How DNS undermines Tor's anonymity: steps towards an effective attack

Catarina Gonçalves
University of Porto
up201906638@fc.up.pt

Cristina Pêra
University of Porto
up201907321@fc.up.pt

Deborah Dore
University of Porto
up202202823@fc.up.pt

***Abstract*—An ongoing topic in Tor is threats to anonymity. The modern Tor user needs to exercise caution if they don't want to reveal their identity. DNS and their recursive resolver may be a contributing factor to a problem. In actuality, DNS requests are frequently transmitted in plain text. In this research, we want to determine the potential damage that an attack to DNS request can cause by implementing a model with features previously introduced by Siby, Juarez, Diaz, Vallina-Rodriguez and Troncoso.**

***Index Terms*—Tor, DNS, privacy, anonymity, fingerprint**

I. Introduction

The Onion Router project, known as Tor, is the most popular anonymity network today. Due to its nature, Tor is mostly used by anyone who wants to protect their privacy online. The "Onion Router" project is where it gets its name. Every package delivered over the Tor network is encrypted many times, creating several encryption levels that only a particular node can decipher. The IP address of the user is kept private since the node that decrypts the layer can only view the next layer to which the packet must be transmitted.

Even though Tor has also been used for malicious purposes, it still represents a significant step forward for democracy. When used for good, it can give the minority a platform to speak out without worrying about being found out and penalized. Tor was purposefully built to guard against risks to user anonymity, but it is still vulnerable to attackers who can monitor incoming and outgoing network traffic and correlate these flows. The consequences of this attack are multiple and severe.

The Domain Name System is a mechanism that changes website domain names into IP addresses. By watching DNS requests, which are frequently sent in plain text, an attacker could be able to determine which website a person is accessing - these attacks are called *fingerprint attacks*. As a matter of fact, DNS requests can be used to reveal a multitude of information about the user and the visited websites.

Encrypting DNS is one of the many ways to avoid sending DNS requests in plain text. Nowadays, most popular DNS providers, such as Google and Cloudflare, have adopted encryption mechanisms for DNS queries. Tor itself performs DNS requests through the relay node using whatever ISP's resolver is configured on the machine, as well as a public ISP address such as Google DNS resolver 8.8.8.8. Other countermeasures against DNS fingerprint are: *EDNS Padding* that adds padding to both DNS clients and resolvers, *DNS over TLS* which is a protocol for encrypting DNS requests that uses TLS to encrypt and authenticate communications and *DNS over HTTPS* that encrypts DNS communication by routing DNS requests through an encrypted HTTPS session. Because they all attempt to mask the real size of the packets and so hide information that the attacker may use to carry out an attack, these techniques—individually or in combination—are helpful.

In this work, we present the implementation and evaluation of a fingerprint attack using classifiers trained on the features introduced by Siby, Juarez, Diaz, Vallina-Rodriguez and Troncoso [4]. Since the DNS responses are smaller than web resources, these usually fit inside a simple TLS record therefore common features are inadequate. The authors of the papers proposed the use of n-grams of TLS record lengths in a trace. Packets going from the client to the resolver will have a positive n-gram, and a negative n-gram for packets going from the resolver to the client. An example is (-64,88,33,-33).

We will also draw insights from the work of Greschbach, Pulls, Roberts, Winter, and Feamster, who in their paper's main idea [3] expanded a K-Nearest Neighbors Classifier by enabling it to accept input from a list of websites that was obtained from watching DNS queries. We will adjust this model to make it compatible with our features. The two models will then be compared.

We shall distinguish between *open world attacks* and *closed world attacks* when addressing threats. In the first scenario, the attacker is aware of every website that the user might access, whereas in the second scenario, the user is able to access websites that the attacker is not monitoring. As a result, in the first scenario, the attacker can accurately identify the website the victim is viewing, whereas in the second scenario, the attacker can only tell apart between monitored and unmonitored websites.

The emphasis of our analysis will be open-world assaults because these are attacks that actually occur nowadays in

which the attacker has a list of websites and attempts to determine whether or not the user is visiting any of them. This type of attack can be quite dangerous: consider a scenario in which the government intends to restrict access to specific websites. Although countermeasures to avoid leaking information from communications are growing increasingly advanced, DNS requests are still frequently transmitted in plain text. The government can take advantage of this weakness to restrict access to monitored websites, particularly those that it does not want its citizens to see.

The purpose of this study is to attempt to strengthen this attack in order to draw attention to the possibility that attacks could get progressively worse.

II. Attack's implementation

This section will explain in detail the steps towards the construction of a model that is able to perform a fingerprint attack on a given dataset.

A. Dataset Collection

In order to setup the data collection for further feature extraction, we used *Vagrant* with *VirtualBox* as a provider. The *Vagrantfile* is configured with the chosen settings[1] for the experiment. The *Vagrantfile* specifies the number of virtual machines to use, the Linux distribution to employ, and the script to launch upon provisioning. Vagrant can be launched by executing the command `vagrant up` while in the same directory as the *Vagrantfile*. After provisioning, we must enter into the virtual machine and run the *browse_chrome* script while specifying the location where to keep the experiment's logs.

The *bootstrap file*, which is launched when initializing the virtual machines, contains instructions for configuring the Cloudflare Tunnel, downloading the experiment's libraries, and launching the script that will be used to make DNS queries. This script makes DNS requests using the Chrome Driver and domains that are among the top 1500 most-visited websites according to Alexa [10]. This will produce a large number of PCAP files that include packet information that will be extracted to create the dataset. The PCAP files will be added to the *pcap* directory, and the *logs* directory will store information on how the PCAP files were created.

B. Feature Extraction

Now that we've gathered all of the essential packets, it's time to extract the features we'll require for the next step of the attack. Each *PCAP file* contains packet data for a network. The name of the *PCAP file* identifies the requested website to the DNS resolver.
The information was extracted using a script. The script creates a dataset containing the TLS record lengths in a trace stored in the *PCAP file*. The resulting dataset is a *JSON file* with three fields: *sent*, *received*, and *order*. For each packet in the PCAP file, the length of the TLS record is stored in the field *sent* if the packet was sent from the user to the resolver or in the field *received* if the packet was received by the user. The communication sequence is stored in the column *order*: 1 for packets going from the client to the resolver and -1 in the opposite direction.

The dataset will be included in the list of websites being monitored if it originated through a DNS request referencing one of the websites specified in the Alexa list for the top 1500 websites [10]. Other website will produce the list of unmonitored websites.
Due to the short amount of time, Siby, Juarez, Diaz, Vallina-Rodriguez and Troncoso's LOC1 and OW datasets were used to integrate the dataset produced in this phase.

C. Implementation & Results

Two classifiers were used during this experiment: a Random Forest Classifier and a K Nearest Neighbors Classifier.

1) Random Forest Classifier: To begin, we replicated the classifier used in the work by Siby, Juarez, Diaz, Vallina-Rodriguez, and Troncoso [4].
This classifier is a Random Forest Classifier, which consists of an ensemble of decision trees that calculate the outcome by majority voting: the resulting class corresponds to the one that most trees elect. Because of its capacity to generalize to previously unknown samples and its resistance to overfitting, this type of classifier outperforms single decision trees most of the time [12].

The model learns what distinguishes a monitored website from an unmonitored one by using the n-grams, the previously extracted features. This is known as binary classification. In this first test, the n-grams have a length equal to 1. For example, for the trace (64,88,33,-33), the uni-grams are ((64),(88),(33),(-33)). This is an *open-world scenario* attack, and the model must determine from these traces if the user is visiting a website on the list of ones being tracked or not.

Instead of only adjusting the number of trees as in the previously referenced work, a randomized search has been carried out to find whose combination of hyperparameters maximizes the effectiveness of this model, thereby strengthening the classifier. The hyperparameters taken in considerations were: the number of trees, the maximum number of features to consider when looking for the best split, the maximum depth of each tree, the minimum number of samples required to be at a leaf node, the minimum number of samples required to split an internal node and whether bootstrap samples are used when building trees.

According to the result of the research put forth by Siby, Juarez, Diaz, Vallina-Rodriguez, and Troncoso, a classifier

[1]This configuration will only work on Intel processors due to incompatibility of *VirtualBox* with other systems. Also, the chosen linux distro for the experiment doesn't run on Apple M1.

trained on this novel set of features has a 0.7 ability to distinguish between classes in open-world attacks without countermeasures, showing a reasonable ability of the classifier to identify between monitored and unmonitored websites.

The classifier developed in this study, which has been trained using a hyperparameter search, has an Average F1-Score of 0.84, 0.84 for the monitored class and 0.85 for the unmonitored class. The Random Forest has an accuracy of 84%, demonstrating how attacks can be strengthened even more. It uses only 72 estimators instead of the 100 estimators used by the authors of the previously cited paper, making the training and testing of the model faster.

This classifier can distinguish between monitored and unmonitored websites with greater precision[2] and recall[3] than previously thought possible. In fact, its precision is higher for the monitored class (0.86) rather than the unmonitored class (0.83). But the recall is lower for the monitored class (0.82) rather than the unmonitored class(0.87). Therefore, the model is typically correct when it classifies a sample as belonging to the monitored class, but it returns few monitored samples as it sometimes classifies monitored classes as unmonitored.

The trade-off between a predictive model's positive predictive value and true positive rate for a given set of probability thresholds is summarized by precision-recall curves. The precision-recall curve for the Random Forest Classifier is shown in figure 1.

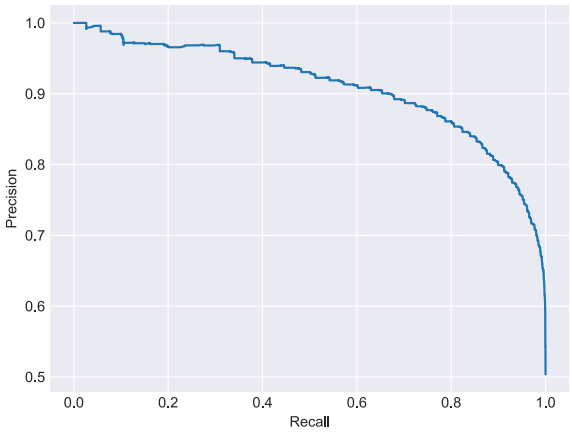

Fig. 1. Precision Recall Curve for Random Forest Classifier using uni-grams

2) K-Nearest Neighbors: A K-Nearest Neighbors Classifier was trained using the same features used for the Random Forest Classifier, with influence from the work of Greschbach, Pulls, Roberts, Winter, and Feamster [3]. The objective was to determine whether various model types might yield better or worse attacks. Neighbors-based categorization falls under the category of instance-based learning, often known as non-generalizing learning, because it doesn't try to build a broad internal model instead just maintains examples of the training data. A query point is assigned to the data class that has the most representation among its nearest neighbors based on a simple majority vote of those neighbors [11].

A randomized search was conducted to identify the optimal hyperparameters for the K-Nearest Neighbors as well. In particular, the following parameters were tuned: the weight function used in prediction, the number of neighbors and the metric to utilize for calculating distance.

According to the results, the classifier categorizes the monitored sample more accurately than the unmonitored sample, with an F1-Score of 0.82 for the monitored class and 0.80 for the unmonitored class. This classifier has a precision of 0.78 for the monitored class and 0.85 for the unmonitored class. But its recall is higher for the monitored class (0.86) and it is lower for the unmonitored class (0.76). It has overall a 81% accuracy. This data suggests that although the model returns a large number of monitored samples for the monitored class, these samples are frequently inaccurate when compared to the correct outcome.

Figure 2 shows the trade-off between precision and recall for this classifier.

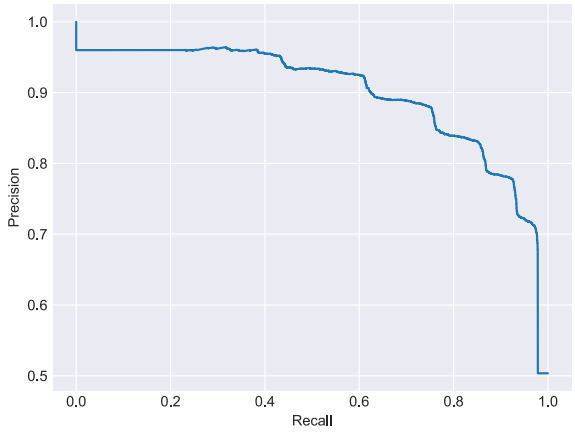

Fig. 2. Precision Recall Curve for KNearest Neighbor using uni-grams

3) Experiment with bi-grams: The same experiments were conducted using bi-grams instead of uni-grams. Using the same example as before, for the trace (64, 88, 33, -33) the uni-grams are represented by ((64), (88), (33), (-33)) while the bi-grams are ((64, 88),(88, 33),(33, -33)).

In terms of the monitored class, the results reveal an Average F1-Score of 0.83 for the Random Forest Classifier and 0.82 for K-Nearest Neighbor. Regarding the unmonitored

[2]fraction of relevant instances among the retrieved instances
[3]fraction of relevant instances that were retrieved

class, the results are 0.85 Average F1-Score for the Random Forest and 0.80 Average F1-Score for K-Nearest Neighbors.

Table I summarizes the results obtained during this work.

Type of model	n-grams	Average Precision	Average Recall	Average F1-Score	Accuracy
Random Forest Classifier	1	0.84	0.84	0.84	0.84
Random Forest Classifier	2	0.84	0.84	0.84	0.84
K-Nearest Neighbors	1	0.82	0.81	0.81	0.81
K-Nearest Neighbors	2	0.82	0.81	0.81	0.81

TABLE I
RESULTS OF EXPERIMENTS

Table I confirms that the model that performs a better attack, even with minimal difference, is the Random Forest Classifier. This model has better accuracy, precision, recall and F1-Score than the others. To evaluate the best features, the uni-grams or the bi-grams, we must look more in detail to the scores for the specific classes.

	precision monitored class	precision unmonitored class	recall monitored class	recall unmonitored class
Random Forest with uni-grams	0.86	0.83	0.82	0.87
Random Forest with bi-grams	0.86	0.82	0.81	0.87

TABLE II
PRECISION AND RECALL FOR THE RANDOM FOREST CLASSIFIER WITH DIFFERENT NGRAMS

As we can see from Table II, the difference between using uni-grams or bi-grams is minimal. The only notable difference is that Random Forest with uni-grams trains more quickly, which can be crucial since an attacker could train models for an attack more rapidly and therefore likely boost accuracy.

This small difference can be also visualized graphically using the ROC curves in Figure 3. The Receiver Operating Characteristic Curve demonstrate how well the model can distinguish between different classes. Figure 3 confirms our hypothesis about the model's minimal difference and reinforces our decision to consider uni-grams as the best features for this specific model due to the reduced training time required.

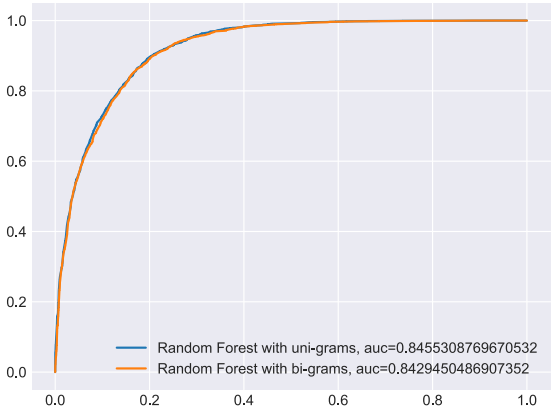

Fig. 3. Comparison between the usage of uni-grams and bi-grams using a Random Forest Classifier

III. CONCLUSION

In this paper, we discussed how protection against fingerprint attacks is still an unresolved question that could cause anonymity leaks. Regardless of the potential impact, DNS queries are often delivered in plain text. Even Tor's relay typically puts trust in open DNS servers.

We first demonstrated how to gather the dataset using Vagrant and VirtualBox in order to place up an attack using machine learning. The Top 1500 Most Visited Websites according to Alexa were used to acquire the dataset using DNS requests. Then we described how to extract data from the network packets that were gathered and utilize that data to create a new collection of features, called ngrams, in order to train a classifier. The K-Nearest Neighbor and Random Forest Classifier models were taken into account. After a careful evaluation, this last one turned out to be the most accurate. Both models were trained first using uni-grams and then with bi-grams. Despite the fact that both features were determined to be effective, uni-grams made it possible for the model to learn more quickly.

In conclusion, we highlighted how one might strengthen an attack using a DNS request to determine if the user is visiting a site on the list of ones that are being monitored. A potential organization may create a list of websites, record DNS requests, and determine if a user is accessing a website from the list. For the anonymity of users and everything it implies, this might be dangerous. User browsing might expose important details about the users that need to be kept private. More steps are yet to be taken to safeguard DNS requests and the privacy of users.

REFERENCES

[1] "The Tor Project: Privacy & Freedom Online," Tor Project — Anonymity Online. [Online]. Available: https://www.torproject.org/. [Accessed: 11-Oct-2022].

[2] E. Jardine, "The dark web dilemma: Tor, anonymity and online policing," SSRN Electronic Journal, 2015.

[3] B. Greschbach, T. Pulls, L. M. Roberts, P. Winter, and N. Feamster, "The effect of DNS on Tor's anonymity," Proceedings 2017 Network and Distributed System Security Symposium, 2017.

[4] S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, and C. Troncoso, "Encrypted dns -> privacy? A traffic analysis perspective," Proceedings 2020 Network and Distributed System Security Symposium, 2020.

[5] A. Mohaisen and K. Ren, "Leakage of .onion at the DNS root: Measurements, causes, and countermeasures," IEEE/ACM Transactions on Networking, vol. 25, no. 5, pp. 3059–3072, 2017.

[6] I. Karunanayake, N. Ahmed, R. Malaney, R. Islam, and S. K. Jha, "De-anonymisation attacks on Tor: A Survey," IEEE Communications Surveys; Tutorials, vol. 23, no. 4, pp. 2324–2350, 2021.

[7] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in USENIX Security. USENIX, 2014. URL: https://nymity.ch/tor-dns/pdf/Wang2014a.pdf

[8] "Doh! DNS over HTTPS explained," RIPE Labs. [Online]. Available: https://labs.ripe.net/author/gih/doh-dns-over-https-explained/. [Accessed: 19-Oct-2022].

[9] J. Hatwell, M. M. Gaber, and R. M. Azad, "Chirps: Explaining random forest classification," Artificial Intelligence Review, vol. 53, no. 8, pp. 5747–5788, 2020.

[10] "Alexa top websites," Alexa Top Websites – Most Popular Sites List 2022. [Online]. Available: https://www.alexatopwebsites.com/alexa-top-1001-1500-most-visited-websites

[11] "K Nearest Neighbors," scikit. [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html#classification.

[12] "Ensemble Methods," scikit. [Online]. Available: https://scikit-learn.org/stable/modules/ensemble.html#forest.