (like decision trees) to produce strong decisions [26].
- Multi-Layer Perceptron (MLP): A class of unsupervised, feed-forward neural networks. An MLP consists of input, hidden, and output layers. Each node in the hidden and output layers contain a non-linear activation function to produce the output of the node [28].

The experiment is repeated several times with different parameters to tune the classifiers and optimize hyperparameters. For KNN classifier, an important parameter to consider is the number of neighbors k, we ran the algorithm with $k = \{1, 2, ..., 10\}$ and the best results obtained when $k = 1$. Another essential parameter is the base estimator in AdaBoost algorithm. When the default base estimator was used (i.e. Decision Trees with maximum depth of 1), AdaBoost did not produce satisfactory results. Thus, we replaced the default base estimator with Support Vector Classifier (SVC) which increased the accuracy by approximately 36%. As for MLP, we performed grid search to find the optimal hyperparameters. We found that the best results were when the hyperparameters were set to the following: *(Hidden layer sizes* $=(10, 30, 10)$*, Maximum iterations (epochs)* $= 100$*, activation function = ReLu, solver = adam, and alpha* $= 0.05$*).*

C. Performance Metrics

The performance of the classification models was evaluated using the following metrics:

- Accuracy: The percentage of the correct decisions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Precision: The number of positive instances predicted as positive attacks, divided by the total number of positives.

$$Precision = \frac{TP}{TP + FP}$$

- Recall (Sensitivity): The number of correctly predicted positive attacks, divided by itself plus the number of missed attacks.

$$Recall = \frac{TP}{TP + FN}$$

- F1 Score: The wighted average of precision and recall.

$$F1\ Score = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

VII. Classification Results

A comparison of the performance of the classification models discussed in section VI-B is analysed. The results are discussed in the following sections.

A. C2 vs Normal Browsing

We observed distinct differences in the nature of traffic produced by Merlin compared to regular browsing activities. One key difference is that Merlin has much fewer packets than legitimate traffic. This was also observed by [7]. In addition, Merlin breaks up data into packets of small lengths between 52 and 1280 bytes, which is a basic intrusion detection evasion technique [29]. On the other hand, normal browsing over QUIC contains larger packets that can reach a maximum length of 1378 bytes. Moreover, the average UDP length in a normal flow can reach approximately 1400 bytes while malicious segments are of length $400 - 500$ bytes. This justifies the high accuracy results obtained by all classifiers as shown in table I. However, it is important to note that these values are not constant and can be modified from Merlin's configuration settings.

Performance	Classifier					
	RF	DT	KNN	SVM	AdaBoost	MLP
Accuracy	98.78%	99.11%	**99.67%**	97.22%	98.64%	99.11%
Precision	98.78%	99.11%	**99.67%**	97.35%	98.69%	99.13%
Recall	98.78%	99.11%	**99.67%**	97.19%	98.62%	99.10%
F1 Score	98.78%	99.11%	**99.67%**	97.22%	98.64%	99.11%

Table I: Performance comparison in C2 vs Normal Brwosing

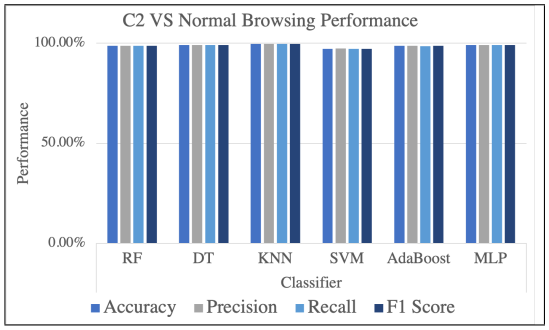
B. Identifying C2 Commands within Merlin

To further detect malicious commands within merlin, we examined three different activities: executing shell commands, downloading, or uploading files to the compromised host. Table II summarizes the obtained results.

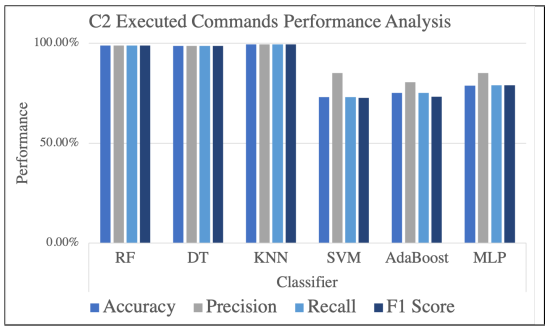
Performance	Classifier					
	RF	DT	KNN	SVM	AdaBoost	MLP
Accuracy	98.89%	98.78%	**99.56%**	73.00%	75.11%	78.89%
Precision	98.90%	98.78%	**99.56%**	85.06%	80.51%	85.06%
Recall	98.89%	98.78%	**99.56%**	73.10%	75.22%	78.95%
F1 Score	98.89%	98.78%	**99.56%**	72.61%	73.18%	79.01%

Table II: Performance Comparison in C2 Commands

The bar plots in Figure 4 report the performance of classifiers in C2 vs normal browsing, and C2 Executed Commands. The overall results show that among the classifiers, SVM was the least performing with an average accuracy of 73%, and KNN outperformed all classifiers with an accuracy of 99%.



(a) Performance Analysis for C2 traffic vs Normal browsing Activities.



(b) Performance Analysis for Predicting the type of command executed within Merlin Server

Figure 4: Bar Plots Demonstrating Performance Results

VIII. Discussion

Our research project has focused on composing a statistical fingerprint for network traffic to detect QUIC C2 flows. This section elaborates on the results and highlights the strengths of our method, as well as discussing possible evasions.

The aforementioned results show the effectiveness of the proposed fingerprinting approach in detecting malicious C2 traffic. However, the classification was done on just one C2 server. In order to verify the generalizability of the fingerprint, more tests can be done on different C2 servers running QUIC protocol with different configurations.

The proposed fingerprint relies heavily on statistical analysis of network traffic. If C2 flows are modified to imitate similar behaviour to regular client-server communications, they may be able to blind with normal traffic and evade detection. Sine the features that constitute the fingerprint are mutable, it will be more useful to incorporate other detection methods along with statistical fingerprinting to enhance the detection process. Such methods may include investigating persistent connections as they indicate potential C2 connections. Moreover, time is an important factor in identifying C2 traffic. In our method, flow duration was not considered in the fingerprint. Adding this feature along with total and cumulative time for each flow will assist in identifying persistent C2 connections.

In order to study the effect of reducing the features that constitute the fingerprint, we ran experimental tests using the full features set and compared the precision for each classifier with the precision obtained from the reduced fingerprint. From the comparison results in table III, we observe that the precision slightly increases for all classifiers in the case of C2 vs normal browsing classification. DT and Adaboost classifiers are able to achieve 100% precision. However, for C2 commands classification, precision dropped to approximately 4% and 8.5% for Adaboost and MLP, respectively.

IX. Conclusion

QUIC protocol is currently representing one of the most potential solutions to provide security services while maintaining low latency overhead. In this paper, we define a statistical fingerprint of network traffic, and assess its effectiveness in detecting malicious QUIC traffic by training six classification models. Our results show that most classifiers achieved high accuracy in identifying malicious C2 flows. The main contributions of this work are twofold:

- A statistical fingerprinting approach is proposed that assists in recognizing malicious QUIC C2 channels.
- The constructed fingerprint is evaluated by training six machine learning classifiers and investigating their performance on the proposed fingerprint.

For future work, we suggest validating the proposed fingerprint by exploring QUIC traffic for more C2 servers. In addition, this work can be extended by examining the effectiveness of the proposed fingerprint in detecting more sophisticated attacks such as Remote Access Trojans (RATs).

References

[1] P. Kumar and B. Dezfouli, "Implementation and analysis of quic for mqtt," *Computer networks (Amsterdam, Netherlands : 1999)*, vol. 150, pp. 28–45, 2019.

[2] "Quic, a multiplexed stream transport over udp - the chromium projects," https://www.chromium.org/quic, (Accessed on 08/09/2020).

[3] B. Ford, "Structured streams: a new transport abstraction." ACM, 2007, pp. 361–372.

[4] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a spdy'ier mobile web?" *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 2010–2023, 2015.

[5] "Rfc 4960 - stream control transmission protocol," https://tools.ietf.org/html/rfc4960, (Accessed on 08/09/2020).

[6] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is quic? provable security and performance analyses." IEEE, 2015, pp. 214–231.

[7] L. Decker, "Quic & the dead: Which of the most common ids/ips tools can best identify quic traffic?" *SANS Institute*, 2020.

[8] S. Arisu, E. Yildiz, and A. C. Begen, "Game of protocols: Is quic ready for prime time streaming?" *International Journal of Network Management*, vol. 30, no. 3, p. e2063, 2020.

[9] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "Developing a predictive model of quality of experience for internet video," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 339–350, 2013.

Fingerprint	Precision						Classification
	RF	DT	KNN	SVM	AdaBoost	MLP	
Complete	99.65%	100.00%	99.67%	99.30%	100.00%	99.83%	C2 vs Normal
Reduced	98.78%	99.11%	99.67%	97.35%	98.69%	99.13%	
Complete	98.78%	99.11%	99.67%	85.06%	84.88%	76.54%	C2 Commands
Reduced	98.78%	99.11%	99.67%	97.35%	98.69%	99.13%	

Table III: Complete vs. Reduced fingerprinting precision

[10] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 183–196.

[11] "Command and control [c&c] server - definition - trend micro usa," https://bit.ly/37dtwes, (Accessed on 08/11/2020).

[12] "Github - ne0nd0g/merlin: Merlin is a cross-platform post-exploitation http/2 command & control server and agent written in golang." https://github.com/Ne0nd0g/merlin, (Accessed on 08/11/2020).

[13] M. J. Muzammil, S. Qazi, and T. Ali, "Comparative analysis of classification algorithms performance for statistical based intrusion detection system," in *2013 3rd IEEE International Conference on Computer, Control and Communication (IC4)*. IEEE, 2013, pp. 1–6.

[14] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 5–16, 2007.

[15] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting," *Computer Networks*, vol. 53, no. 1, pp. 81–97, 2009.

[16] S. Huang, K. Chen, C. Liu, A. Liang, and H. Guan, "A statistical-feature-based approach to internet traffic classification using machine learning," in *2009 International Conference on Ultra Modern Telecommunications & Workshops*. IEEE, 2009, pp. 1–6.

[17] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy p2p botnets using statistical traffic fingerprints," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2011, pp. 121–132.

[18] L. Boero, M. Cello, M. Marchese, E. Mariconti, T. Naqash, and S. Zappatore, "Statistical fingerprint-based intrusion detection system (sf-ids)," *International Journal of Communication Systems*, vol. 30, no. 10, p. e3225, 2017.

[19] M. E. Ahmed, S. Ullah, and H. Kim, "Statistical application fingerprinting for ddos attack mitigation," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1471–1484, 2018.

[20] K. M. A. Kamal and S. Almuhammadi, "Vulnerability of virtual private networks to web fingerprinting attack," in *Advances in Security, Networks, and Internet of Things*. Springer, 2021, pp. 147–165.

[21] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 605–616.

[22] "Release quic protocol·ne0nd0g/merlin," https://github.com/Ne0nd0g/merlin/releases/tag/v0.6.0, (Accessed on 09/27/2021).

[23] "The selenium browser automation project :: Documentation for selenium," https://www.selenium.dev/documentation/en/, (Accessed on 09/27/2021).

[24] "subprocess — subprocess management — python 3.8.6 documentation," https://docs.python.org/3/library/subprocess.html, (Accessed on 09/27/2021).

[25] "Pexpect version 4.8 — pexpect 4.8 documentation," https://pexpect.readthedocs.io/en/stable/, (Accessed on 09/27/2021).

[26] T. Hastie, R. Tibshirani, and J. Friedman, *Random Forests*. New York, NY: Springer New York, 2009, pp. 587–604. [Online]. Available: https://doi.org/10.1007/978-0-387-84858-7_15

[27] V. A. Dev and M. R. Eden, "Gradient boosted decision trees for lithology classification," in *Proceedings of the 9th International Conference on Foundations of Computer-Aided Process Design*, ser. Computer Aided Chemical Engineering, S. G. Muñoz, C. D. Laird, and M. J. Realff, Eds. Elsevier, 2019, vol. 47, pp. 113 – 118. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128185971500199

[28] P. Marius, V. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, 07 2009.

[29] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, "Evasion techniques: Sneaking through your intrusion detection/prevention systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1011–1020, 2011.