Glove: A Bespoke Website Fingerprinting Defense

Rishab Nithyanand
Stony Brook University
rnithyanand@cs.stonybrook.edu

Xiang Cai
Stony Brook University
xcai@cs.stonybrook.edu

Rob Johnson
Stony Brook University
rob@cs.stonybrook.edu

ABSTRACT

Website fingerprinting attacks have emerged as a serious threat against web browsing privacy mechanisms, such as SSL, Tor, and encrypting tunnels. Researchers have proposed numerous attacks and defenses, and the Tor project now includes both network- and browser-level defenses against these attacks, but published defenses have high overhead, poor security, or both. In this paper we present preliminary results of *Glove*, a new SSH based defense. Glove is based on the observation that current defenses are expensive not because website traces are different, but because the defense, operating blindly, does not know how to add cover traffic and therefore, puts it everywhere. Instead, Glove uses existing knowledge of a websites traces to add cover traffic conservatively while maintaining high levels of security. Further, Glove satisfies the information theoretic definitions of security defined in prior work [3] – i.e., it is resistant to any fingerprinting adversary. Our simulations show that Glove performs better than all currently proposed SSH based defenses in terms of the security-overhead trade-off.

1. INTRODUCTION

Glove demonstrates that efficient and secure website fingerprinting defenses are possible. The fundamental idea behind Glove is simple: although web pages vary widely in size and structure, they can be clustered into large groups of highly similar web pages. A defense system therefore need add only a small amount of cover traffic to make all the pages in a cluster indistinguishable to an attacker. When a user loads a page using Glove, the attacker can identify the cluster to which the page belongs, but gains no additional information about which page within that cluster is loaded.

Glove consists of an offline *training* phase and an online *defending* phase. During the training phase, Glove collects traces of web pages, clusters the pages by their network-level features, and computes, for each cluster, a transcript of packet sizes and timings that it replays whenever a user loads one of the pages in that cluster during the defending phase. Glove can be viewed as an extension of the main idea behind efficient traffic morphing [7]. Durng the training phase, the morphing system learns the distribution of packet sizes of two websites and computes a transformation matrix that can efficiently convert one distribution into the other. During the defending phase, it uses the matrix to fragment or pad packets generated by the first website so that the resulting sizes match the distribution of the second website. Glove extends this offline-online approach to cover all traffic features, not just packet sizes. It may also be viewed as simply optimizing for the common case in BuFLO based defenses [4, 2]. Both BuFLO based defenses send packets between the proxy end-points at fixed schedules that reveal little information about which website is being loaded. Glove optimizes this approach by using prior knowledge about popular websites to select a packet schedule that uses less bandwidth but still hides the identity of the website. For websites that Glove does not have prior knowledge, it falls back to BuFLO type defenses.

2. THE GLOVE DEFENSE

We believe the efficacy of a defense does not equal to its security against a specific attack. Future attacks may exploit the vulnerability of the defense even if it was secure in the past. As a result, Glove is designed to defeat an ideal attacker, who can not distinguish two websites if and only if they generate the exact same sequence of network traffic observations. Glove is a provably secure defense, because it gives an upper bound on the success rate of any attacks, regardless of their attacking methods.

The network traffic generated by loading a web page consists of a sequence of packets, which we call a trace. Packets in a trace can be divided into two categories: request packets that transmitted from a client to a web server, and response packets that directed to the client. From a different point of view, we can say the contents of a web page are covered by its trace. We also define *super-trace* as follows:

DEFINITION 1 (SUPER-TRACE). *We say that a trace S is the super-trace of traces $t_1, \ldots, t_n$ iff each trace t_i may be transformed into trace S by some sequence of the following actions:*

- ***Inserting:*** *Inserting request or response packets.*
- ***Merging:*** *Merging consecutive requests or responses.*
- ***Splitting:*** *Splitting a packet into a set of smaller sized packets such that the sum of their sizes is maintained.*
- ***Delaying:*** *Increasing delays between a response packet and its succeeding request packet, and vice-versa.*

Clearly, from Definition 1 we can see that, trace t_i of web page w_i can be replaced by *super-trace* S (at the cost of additional bandwidth and latency overhead). This is because S has enough data packets to cover the contents in w_i, while maintaining the dependencies between requests and responses in t_i. To protect n websites $w_1, \ldots, w_n$, Glove does two things: First, it divides the traces of these websites into

k clusters based on rules we will describe later. Second, for all the traces within a cluster, Glove computes a single *super-trace*.

Observe that Glove plays the same trace S_c whenever a web page in cluster c is loaded, thus generating the same observation to an attacker. Glove therefore meets the information theoretic definitions of security as defined in prior work [3]. For e.g., let C be the smallest cluster among all k clusters. Because loading all web pages in C yield the same observation to an attacker A, the probability that A can correctly guess which web page is loaded is $\frac{1}{|C|}$. Since C is the smallest cluster, Glove is a *uniformly ϵ-secure* defense, where $\epsilon = \frac{1}{|C|}$ here. Similarly, Glove can be tuned to achieve *non-uniformly ϵ security*.

2.1 Clustering Webpages

On a high level, to cluster webpages based on their network-level features, we employ *k-medoids* clustering on the *Dynamic Time Warping (DTW)* based distance matrix computed on input webpage traces.

2.1.1 Finding Representative Traces

In reality, not all webpages are static and the network condition changes from time to time, so the trace varies each time a webpage is loaded. Before clustering webpages, we need to choose a "representative" trace for each webpage, i.e., the trace that is likely to be most easily transformed into other traces generated by the same page. To do this, we load a webpage n times and record n traces. We then compute pair-wise Damerau-Levenshtein edit distances [6] among them, and find the trace t with the minimum average distance to others. t is chosen as the representative trace for the web-page.

2.1.2 k-medoids clustering

Since a trace can be viewed as a time-series, we compute the Dynamic Time Warping (DTW) [1] distances between every pair of representative traces. Once this pairwise distance matrix is computed, we use the k-medoids [5] algorithm to group similar webpages into a pre-determined number of clusters. The number of clusters determines the security and overhead of the Glove defense.

Simulations revealed that the clusters generated by DTW had lower overhead super-traces than other distance metrics. This is likely because DTW implicitly takes into account the time between packets while computing edit distances – an important factor when considering that supertraces need to maintain the inter-packet time dependencies between requests and responses in their constituent traces.

k-medoids is preferred for clustering as it was designed with the idea of enabling custom distance metrics between points (unlike k-means and other clustering methods which assume a euclidean space).

In our implementation of k-medoids, the cost of a cluster configuration was the lower-bound on bandwidth overhead which can be computed as: $\sum_i^k |c_i|(\max(req_{j \in c_i}) + \max(res_{j \in c_i}))$. Here, $c_1, \ldots, c_k$ are the k clusters and req_j, res_j denote the number of request and response bytes of the j^{th} site. The idea is that any super-trace of a group of traces must contain atleast as many request and response bytes as its constituent traces with the most number of request and response bytes.

2.2 Computing Super-traces

The *super-trace* of a cluster is a single trace which covers all traces contained in that cluster. If all webpages are static, a defense that *plays* this supertrace while loading any webpage within the cluster, effectively hides all information about the page being loaded (except the cluster it belongs to). However, since most webpages are dynamic, we compute a supertrace which aims to conservatively cover a large (tunable) percentage of all traces that one of its constituent webpages might generate. To do this, we use the heuristic demonstrated in Algorithm 1 to approximate the minimum bandwidth super-trace for each cluster. The following notation is used:

- Minimum site coverage (μ_{min}): This parameter determines the minimum number of traces of each webpage to be covered by the super-trace. The parameter μ denotes the average coverage of all pages. Larger μ_{min} values provide more resistance to the dynamicity of webpages, often times at the cost of larger overheads (not always).
- Bandwidth-Latency tuner (τ): This parameter allows us to tune the defense to produce super-traces that optimize some combination of bandwidth and latency overheads. The lower the value of τ, the lower the latency overhead (at the cost of bandwidth), and vice versa. The range of τ is 1 to 100.
- T is the set of input traces that the super-trace is computed over. This is initially $\emptyset$. ST denotes the currently computed super-trace. R_i is the set containing all the recorded traces of site i. *covmin* is the index of the site which is least covered by the current ST. This value may be initialized randomly.
- F denotes the current frontier packet of each trace in T and len_i denotes the number of packets in the i^{th} trace in T.
- The function `Find-Direction` returns +1 if more than $\frac{1}{6}$ up-stream packets, else returns -1. Function `Find-Time` returns the τ^{th} percentile time of frontier packets in direction P_D. `Find-Size` returns the maximum packet size in the frontier with time $\leq P_T$ and direction $= P_D$. Finally, function `Update-Frontiers` updates the frontier of each trace (F_i) to the last packet not covered by the current ST.

The algorithm is simple. The super-trace is computed over a set of input traces. In each iteration we add a trace from the least covered site into this input, until all sites have satisfied the minimum coverage parameter μ_{min}.

Now, for each of these input traces, a counter (starting at the first packet) indicating the current frontier is maintained. We count the number of frontier packets in each direction. If more than $\frac{1}{6}$ of the packets are up-stream packets, we add an up-stream packet to the super-trace, otherwise, we add a down-stream packet. We use the parameter $\frac{1}{6}$ because in our input traces we found that the average ratio of up-stream to down-stream packets was $\frac{1}{6}$. The time at which this newly added packet is to be sent is set to be the time of the τ^{th} percentile frontier packet (assuming that frontier packets are ordered by time) in the chosen direction. The size of the newly added packet is taken to be maximum size

Algorithm 1 Algorithm to compute the super-trace of a cluster

function INPUT-GEN(μ_{min}, τ, $covmin$, T, $\{R_1, \ldots, R_n\}$)
  if $coverage_{covmin} < \mu_{min}$ **then**
    $T \leftarrow T \cup t$, where $t \in R_{covmin}$;
    $ST \leftarrow$ COMPUTE-ST(τ, T)
  else
    return ST
  end if
end function
function COMPUTE-ST(τ, T)
  $ST \leftarrow \emptyset$, $F \leftarrow \{1, \ldots, 1\}$
  while $F \neq \{len_1 + 1, \ldots, len_m + 1\}$ **do**
    $P_D \leftarrow$FIND-DIRECTION(T)
    $P_T \leftarrow$FIND-TIME(τ, T, P_D)
    $P_S \leftarrow$FIND-SIZE(T, P_T, P_D)
    $ST \leftarrow ST||(P_D, P_T, P_S)$
    $F \leftarrow$ UPDATE-FRONTIERS(ST, T)
  end while
end function

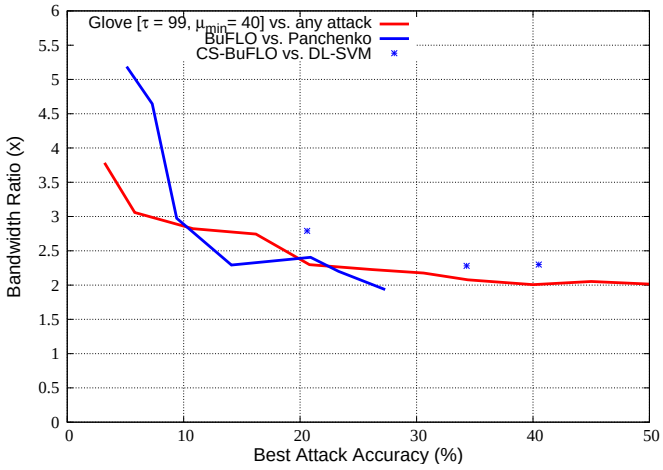


Figure 1: Attacker Accuracy vs. Bandwidth Ratio

(rounded to the nearest 50 bytes) of all the frontier packets in the chosen direction. The above process is repeated until the frontier of all the input traces have passed their final packet. The final trace ST is guaranteed to cover at-least μ_{min}% of the traces of each site (although, in practice the average coverage turns out to be far higher).

3. SIMULATION RESULTS

For our simulations, we collected 50 traces each from the Alexa top 500 functioning, non-redirecting webpages. The browser cache was cleared between page-loads. Next, each of the 50 traces for each site were then ranked by their *representativeness*, and clustering was done. For our simulations, we varied the number of clusters (k) in the range of 16 and 250, giving us defenses which were between .032 and .5-non uniformly secure. Once the clusters were found, sets of supertraces were generated for each of the clusters while varying μ_{min} and τ. Finally, statistics corresponding to expected site coverage, bandwidth, and latency ratios were computed.

Security-Overhead Trade-off: Figure 1 compares the trade-off between efficiency of Glove, BuFLO, and CS-BuFLO and the levels of security provided. Note that in this plot, Glove is the only defense that provides information theoretic security *against any attacker*, while the data for BuFLO and CS-BuFLO (obtained from [4] and [2]) are relevent only against Panchenko and DL-SVM attackers, respectively. Therefore, while the BuFLO and CS-BuFLO trade-offs might vary depending on the attacker, the Glove trade-off holds against any attacker. From the figure, it is clear that Glove provides significantly better trade-off costs than CS-BuFLO (vs. DL-SVM, for any security level) and BuFLO (vs. Panchenko, when attack accuracy $\leq 10\%$). When security is less critical (attack accuracy $> 10\%$), however, Glove and BuFLO (vs. Panchenko) provide similar trade-offs.

Tunability of Glove: In figure 2, we demonstrate the effect of varying the μ_{min} and τ parameters of Glove (on its bandwidth and latency ratios). In particular, the figures show that varying μ_{min} to increase resistance to dynamic content in web-pages has little effect on the bandwidth overhead of the defense, while significantly increasing its latency overhead (when $\tau = 99$ – i.e., the defense is optimized for minimizing bandwidth costs). Further, we see that τ allows user experience tuning – i.e., reducing τ causes a significant drop in latency overheads (increasing the usability of the defense) at the cost of increased bandwidth.

4. DISCUSSION AND CONCLUSIONS

In this paper we presented Glove– a website fingerprinting defense that illustrates a promising new approach towards building efficient fingerprinting defenses. In particular, Glove is the first SSH-based defense to demonstrate:

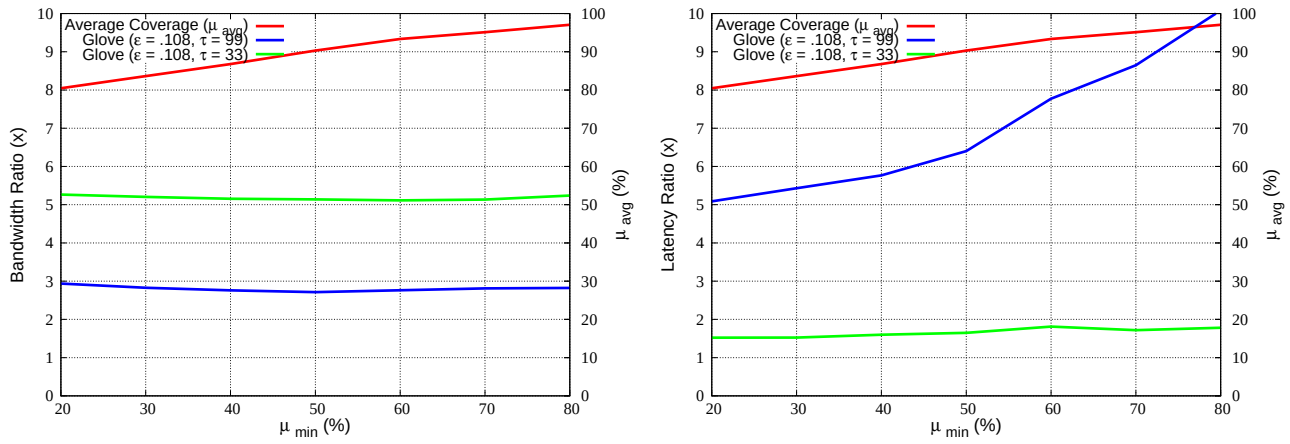
Information-Theoretic Security: Unlike previous defenses, Glove provides information-theoretic security guarantees, as defined in [3]. This is achieved by computing a single supertrace for each computed cluster and playing this each time a page contained in the cluster is loaded. As a result, in the absence of prior (or, outside) knowledge, any (current or future) website fingerprinting attackers success rate is bounded by the size of the computed cluster.

Good Security-Overhead Trade-off: The results illustrated in figure 1 demonstrate the validity of our conjecture that using prior information about the structure of a web-page to add cover traffic conservatively yields *better* website fingerprinting defenses. This approach, used by Glove, results in it being more secure and efficient than any previously proposed SSH based defense.

High Tunability: Unlike previous defenses, Glove allows users to tune their browsing experience by exposing the τ parameter. Lower τ values significantly reduce latency at the cost of bandwidth, and vice-versa. This is especially useful in current scenarios where it is likely the case that bandwidth costs are inconsequential, while even moderately increased user experienced latency may result in significantly decreased usability of the defense.

However, Glove also has limitations that may hamper its adoptability in the real world. We discuss these below.

Infrastructure Requirements: Glove greatly reduces its overhead by utilizing prior knowledge of web-page structures. This requires the *Glove infrastructure* to be able to collect traces of defended web-pages, cluster these pages, and compute corresponding super-traces. These tasks are fairly computationally intensive, making them infeasible for standard server side proxy nodes to perform on a regular basis. Instead, using a powerful dedicated central node to compute



(a) Bandwidth Ratios

(b) Latency Ratios

Figure 2: Overhead Ratios and Average Coverage while varying μ_{min} and τ on an $\epsilon = .108$ Glove

clusters/super-traces and distribute them to Glove proxies (e.g., via Tor bridges – since the security of Glove is independent of the secrecy of the computed super-traces) is more efficient. However, such a node may not be easily available in the real world. In the absence of such nodes, Glove has to fall back on less powerful server side proxies which take as input a list of urls (from the client) and returns a single super-trace (to be played when the client loads a page in the input urls). However, in these cases, while Glove retains its information-theoretic security guarantees, it is no longer able to user prior knowledge of web-page structures to provide low overheads. To circumvent these problems, one may consider developing distributed clustering and super-trace algorithms for use with Glove.

Effect of Dynamic Content: Two types of dynamicity affect the Glove defense. First, the information-theoretic guarantees provided by Glove hold only when $\mu_{min} = 100\%$. This, however, is possible to guarantee only when web-pages do not contain dynamic content (e.g., JS, AJAX, etc.). One way to address this problem is to force users to disable scripts while browsing – although this is reasonable for a few users (who use Glove to avoid life-or-death situations), it is not reasonable for other more casual users of Glove. For such users, a more suitable option is to have the Glove server side proxy simply ignore any packets that may still be required to be transmitted after the super-trace has been played to completion. While this may occasionally result in a user viewing only a partially loaded page, this is rare enough to prevent a significant drop in the usability of the system (e.g., at $\mu_{min} = 80\%$ in figure 2, we see this happens less than 3% of the time, on average). Second, when the structure of a Glove defended web-page changes significantly, its trace may change to a large enough degree that less than $\mu_{min}\%$ of its page loads are covered by its current super-trace. In this case, re-clustering/super-tracing of all defended pages may be in order. While our observation is that this is rare, its occurrence does result in the need for the Glove infrastructure to occasionally perform this re-clustering, super-tracing, and distribution.

Code and Data Release: To ensure reproducibility of our results and ease comparative evaluation, the following resources will be made available (at `https://bitbucket.org/rishabn/glove`), on publication of this paper: a fully working simulation and implementation of Glove, traces collected, clusters generated, and complete simulation results.

5. REFERENCES

[1] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[2] Xiang Cai, Rishab Nithyanand, and Rob Johnson. Cs-buflo: A congestion sensitive website fingerprinting defense. In *In Submission*, 2014.

[3] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *ACM CCS*, 2014.

[4] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *IEEE Security and Privacy*, 2012.

[5] Leonard Kaufman and Peter Rousseeuw. Clustering by means of medoids. *Statistical Data Analysis Based on the L1-Norm and Related Methods*, 1987.

[6] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33:31–88, March 2001.

[7] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, 2009.