Improving Meek With Adversarial Techniques

Steven R. Sheffey
Middle Tennessee State University

Ferrol Aderholdt
Middle Tennessee State University

Abstract

As the internet becomes increasingly crucial to distributing information, internet censorship has become more pervasive and advanced. Tor aims to circumvent censorship, but adversaries are capable of identifying and blocking access to Tor. Meek, a traffic obfuscation method, protects Tor users from censorship by hiding traffic to the Tor network inside an HTTPS connection to a permitted host. However, machine learning attacks using side-channel information against Meek pose a significant threat to its ability to obfuscate traffic. In this work, we develop a method to efficiently gather reproducible packet captures from both normal HTTPS and Meek traffic. We then aggregate statistical signatures from these packet captures. Finally, we train a generative adversarial network (GAN) to minimally modify statistical signatures in a way that hinders classification. Our GAN successfully decreases the efficacy of trained classifiers, increasing their mean false positive rate (FPR) from 0.183 to 0.834 and decreasing their mean area under the precision-recall curve (PR-AUC) from 0.990 to 0.414.

1 Introduction

Tor [6] is often used to hide a user's internet traffic in order to gain privacy or circumvent censorship [7]. However, the IP addresses of Tor relays are public, and authorities such as China have used this information to prevent access to Tor [8] [29]. In response, researchers and activists developed traffic obfuscation methods that use techniques such as encryption and protocol mimicry to make it more difficult for censors to prevent access to Tor [7]. Many of these traffic obfuscation methods are made available with Tor Browser as "pluggable transports" [24].

One such pluggable transport is Meek, which offers strong protection against metadata-based Deep Packet Inspection (DPI) attacks using domain fronting [10]. Domain fronting obfuscates traffic by hiding traffic intended for a forbidden host inside the encrypted payload of an HTTPS connection to a permitted host [10]. This is achieved by manipulating the `Host` header of the underlying encrypted HTTP payload in order to take advantage of cloud hosting services that forward HTTP traffic based on this header. Domain fronting exploits censors' unwillingness to cause collateral damage, as blocking domain fronting would require also blocking the typically more reputable, permitted host. From the point of view of an adversary using DPI and metadata-based filtering, there is no difference between Meek and normal HTTPS traffic. All unencrypted fields in Meek traffic that could indicate its true destination such as DNS requests, IP addresses, and the Server Name Indication (SNI) inside HTTPS connection headers do not reveal its true destination.

However, recent work has shown that Meek is vulnerable to machine learning attacks that use side-channel information such as packet size and timing distributions to differentiate Meek traffic from normal HTTPS traffic [20], [28], [30]. Wang et al. [28] were able to differentiate Meek traffic from normal HTTPS traffic with a FPR as low as 0.0002 using a CART decision tree, while Yao et al. [30] achieved an accuracy of 99.98% on the same task using a hidden Markov model. Nasr et al. [20] were able to deanonymize Meek traffic with a FPR of 0.0005 using a neural network. Machine learning attacks against Meek pose a dangerous threat to its effectiveness, and strengthening Meek against these attacks remains an open problem.

Traffic obfuscation is fundamentally a conflict between an adversary attempting to detect unwanted traffic and a user attempting to modify their traffic in a way that circumvents this [7]. Generative Adversarial Networks (GANs) operate under a similar paradigm. GANs are typically composed of two components: a generator and a discriminator [12]. The goal of the generator is to generate realistic looking data, while the goal of the discriminator is to determine whether this synthetic data is real or fake. By training the generator and discriminator in unison, each can learn from the other until an equilibrium is reached.

In this work, we evaluate the efficacy of GANs in identifying and correcting identifiable patterns in Meek traffic.

We achieve this by first developing a method capable of generating large amounts of normal HTTPS and Meek packet captures, with a focus on efficiency and reproducibility. We then extract statistical signatures modeling the identifiable side-channel features of this traffic. Finally, we train a modified version of StarGAN [4], and evaluate its ability to reduce the FPR and PR-AUC of multiple trained classifiers.

2 Data Collection

The objective of our data collection method is to capture traffic generated by navigating to the same websites over both normal HTTPS and Meek, and to do so efficiently and reproducibly. Previous work analyzing Meek traffic uses sequential scripts [26], [28] or does not appear to specify their Meek data collection process [20], [30]. We use Docker [13] to allow our data collection process to be performed in parallel, and in a reproducible environment.

Docker is a platform based on reproducible environments known as containers [13]. Our data collection method is composed of two types of containers: a work queue and workers. The work queue manages and distributes a queue of data collection work. Each work item in the queue contains a URL and a proxy type. Workers navigate to URLs using the given proxy type, and produce packet captures for each piece of work.

Because Meek tunnels traffic through a single HTTPS connection [10], gathering individual Meek connections requires restarting the Meek process. This limitation presents a major bottleneck to our data collection process. In order to circumvent this, we use Docker Compose [14] to allow for any number of data collection workers to operate in parallel.

During data collection, each worker repeats the process shown in Figure 1.

1. Request URL and proxy from the work queue. If there is no more work, exit.
2. Start tcpdump to capture all packets.
3. Start Tor/Meek if applicable.
4. If using Meek, wait for 10 seconds to ensure Tor has been properly initialized, and to provide some measure of reducing network load.
5. Start Firefox using Selenium.
6. Navigate to the URL.
7. Wait for either an element with a common tag (`<script>`) to load, or 60 seconds to pass. This is done to speed up the data collection process and avoid getting stuck.
8. Thoroughly shut down Firefox, Tor/Meek, and tcpdump, in that order.
9. Send a report to the work queue containing information about the work done, and the filename of the generated PCAP file.

Figure 1: Overview of worker program.

2.1 Datasets

We collect datasets from a residential desktop (H), a university office desktop (U), and an Amazon Web Service (AWS) server (A). Datasets H and U were collected using Docker installed on NixOS hosts, while dataset A was generated using an AWS `m5.2xlarge` instance provisioned by `docker-machine`. Each dataset contains 20000 samples, created by navigating to the top 10000 websites of the Alexa top 1M dataset [1] using both regular HTTPS and Meek using the meek-azure bridge from Tor Browser. Datasets H and U were collected in Middle Tennessee, while dataset A was generated from the AWS `us-east-1` region (North Virginia) [25].

Our datasets contain HTTPS traffic, generated with and without Meek. However, HTTPS traffic does not encompass the scope of all Meek traffic. We only collect traffic from connections to the homepages of popular websites, but Tor Browser users may navigate to other pages, use hidden services, or communicate using other protocols. Our data collection framework may be extended to include hidden services, but is not suited to non-web traffic.

3 Feature Extraction

In this work, we analyze the following side-channel traffic features: histograms of TCP payload sizes and per-direction packet inter-arrival times. We ignore basic packet metadata such as IP addresses or TLS parameters. While Meek traffic may have distinct values for these features compared to the wide variety of HTTPS clients on the internet, we assume that these basic fields could be trivially modified. The side-channel features we analyze are acknowledged in the original implementation of Meek [10] and used by Wang et al. [28], Nasr et al. [20], and Yao et al. [30] to identify Meek. These features are identifiable weaknesses in Meek, but their statistical distribution may be modified through traffic shaping techniques; for example, Verma et al. [27] propose inserting extra data (chaff) into packets or delaying packet transmission in order to match a distribution generated by an adversarial neural network. HTTPOS [17] applies a similar technique to HTTP traffic.

We use Bro, a DPI engine, to aggregate packets from each PCAP into a set of HTTPS connections [21]. We then associate each packet with an HTTPS connection using its source

IP, destination IP, source port, destination port, and timestamp. All packets unrelated to HTTPS connections are ignored. As much more information is found in smaller payload lengths and inter-arrival times than larger ones, we aggregate these features into logarithmic bins. For TCP payload lengths, we use bins of size 10 from 0 to 100 bytes, size 100 from 100 to 1000 bytes, size 1000 from 1000 to 10000 bytes, and a single bin for packets larger than 10000 bytes. For packet inter-arrival times, we use bins of size 1 from 0 to 10 ms, size 10 from 10 to 100 ms, size 100 from 100 to 1000 ms, and a single bin for inter-arrival times above 1000ms. These bin sizes are similar to those used by Wang et al. [28].

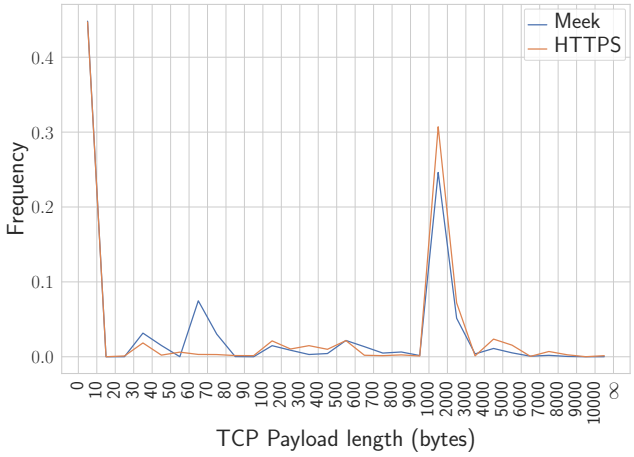

Figure 2: Average TCP payload length frequency

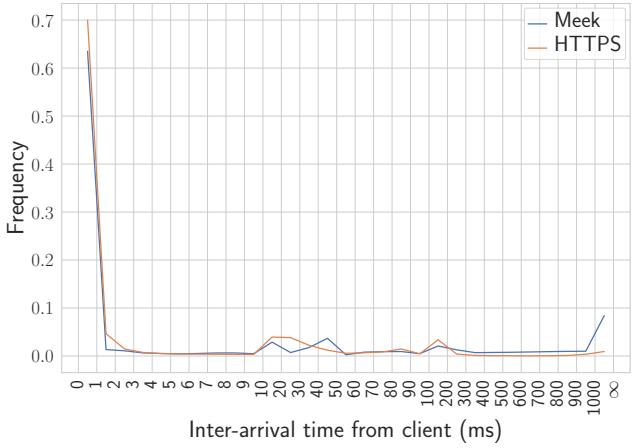

Figure 3: Average inter-arrival time frequency (from client)

Figures 2, 3, and 4 show the average frequencies of TCP payload sizes, inter-arrival times from client, and inter-arrival times to client, respectively over dataset H, defined in Section 2.1. One difference between normal HTTPS and Meek traffic can be seen in Figure 2 where Meek traffic has a much larger proportion of packets with payload size between 60 and

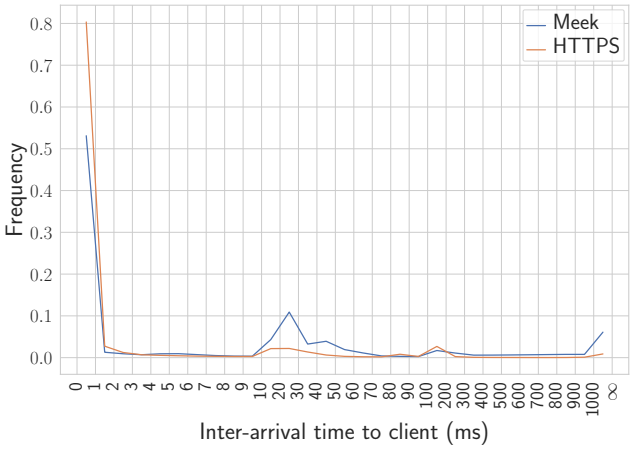

Figure 4: Average inter-arrival time frequency (to client)

70 bytes. Additionally, The inter-arrival times of Meek traffic in Figure 3 and Figure 4 seem to indicate a higher latency in Meek traffic. Our observed differences in TCP payload length distribution differ from the TCP payload length distribution measured by Fifield et al. [10], where Meek traffic exhibited a much larger number of payloads around 1400 bytes and a lack of payloads around 50 bytes. This may be due to a difference in data sources [10], or modifications to Meek [9]. Fifield et al. [10] compared Google traffic from Lawrence Berkeley National Laboratory to traffic generated by navigating to the Alexa top 500 over Meek. Since Meek's creation, it has introduced many changes such as HTTP/2 support [9], which can result in different traffic characteristics [18].

4 Adversarial Transformation

While typical GANs contain a generator, which generates adversarial data from noise, our model uses a transformer, which transforms a traffic signature from one class to another. This model is similar to an adversarial transformation network [2]. We use StarGAN [4] as a basis for our model. StarGAN is an adversarial transformation model that transforms images by selectively applying features from various domains and datasets. While StarGAN can support modifying any number of features, we only modify one binary feature: whether the traffic signature represents Meek traffic or normal HTTPS traffic. This allows us to significantly simplify StarGAN while still taking advantage of its powerful adversarial transformation features.

4.1 Architecture

In our model, we replace StarGAN's complex, multi-layered convolutional layers with a simple fully-connected hidden layer. While convolutional neural networks are useful for image classification tasks [16], our traffic signatures are very

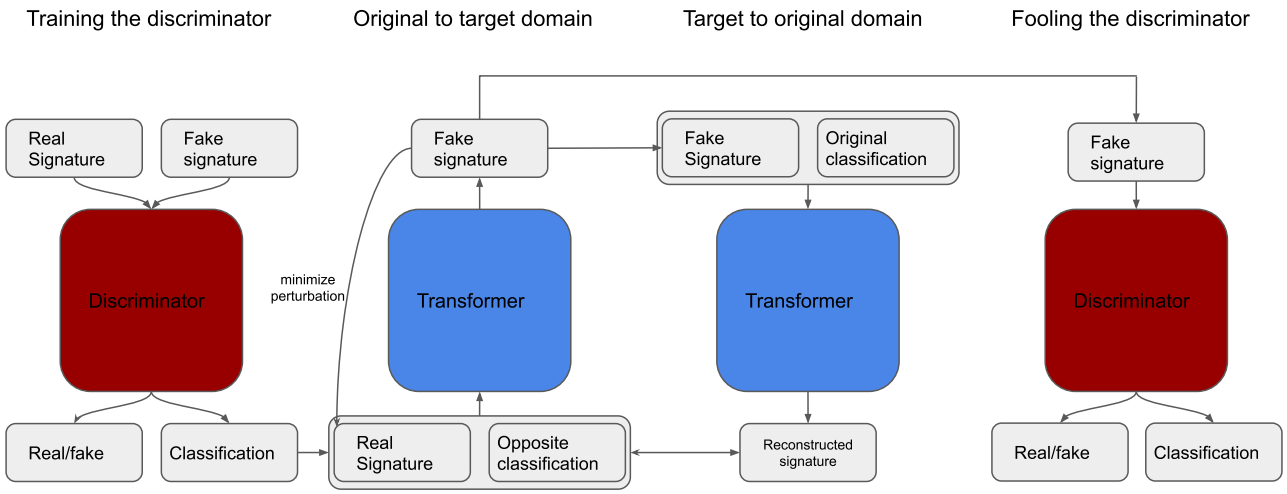

Figure 5: Training process for our GAN, adapted from the original figure in [4].

simple and can be classified using a very small number of parameters. The discriminator accepts a signature, contains a single fully-connected hidden layer of size 16, and outputs a label Y (the probability that the signature represents a Meek flow), and a source S (the probability that a signature was modified using a transformer). The transformer accepts a signature (X) and a target class, contains a single fully-connected hidden layer of size 128, and outputs a modified signature (X'). The transformer contains a larger hidden layer in order to avoid losing information during reconstruction.

4.2 Training

We train our model using a modified version of the StarGAN training process. For each training iteration, we train the discriminator using the following steps:

1. Retrieve a batch of 16 signatures X and labels Y from the training set. A label is 0 if the flow is a regular flow, and 1 if the flow is Meek.

2. Predict the flow's label and source, and calculate loss for the predictions. BCE is binary cross-entropy.

$$Y_{predicted}, S_{predicted} = D(X)$$
$$DLoss_{cls} = BCE(Y, Y_{predicted})$$
$$Dloss_{src} = -mean(S_{predicted})$$

3. Generate 16 random labels Y_{random}

4. Use T to transform X given Y_{random}

$$X' = T(X, Y_{random})$$

5. Calculate loss for the discriminator's source prediction over the transformed signatures. Class is ignored here, because the class of traffic does not matter if it is determined to be fake.

$$S'_{predicted} = D(X')$$
$$Dloss'_{src} = mean(S'_{predicted})$$

6. Calculate gradient penalty loss $DLoss_{gp}$, as defined in [4].

7. Calculate the final loss function for the discriminator

$$Dloss = DLoss_{src} + Dloss_{cls} + DLoss'_{src} + 10DLoss_{gp}$$

8. Perform gradient descent over the discriminator's weights to minimize $DLoss$ using the Adam optimizer [15].

Every 5 iterations, we train the transformer using the following steps:

1. Calculate loss for the discriminator's label and source prediction over the transformed signatures.

$$Y'_{predicted}, S'_{predicted} = D(X')$$
$$TLoss_{cls} = BCE(Y_{random}, Y'_{predicted})$$
$$TLoss_{src} = -mean(S'_{predicted})$$

2. Calculate the perturbation loss. This measures the mean absolute distance between unmodified and transformed traffic.

$$TLoss_{pert} = mean(|X - X'|)$$

3. Transform the transformed signature back to its original class using the transformer, and measure the mean absolute difference between the original and reconstructed signature.

$$X_{rec} = T(X', Y)$$
$$TLoss_{rec} = mean(|X - X_{rec}|)$$

4. Calculate the final loss function for the transformer

$$TLoss = TLoss_{cls} + TLoss_{src} + 10TLoss_{pert} + 10Tloss_{rec}$$

5. Perform gradient descent over the transformer's weights to minimize $TLoss$ using the Adam optimizer [15].

The signatures generated by our transformer represent a new distribution of packet sizes and timings that, if matched, would make Meek traffic appear similar to regular HTTPS traffic. However, introducing delays or extra data into a traffic stream in order to match this distribution introduces overhead [27]. In order to minimize this, we introduce an additional objective into our transformer's loss function called perturbation loss, defined above in step 2 of the training process. This measures the mean absolute difference between the original signature and the transformed signature. By introducing perturbation loss, we train the transformer to make minimal modifications to the traffic signature while simultaneously fooling the discriminator. By minimizing changes made by the transformer, we reduce the amount of work a traffic shaping method would have to do to modify the Meek traffic stream in order to fool classifiers.

In order to reduce overfitting, a situation in which neural networks generalize poorly due to relying on noise present in the training set, we introduce early stopping measures [3]. Our training process iterates repeatedly over the training set until both the discriminator loss $DLoss$ and transformer loss $TLoss$ have not decreased by 0.0001 over 2000 batches. This is to ensure that D and T cease training when they have reached an equilibrium.

4.3 Evaluation

To avoid biasing our experiments by evaluating models using data that they have been trained on, we split each dataset into three parts:

- 30% GAN training set (G_{train})
- 20% Classifier training set (C_{train})
- 50% Classifier testing set (C_{test})

Our training and evaluation process is composed of 8 steps:

1. Train the Discriminator (D) and Transformer (T) using G_{train}, as described in Section 4.2.
2. Train a neural network classifier and decision tree with C_{train}
3. Split the classifier training set in half

$$C_{train1}, C_{train2} = split(C_{train})$$

4. Transform one half of the classifier training set into the opposite class using the transformer, while retaining the original (unmodified labels). This is to simulate an adversary who is aware of the traffic modification scheme, and aims to classify modified traffic as its original class.

$$X_{train2}, Y_{train2} = C_{train2}$$
$$X'_{train2} = T(X_{train2}, 1 - Y_{train2})$$
$$C'_{train2} = X'_{train2}.Y_{train2}$$
$$C'_{train} = C_{train1} \bigcup C'_{train2}$$

5. Train a neural network classifier over C'_{train}
6. Evaluate the PR-AUC and FPR of all classifiers over C_{test}
7. Transform C_{test} using T

$$C'_{test} = T(C_{test})$$

8. Evaluate the PR-AUC and FPR of all classifiers over C'_{test}

The neural network classifiers are simple dense neural networks that accept a signature, contain a hidden layer identical to the discriminator, and output the probability that the signature represents Meek traffic. The decision tree is the default decision tree provided by scikit-learn [22], which is identical to the best-performing classifier type in Wang et al [28].

To avoid overfitting when training C, we separate C_{train} into a smaller C_{train} with 90% of its original size, and C_{val} containing 10% of C_{train}. Each epoch, we evaluate the loss of N using C_{val} to calculate the validation loss. If the validation loss has not decreased by 0.001 in 5 epochs (full iterations over C_{train}), we stop training the classifier.

We evaluate classifiers using PR-AUC and FPR. PR-AUC is a particularly useful metric when dealing with a domain in which the number of negative samples vastly outweighs the number of positive samples [5]. This suits traffic obfuscation well, as most internet users do not use Meek. Additionally, PR-AUC takes prediction confidence into account, and graphs precision vs recall based on a classifier's ability to confidently provide predictions [5]. FPR is commonly used when evaluating obfuscation methods, as falsely blocking a connection can cause degraded network performance [28]. Additionally, existing work [28] uses PR-AUC and FPR to measure obfuscator classification performance, allowing our work to be more readily comparable.

Finally, to increase confidence in our results, we use a method similar to K-fold validation. We shuffle each dataset, then repeat the training and evaluation process using all 6 orderings of G_{train}, C_{train}, and C_{test}. Our final results are the average of each evaluation metric over all orderings.

5 Results

The effects of our transformer on classifier PR-AUC and FPR are shown in Tables 1 and 2 respectively. "Naive NN" is the neural network classifier trained only on unmodified signatures, while "Informed NN" is the neural network classifier trained on both unmodified and modified signatures. Our transformer successfully hinders all tested classifiers on all datasets.

Data	Classifier	Baseline PR-AUC	Modified PR-AUC
H	Naive NN	0.999	0.309
	Informed NN	0.915	0.583
	Decision Tree	0.998	0.476
U	Naive NN	1.000	0.309
	Informed NN	0.999	0.428
	Decision Tree	1.000	0.503
A	Naive NN	1.000	0.309
	Informed NN	0.999	0.309
	Decision Tree	0.999	0.503
Avg	Naive NN	1.000	0.309
	Informed NN	0.971	0.440
	Decision Tree	0.999	0.494

Table 1: Effect of transformer on PR-AUC

Figures 6, 7, and 8 compare normal signatures from dataset H and transformed Meek signatures from dataset H ($T(H_{meek})$), with a transformer T that has been trained on the entirety of dataset H. Compared to Figures 2, 3, and 4, the differences between modified Meek and Normal traffic are much less pronounced. One notable traffic signature modification can be seen in Figure 6, where the difference in payload lengths between 60 bytes and 70 bytes has been reduced. Modified Meek inter-arrival times, shown in Figures 7 and 8 are much closer to those of normal traffic, and the frequency of inter-arrival times above 1000 ms has been reduced.

6 Discussion

The baseline classification results in Figures 1 and 2 show that Meek is easily identifiable using machine learning attacks. In every case, our classifiers achieved a near perfect PR-AUC and FPR on unmodified data. Wang et al. [28], Yao et al. [30], and Nasr et al. [20] also achieve impressive classification results. However, this strength can also be a weakness. Machine learning models are prone to overfitting [3], making them sensitive to perturbation. For example, over all datasets, the naive neural network achieves a PR-AUC of 1.00 on unmodified data while the informed neural network achieves a PR-AUC of 0.971. However, when classifying modified data, the neural network trained with unmodified data achieves a PR-AUC of 0.309, while the neural network trained using both unmodified and modified data achieves a PR-AUC of 0.440. However, the informed neural network tends to perform poorly in terms of false positive rate compared to the naive neural network. This may be due to catastrophic interference [11] caused by conflicting information between the unmodified and modified training set.

However, because we ignore hostnames, we lose some iden-

Data	Classifier	Baseline FPR	Modified FPR
H	Naive NN	0.005	1.000
	Informed NN	0.654	0.667
	Decision Tree	0.001	0.999
U	Naive NN	0.000	1.000
	Informed NN	0.351	0.501
	Decision Tree	0.000	1.000
A	Naive NN	0.002	1.000
	Informed NN	0.630	0.833
	Decision Tree	0.001	0.510
Avg	Naive NN	0.002	1.000
	Informed NN	0.545	0.667
	Decision Tree	0.001	0.836

Table 2: Effect of transformer on FPR

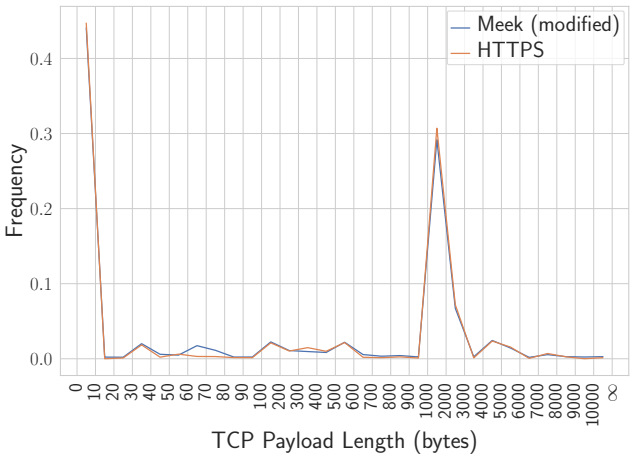

Figure 6: Average TCP payload length frequency over dataset H

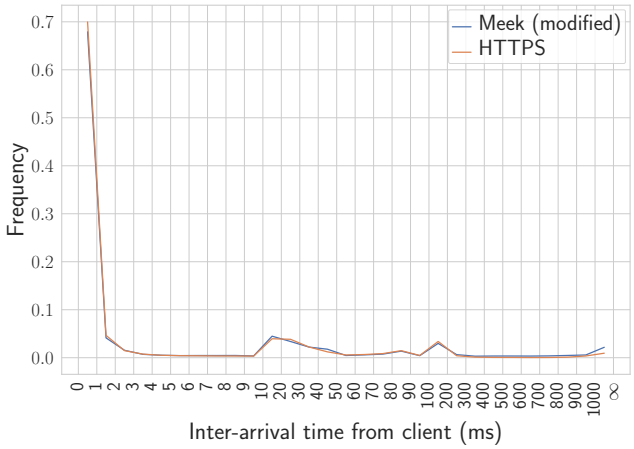

Figure 7: Average inter-arrival time frequency (from client) over dataset H

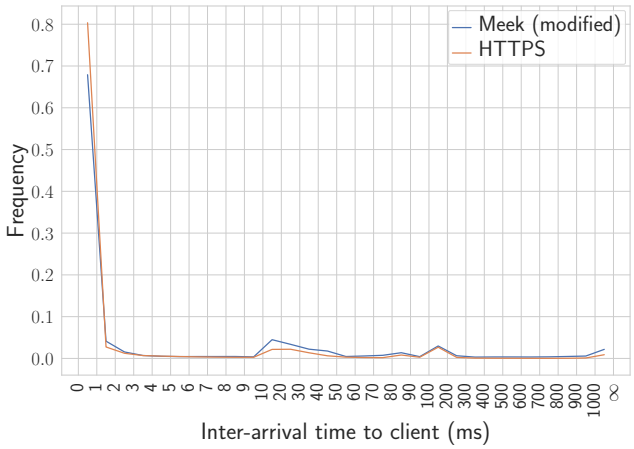

Figure 8: Average inter-arrival time frequency (to client) over dataset H

tifiable features. During training, we compare Meek traffic to all regular HTTPS traffic, rather than with HTTPS traffic to the Meek fronting host. For example, the meek-azure bridge uses `ajax.aspnetcdn.com` as the fronting host [23]. This host typically serves "popular third party JavaScript libraries such as jQuery" [19]. Traffic that mimics average HTTPS traffic to all domains may appear unusual to an adversary compared to typical traffic through this host. In this work, we assume that an increase in false positive rate is sufficient to make classification of Meek traffic less feasible, but future work may target hosts on CDN used for domain fronting during data collection.

Additionally, our dataset lacks geographical diversity. All traffic was generated from the Eastern US, and models generated from this data may not be useful to Meek users in other countries. While the Alexa top 1M dataset [1] contains websites from around the world, the data collection workers are set to use an English locale, which may result in latency differences compared to requesting the webpages in other languages.

7 Conclusions

In this work, we develop a data collection framework capable of efficiently producing reproducible packet captures of Meek and normal HTTPS traffic. We evaluate multiple classification methods over this captured traffic and train classifiers capable of identifying Meek. We then show that our adversarial modification scheme is capable of modifying traffic signatures in a way that reduces average classifier PR-AUC from 0.990 to 0.414 and increases average classifier FPR from 0.183 to 0.834.

While we focus on Meek and normal HTTPS traffic in this work, our adversarial modification scheme and data collection framework can potentially be applied to any Tor pluggable transport in order to identify and correct for weaknesses. In the future, adversarial models could be applied to shape traffic in real-time in order to improve any obfuscation method that relies on protocol mimicry or tunneling.

As adversaries performing censorship become more advanced, researchers developing obfuscation methods must become aware of their capabilities. Performing classification and transformation simultaneously using adversarial machine learning can allow researchers to model theoretical capabilities of both the censor and the obfuscator.

Availability

All code used to produce the results in this work including the traffic generation framework, feature extractor, and machine learning code is open source, and can be accessed at

https://github.com/starfys/packet_captor_sakura

References

[1] Alexa Internet, Inc. Keyword research, competitive analysis, website ranking. https://www.alexa.com. Accessed: 2019-05-01.

[2] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

[3] Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.

[4] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[5] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[7] L. Dixon, T. Ristenpart, and T. Shrimpton. Network traffic obfuscation and automated internet censorship. *IEEE Security Privacy*, 14(6):43–53, Nov 2016.

[8] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing china's blocking of unpublished tor bridges. In *8th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 18)*, 2018.

[9] David Fifield. pluggable-transports/meek - https transport. https://gitweb.torproject.org/pluggable-transports/meek.git/commit/?id=cea86c937dc278ba6b2100c238b1d5206bbae2f0. Accessed: 2019-05-10.

[10] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015.

[11] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[13] Docker Inc. Docker | what is a container? https://www.docker.com/resources/what-container. Accessed: 2019-04-21.

[14] Docker Inc. Docker compose | docker documentation. https://docs.docker.com/compose/. Accessed: 2019-07-05.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, and Roberto Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, volume 11. Citeseer, 2011.

[18] Diogo Belarmino Coelho Marques. Learning from http/2 encrypted traffic: a machine learning-based analysis tool. 2018.

[19] Microsoft. Microsoft ajax content delivery network | microsoft docs. https://docs.microsoft.com/en-us/aspnet/ajax/cdn/overview. Accessed: 2019-07-02.

[20] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976. ACM, 2018.

[21] Vern Paxson. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] Tor Project. doc/meek – tor bug tracker & wiki. https://trac.torproject.org/projects/tor/wiki/doc/meek. Accessed: 2019-07-01.

[24] Tor Project. Tor project: Pluggable transports. https://2019.www.torproject.org/docs/pluggable-transports.html.en. Accessed: 2019-04-20.

[25] Amazon Web Services. Aws regions and endpoints - amazon web services. https://docs.aws.amazon.com/general/latest/gr/rande.html. Accessed: 2019-07-05.

[26] Khalid Shahbar and A Nur Zincir-Heywood. Traffic flow analysis of tor pluggable transports. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 178–181. IEEE, 2015.

[27] G. Verma, E. Ciftcioglu, R. Sheatsley, K. Chan, and L. Scott. Network traffic obfuscation: An adversarial machine learning approach. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 1–6, Oct 2018.

[28] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 57–69, New York, NY, USA, 2015. ACM.

[29] Philipp Winter and Stefan Lindskog. How china is blocking tor. *arXiv preprint arXiv:1204.0447*, 2012.

[30] Zhongjiang Yao, Jingguo Ge, Yulei Wu, Xiaodan Zhang, Qiang Li, Lei Zhang, and Zhuang Zou. Meek-based tor traffic identification with hidden markov model. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 335–340. IEEE, 2018.