Practical Censorship Evasion Leveraging Content Delivery Networks

Hadi Zolfaghari
University of Massachusetts Amherst
hadi@cs.umass.edu

Amir Houmansadr
University of Massachusetts Amherst
amir@cs.umass.edu

ABSTRACT

CDNBrowsing is a promising approach recently proposed for censorship circumvention. CDNBrowsing relies on the fact that blocking content hosted on public CDNs can potentially cause the censors collateral damage due to disrupting benign content publishers. In this work, we identify various low-cost attacks against CDNBrowsing, demonstrating that the design of practically unobservable CDNBrowsing systems is significantly more challenging than what thought previously. We particularly devise unique website fingerprinting attacks against CDNBrowsing traffic, and discover various forms of information leakage in HTTPS that can be used to block the previously proposed CDNBrowsing system. Motivated by the attacks, we design and implement a new CDNBrowsing system called CDNReaper, which defeats the discovered attacks.

By design, a CDNBrowsing system can browse only particular types of webpages due to its proxy-less design. We perform a comprehensive measurement to classify popular Internet websites based on their browsability by CDNBrowsing systems. To further increase the reach of CDNBrowsing, we devise several mechanisms that enable CDNBrowsing systems to browse a larger extent of Internet webpages, particularly partial-CDN webpages.

1. INTRODUCTION

Internet censorship continues to remain the biggest threat to the Internet freedom across the globe [7, 8, 25]. A recent promising approach for censorship circumvention [21], which we call *CDNBrowsing*, relies on the collateral damage of disrupting ubiquitous content delivery networks (CDN). In this approach, censored clients obtain a forbidden Internet webpage that is hosted on a CDN network by connecting to some *arbitrary* edge server of the hosting CDN system—as opposed to obtaining it from the optimal CDN edge server identified through DNS resolution. This mechanism is aimed at defeating the main censorship mechanisms as follows: First, CDNBrowsing clients bypass any DNS-based censorship by refraining from DNS lookups for the censored content. Second, censors will refrain from IP blocking the CDN edge servers that host censored webpages since this will block all the other (potentially many) non-forbidden webpages hosted on those edge servers as well. Finally, the use of encryption by the hosting CDN providers (e.g., through HTTPS) can bypass DPI-based censorship mechanisms like keyword and URL filtering.

In this paper, we show that despite being a promising new direction for censorship circumvention, designing practical CDNBrowsing systems is significantly more challenging than previously suggested [21]. We demonstrate the possibility of various low-cost identification attacks tailored to CDNBrowsing, and demonstrate that the attacks can effectively detect and block *CacheBrowser*, the first CDNBrowsing system proposed by Holowczak et al. [21]. Specifically, our thorough analysis of in-the-wild CDN systems shows that their deployments of the HTTPS protocol leaks the identity of the websites being served to the clients, therefore, allowing the censors to block an unprotected CDNBrowsing system like CacheBrowser. Additionally, we introduce a unique form of *website fingerprinting* attack against CDNBrowsing traffic, and show its significantly high accuracy against CacheBrowser with processing overheads two orders of magnitude lower than traditional website fingerprinting techniques [39]. We conclude that *an effective CDNBrowsing system should be tailored to every single CDN system* it tries to leverage for circumvention, as opposed to CacheBrowser's one-size-fits-all solution; this is due to the diversity of CDN deployments as shown in our analysis.

To that end, we design and deploy a new CDNBrowsing system called *CDNReaper* that protects against the discovered attacks. CDNReaper takes into account the specific deployment of HTTPS by each real-world CDN system, and implements CDN-aware mechanisms that remove the leaked information about (forbidden) CDN destinations browsed by CDNReaper. Also, CDNReaper defeats the CDNBrowsing-specific website fingerprinting attacks by making modifications to the web objects requested by the censored clients. We have implemented CDNReaper as a fully functional, end-user system.

By design, a CDNBrowsing system can only browse particular censored webpages, e.g., those hosted on specific CDN systems. We perform *the first comprehensive analysis* on the reach of CDNBrowsing systems by inspecting the top 10,000 Alexa websites and classifying them into various categories based on their readiness to be browsed by CDNBrowsing systems. To further expand the reach of CDNBrowsing, we

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978365

devise several mechanisms that enable CDNBrowsing systems to browse a larger scope of Internet webpages. In particular, we implement mechanisms to enable CDNBrowsing of *partial-CDN websites*, i.e., those with only some of their main content hosted on CDN, which constitute a large fraction of Internet webpages based on our study. For instance, some video streaming or Internet TV websites host their multimedia content on CDN networks, but not their front-page HTML. We particularly introduce *content wrappers* and *dynamic mirrors* to enable CDNBrowsing of partial-CDN websites, and demonstrate their feasibility for popular partial-CDN webpages like bbc.co.uk and tumblr.com. We show that these mechanisms come with very small overheads compared to fully CDNBrowsable webpages. We also investigate how CDNReaper can be used for browsing websites hosted on private CDN networks, such as YouTube.

In summary, we make the following main contributions:

- We identify various low-cost attacks tailored for CDNBrowsing systems, and demonstrate their effectiveness against CacheBrowser [21]. Particularly, we devise a CDNBrowsing-specific website fingerprinting attack that is two orders of magnitude faster than the state-of-the-art fingerprinting system of Wang et al. [39], despite its higher accuracy.
- We have designed and deployed an advanced CDNBrowsing system called CDNReaper. Our system is designed to defeat the CDNBrowsing-specific attacks we discovered. We have implemented CDNReaper as a fully functional system with Chrome and Firefox plugins. We have also designed an advanced bootstrapper for CDNReaper. Our bootstrapper is designed as a CDNBrowsable service itself, therefore it is highly unblockable.
- Through comprehensive evaluations, we classify the top 10,000 Alexa websites into six categories based on their readiness to be browsed through CDNBrowsing.
- We have devised several mechanisms to extend the reach of CDNBrowsing. Particularly, we introduce *content wrappers* and *dynamic mirrors*, which enable CDNBrowsing of partial-CDN webpages with negligible overheads.

2. BACKGROUND ON CDNBrowsing

2.1 Internet Censorship

The Internet plays a crucial role in today's social and political movements by facilitating the free circulation of speech, information, and ideas. Consequently, repressive regimes and totalitarian governments regulate, monitor, and restrict access to the Internet [1, 4, 13, 27, 32, 35, 41], which is broadly known as *Internet censorship*. The main technical approaches [14, 26] to enforce censorship are *IP address blocking*, *DNS interference*, and *deep-packet inspection (DPI)*. The censors use these techniques not only to block access to forbidden Internet destinations, but also to disable systems that aim at helping censored users bypass censorship, i.e., *circumvention systems* [2, 3, 15, 16, 19, 23, 24, 29, 31, 33, 36, 38, 40]. Therefore, the basic feature of an effective circumvention system is to be unblockable by the censors. Unfortunately, the leading circumvention technologies are effectively blocked [9, 41, 43] by competent state-level censors across the globe (most notably, China and Iran), thanks to the affordable state-of-the-art censorship technologies manufactured and sold by competing multinational technology companies. Even the most advanced, state-of-the-art circumvention systems designed in the academia, such as Tor pluggable transports [16, 19, 31, 38, 40], are thwarted [20, 22, 37] by competent state-level censors.

2.2 CDNBrowsing Circumvention

CDNBrowsing is a new approach for censorship circumvention, recently proposed by Holowczak et al. [21] as a system called CacheBrowser. CDNBrowsing is based on the observation that webpages hosted on typical CDN platforms *share* the same set of IP addresses. Therefore, to block a forbidden webpage hosted on a CDN system, the censors will *not* be able to use **IP filtering** since it will also block all the (potentially many) non-forbidden webpages that are served through the same set of shared IP addresses. The censors, however, may be able to use **DNS interference** to block access to specific forbidden CDN-hosted webpages. Fortunately, such DNS interference can be easily circumvented: as Holowczak et al. show, typical CDN-hosted webpages can be obtained from *arbitrary* CDN edge servers. Therefore, a censored client can ignore the DNS resolution stage (which is anyways interfered with by the censors) and directly request the forbidden CDN website from *some* arbitrary CDN edge server that the client already knows. Ignoring the DNS resolution state by a CDNBrowsing client can potentially degrade the QoS performance of the (otherwise censored) CDNBrowsing connection compared to regular connections (since DNS resolution is meant to identify the optimal CDN IP address), however, Holowczak et al. show that the performance is still superior to that of traditional circumvention systems like Tor. Finally, the use of encryption in HTTPS connections is meant to prevent the censors from **deep-packet inspection** of HTTPS CDN content, therefore, rendering URL/keyword filtering infeasible

2.3 Advantages of CDNBrowsing

CDNBrowsing is a new paradigm in censorship circumvention. It differs from traditional circumvention systems [15, 23, 24, 31, 33, 36, 38, 40] like Tor, Psiphon, and VPNs in that it *makes no use of circumvention proxies*. This offers CDNBrowsing unique advantages over traditional circumvention systems: **Better quality of service:** Since CDNBrowsing makes no use of —congested, bandwidth-limited—circumvention proxies, CDNBrowsing connections can offer better quality of service (e.g., lower latency) compared to traditional circumvention systems that make use of proxies. This is demonstrated [21] against Tor. **Lower cost of operation:** In proxy-based circumvention systems, someone should pay for the operational costs of circumvention proxies, i.e., either the users (e.g., VPN users), or some volunteers (e.g., Tor relays). In a CDNBrowsing system, however, the operational costs are minimized due to the absence of proxies. **Better sustainability:** Third-party entities who run proxies for traditional circumvention systems like Tor and Psiphon may lose interest in doing so over time, e.g., due to the high-cost of operation or censor coercion. CDNBrowsing, on the other hand, has minimal dependence on third-party operators. **Ease of deployment:** Many recent proposals for circumvention have not seen real-world deployment as their deployments require substantial support from third-parties. For instance, a proxy-based proposal like FreeWave [24] needs to be deployed on volunteer proxies, Infranet [18] relies on volunteer webpages, and decoy routing [23, 42] cannot function without help from volunteer ISPs. CDNBrowsing systems, however, can function with little reliance on volunteers.

2.4 Limitation of CDNBrowsing

Unlike traditional circumvention systems, a CDNBrowsing system can *only* browse certain webpages, e.g., those that are hosted on public CDN providers. Therefore, the owners of censored websites may have to adapt their website deployments (e.g., by hosting them on public CDN systems) to make their content accessible by a CDNBrowsing system. In other words, CDNBrowsing systems are *publisher-centric* [21]: content publishers (e.g., website owners) will have to pay the expenses for their content to become accessible by censored users, as opposed to being oblivious to censorship and relying on volunteer third-parties like Tor bridges to undertake the expenses.

Why CDNBrowsing is still promising despite this limitation: First, only a very small fraction of all Internet webpages are blocked by a typical censoring country like China or Iran. Therefore, an ideal circumvention system is not the one that is able to browse *all* Internet websites, but the one that can browse all (or most) of the *censored* Internet destinations. Traditional circumvention systems like Tor and VPN proxies enable access to all websites —including the majority non-censored websites —at the price of lower quality of service, higher cost of operation, etc. Second, the "publisher-centric" circumvention paradigm in which content publishers get involved in unblocking their content is *not unrealistic*. In fact, an increasing number of censored content publishers have recently started to take measures to unblock their censored content, e.g., media websites like bbc.co.uk, manototv.com, and dw.tv are currently working with circumvention companies like Lantern and Psiphon to make their own censored content unblocked. Therefore, a practical CDNBrowsing system like our CDNReaper may be adopted by censored content publishers in the future, e.g., through censored website administrators migrating their websites to public CDNs. Finally, as we show in this paper, a significant fraction of readily non-CDNBrowsable websites are partially hosted on CDNs. We build mechanisms that enables CDNReaper to browse such websites with only minimal overhead compared to fully-CDNBrowsable websites, therefore expanding the reach of CDNBrowsing.

2.5 Comparison to Domain Fronting

Fifield et al. [19] recently proposed a new approach for setting up circumvention proxies, called *domain fronting*. In this approach, circumvention proxies, e.g., Tor bridges, are run as web services that share IP addresses with other non-circumvention services. Particularly, Fifield et al. implement a domain fronting pluggable transport for Tor, called *meek*, which is implemented over Microsoft Azure, Google App Engine and, Amazon CloudFront. Similar to CDNBrowsing, IP filtering of domain fronted proxies causes collateral damage to the censors as it will also block the non-circumvention services hosted on the shared IP addresses.

Domain fronting over CDN resembles the approach taken in CDNBrowsing. However, they offer distinctive advantages and disadvantages compared to each other. Domain fronting is essentially a proxy-based approach, therefore it suffers from the weaknesses enumerated in Section 2.3 in comparison to CDNBrowsing. Particularly, deployment of a domain fronting system like meek is significantly costlier than a CDNBrowsing system. For instance, the meek pluggable transport has cost a total of $26,536 in proxying expenses since its inception ($2,479 per month in the latest report) [30], even despite a discounted rate due to a "free research grant" [19] and a 1.5–3 MB/s bandwidth cap on users' traffic. We show in Section 5 that even the mirrored CDNBrowsing websites cost close to zero. Also, CDNBrowsing systems offer better quality of service compared to domain fronting due to their proxy-less implementation as shown by Holowczak et al. [21] (meek is particularly much slower due to its 3 MB/s bandwidth limit).

On the other hand, unlike domain fronting which is a generic proxy, CDNBrowsing can access only particular types of Internet websites as discussed in Section 2.4. We will perform a comprehensive evaluation of our CDNBrowsing system's reachability later in Section 5. Also, the identification attacks that we discuss in Section 3 are unique to CDNBrowsing and to the most part inapplicable to domain fronting. We will elaborate on this in Section 3.3.

2.6 CDN Basics and Terminologies

We refer to the autonomous organization operating a content delivery network as a CDN **provider**, and to the content publishers paying the CDN providers for hosting their web content as CDN **customer**s. For instance, bloomberg.com is a "customer" of the Akamai CDN "provider." A **client** is an Internet user who connects to CDN providers to access the content they host.

A typical CDN network is composed of several key components. **Edge server**s are the computer servers that cache and serve customer content to the clients. A CDN's edge servers are usually set up at disperse geographic locations to optimize performance and reliability [6]. A CDN provider's **mapping system**, which serves DNS queries, directs clients to various edge servers based on factors like proximity and the load on the servers.

Commercial CDN providers [5] like Akamai and CloudFront use their CDN infrastructure to serve a large number of heterogeneous customers; we refer to them as **shared CDN**s. In a shared CDN system, edge server IP addresses are shared among the customers, i.e., the mapping system may assign the same IP addresses to various customer websites. On the other hand, a **private CDN** is one that only serves the content for a specific content publisher, e.g., Google has its own private CDN.

3. IDENTIFICATION ATTACKS

In Section 2.2 we discussed the main principles behind CDNBrowsing's resistance to the core censorship mechanisms, i.e., IP blocking, DNS interference, and DPI-based filtering. In this section, we introduce several unique attacks against CDNBrowsing, demonstrating that the design of effective, unblockable CDNBrowsing systems is *significantly more challenging*. We particularly show that our attacks can identify and block CacheBrowser [21]. Later in Section 4 we design a CDNBrowsing system that defeats these attacks.

3.1 Destination Leakage in HTTPS

Despite the use of encryption in HTTPS, we demonstrate that an unprotected CDNBrowsing system like CacheBrowser can be blocked using deep-packet inspection and IP filtering.

CDN	A1	A2	B1	B2
Akamai	✓	✓	✗	✓
Cloudfront	✗	✓	✓	✓
CloudFlare	✗	✓	✓	✓
Edgecast	✓	✓	✗	✓
Fastly	✓	✓	✓	✓
Incapsula	✓	✓	✓	✓
MaxCDN	✓	✗	✓	✓

Table 1: Deploying HTTPS by major CDNs.

This is due to how real-world CDNs implement the HTTPS protocol.

3.1.1 How Real-World CDNs Deploy HTTPS

Implementing HTTPS over CDN networks poses significant technical challenges as demonstrated by Liang et al. [28] recently. First, content publishers need to delegate trust (e.g., TLS certificates) to the CDN edge servers in order to be able to provide clients with authenticated, secure access to HTTPS websites. Second, as each CDN edge server is likely to handle TLS requests for multiple independent domains (e.g., belonging to different content publishers), it must be able to return valid certificates for each of the requested domains. In the following, we discuss the four main mechanisms used by real-world CDNs to deploy HTTPS. Table 1 shows the mechanisms used by the major CDN providers.

A. Use of Shared TLS Certificates.

A1. CDN Domain Certificates. In this approach, the CDN provider obtains a certificate certifying its wildcard domain, e.g., *.akamaihd.net. To use this wildcard certificate, a publisher customer of this CDN will need to use a subdomain of the certified CDN domain to publish its content, e.g., fbstatic-a.akamaihd.net. Trivially, this allows multiple independent customers of a CDN to use the same certificate. Content publishers mainly use this mechanism to host the inner objects of their webpages, e.g., the CSS files, images, and scripts that are embedded inside HTML webpages. The wildcard certificates are usually offered for free to content publishers.

A2. SAN Certificates. The Subject Alternative Name (SAN) extension to X.509 allows multiple domain names to be included in a single certificate [12]. Using this extension, a CDN system can obtain a single TLS certificate that collectively certifies domain names for multiple CDN customers. A CDN edge server in possession of a SAN certificate can then serve HTTPS for all of the customer domains included in the certificate.

B. Use of Individual TLS Certificates.

B1. Server Name Indication (SNI). SNI [17] is an extension to the TLS protocol. It allows a web server hosting multiple HTTPS domain names to return the right individual TLS certificates to a client requesting one of the domains. The use of SNI is arguably the most suitable method for HTTPS deployment by CDNs since it allows content publishers to use their individual certificates with no additional cost to the CDN. However, SNI is not yet widely deployed by CDNs even though most modern web browsers and TLS libraries support this extension.

B2. Dedicated IP addresses. Some in-the-wild CDNs dedicate specific IP addresses to each of their customers, allowing them to serve individual certificates for their customer content publishers. During each TLS handshake, an edge server will return a TLS certificate that corresponds to the destination IP address targeted by the client. CDN providers usually charge their customers for dedicated IP addresses.

3.1.2 How These Deployments Leak Destination

The described HTTPS deployments may leak the identity of the customer CDN websites in one of the following ways, enabling low-cost censorship: **(1) TLS certificate:** In the A2, B1, and B2 HTTPS deployments, the TLS certificate returned by a CDN edge server may reveal the identity of the customer domain name. This enables the censors to use DPI to block CDNBrowsing connections to forbidden CDN customers without disrupting connections to non-forbidden customers. **(2) The SNI field:** In the B1 deployment, the SNI field carries the requested domain name in cleartext, enabling censorship of the forbidden CDN domains. **(3) Dedicated IPs:** In the B2 deployment, censors can identify forbidden CDNBrowsing connections based on the (discoverable) mapping between IP addresses and forbidden customers.

These leakage attacks are *low-cost* and readily deployable over major off-the-shelf censorship boxes: The "TLS certificate" and "SNI field" tests each require a DPI box to inspect—a single packet—from an HTTPS connection, and the "dedicated IPs" test can be deployed using the standard IP address filtering.

We therefore conclude that *an effective CDNBrowsing system should be tailored to particular deployments of HTTPS by various real-world CDN systems.* Particularly, we find that CacheBrowser's one-size-fits-all solution makes it susceptible to the enumerated leakage attacks, enabling the censors to block it at *low-cost* against the major CDNs listed in Table 1.

3.2 Domain-Based Website Fingerprinting

Today's modern webpages are composed of several web objects including advertisements, CSS, and Javascript objects usually hosted on multiple domains. This requires a web browser to make multiple HTTP requests, potentially with multiple web servers, to render a typical webpage. This enables a **unique website fingerprinting attack** against CDNBrowsing systems, which we call *domain-based website fingerprinting.* The main intuition behind the attack is that different CDN customers hosted by the same CDN are likely to be composed of various objects. Therefore, the censors can fingerprint each CDN webpage based on the number of packets it exchanges with various domains. For instance, Figure 1 compares three CDN webpages regarding the number of packets they receive from various domains.

Note that domain-based website fingerprinting only applies to CDNBrowsing systems: traditional proxy-based circumvention systems like Tor and VPNs combine all of the objects of a circumvented webpage into a single encrypted connection to the proxy, therefore concealing from the censors the packets exchanged with various domains.

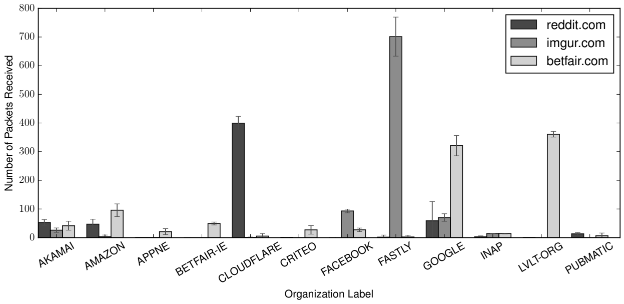


Figure 1: The number of packets received from 10 different organizations (domains) for three CDN webpages.

Implementing a domain-based fingerprint: We design a domain-based website fingerprinting algorithm to identify CDNBrowsing destinations. Our domain-based fingerprint is based on a decision-tree classifier and uses the amount of incoming and outgoing traffic from/to various domains as the feature vector.

In our analysis, we label each packet based on the domain of its destination IP address using `whois`. For example, a request to ad.doubleclick.net, an advertisement business owned by Google, will be labeled `GOOGLE` and a request to analytics.twitter.com will be labeled `TWITTER`. We also group together network domains that belong to the same organization but have different `whois` names for users in different regions, e.g., `LINODE-US` and `LINODE-UK`.

To evaluate our domain-based website fingerprinting algorithm, we chose a set of 100 pages from among websites blocked in China and Iran with full HTTPS support. From the traces of the 100 monitored pages we extracted 390 unique domain labels which we reduce to 249 after grouping them together as described above. This results in 498 features (incoming and outgoing traffic for each domain) across our dataset. For each of the webpages in our dataset we browse the page 40 times, where we use 30 of the runs for training and the remaining 10 during the detection. Our algorithm is able to identify CDNBrowsing destinations with a high accuracy of 0.991 ± 0.002.

Comparing with regular website fingerprinting: We also run the state-of-the-art website fingerprinting attack of Wang et al. [39] over the same set of CDNBrowsing webpages described above. The attack runs a k-NN classifier and is shown to have superior performance compared to previous website fingerprinting algorithms. We achieve a 0.94 ± 0.002 accuracy in identifying the CDNBrowsing webpages in our dataset using Wang et al.'s attack (within 2000 rounds for weight adjustments). As can be seen, our domain-based fingerprinting algorithm slightly outperforms the state-of-the-art fingerprinting attack of Wang et al. (0.991 ± 0.002 compared to 0.94 ± 0.002) even though our feature set was smaller than Wang et al.'s. More importantly, **our simple domain-based algorithm is two-orders of magnitude faster than the state-of-the-art website fingerprinting** of Wang et al. Specifically, while Wang et al.'s algorithm takes 90 CPU seconds for training and 0.05 CPU seconds for classification on an Intel Xeon 3.5 GHz processor, our algorithm takes only 0.60 and 10μ CPU seconds for training and classification, respectively.

3.3 Relevance of the Attacks to Domain Fronting

As introduced in Section 2.5, domain fronting [19] runs circumvention proxies over web services with shared IP addresses, such as app engines, cloud services, and CDNs. Particularly, meek is a domain fronting pluggable transport for Tor that, among other things, runs over Amazon CloudFront.

As domain fronting takes a similar approach to CDNBrowsing, we discuss how the attacks discussed in this section are relevant to it.

Destination Leakage in HTTPS. The HTTPS leakage attacks of Section 3.1 are also relevant to CDN-based domain fronting, however, they are *not* as critical compared to CDNBrowsing. In CDN-based domain fronting, it suffices to find as few as a single CDN provider that does not leak website identity through HTTPS (i.e., the single CDN provider can be used to set up domain fronting proxies like meek [19]). By contrast, CDNBrowsing needs to be able to obtain censored content from various CDN providers without leaking destination identity.

Website Fingerprinting. The domain-based website fingerprinting attack introduced in Section 3.2 is unique to CDNBrowsing and does *not* work against domain fronting systems. This is because, like other proxy-based circumvention systems, domain fronting proxies bundle all web objects of a circumvention session into a single encrypted connection, i.e., a Tor connection in meek. Therefore, the censors will not be able to identify the destination domains of the target web session, preventing them from performing our domain-based website fingerprinting attack. In fact, the domain-based website fingerprinting attack is exclusively unique to CDNBrowsing and does not work on *any* other circumvention system.

On the other hand, traditional website fingerprinting mechanisms such as Wang et al.'s [39] are applicable to domain fronting (and other proxy-based circumvention systems), and have been investigated in previous work.

4. CDNReaper: A PRACTICAL CDNBrowsing SYSTEM

We design and implement CDNReaper, a CDNBrowsing system that defeats the attacks introduced in Section 3. Our code is available online at https://github.com/CacheBrowser.[1]

4.1 Threat Model

We consider CDNReaper clients to be inside censoring ISPs. A **censoring ISP** deploys the main censorship techniques of IP filtering, DNS interference, and Deep-Packet Inspection (DPI) [14, 26] to prevent its users from accessing forbidden Internet destinations, also to stop them from using circumvention systems. The censors can also perform active probing mechanisms [22, 41], as well as offline attacks such as machine-learning based website fingerprinting [39].

We assume that **censors are rational**, i.e., they refrain from actions that will interfere with non-prohibited Internet activities of their benign, non-circumvention citizens. In particular, we assume that the censors do not block *all* encrypted traffic as encryption is essential for various popular non-forbidden Internet services. Additionally, we assume

[1] We re-use the existing CacheBrowser repository and project name due to its popularity.

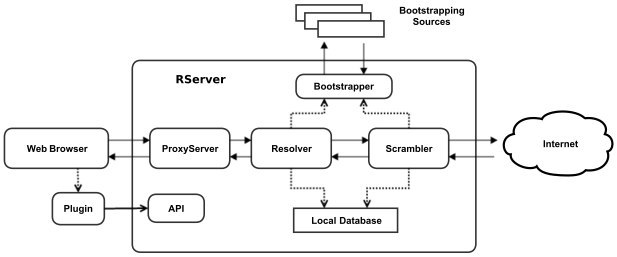


Figure 2: CDNReaper's architecutre.

that the censors do not entirely block a commercial CDN provider (e.g., by IP blacklisting all edge servers), since this will block all the non-forbidden webpages hosted on it. The censors, however, may perform—selective—blocking of CDN domain names and encrypted connections.

We assume that (at least some of) **commercial CDN providers are oblivious to censorship**. An oblivious CDN provider does not fully cooperate with either the censors or circumvention systems. No CDNBrowsing system can take advantage of a CDN provider who is not willing to be used for CDNBrowsing; such a CDN provider can simply remove all of the forbidden content publishers from its network in the first place, e.g., as done by Chinese CDN providers [21]. Also, a non-oblivious CDN provider may only cooperate with some censoring governments, but not all of them. For instance, Akamai is only known to be partially cooperating with the Chinese censors [21].

4.2 High-level Design

To prevent the attacks introduced in Section 3, CDNReaper needs to be able to make modifications to the HTTPS traffic of the clients. CacheBrowser's simple design [21], which simply modifies /etc/hosts to change DNS mappings, does not offer the flexibility to modify HTTPS connections. We therefore design CDNReaper based on a clean-slate architecture, which is shown in Figure 2. The core of CDNReaper is *RServer*, which runs as a local HTTP/HTTPS proxy on the client's machine. Our design is in nature similar to Tor client software that uses a SOCKS Proxy. RServer receives Internet requests (e.g., HTTP GETs) from client applications such as a web browser, and proxies them to the Internet possibly after making the required changes on the traffic to ensure unobseravbility. Unlike CacheBrowser's one-size-fits-all approach, CDNReaper treats connections to various CDN providers differently by tailoring the defense mechanisms to each CDN's particular HTTPS implementation. Our system's use of an HTTP proxy allows it to be used by any application that talks HTTP, particularly web browsers and download managers. A web browser can simply use CDNReaper as an HTTP proxy; we have specifically developed Chrome and Firefox plugins that interact with CDNReaper's RServer to provide additional features, such as enabling CDNReaper for specific sites.

As depicted in Figure 2, the three main components of RServer communicate as a chain, each processing the requests sequentially. ProxyServer is an HTTP/HTTPS proxy server that receives requests from user applications, e.g., a browser. It intercepts HTTPS requests by acting as a *man-in-the-middle* block using a locally generated and privately stored certificate, which is trusted by the client (each client generates her own local certificate). ProxyServer forwards a received HTTP request to Resolver, which will search in Local Database for information on how to process the connection to the requested domain name (if no entry is found, the Bootstrapper will be used as described later). Resolver will replace the requested domain name with the resolved IP address. Resolver also applies per-CDN rules on HTTPS connections to ensure unobservability, as will be discussed in Section 4.3. Note that the requests for non-censored domains will be processed normally through regular DNS queries.

A key component of CDNReaper's design is Scrambler. It manipulates HTTP traffic to defeat website fingerprinting, e.g., by modifying the objects queried on behalf of client applications. We detail Scrambler in Section 4.4.

If the information needed to browse a CDN website is missing in Local Database, RServer will use Bootstrapper to obtain the information. The obtained information will be cached in Local Database with a TTL. Bootstrapper is designed as a CDNBrowsable service itself, as will be detailed in Section 4.5.

We also build an API for CDNReaper to enable communications with third-party applications like browsers and download managers. We particularly build a Chrome extension and a Firefox plugin that use this API to communicate with RServer.

4.3 How CDNReaper Removes HTTPS Leakage

In Section 3.1 we discussed how the real-world deployments of HTTPS by in-the-wild CDN networks may leak the identity of the visited CDN websites, enabling the censorship of an unprotected CDNBrowsing system. CDNReaper is designed to prevent such leakage.

Our investigation of the most popular CDN systems in Section 3.1 demonstrates that the deployment of HTTPS varies across in-the-wild CDN providers. This suggests that a one-size-fits-all solution like that of CacheBrowser is prone to failure and *a practical CDNBrowsing system should tailor its mechanism to specific CDN systems.*

We start by describing how CDNReaper deals with removing identity leakage for the two CDN systems of Akamai and CloudFlare, which are examples of easy and challenging CDNs for CDNReaper. We will then describe CDNReaper's approach towards other CDN systems. In our experiments, we browse the top 10,000 Alexa websites from 20 geographically-dispersed Planetlab [10] nodes in order to identify multiple edge servers of different CDNs. For each CDN, we establish HTTPS connections to the identified edge servers, inspect the connections, and evaluate the impact of our countermeasures.

Akamai (An easy case). Our analysis finds that Akamai's mapping system treats HTTPS (secure) webpages differently than HTTP (insecure) webpages. Specifically, while HTTP-only customers of Akamai share edge server IP addresses, each HTTPS customer of Akamai is assigned to a dedicated, non-shared set of IP addresses. This simplifies the handling of customer TLS certificates for Akamai; we observe that once a client connects to an IP address dedicated to a specific Akamai HTTPS customer (e.g., https://customer1.com), the edge server will automatically return the TLS certificate for the `customer1` domain to the client regardless of the value of the SNI field (e.g., even if the SNI specifies a `customer2` domain name).

Ignoring the SNI field by Akamai edge servers allows CDNReaper to simply replace a forbidden website's SNI entry with a non-forbidden domain name, or even to remove the SNI entry. Still, the dedicated IP addresses of a forbidden website as well as the returned certificate containing the forbidden domain name enable censorship. To counter, CDNReaper requests a forbidden HTTPS website from edge server IP addresses other than those dedicated by the mapping system for it. We performed comprehensive experiments to see if this results in successful connections. We find that *all* insecure edge servers of Akamai respond correctly to HTTPS requests for any of the 400 Akamai customer websites we tried. An insecure edge server simply returns an Akamai wildcard certificate (e.g., an akamaihd.net certificate) for these connections, effectively hiding the actual domain names. For secure edge servers (i.e., IP addresses dedicated to other customers), we identified 410 edge servers (out of a total of 6,800 secure edge servers) that correctly responded to the HTTPS requests for other domain names, while the rest returned an `HTTP 403 Permission Denied` error.

For both secure and insecure edge servers, the returned certificate is different than the domain name requested by the client (i.e., it is either the certificate for another Akamai website or an Akamai wildcard certificate). CDNReaper simply ignores the received certificate to avoid a browser error. Note that this does *not* weaken the security of HTTPS: CDNReaper still verifies that the received certificate is in possession of an Akamai edge server, i.e., it is an Akamai domain certificate or a certificate of another Akamai customer.

CloudFlare (A challenging case). Among all CDN providers we evaluated, CloudFlare has the least flexibility to be used for CDNBrowsing. CloudFlare provides both free and paid HTTPS services to its customers. For its free HTTPS service, CloudFlare shares edge server IP addresses among its customers. To be able to return the right certificate to the clients, CloudFlare combines the use of the SNI and SAN certificates, as described in Section 3.1. A CloudFlare edge server uses the SNI entry to return a TLS certificate to a requesting client, which will be a SAN certificate shared with multiple CDN customers. Unlike Akamai and all the other CDN systems we evaluated, CloudFlare edge servers enforce a *strict access control* based on the SNI entry. That is, if a client requests a webpage whose domain is different from what specified in the SNI, the edge server will close the connection with a `403 Permission Denied` error message. Also, CloudFlare's shared edge servers deny serving HTTPS connects that miss an SNI field. CloudFlare's paid HTTPS service, on the other hand, assigns dedicated IP addresses to the customers. The dedicated edge servers serve the content even if the SNI entry is missing, however, they solely serve the content for the assigned customer domain name. Therefore, requesting a forbidden CDN content from the dedicated edge server of another customer domain will result in a `403` error.

Based on these observations, no CDNBrowsing system will be able to offer circumvention for CloudFlare against a competent censor who deploys DPI over the SNI field. Nonetheless, CDNReaper defeats IP address filtering and DNS interference for all HTTP and HTTPS connections to CloudFlare.

CDNReaper's CDN-Aware Treatment: CDNReaper uses the following mechanisms to prevent the three described types of leakage in TLS. **SNI leakage:** To remove identity leakage through SNI, CDNReaper replaces forbidden domain names in the SNI fields of its connections with non-forbidden domain names. The replaced domain name is selected based on the policies of the hosting CDN. **Deterministic IP addresses:** In *all* of the CDN systems we evaluated, clients can ask any "shared" edge server for arbitrary HTTPS content. For some CDNs, an HTTPS website can be accessed even through the edge severs dedicated to other customers. CDNReaper picks the edge server IP addresses according to each CDN's setting. **TLS certificate leakage:** Our countermeasures for SNI and deterministic IP leakages automatically mitigate information leakage in TLS certificates as well. As described before, dedicated edge servers return TLS certificates based on either the domain name specified in the SNI or the IP address. For instance, if the SNI field for a forbidden Akamai connection is changed to akamai.net, the edge server will return the certificate for akamai.net, not the forbidden website's certificate. For most CDNs, the shared edge servers by default return a certificate for the CDN wildcard domain.

For each connection, CDNReaper applies one or multiple of the following techniques based on the characteristics of the hosting CDN:

- **T1.** If the shared edge servers of the CDN accept HTTPS requests for arbitrary customer websites, ask forbidden content from an arbitrarily edge server. The edge server will respond with a CDN wildcard domain certificate.
- **T2.** If the dedicated edge servers accept HTTPS requests for other customer websites, contact the dedicated IP address of a non-forbidden domain to request content for a forbidden domain.
- **T3.** If the edge servers allow connections to have empty SNI fields, remove the SNI entry in forbidden HTTPS connections.
- **T4.** If the edge servers allow non-matching SNI entries, replace a forbidden connection's SNI with a non-forbidden domain name.

Table 2 shows the techniques that work for each of the evaluated CDNs. For CDNReaper to be able to browse forbidden websites on a particular CDN network, that CDN should support at least one of *T1* or *T2*. For CDNReaper's connections to be unobservable to competent DPI-powered censors, the hosting CDN network should support at least one of *T3* or *T4*.

4.4 How CDNReaper Defeats Traffic Analysis

In Section 3.2 we demonstrated the possibility of unique website fingerprinting attacks against CDNBrowsing systems. Specifically, we demonstrated a "domain-based" website fingerprinting attack, which works by recording the amount of traffic a CDNBrowsing client exchanges with different Internet domains. We also showed the feasibility of traditional website fingerprinting attacks [39] against CDNBrowsing, however, we demonstrated our CDNBrowsing-specific domain-based fingerprinting attack to be *two orders of magnitude* faster than the traditional fingerprinting attacks (and even slightly more accurate).

CDN	T1	T2	T3	T4
Akamai	✓	Some	✓	✓
Cloudfront	✓	✓	Some	✓
Cloudlfare	✓	✓	Some	✗
CDNetworks	✓	✓	✓	✓
Edgecast	✓	✓	✓	✓
Fastly	✓	✓	✓	✓
Incapsula	✓	✓	✓	✓

Table 2: The applicability of CDNReaper's countermeasures to various CDN systems. A CDN should support at least one of *T1* or *T2* to enable CDNBrowsing, and should support at least one of *T3* or *T4* for defeating the attacks.

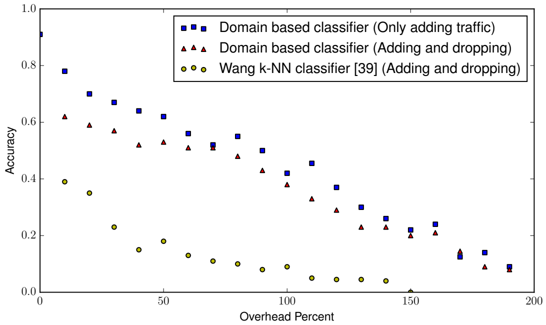


Figure 3: Accuracy of website fingerprinting attacks against CDNReaper for various overheads.

The `Scrambler` component in our CDNReaper design aims at resisting the introduced website fingerprinting attacks. `Scrambler` modifies the amount of traffic a CDNReaper client exchanges with different end-points, i.e., by injecting redundant traffic on some domains and/or removing (non-critical) traffic from other domains. `Scrambler`'s modifications are performed with two constraints: (1) not disrupting a client's normal browsing experience, and (2) limiting traffic overhead to a rate specified by the client.

Injecting traffic: The user configures a maximum overhead for `Scrambler`'s modifications (as we will show in our analysis, the overhead impacts resistance to fingerprinting attacks). For each network request from the browser, the `Scrambler` component may decide to send a *decoy* request along with the actual request, ensuring that the overhead does not surpass the user-specified limit. The decoy web requests are chosen from a list of n_{dom} popular domains maintained by `Scrambler` (n_{dom} is proportional to the overhead limit, as discussed later). For each of the n_{dom} domains, `Scrambler` remembers multiple URLs in order to send the decoy requests. `Scrambler` keeps record of the volume of bandwidth exchanged with each of the n_{dom} domains during the latest time interval of T seconds (we use $T = 10s$). The decoy destinations are selected from the list of decoy URLs such that the overall bandwidth is roughly uniformly distributed among the n_{dom} decoy domains. To do so, `Scrambler` picks from among domains with the lowest traffic in the past T seconds as the decoy domain.

Dropping traffic: Our analysis of the frontpage of the top 1,000 Alexa websites shows that on average close to 24% of the requests made by them are advertisement and analytical requests. As such objects have little impact on users' browsing experience, `Scrambler` may drop them in order to improve resistance to domain-based website fingerprinting attacks. For each dropped connection, `Scrambler` may add a decoy connection, as described earlier, subject to keeping traffic overhead below the user-specified limit. To refrain from dropping important web objects, `Scrambler` maintains a list of advertisement and analytical URL patterns that gets updated over time. A client can modify the list.

`Scrambler`'s performance in defeating attacks can get further improved by dropping/modifying other non-critical web requests. Particularly, some popular web objects, such as common JS or CSS libraries, are hosted on multiple domains, therefore `Scrambler` can change the requested domain. For instance, a request for some version of the JQuery library from the Google repository[2] can be modified to that of Microsoft[3] or Cloudflare.[4] We leave this to future work.

Evaluation: We evaluate the performance of the website fingerprinting attacks of Section 3.2 against the defenses deployed by our `Scrambler`. We perform our machine-learning based attacks using the same parameters as in Section 3.2. We vary n_{dom} from 10 to 150 proportional to the overhead limit. Figure 3 shows the accuracy of the attacks on CDNReaper for different overheads. Compared to the results in Section 3.2 against CacheBrowser (i.e., accuracies of 0.991 and 0.94 for domain-based and Wang et al. [39], respectively), the accuracy of both of the attacks drops significantly even for low overheads. As expected, increasing the overhead limit improves resistance to the attacks. Also, we can see that our CDNBrowsing-specific domain-based attack is still more successful compared to traditional fingerprinting attacks. This is because the `Scrambler` can not remove or highly modify the critical domains part of a web session. We also see that dropping traffic is more effective for lower overheads. Note that even a 200% overhead is not drastic, i.e., it implies that on average the client will need to load two extra pages per censored webpage.

4.5 A Practical Bootstrapper for CDNReaper

To be able to serve a forbidden webpage, CDNReaper client software needs to know the following information about the forbidden webpage: the CDN provider hosting the webpage (if any), the type of CDN hosting (i.e., partial or full CDN deployment), the settings of the hosting CDN (Table 2), and some (multiple) edge server IP addresses for the hosting CDN. This information is kept in `Local Database`. When a particular forbidden webpage is requested for the *first time*, CDNReaper's `Local Database` may not have all of the required information to serve that request. In this case, CDNReaper will use its `Bootstrapper` component (Figure 2) to obtain the missing information and record it in `Local Database` with a TTL. If `Bootstrapper` is unable to fulfill a request, either due to the requested domain not being hosted on a CDN or due to insufficient information, this will also be recorded in the `Local Database` to avoid redundant queries.

[2] ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js

[3] ajax.aspnetcdn.com/ajax/jQuery/jquery-1.12.2.min.js

[4] cdnjs.cloudflare.com/ajax/libs/jquery/1.12.2/jquery.js

CDNReaper's `Bootstrapper` is a CDNBrowsable service itself, therefore it offers the same resistance to censorship as CDNReaper. Specifically, our `Bootstrapper` connects to a *remote bootstrapping server* that is hosted on a "good" CDN network based on Table 2. Our bootstrapping server is hosted on Amazon Cloudfront, but one can run multiple remote bootstrapping servers each on a different CDN network to increase reliability.

Our remote bootstrapping server runs a RESTful Web API [34] to respond to queries. A client's `Bootstrapper` will establish a CDN-based HTTPS connection to the remote bootstrapping server in order to query information about blocked domain names. CDNReaper also supports local `Bootstrapper` files to be used as a bootstrapping source, along with a client interface for manually entering records for forbidden domain names.

5. EXTENDING THE REACH OF CDNBrowsing

As discussed earlier in Section 2.2, an inherent limitation of a CDNBrowsing system is that it can only browse certain types of censored webpages. In this section, we start by a comprehensive analysis of Internet webpages to evaluate their readiness to be browsed by a CDNBrowsing system like CDNReaper. Next, we will present several mechanisms that we have deployed to significantly increase the fraction of Internet webpages that can be browsed by a CDNBrowsing system.

5.1 Classifying Internet Websites

We evaluate the homepages of the top 10,000 Alexa websites and classify them into six classes based on their compatibility with CDNBrowsing.

- **Class 1** (full-CDN, protected HTTPS) A webpage that is fully hosted on shared-CDNs, and is accessible through HTTPS. Additionally, the hosting CDN provider enables CDNReaper to remove HTTPS information leakage as described in Section 4.3.
- **Class 2** (full-CDN, leaking HTTPS) A webpage that has all of its content hosted on shared-CDNs and one that can be accessed through HTTPS. However, the hosting CDN does not enable CDNReaper to remove HTTPS leakage as discussed in Section 4.3.
- **Class 3** (full-CDN, HTTP-only) A webpage that has all of its content hosted on shared-CDN networks, but without HTTPS support.
- **Class 4** (partial-CDN) A webpage that has only some of its content hosted on shared-CDN networks. Often, partial-CDN webpages host their sizable content such as images and videos on a CDN, and host user-specific content on their own origin servers.
- **Class 5** (private-CDN) A webpage that is hosted on a private-CDN network as defined in Section 2.6. Note that Class 5 overlaps with the two Classes of 4 and 6.
- **Class 6** (non-CDN) A webpage that has no content hosted on any shared-CDN networks.

Figure 4 shows the fraction of the webpages belonging to each of the defined Classes within the top 10,000 Alexa websites. The webpages in Class 1 can be browsed using CDNReaper with plausible unobservability against a competent censor deploying advanced DPI. However, this constitutes a small fraction of all webpages. The webpages in Class 2 can be browsed by CDNReaper only against a censor who does not inspect HTTPS fields (e.g., certificates and SNI) for forbidden domain names. Also, the webpages in Class 3 can be browsed by CDNReaper only in the presence of the censors who do not deploy URL/keyword filtering. We expect the fraction of pages in Class 3 to decrease over time with the growing popularity of encryption on the web.

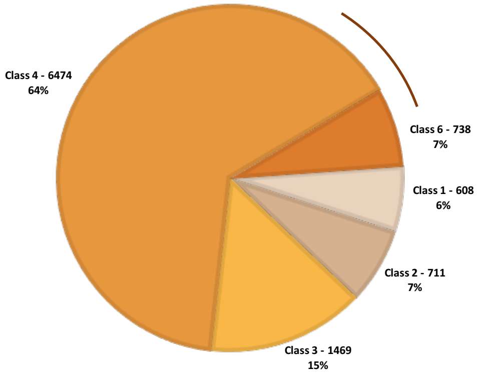


Figure 4: Classifying the top 10,000 Alexa websites based on their readiness to be browsed by CDNReaper.

In the following, we discuss several mechanisms that expand the reachability of CDNReaper by enabling it to browse websites in Class 4 to Class 6.

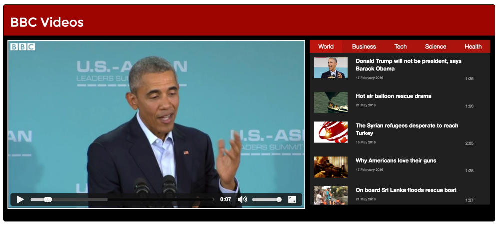
5.2 Supporting Partial CDN Webpages

As shown in Figure 4, a large fraction (64.7%) of the inspected websites are in Class 4, i.e., are partially hosted on shared-CDNs. Through manual inspection of these webpages, we observe that for many of these partial-CDN webpages, the CDN-hosted content is the core content of the webpage, potentially of the highest interest to the censored clients. Particularly, many partial-CDN webpages host sizable multimedia content like images, documents, and video streams on CDN in order to reduce hosting expenses and increase reliability. Based on this observation, we use the mechanisms described in the following to enable CDNBrowsing of partial-CDN websites.

5.2.1 Content Wrappers

As discussed earlier, many of the partial-CDN websites we evaluated host their high-interest content on CDNs. For instance, the popular flickr.com hosts its image and video content on shared-CDN networks, but not the rest of the website. We introduce *content wrapper*s to enable CDNBrowsing of such partial-CDN websites. A content wrapper is a simple HTML page that renders the CDN-hosted parts of a partial-CDN webpage, e.g., the images and videos. One can generate content wrappers for various censored webpages and distribute them among censored clients through out-of-band channels, e.g., email. We call such content wrappers as *offline* wrappers.

Alternatively, one can run a *live* content wrapper as a CDNBrowsable webpage, e.g., by hosting it on a shared-CDN. Note that even live content wrappers have minimal operational costs to the volunteers who set them up. This is because the bandwidth-heavy parts of the wrapped websites



Donald Trump will not be president, says Barack Obama
17 February 2016 1:35

Hot air balloon rescue drama
21 May 2016 1:50

The Syrian refugees desperate to reach Turkey
16 May 2016 2:05

Why Americans love their guns
17 February 2016 1:28

On board Sri Lanka floods rescue boat
21 May 2016 1:37

Figure 5: A content wrapper for the partial-CDN webpage of bbc.co.uk.

are simply downloaded from the CDN network paid by the original content publisher, and the volunteer only has to host small HTML wrappers, which can be hosted on a low-QoS, free CDN network. To demonstrate, we have made a content wrapper for bbc.co.uk, a partial-CDN website blocked in China and Iran, as shown in Figure 5. Our wrapper renders the bandwidth-intensive videos of BBC directly from Akamai, which is paid directly by BBC. We have hosted a live version of our wrapper on a free hosting service at https://bbcvids.site.

5.2.2 Dynamic Mirroring of Non-CDN Content

Content wrappers, introduced above, are ideal for partial-CDN webpages whose non-CDN content are static (i.e., do not change frequently over time). We devise a dynamic mirroring mechanism to support partial-CDN webpages whose non-CDN content may change dynamically. Our dynamic mirror is similar to a "live" wrapper in that the clients will obtain the CDN-hosted content directly from the partially hosting CDN. A dynamic mirror is also hosted on a shared-CDN to be browsable by CDNReaper. By contrast, a dynamic mirror obtains the non-CDN parts of the webpage directly from the content publisher upon client requests.

To demonstrate the simplicity and effectiveness of this approach, we set up a dynamic mirror for https://www.tumblr.com, which is a partial-CDN blogging website censored in Iran. We deploy our mirror on Heroku,[5] a CDNBrowsable cloud application platform that offers free hosting service for low-bandwidth applications. To run our dynamic mirror, we deployed a `NodeJS` server on get-tumblr.herokuapp.com, which fetches and returns the —non-CDN —HTML content of Tumblr blogs. For example, the blog https://techedblog.tumblr.com is mirrored at https://get-tumblr.herokuapp.com/techedblog, which is browsable by CDNReaper. Once the mirrored HTML is loaded, the *rest of the content will load from the shared-CDN* that partially hosts Tumblr.

Automated Mirror Creation: To facilitate the creation of mirrors, we have built an online tool called *MirrorMySite*, hosted at http://mirror-my.site, which automates the creation of such mirrors. To use this tool, the user is required to create a free Heroku account. The user will then simply enter the webpage URL intended to be mirrored, and the mirror will be deployed as a NodeJS server on a Heroku subdomain of the user's choice, e.g. mybbc.herokuapp.com. The link, then, can be used by one or many censored clients.

Comparison to Static Mirrors: The *Collateral Freedom project* [11] also mirrors several popular censored websites on cloud platforms like Amazon EC2 and GitHub. However, such mirrors (1) only support static web content like HTML pages, and (2) are not live and need to get manually updated frequently by their operators.

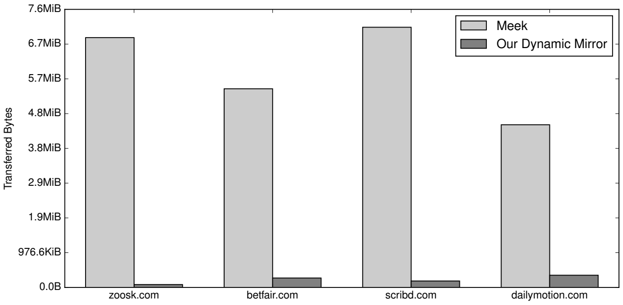


Figure 6: Comparing our dynamic mirrors with meek [19] regarding traffic load for sample partial-CDN websites. The traffic load consists of the upstream and downstream traffic used by the third-party proxy in order to fulfill a client's requests.

Comparison to Domain Fronting: As discussed in Section 2.5 a major advantage of CDNBrowsing over CDN-based domain fronting is its significantly lower operational costs. This is because domain fronting proxies simply relay *all* of the circumvented traffic for the censored users, therefore the volunteer relays need to pay the significant operational costs (bandwidth, CPU, etc.). As mentioned earlier, the meek pluggable transport has so far cost Tor a total of $26,536 in operational charges ($2,479 for the last reported month) [30] even despite using a discounted rate due to a "free research grant" [19] and a 1.5–3 MB/s bandwidth cap on users' traffic. CDNBrowsing of full-CDN webpages has *zero* cost to CDNReaper since all traffic is directly obtained from the edge servers paid by the censored content publishers. Even for partial-CDN websites, the dynamic mirrors do not impose significant bandwidth costs compared to proxy-based circumvention systems. Figure 6 compares the traffic load proxied by our dynamic mirror versus meek for multiple partial-CDN websites. As can be seen, the loads on our mirrors are almost negligible compared to meek; our dynamic mirrors can be set up on free CDNs, as demonstrated above, due to their very low bandwidth.

5.2.3 Importing Login Credentials

An increasing number of content publishers maintain user accounts for their clients, i.e., users will have to log in to their websites. Examples includes social networking websites, online banking, etc. In order to protect their sensitive client information (e.g., passwords), such content publishers commonly refrain from hosting their log-in webpages on third-party CDN networks, even if the rest of their content is hosted on CDN. That constitutes a significant fraction of partial-CDN websites. To enable CDNReaper to browse such partial-CDN websites, our CDNReaper browser plugin allows users to import their log-in credentials that are obtained through out-of-band channels or previous sessions as website cookies.

5.3 Supporting Private CDNs

Some content publishers use private CDNs (as defined in Section 2.6) to host their content, either partially or en-

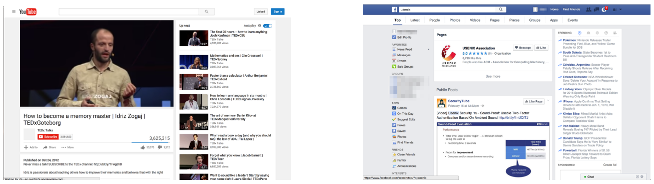
[5] https://www.heroku.com/

tirely. For instance, this includes 10% of the top 10,000 websites we analyzed (Class 5 in Figure 4). Holowczak et al. [21] argue that a CDNBrowsing system can not circumvent private CDNs, since the censors can IP-filter all the edge servers of a private CDN at no collateral damage of blocking non-forbidden websites. However, we demonstrate that CDNReaper can be utilized for browsing important private CDNs. First, some private CDNs are so dispersedly implemented that IP blocking all of their edge servers will be both resource-intensive and error-prone to the censors. Second, and more importantly, even some of the private CDNs are shared between multiple sibling content publishers, therefore, IP filtering will cause collateral damage. Even though the number of websites unwillingly blocked in this case would be smaller compared to an IP-filtered shared-CDN, we find that the co-hosted websites are among the most popular non-forbidden websites, including Instagram, Google, Microsoft, and Yahoo.

We evaluate the top 10,000 Alexa websites for private CDNs. Since the internal architecture of most content publishers are not publicly disclosed, we consider a website to be on a private CDN based on the variety of the IP addresses serving it. For each website that is not hosted on one of the known shared-CDNs [5], we browse it from 20 geographically dispersed PlanetLab nodes and collect the hosting IP addresses. If 25% or more of the collected IP addresses for a website are different we consider the website to be hosted on a private-CDN (we choose the 25% threshold by manually inspecting multiple known private-CDN websites). This leads us to 960 private CDNs in Alexa's top 10,000, approximately 10% of the websites. Note that the list of identified private-CDN websites is a subset of both Class 4 and Class 6 in Figure 4. This is because some of the partial-CDN websites in Class 4 host part of their content on shared CDNs and part on private CDNs.

We particularly find instances of sibling content publishers who share private-CDNs where only some of the sharing content publishers are considered to be forbidden by the censors. As one example, the different websites owned by Google Inc., e.g., Google Search and YouTube, are hosted on the same private CDN. We perform experiments from a node in Iran, who blocks only some Google websites; particularly, Iran censors www.youtube.com but not www.google.com. We observe that Iran's censorship of www.youtube.com is performed only through DNS interference, but not IP filtering, presumably to avoid the collateral damage of blocking non-forbidden Google products like www.google.com. We use CDNReaper to access the censored www.youtube.com from our Iranian node by connecting to arbitrary IP addresses of Google's private CDN that serve www.google.com; this is shown in Figure 7a.

As another example of co-located private-CDN websites, we case study Facebook's private CDN. Both facebook.com and messenger.com are owned by Facebook, and our experiments show that both websites use the same private CDN. As shown in Figure 7b, we were able to successfully use CDNReaper to browse facebook.com using CDNReaper by connecting to the edge server IP addresses obtained for www.messenger.com. Note that facebook.com is also a Class 4 website, i.e., it hosts all of its static content (images, videos, etc.) on the Akamai shared-CDN, but its HTML content is hosted on a private-CDN.



(a) YouTube loaded using google.com IPs

(b) Facebook loaded using messenger.com IPs

Figure 7: CDNReaper loading blocked private-CDN websites

6. CONCLUSIONS

We discover various low-cost attacks against CDNBrowsing systems. Particularly, we demonstrate that the specific implementation of HTTPS by in-the-wild CDN providers may leak the identity of the websites browsed thorough CDNBrowsing, therefore, arguing that CDNBrowsing systems should be tailored to specific CDN systems, in contrast to the one-size-fits-all solution in previous work [21]. We also devise CDNBrowsing-specific website fingerprinting attacks that are able to identify CDNBrowsing traffic with very high accuracy, faster than traditional website fingerprints. To counter the attacks, we design and fully implement a new CDNBrowsing system called CDNReaper.

We additionally perform comprehensive measurements to classify Internet websites based on their readiness for CDNBrowsing. To further increase the reach of CDNBrowsing, we devise several mechanisms that enable CDNBrowsing of partial-CDN webpages.

7. ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their insightful feedback. This work was supported in part by NSF CAREER grant CNS-1553301.

8. REFERENCES

[1] S. Aryan, H. Aryan, and A. Halderman. Internet censorship in Iran: A first look. In *FOCI*, 2013.

[2] C. Brubaker, A. Houmansadr, and V. Shmatikov. CloudTransport: Using Cloud Storage for Censorship-Resistant Networking. In *PETS*, 2014.

[3] S. Burnett, N. Feamster, and S. Vempala. Chipping Away at Censorship Firewalls with User-Generated Content. In *USENIX Security*, 2010.

[4] R. Carroll. China steps up web censorship and blocks HSBC. http://www.theguardian.com/world/2014/nov/18/china-blocks-hsbc-web-crackdown-censorship, November 2014.

[5] Latest List of Vendors in the Content Delivery Ecosystem. http://blog.streamingmedia.com/2014/07/cdnvendors.html.

[6] 5 Reasons to Implement a Content Delivery Network (CDN). http://blog.newrelic.com/2012/12/18/5-reasons-to-implement-a-cdn/.

[7] Joint Statement on Censorship and Science: A Threat to Science, the Constitution, and Democracy. http://humanrightshouse.org/Articles/16300.html, April 2011.

[8] Threats to Internet freedom – political censorship and government control over infrastructure. http://humanrightshouse.org/Articles/16300.html, April 2011.

[9] China Continues Its Crackdown On VPN Services. https://techcrunch.com/2015/09/07/china-continues-its-crackdown-on-vpn-services/, September 2015.

[10] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *SIGCOMM CCR*, 33(3):3–12, 2003.

[11] Collateral Freedom. https://openitp.org/pdfs/CollateralFreedom.pdf, 2013.

[12] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. Updated by RFC 6818.

[13] J. Cowie. Egypt Leaves the Internet. http://www.renesys.com/blog/2011/01/egypt-leaves-the-internet.shtml, January 2011.

[14] Defeat Internet Censorship: Overview of Advanced Technologies and Products. http://www.internetfreedom.org/archive/Defeat_Internet_Censorship_White_Paper.pdf, 2007.

[15] R. Dingledine and N. Mathewson. Design of a Blocking-Resistant Anonymity System. https://svn.torproject.org/svn/projects/design-paper/blocking.html.

[16] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton. Protocol Misidentification Made Easy with Format-Transforming Encryption. In *CCS*, 2013.

[17] D. Eastlake. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066 (Proposed Standard), Jan. 2011.

[18] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing Web Censorship and Surveillance. In *USENIX Security*, 2002.

[19] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson. Blocking-resistant Communication through Domain Fronting. In *PETS*, 2015.

[20] J. Geddes, M. Schuchard, and N. Hopper. Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention. In *CCS*, 2013.

[21] J. Holowczak and A. Houmansadr. CacheBrowser: Bypassing Chinese Censorship without Proxies Using Cached Content. In *CCS*, 2015.

[22] A. Houmansadr, C. Brubaker, and V. Shmatikov. The Parrot is Dead: Observing Unobservable Network Communications. In *IEEE S&P*, 2013.

[23] A. Houmansadr, G. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention Infrastructure Using Router Redirection with Plausible Deniability. In *CCS*, 2011.

[24] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer. I Want My Voice to Be Heard: IP over Voice-over-IP for Unobservable Censorship Circumvention. In *NDSS*, 2013.

[25] S. Kelly, M. Earp, L. Reed, A. Shahbaz, and M. Truong. Tightening the Net: Governments Expand Online Controls (Freedom on the Net 2014). https://freedomhouse.org/sites/default/files/FOTN_2014_Full_Report_compressedv2_0.pdf, 2014.

[26] C. Leberknight, M. Chiang, H. Poor, and F. Wong. A Taxonomy of Internet Censorship and Anti-censorship. http://www.princeton.edu/~chiangm/anticensorship.pdf, 2010.

[27] C. Lecher. Internet censorship reaching dangerous levels in Turkey. http://www.todayszaman.com/national_internet-censorship-reaching-dangerous-levels-in-turkey_393727.html, July 2014.

[28] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu. When HTTPS Meets CDN: A Case of Authentication in Delegated Service. In *IEEE S&P*, 2014.

[29] R. McPherson, A. Houmansadr, and V. Shmatikov. CovertCast: Using Live Streaming to Evade Internet Censorship. In *PETS*, 2016.

[30] Summary of meek's costs, March 2016. https://lists.torproject.org/pipermail/tor-project/2016-April/000271.html.

[31] H. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. SkypeMorph: Protocol Obfuscation for Tor Bridges. In *CCS*, 2012.

[32] Z. Nabi. The anatomy of Web censorship in Pakistan. In *FOCI*, 2013.

[33] A Simple Obfuscating Proxy. https://www.torproject.org/projects/obfsproxy.html.en.

[34] RESTful Web APIs. http://restfulwebapis.com/.

[35] P. Sands. Syria Tightens Control over Internet. http://www.thenational.ae/news/world/middle-east/syria-tightens-control-over-internet, September 2008.

[36] Ultrasurf. http://www.ultrareach.com.

[37] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton. Seeing through Network-Protocol Obfuscation. In *CCS*, 2015.

[38] Q. Wang, X. Gong, G. Nguyen, A. Houmansadr, and N. Borisov. CensorSpoofer: Asymmetric Communication Using IP Spoofing for Censorship-Resistant Web Browsing. In *CCS*, 2012.

[39] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security*, 2014.

[40] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh. StegoTorus: A Camouflage Proxy for the Tor Anonymity System. In *CCS*, 2012.

[41] P. Winter and S. Lindskog. How the Great Firewall of China Is Blocking Tor. In *FOCI*, 2012.

[42] E. Wustrow, S. Wolchok, I. Goldberg, and J. Halderman. Telex: Anticensorship in the Network Infrastructure. In *USENIX Security*, 2011.

[43] S. Yang. China Has Escalated Internet Censorship to a New Level. http://www.businessinsider.com/china-blocks-vpns-2015-1, January 2015.