Adversarial Obfuscation for Domain Fronting

Steven R. Sheffey
Middle Tennessee State University

Ferrol Aderholdt
Middle Tennessee State University

Abstract

As the becomes more important as a source of information, internet censorship has become more pervasive. Tor aims to circumvent censorship, but adversaries are capable of blocking Tor traffic. Pluggable transports are used to obfuscate tor traffic to circumvent this. Meek uses domain fronting to emulate benign and trusted HTTPS traffic, with the hope that adversaries will avoid blocking it in order to avoid the collateral damage of blocking a reputable website. However, attacks using machine learning to differentiate domain-fronted traffic from regular HTTPS traffic pose a threat to the effectiveness of domain fronting. We propose the use of adversarial machine learning to model and reduce this threat.

1 Introduction

Meek provides excellent protection against metadata-based DPI attacks and resilience to censorship by hiding traffic inside encrypted HTTPS flows to reputable websites. However, due to its distinct activity pattern, it can be trivially identified by machine learning algorithms analyzing its traffic patterns.

Adversarial neural networks are powerful for modeling scenarios in which one party is trying to classify data, and another party is trying to produce fake or modified data that can fool the classifier. The result of training an adversarial model is a discriminator that is somewhat resilient to fakes, and a generator that can generate fake data, or modify data in a way that changes its perceived class. We believe that adversarial models can model internet censorship and help provide obfuscators such as Meek information to reduce its detectability.

2 Background

2.1 Censorship

- Internet censorship has become pervasive (china, iran, saudi arabia, russia, etc)
- The Tor network is used to circumvent internet censorship
- Censors have blocked Tor

2.2 Obfuscation

Obfuscators are used to circumvent protocol censorship or increase privacy. Tor uses Pluggable Transports

2.2.1 Forms of Obfuscation

- Randomization (obfsproxy, scramblesuit)
- Mimicry (Skypemorph, Marionette, FTE)
- Tunneling (Meek)
- Traffic shaping, chaff
- TODO: snowflake, other newer obfuscators

TODO: go into detail about meek

2.2.2 Attacks against Obfuscation

- Syntactic/Semantic (badly obfuscated traffic doesn't make sense semantically in its transformed protocol)
- Entropy
- Traffic statistical analysis
- TODO: others

TODO: go into detail about statistical analysis/machine learning

Wang et al. [?] were able to identify Meek traffic with a FPR (false positive rate) of 0.0002 on meek-amazon and 0.00006 on meek-google. This indicates that traffic analysis attacks using machine learning pose a major threat to traffic obfuscation and anti-censorship efforts.

3 Related Work

TODO: talk move about wang et al

TODO: find more papers identifying tor with machine learning

TODO: talk about the other adversarial transformation paper

4 Data Collection

We implement a framework that allows for reproducible, parallel generation of packet captures for HTTPS traffic with and without tor and pluggable transports.

4.1 Framework

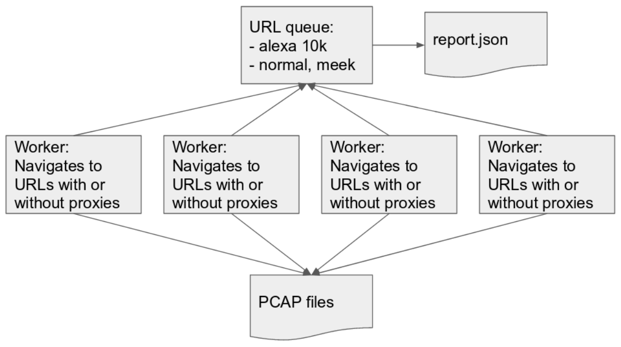
URL queue:
- alexa 10k
- normal, meek
report.json
Worker: Navigates to URLs with or without proxies
Worker: Navigates to URLs with or without proxies
Worker: Navigates to URLs with or without proxies
Worker: Navigates to URLs with or without proxies
PCAP files

Figure 1: Architecture of data collection containers

The data collection framework is composed of two types of containers: *URL Queue*, a work queue containing URLs and proxy types, and *Workers* that navigate to URLs using the given proxy and recort packet captures. Each type of container is run using Docker. Docker allows for reproducibility of the data collection process, parallelism, and isolation of the traffic captures by each worker. Each worker requests a piece of work from the URL queue. The worker then starts `tcpdump` to record all network traffic. If the work type requires startng TOR, then TOR is started with the given pluggable transport. In this work, we use `meek-azure` to tunnel our TOR traffic through Meek. Because the first start of TOR downloads and caches a large amount of data, we mark the first Tor work on each worker to allow it to be potentially accounted for. Once the TOR process has been started, we start Firefox using Selenium. After starting the browser, if TOR is used, we wait 10 seconds to ensure the pluggable transport has been properly initialized. Then, we use Selenium to navigate to the webpage. Because some webpages are constantly downloading content in a way that prevents Selenium from marking the page as finished loading, we wait for either common tag to appear (`<script>`) or TODO seconds to pass. After the page has been loaded, the worker shuts down the browser, TOR, and TCPDUMP. The worker then sends a report to the URL queue service containing information about the work done, and the filename of the `pcap` file. The overall architecture of the data collection framework is shown in Figure **??**.

4.2 Datasets

Each of our datasets contains packet captures for navigating to the top 10000 websites of the Alexa top 1M both without any proxies, and through Tor over Meek. We collect datasets from a home wired network (H_{normal}, H_{meek}), a university wired network (U_{normal}, U_{meek}), and an AWS `m5.2xlarge` instance (A_{normal}, A_{meek}).

4.3 Feature Extraction

In order to extract packet-level features, we use *BRO*, a DPI engine, to generate a set of connections for each `PCAP`. Using this list of connections, we use a program to associate packets with connections based on source IP, destination IP, source port, destination port, and timestamp. For each HTTPS connection, we sort the packets by timestamp, and then extract features from each packet. The features extracted are TCP payload size, and time since the last packet of the same direction.

The values for all connections in a `pcap` are aggregated into a histogram with logarithmic bins. Logarithmic bins are used because much more information is found in the smaller packet sizes and inter-arrival times than larger ones.

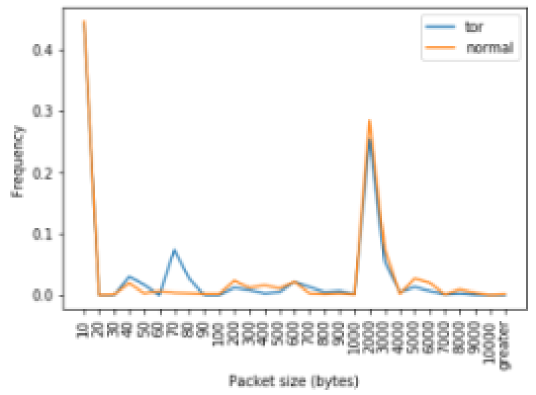

Figure 2: Averaged packet sizes from dataset H

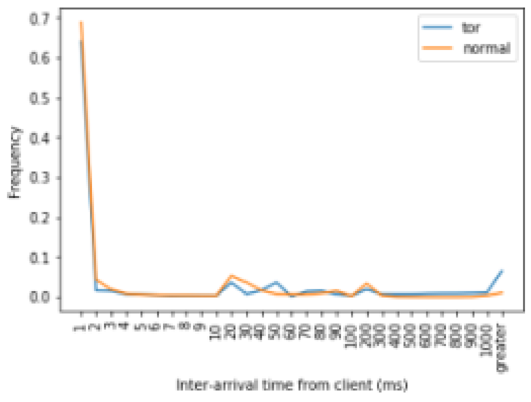

Figure 3: Averaged inter-arrival time (from client) from dataset H

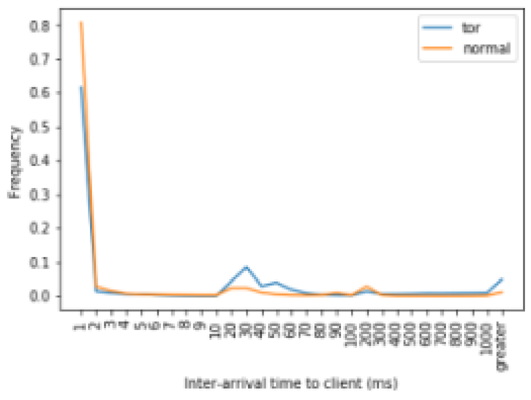

Figure 4: Averaged inter-arrival time (to client) from dataset H

TODO: explain feature these graphs.

5 Adversarial Model

5.1 Architecture

We use StarGAN [?] as a basis for obfuscation. StarGAN allows for adversarial transformation through application of a variety of features, though for this model, we only use a binary set of features (Meek or normal). Additionally, StarGAN performs well on datasets from multiple domains, which has potential applications in transforming obfuscated traffic of multiple types or locations.

Compared to StarGAN, we make some modifications to the model process. In our model, rather than complex convolutional neural networks, we use simple fully connected neural networks in our discriminator and generator. In addition, to ensure that minimal perturbation is made to the signatures, we add an additional constraint to StarGAN's generator loss function that minimizes mean absolute difference between the transformed signature and the input signature.

The classifier used to evaluate the model's efficiency is a basic feed forward neural network, with similar architecture to the discriminator. We also use a random forest classifier in the classification stages, to provide a measure of transformation effect on models that are not neural networks.

5.2 Training and Evaluation

The training and evaluation process is composed of 5 steps

1. Train the GAN (Discriminator and Transformer)
2. Train classifiers
3. Evaluate correctness of classifier
4. Evaluate correctness of classifier on transformed traffic
5. Evaluate correctness of classifier on transformed traffic with flipped labels

In order to validate this process, we split the dataset into three sets:

- 50% GAN training set
- 30% Classifier training set
- 20% Classifier training set

We shuffle the dataset randomly and then use K-fold validation to perform our training and evaluation process on all 6 orderings of these dataset splits.

6 Results

Our models succeded at fooling all tested classifiers with minimal perturbation

	Baseline	Transformed	Transformed & flipped class
Classifier	0.99	0.3	0.99
Discriminator	0.99	0.3	0.99
Random Forest	0.99	0.3	0.99

Figure 6: Effect of transformer on PR-AUC on dataset H

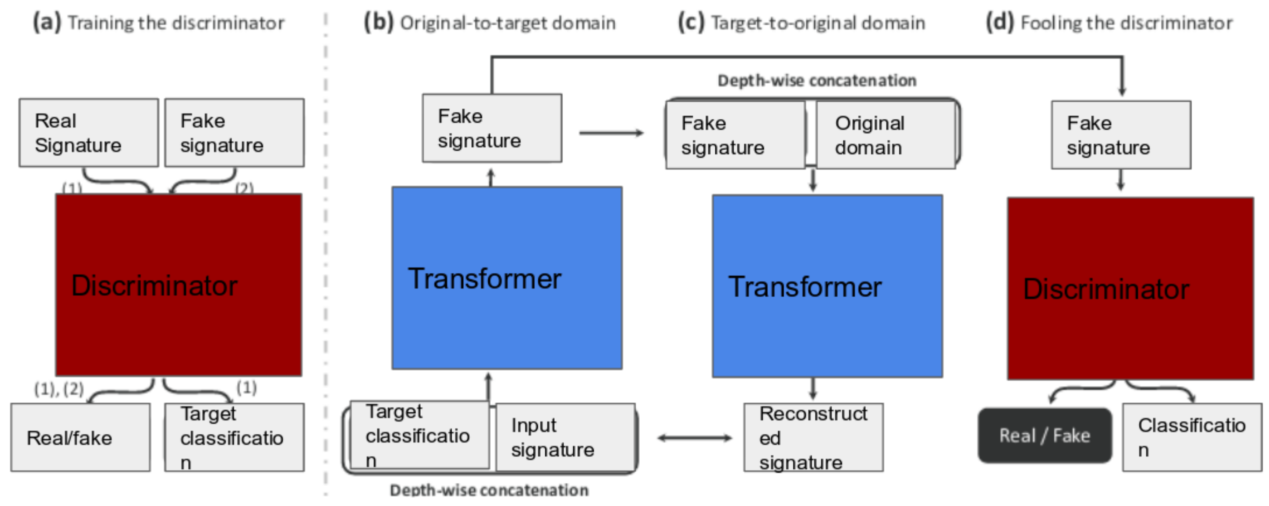

Figure 5: Architecture of our adapted version of StarGAN

	Baseline	Transformed	Transformed & flipped class
Classifier	0.99	0.3	0.99
Discriminator	0.99	0.3	0.99
Random Forest	0.99	0.3	0.99

Figure 7: Effect of transformer on PR-AUC on dataset U

	Baseline	Transformed	Transformed & flipped class
Classifier	0.99	0.3	0.99
Discriminator	0.99	0.3	0.99
Random Forest	0.99	0.3	0.99

Figure 8: Effect of transformer on PR-AUC on dataset A

7 Conclusions

Our results indicate that adversarial neural networks show promise in identifying and modifying statistical signatures of traffic that could be used to make the traffic more identifiable. Transformations performed by the neural network validate previous work's conclusions that the amount of ACKs in Meek traffic leaves it extremely vulnerable to machine learning attacks.

Adversarial perturbation could be used to modify Meek traffic in a way that reduces its detectability.

Availability

The traffic generation framework, feature extractor, and machine learning code for this work is open source, and can be accessed at

`https://github.com/starfys/packet_captor_sakura`