User profiling in the time of HTTPS

Roberto Gonzalez
NEC Labs. Europe
roberto.gonzalez@neclab.eu

Claudio Soriente
Telefonica Research
claudio.soriente@telefonica.com

Nikolaos Laoutaris
Telefonica Research
nikolaos.laoutaris@telefonica.com

ABSTRACT

Tracking users within and across websites is the base for profiling their interests, demographic types, and other information that can be monetised through targeted advertising and big data analytics. The advent of HTTPS was supposed to make profiling harder for anyone beyond the communicating end-points. In this paper we examine to what extent the above is true. We first show that by knowing the domain that a user visits, either through the Server Name Indication of the TLS protocol or through DNS, an eavesdropper can already derive basic profiling information, especially for domains whose content is homogeneous. For domains carrying a variety of categories that depend on the particular page that a user visits, *e.g.*, news portals, e-commerce sites, *etc.*, the basic profiling technique fails. Still, accurate profiling remains possible through transport layer fingerprinting that uses network traffic signatures to infer the exact page that a user is browsing, even under HTTPS. We demonstrate that transport fingerprinting remains robust and scalable despite hurdles such as caching, dynamic content for different device types *etc.*Overall our results indicate that although HTTPS makes profiling more difficult, it does not eradicate it by any means.

1. INTRODUCTION

Online user profiling is a profitable business extensively carried out by third parties such as search engines, ad networks and network providers. It leverages browsing activities to infer user interests and intentions. Since HTTP traffic has no privacy provisions, any third party can pry on the connections to a websever and profile users. HTTPS enhances online user privacy by encrypting the communication between a browser and a webserver. Major internet stakeholders are pushing for an *HTTPS everywhere* web with the promise of increased security and privacy and, therefore, of mitigating the problem of user profiling by third parties.

In this paper we assess the extent by which HTTPS prevents third parties from profiling users based on the websites they visit. We show that the widely used Server Name Indication (SN) extension of the TLS protocol leaks user interests to third parties which eavesdrop on the (encrypted) connection between a client and an HTTPS webserver. The SN extension improves address-space utilization as it allows to consolidate several HTTPS webservers at a given IP address. However, SN also hinders user privacy as it leaks the domain requested by a user, despite the HTTPS pledge of a secure and private connection.

The privacy leakage due to SN is especially severe in websites with *homogenous* content across its pages. For example, a connection request to `www.foxsports.com` tells a lot about the user interests, regardless of the actual page the user is browsing within that website. For a website with more varieties across its pages, the domain requested by a user may not tell enough about the interests of that user. For example, a connection to `www.amazon.com` may not tell much about a user. However a connection to `www.amazon.com/books` would reveal interests in books and a connection to `www.amazon.com/baby` may indicate an intent to buy baby items. We show that by using fingerprinting techniques, a network eavesdropper can accurately tell the page a user is browsing within a domain and, therefore, build a refined user profile.

Users profiling despite HTTPS is achievable, given enough bandwidth to fingerprint the websites under observation. If bandwidth is an issue, we also define an optimization problem that allows an eavesdropper to periodically pick the websites to fingerprint in order to maximize the number of users that are correctly profiled over time.

Overall, our findings show that HTTPS, while being a formidable tool to strengthen the security of web applications, cannot protect users against online profiling by third parties.

2. BACKGROUND AND MODEL

User profiling.

Profiling systems often use a closed-source mapping between URLs and interest categories. We follow the approach of previous work [1] and instantiate the mapping using the *Display Planner* of Google AdWords [2] – an online tool that given a URL returns the set of categories assigned by AdWords to that URL. Categories are arranged in a hierarchy and each URL has, on average, 10 assigned categories. The Display Planner also provides the inverse mapping, i.e., given a category it provides a list of websites that belong to that category.

HTTPS and Server Name Indication.

HTTPS enhances HTTP with the Transport Layer Security (TLS) protocol. TLS provides a secure pipe to a server that is usually authenticated via an X.509 certificate. The secure pipe is established via a TLS *handshake* – a procedure that allows the client and the server to establish cryptographic keys to encrypt and authenticate data exchanged through the pipe.

Given the ever increasing awareness on the privacy issues of HTTP, major web stakeholders are mandating secure (*i.e.*, HTTPS) connections to serve their websites [3, 4]. Furthermore the ToR project and EFF promote the HTTPS Everywhere extension [5, 6], that automatically redirects browsers to the HTTPS version of a website when available. One goal of this collective effort towards an HTTPS web is to increase online privacy with respect to network eavesdroppers. HTTPS ensures that a user is connected to the legitimate webserver and that the exchanged information cannot be eavesdropped by third parties. [1]

Server Name Indication (SN) is an extension of the TLS protocol by which a client specifies the hostname it is attempting to connect in the `client_hello` message (the first message of a TLS handshake). The extension is widely used by modern browsers and allows a server to serve multiple HTTPS websites, each with its own X.509 certificate, from the same IP address. The SN is, therefore, sent in cleartext and can be eavesdropped by any party tapping on the network between the client and the server.

System Model.

We consider a network eavesdropper that tries to profile users tapping on their network connections. We assume an *HTTPS everywhere* web where users connect to any website via HTTPS. The network eavesdropper, therefore, does not have access to the cleartext traffic exchanged between the user browser and the webservers but only sees encrypted flows. However, we assume the eavesdropper can infer the hostname requested by the user by looking at the SN in the `client_hello` message. In case SN is not used, client queries to DNS (recall that DNS has no provisions for confidentiality) or simply a `whois` on the destination IP address may reveal the hostname requested by the user.

We simplify the structure of a website and the user browsing behaviour as follows. Each website has a main page and a set of 1-st level pages (i.e., the pages linked on the main page). We do not consider pages of the website beyond the ones linked on the main page, but our results can be easily generalized to account for more complex website structures. Similar to previous work [7, 8, 9, 10, 11], we assume a user visits one page at a time for each domain. [2] This could be either the main page, or any of the 1-st level pages. The eavesdropper tries to infer the page visited by the user and assigns to her profile the corresponding set of categories according to Google AdWords.

3. USER PROFILING BY SERVER NAME

Looking at the SN in the `client_hello` message, a basic eavesdropper learns the website a user is browsing and assigns the categories of the main page to the user profile. If the user were actually browsing a page different from the main one, the profile built by the basic eavesdropper may not be accurate.

A first step towards understanding the accuracy of user profiling in an HTTPS everywhere web consists in assessing the difference between the categories of a website main page (e.g., the categories of `www.nbcnews.com/`) and the ones of any of its 1-st level pages (e.g, the categories of `www.nbcnews.com/politics/`).

In this experiment we have collected the list of top websites returned by AdWords for each of its 24 first level categories.Within each list, we have selected the 100 most popular websites based on their rank in Alexa [12]. For each of the resulting 2.4K websites we have fetched the URL of all the links available on the main page that remains within the same host. We did not consider external links like the ones to CDNs. Each of the collected URLs (totalling to more than 110K URLs) was submitted to the AdWords Display Planner to obtain its set of categories.

For each of the 24 top level categories of AdWords, Figure 1 shows the distribution of the Jaccard index among the categories assigned to the main page of a website and the categories assigned to its 1-st level pages. A Jaccard index close to 1 means that simply assigning the categories of the main page to a user creates a quite accurate profile, regardless of the actual page the user is browsing within that website. A

[1]Security guarantees of HTTPS do not take into account phishing attacks or flaws in the public key certification system.

[2]Discerning traffic when multiple pages of a domain are fetched simultaneously using HTTPS, remains an open problem.

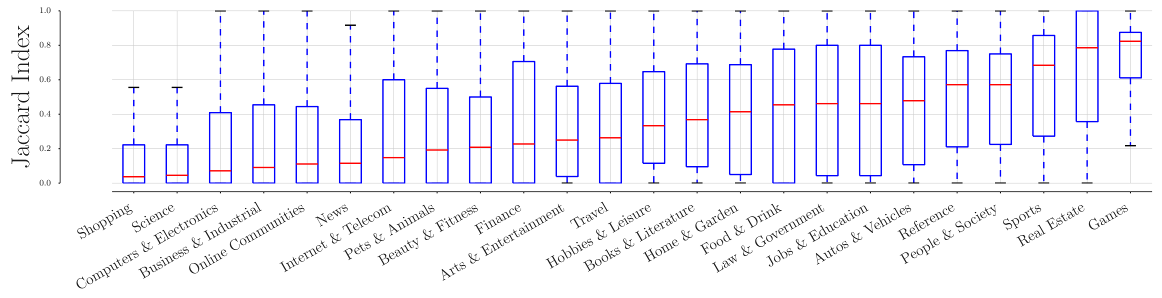


Figure 1: Distribution of the Jaccard index among the categories of the main page and the 1st-level pages.

Jaccard index close to 0 means that the same profile technique may lead to a less accurate user profile.

Figure 1 shows a great variance depending on the main category of the website. Users visiting Sports, Real Estate or Games websites could be profiled very accurately only by knowing the website their are connected to. However, when a user visits any page within a website related to, *e.g.*, Shopping, Computers & Electronics or News, the user profile built by assigning her the categories of the main page is likely to be inaccurate.

4. TRAFFIC FINGERPRINTING

In this section we show how to improve profiling accuracy by guessing the exact page a user is browsing using traffic fingerprinting. Traffic fingerprinting [7, 8, 9, 10] is an active research area on techniques to infer information (such as the visited page on an encrypted connection) by solely observing traffic patterns at the network/transport level.

Fingerprinting involves a training phase during which the adversary builds a fingerprint of each of the monitored pages. This is accomplished by fetching multiple times the monitored pages and recording features of the generated traffic such as packet size or inter-arrival times. Later, the adversary eavesdrops on the client's connection, extracts the same features from the client's traffic, and tries to match the client trace to one of the fingerprints computed during the training phase. Differences between the training data and the client (or test) data, due to, *e.g.*, different routes or congestions are mitigated using statistical methods.

We use and adapt to our scenario the fingerprinting technique of [11] – the most accurate web fingerprinting framework to date – that uses as features the size and the direction of each packet of a TCP connection. The classifier is, therefore, robust against differences in bandwidth or congestions along the route. The authors of [11] show that page fingerprinting is hard in an *open-world* scenario in which the client can browse any page outside of the set monitored by the eavesdropper. We show that webpage fingerprinting can be reasonably accurate in a *closed-world* scenario in which the eavesdropper monitors all the pages that the client can possibly visit. This assumption is realistic in our settings because the eavesdropper knows the website requested by the user (by looking at the SN in the `client_hello` message) and must infer which page she is browsing within this particular website.

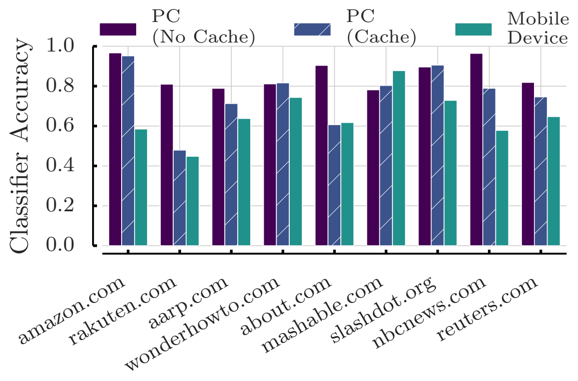


Figure 2: Accuracy of the classifier when fetching pages from a PC (with and without cache) and from a Mobile Device.

The features we extract from the traffic generated by downloading a page include: the number of incoming packets and the number of outgoing ones, the total size of incoming packets and the total size of outgoing ones, and a trace defined over the size and the order of the observed packets.[3] We use an SVM classifier with an RBF kernel parametrized with $\gamma \in [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]$ and $c \in [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]$. For each monitored website, we capture with `tcpdump` the traffic generated by fetching each of the 1-st level pages 50 times and measure the accuracy of the classifier using 10-fold cross validation.

Classifier Accuracy.

For this experiment we pick 9 websites that have low Jaccard index between the main page and the 1-st

[3]Given the space constraints, we refer the reader to [11] for the details on the feature selection process.

level pages (see Figure 1). For each website we train the classifier and test its accuracy in thee different scenarios. We use a PC with Mozzilla Firefox with and without cache, and a mobile device with Google Chrome with cache enabled. In the latter scenario we use the Android emulator to fetch the pages from an emulated Nexus 5 using the build-in feature of the emulator to simulate the conditions of a 3G network.

Figure 2 shows the accuracy of the classifier for the 9 websites in each one of the aforementioned scenarios. We found the lower accuracy for the PC with cache scenario when predicting pages of aarp.com (0.79) while we experienced the highest accuracy for amazon.com (0.97). Caching inevitably hinders the accuracy of the classifier by 10.3% on average, but the average accuracy never drops below 0.48. The accuracy decreases because when parts of a page are in the local cache, the traffic trace available to the classifier becomes shorter and, therefore, more likely to be confused with that of another page.[4] The mobile phone scenario suffers from a similar issue, not due to caching only, but also because mobile versions of a site are typically simpler than their desktop counterparts, and thus they end up producing more similar traffic traces.

From Page Prediction to User Profiling.

In this experiment we take a closer look at the effect of the classifier accuracy on the quality of the user profiles built by the eavesdropper.

Figure 3a shows the confusion matrix for `edition.cnn.com` where pages are sorted lexicographically based on their URL. For the same website and the same sorting of its pages, the matrix in Figure 3b shows the Jaccard index between any pair of 1st-level pages. Due to the sorting, pages under the same branch of the website, say `edition.cnn.com/style` appear sequentially, in both matrices. Figure 3a shows that when the classifier makes a mistake, the output page tends to be *close* to the correct one. For example `edition.cnn.com/style/arts` is often mis-classified as `edition.cnn.com/style/fashion` and viceversa. This is because the features we use to train the classifier look at the structure of a page (*e.g.*, the number and position of textboxes) rather than its content (*e.g.*, the actual text). Therefore, when pages within the same branch of a website share a similar structure, we experience classification mistakes similar to the ones of Figure 3a.

When mis-classification happens, the amount of damage to user profiling accuracy depends on whether the categories of the true page and the categories of the page output by the classifier overlap or not. For example, because of their similar structure, `edition.cnn.com/asia/` is likely to be predicted as `edition.cnn.com/africa/` by the classifier (see box 1 in Figure 3a); however, given that the set of their categories is very similar (see box 1 in Figure 3b), the mistake of the classifier has very little impact on the quality of user profiling. Of course this is not always the case. For example the pages under `edition.cnn.com/style/` (see box 2 in Figure 3a) are likely to be confused with one another by the classifier. This, however, leads to high profiling error because different pages under the "style" branch of the website have little overlap in term of categories (see box 2 in Figure 3b).

Figure 4 depicts the performance of the basic and the advanced profiling techniques when monitoring the 9 websites of Section 3. Dashed bars show the precision and recall of the basic profiling technique described in Section 3. Solid bars show the precision and recall of the advanced profiling mechanism that leverages the web fingerprinting technique described above. User profiling leveraging web fingerprinting clearly outperforms the basic profiling technique.

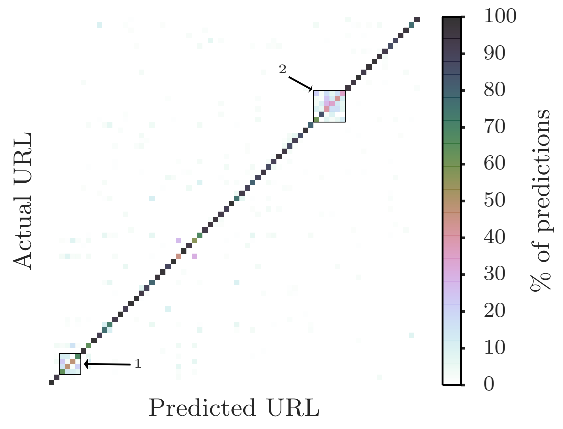
Classifier Freshness.

The difference between the time when the classifier is trained and the time when pages are predicted may affect the prediction accuracy. This is especially true for very dynamic websites (*e.g.*, news or online community websites). In this experiment we discretize time in *epochs* and we assume that website content only change from one epoch to the next one. If the train and the test data are collected in the same epoch, we say that the classifier is *fresh;* otherwise we say that the classifier is *stale*. We define epochs as days. We train the classifier over a snapshot of the website on a given day, and we try to predict pages fetched throughout the following 6 days.

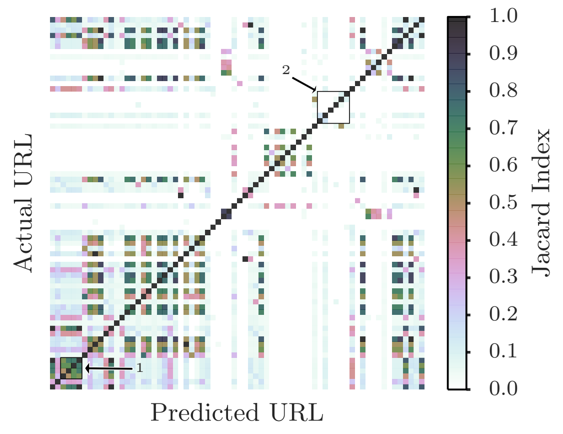
We expect a sensible difference in accuracy between a stale classifier and a fresh one for dynamic pages where content changes every day (*e.g.*, news websites). In the case of websites with static content, the difference between a stale classifier and a fresh one are expected to be less pronounced. To verify this, we add 4 websites with mostly static content (2 corporate and 2 academic ones) to the the 9 websites of the previous experiments.

Figure 5 shows the effect of staleness on the accuracy of the classifier for both a stateful and a dynamic website. The dashed lines represent the percentage of 1st-level pages that remain linked in the main page across days, while the solid ones represent the accuracy of the classifier. We observe the accuracy of the classifier for the dynamic website decreases rapidly while the accuracy for the static one decreases slowly

[4]In the extreme case of a page whose elements are all in the cache, the resulting trace becomes totally indistinguishable from that of any other fully cached page.



(a)



(b)

Figure 3: Confussion matrix (a) of the classifier and the Jaccard index between the categories assigned to two different pages (b) for `edition.cnn.com`. Box number 1 highlights URLs of the type edition.cnn.com/[region]. Box number 2 shows URLs under the branch `edition.cnn.com/style/`

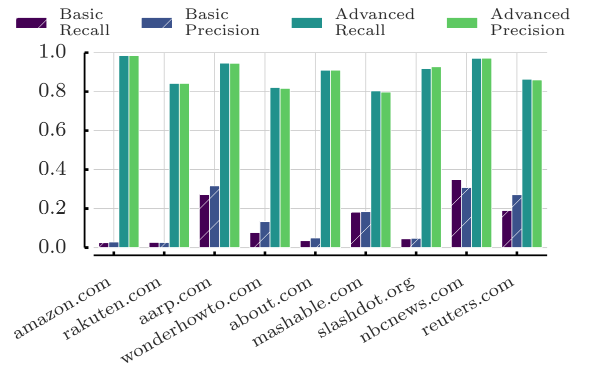


Figure 4: Precision and recall of the baseline eavesdropper and the eavesdropper leveraging website fingerprinting.

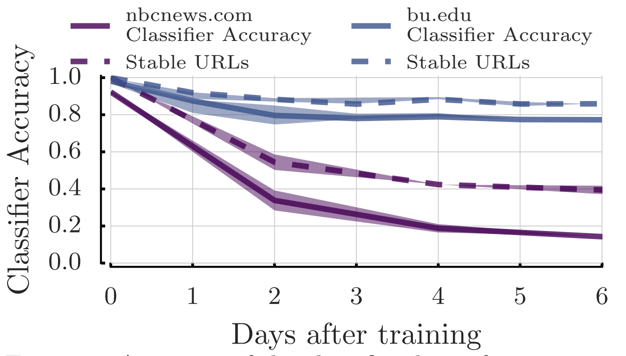


Figure 5: Accuracy of the classifier days after training for a dynamic website (`nbcnews.com`) and a static one (`bu.edu`).

during the first two days and then stabilizes around 80% accuracy. For both lines, the shadows denote the minimum and the maximum of the statistics.

5. OPTIMIZING BANDWIDTH USE

In a real-world deployment, the eavesdropper may not have the bandwidth required to refresh the classifier of each monitored website at every epoch. In the following we formulate an optimization problem for maximizing the profiling quality given a bandwidth constraint.

We consider an eavesdropper that monitors a corpus of n websites $w^1, \ldots, w^n$. Website w^i has a main page p_0^i and s^i 1-st level pages $p_1^i, \ldots, p_{s_i}^i$. We also use $c(p_j^i)$ to denote the set of categories of page p_j^i. When browsing website w^i, the user may visit any page p_j^i, with $j = 0, \ldots, s_i$. Since the connection is encrypted, we do not make any assumption on which are the most popular pages within w^i. If the user visits page p_j^i, the correct categories that should be assigned to that user when browsing w^i are, therefore, $c(p_j^i)$. We consider any category in $c(p_j^i)$ that the profiler assigns to that user as a *true positive*. Similarly, any category *not* in $c(p_j^i)$ that the profiler assigns to that user is a *false positive*.

The basic eavesdropper of Section 3 learns w^i from the `client_hello` message issued by the user browser and assigns the categories of the main page $c(p_0^i)$ to that user. However, because of HTTPS, the baseline profiling system cannot tell which page p_j^i was visited. If we denoted with T^i and F^i the true positive and the false positive, respectively, we have:

- $T^i = \frac{1}{s_i+1} \sum_{j=0..s_i} |c(p_0^i) \cap c(p_j^i)|$
- $F^i = \frac{1}{s_i} \sum_{j=1..s_i} |c(p_j) \setminus c(p_0)|$

The advanced eavesdropper of Section 4 tries to infer the page p_j^i the user has fetched by looking at the encrypted traffic trace. This is done by means of a classifier trained on a snapshot of w^i. As shown in the previous section, the freshness of the snapshot used to train the classifier impacts on its accuracy.

We denote the expected number of true positives and false positives with a classifier that is t_i epochs stale by $\mathcal{T}^i_{t_i}$ and $\mathcal{F}^i_{t_i}$, respectively. Thus we have:

- $\mathcal{T}^i_{t_i} = \sum_{j=0..s_i} \pi(p^i_j, p^i_j)|c(p^i_j)| + \sum_{j=0..s_i} \sum_{l=0..s_i,\ l \neq j} \pi(p^i_j, p^i_l)|(c(p^i_j) \cap c(p^i_l))|$
- $\mathcal{F}^i_{t_i} = \sum_{j=0..s_i} \sum_{l=0..s_i,\ l \neq j} \pi(p^i_j, p^i_l)|c(p^i_l) \backslash c(p^i_j)|$,

where $\pi(p^i_j, p^i_l)$ denotes the probability of predicting page p^i_j as p^i_l (depends on the freshness of the classifier).

Given the expected number of true and false positives, we set B as the bandwidth budget made available to eavesdropper at every epoch, and b^i as the bandwidth required to refresh the classifier for webiste w^i. We also denote by u^i the popularity of website w^i (*i.e.*, the number of users that visit w^i in an epoch).

Upon every epoch, the eavesdropper decides to spend the budget B by training classifiers on a fresh snapshots of a subset X of the monitored websites. If website w^i is included in X, the available budget is reduces by b^i and the expected number of correct categories assigned is $u^i \cdot \mathcal{T}^i_0$, while the expected number of categories miss-assigned is $u^i \cdot \mathcal{F}^i_0$. If website w^i is not included in X, the budget remains untouched and the expected number of correctly assigned and mis-assigned categories is $u_i \cdot \mathcal{T}^i_{t_i}$ and $u^i \cdot \mathcal{F}^i_{t_i}$, respectively, assuming the most recent classifier for w^i is t_i epochs stale.

The selection of X, therefore, tries to maximize the number of true positive and to minimize the number of false negative, while respecting the available budget.

$$\begin{aligned}
&\text{Select } X \subseteq \{1, \ldots, n\} \\
&\text{s.t. Max } \sum_{i \in X} u^i(\mathcal{T}^i_0 - \mathcal{F}^i_0) + \sum_{i \notin X} u_i(\mathcal{T}^i_{t_i} - \mathcal{F}^i_{t_i}) \\
&\text{Where } \sum_{i \in X} b^i \leq B
\end{aligned}$$

If a classifier were never trained on w^i we fall-back to the profiling technique of the naïve eavesdropper so that the number of true positives is $u^i \cdot c(p^i_0)$ and the number of false positive is $|c(p^i_j) \setminus c(p^i_0)|$.

The above problem resembles the well-known 0/1 knapsack problem with the only difference that items that are not selected add a non-zero value to the total gain.

A toy example.

To illustrate the workings and the value of the above optimization we have conducted a simulation based on the 15 websites from previous sections. We empirically assessed the training bandwidth requirement and probabilities of the confusion matrices, while we used Alexa to obtain the popularity of each website.

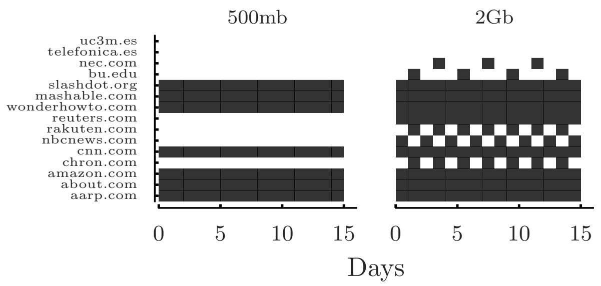


Figure 6: Output of the optimization problem across 15 days for 2 different bandwidth budgets (500mb and 2Gb). A black box represents a website which classifier must be refreshed on that day.

In Figure 6 we use a black box to mark a domain that is being selected for re-training on a particular day. We show which domains get to be classified every day under two different budgets – 500mb and 2Gb, representing 10% and 40%, respectively of the budget needed to re-classify all sites every day.

We observe that bandwidth availability can strongly affect the daily classification pattern. In case of 500mb budget, the same set of websites gets to be picked for classification every day. In the case of 2Gb, however, different websites get to compete for the available budget and thus end up being picked or skipped on different days. The actual resulting pattern depends on the interplay between website popularity, size, and dynamicity of content.

The small number of websites in the above example does not leave a lot of margin for profiling performance difference between optimizing only once vs optimizing every day. We have, however, simulated a larger example that includes 200 pages with a mix of popularities, content dynamicity, and size and have observed that in more complex settings the difference between optimizing only once vs. every day is substantial.

6. CONCLUSIONS

To the best of our knowledge our study is the first one to demonstrate that network eavesdroppers can profile user interests despite HTTPS. We have shown that even off-the-shelf traffic classification algorithms can guess the page that a user is viewing. Caching and dynamic content tailored to the device capabilities make the whole effort harder but the obtained accuracy remains high. We believe that more specialized classification algorithms, coupled with careful optimisation of classification bandwidth can yield accurate and scalable user profiling even in more complex settings than the ones we have considered. We plan to prove our claim by developing a fully functioning prototype.

7. REFERENCES

[1] J. M. Carrascosa, J. Mikians, R. Cuevas, V. Erramilli, and N. Laoutaris, "I always feel like somebodys watching me measuring online behavioural advertising," in *Proc. of ACM CoNEXT'15*.

[2] "Display Planner basics." `https://support.google.com/adwords/answer/3056115?hl=en`. "[Online; accessed 12-May-2016]".

[3] "SSL compliance." `https://support.google.com/richmedia/answer/6015286?hl=en`. "[Online; accessed 12-May-2016]".

[4] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," RFC 7540, RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7540.txt`.

[5] "HTTPS Everywhere." `https://addons.mozilla.org/en-US/firefox/addon/https-everywhere/`. "[Online; accessed 12-May-2016]".

[6] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," tech. rep., DTIC Document, 2004.

[7] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi, "Host fingerprinting and tracking on the web: Privacy and security implications.," in *Proc. of NDSS'12*.

[8] A. Hintz, "Fingerprinting websites using traffic analysis," in *Proc. of PETS'02*.

[9] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proc. of CCSW'09*.

[10] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proc. of ACM WPES'11*.

[11] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, "Website fingerprinting at internet scale," in *Proc. of NDSS'16*.

[12] "Alexa Top Sites." `http://www.alexa.com/topsites`. "[Online; accessed 12-May-2016]".