Domain Shadowing: Leveraging Content Delivery Networks for Robust Blocking-Resistant Communications

Mingkui Wei
Cybersecurity Engineering
George Mason University, Fairfax, VA, 22030

Abstract

We debut *domain shadowing*, a novel censorship evasion technique leveraging content delivery networks (CDNs). Domain shadowing exploits the fact that CDNs allow their customers to claim arbitrary domains as the back-end. By setting the front-end of a CDN service as an allowed domain and the back-end a blocked one, a censored user can access resources of the blocked domain with all "indicators", including the connecting URL, the SNI of the TLS connection, and the `Host` header of the HTTP(S) request, appear to belong to the allowed domain. Furthermore, we demonstrate that domain shadowing can be proliferated by *domain fronting*, a censorship evasion technique popularly used a few years ago, making it even more difficult to block. Compared with existing censorship evasion solutions, domain shadowing is lightweight, incurs negligible overhead, and does not require dedicated third-party support. As a proof of concept, we implemented domain shadowing as a Firefox browser extension and demonstrated its capability in circumventing censorship within a heavily censored country known by its strict censorship policies and advanced technologies.

1 Introduction

Domain fronting (Df) is a censorship evasion technique proposed in 2015 [18], which allows censored users to circumvent censorship by exploiting the following two facts. On the one hand, many content delivery networks (CDNs) solely rely on the `Host` header of an incoming HTTPS request to determine the origin, even though this header is inconsistent with the server name indication (SNI) [13] used to establish the transport layer security (TLS) tunnel. On the other hand, the censor can only see the SNI of an HTTPS connection but not the `Host` header inside the TLS tunnel. A censored user can circumvent censorship by sending an HTTPS request to a CDN requesting an allowed domain, but set the `Host` header to a blocked one. As long as both domains dwell on the same CDN, the CDN will route the request to the blocked domain according to the `Host` header but have the TLS connection still appear to belong to the allowed domain. The blocking-resistance of domain fronting derives from the significant "collateral damage", i.e., to disable domain fronting, the censor needs to block users from accessing the entire CDN, resulting in *all* domains on the CDN inaccessible. Because today's Internet relies heavily on web caches and many high-profile websites also use CDNs to distribute their content, completely blocking access to a particular CDN may not be a feasible option for the censor. Because of its strong blocking-resistant power, domain fronting has been adopted by many censorship evasion systems since it has been proposed [24, 28, 34, 36]. In the last two years, however, many CDNs began to disable domain fronting by enforcing the match between the SNI and the `Host` header [2, 3, 38], which makes domain fronting less effective.

In this paper, we debut *domain shadowing (Ds)* as a novel censorship evasion technique. Similar to domain fronting, domain shadowing also leverages CDNs to achieve censorship evasion. However, domain shadowing differs from domain fronting in that it does not manipulate the SNI and the `Host` header of an HTTPS request. Instead, it exploits a legitimate CDN feature that specifies the connection between the front-end and the back-end domains. Specifically, we found that most CDNs allow users to claim arbitrary domains as the back-end of a CDN service without imposing any limitations. To circumvent censorship, a censored user can set an allowed domain (namely the *shadow domain*) as the front-end, and a blocked domain as the back-end, of a CDN service. By sending HTTP(S) requests to the shadow domain, the CDN will faithfully fetch the web document from the back-end, i.e., the blocked domain, and "repackages" the response into a new response with the URL, SNI, and `Host` header all "rebranded" as the shadow domain, enabling the user to visit the blocked domain "in the name of" the allowed domain.

Compared with other censorship evasion systems, domain shadowing is lightweight and incurs negligible overhead. Besides a valid CDN account, the operation of domain shadowing does not require any support from a dedicated third party, as most other systems do. The essential task for a user to use domain shadowing is appropriately configuring the front-end and back-end domains in his/her CDN account, which is a one-time task and can even be automated using CDN-provided APIs or SDKs. The only performance penalty would be waiting for such configurations to be deployed by the CDN when a domain is being visited for the first time, which costs less than 20 seconds most of the time. On the other hand, the subsequent web browsing can be even faster than directly connecting to the origin server.

As a proof-of-concept, we implemented domain shadowing as a Firefox extension based on Fastly's CDN service [15], which automates all configuration procedures using Fastly's web APIs and is intuitive to use by regular users. We demonstrate that this extension enables censored users to access blocked websites within a heavily censored country known for its strict censorship policies and advanced techniques.

To summarize, our contributions are:

1. We exhibit *domain shadowing* as a novel censorship evasion technique leveraging content delivery networks. We analyze its potential and demonstrate that it can circumvent most censorship techniques.

2. We demonstrate domain fronting can corroborate with domain shadowing and proliferate its resilience. The combined solution, namely the *DfDs (domain fronting + domain shadowing)*, can achieve even stronger blocking-resistance.

3. We implement domain shadowing as a Firefox extension and showcase its capability of circumventing censorship in a heavily censored country. We will open-source our implementation to benefit the research community.

4. We thoroughly evaluate the benefits and limitations of domain shadowing; discuss tactics to stay ahead in the potential arm-race among the censor, the CDN, and the user; and analyze domain shadowing's security impacts to the CDN, the publisher, and the user. Our work paves the way for further development of a full-fledged censorship evasion system based on this newly proposed technique.

The rest of the paper is structured as follows. In Section 2, we introduce background knowledge related to domain shadowing, and in Section 3, we explain in detail how domain shadowing works. In Section 4, we demonstrate our experimental implementation of domain shadowing as a Firefox browser extension, and showcase its capacity to circumvent censorship. In Sections 5, 6, and 7, we discuss domain shadowing's advantages and limitations from the perspective of usability, censorship blocking-resistance, and security impact, respectively. Related works are discussed and compared in Section 8, and we finally concluded our work in Section 9.

2 Background

2.1 Internet Censorship Techniques

Censorship techniques for identifying and blocking website browsing have been extensively studied in many research works [1, 23, 43, 46]. In general, these techniques can be classified into three categories: IP filtering, DNS interference, and deep packet inspection (DPI).

IP filtering checks the IP address a user attempts to connect to and blocks the request if the IP belongs to a blocklist. IP filtering is low-cost, straightforward, and effective if the prohibited website has a static IP known by the censor. In the age of cloud computing, however, IP filtering becomes less effective since webservers hosted on clouds may be assigned with dynamic IPs by the cloud service provider [23, 30].

In *DNS interference* [25, 26], the censor intercepts and inspects the DNS query message sent by a user. If the queried domain is prohibited, the censor may simply refuse to respond or respond with a fake IP [23]. However, the user can skip the DNS query step and directly connect to the webserver's IP address to bypass DNS interference.

Assisted by machine learning and data mining techniques, the *deep packet inspection (DPI)* [46] inspects the content of the packets among the censored network to identify suspicious traffic [20, 25]. However, DPI is unable to inspect encrypted packets such as HTTPS traffic, as long as the underlying encryption algorithm is not compromised.

Although all of the above approaches have their shortages, effective censorship can be achieved by using them holistically. We refer readers to [23, 46] for more comprehensive evaluations regarding country-level censorship techniques.

2.2 Content Delivery Network

Content delivery networks (CDNs) have emerged as a new business model in the recent decade and have undergone substantial growth [27]. Technically, CDN combines the characters of both the reverse proxy [37] and the shared cache [17]. As a reverse proxy, a CDN edge server is placed in front of the origin server and intercepts HTTP(S) requests and responses between the client and the origin server. As a shared cache, an edge server caches static web documents from multiple origin servers and uses these caches to serve duplicate HTTP(S) requests. Domain shadowing mainly exploits CDN's first feature, and we leave detailed explanations to later sections.

2.3 The Rise and Fall of Domain Fronting

A CDN is shared by multiple domains and relies on the `Host` header of an incoming request to determine the domain to forward the request. Domain fronting is a technique proposed by *D. Fifield, et.al.* in 2015 [18], which exploits a "quirky" implementation shared by many CDNs [3] .

As explained at the beginning of Section 1, many CDNs do not check the consistency of the SNI and the `Host` header of an incoming HTTPS request, and only rely on the `Host` header to forward the request. As a result, assuming the two domains, `allowed.com` and `blocked.com`, are both hosted on the same CDN, the user can circumvent censorship and access the blocked domain by sending an HTTPS request to the CDN edge server and requesting `allowed.com` (known as the *front domain*), but set the `Host` header to `blocked.com`.

Blocking domain fronting is difficult because of the "collateral damage" it brings. Specifically, in order to block a user from accessing a blocked domain hosted on a CDN, the censor must block all the domains on the CDN. Otherwise, a single allowed domain can serve as the front domain and makes all other domains on the same CDN accessible. Because CDN service is prevalent in today's Internet, and many valuable domains are also served by CDNs, completely blocking a (large) CDN is infeasible to many censors. Because of its robustness against censorship, domain fronting has been adopted by many censorship evasion systems, including Tor Meek [34], Psiphon [36], Lantern [24], and Signal [28].

In the recent two years, however, many CDNs (e.g., Google Cloud CDN and Amazon Cloudfront) became aware of domain fronting and began to disable it by enforcing the match between the SNI and the `Host` header [2, 3], which forced many censorship evasion systems to halt their service [38] or steer to smaller CDNs that are less costly to block [34].

3 Domain Shadowing

3.1 Threat Model

We depict the threat model in Figure 1, which involves the following roles and assumptions of their capabilities.

3.1.1 Roles

Censor: We assume an advanced censor who applies strict censorship policies, and deploys state-of-the-art technologies. We also assume the censor blocks domains based on a blocklist rather than a whitelist, i.e., a domain is accessible unless the censor explicitly blocks it.

User: The user refers to a regular human user and the web browser used for web browsing. As long as the context allows, we will interchangeably use *censored user* and *user* to refer to the user that locates within a censored area. We assume the user is not tech-savvy but has a reasonable knowledge of computer and Internet operations.

Publisher: The publisher is the owner of a domain. We assume the publisher is neutral and has no particular favor to either the censor or the user. Specifically, if the publisher's domain is blocked by the censor, it neither assists the censor in actively rejecting requests from the censored area nor takes any action to facilitate the user to circumvent the censor.

CDN: The CDN refers to a CDN provider that provides CDN service to all the public domains. We assume that the CDN is accessible by the censored user within the censored area. However, the CDN itself is not censored and can access blocked domains. One example could be the CDN deploys all its edge servers outside of the censored area.

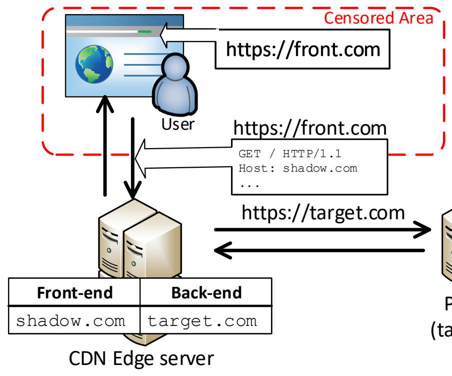


Figure 1: Threat model.

3.1.2 Terminologies

Front-end, *back-end*, and *domain binding*: CDN acts as a reverse proxy and is located between the client and the origin server, as shown in Figure 1. In this paper, the front-end of a CDN refers to its client-facing side, and the back-end refers to its server-facing side. We define the domain binding as the connection between these two domains.

Front/shadow/target domain: We define the front domain as the domain appears in the browser's address bar, which is used for the DNS query and as the SNI for the TLS handshake. The shadow domain is the domain present in the `Host` header of the HTTP(S) request, and also the domain set as the CDN's front-end. The target domain is the blocked domain that the censored user wants to access, and also the domain set as the CDN's back-end. Throughout this paper, we use `front.com`, `shadow.com`, and `target.com` as surrogates to represent these three domains, which do not refer to real world websites.

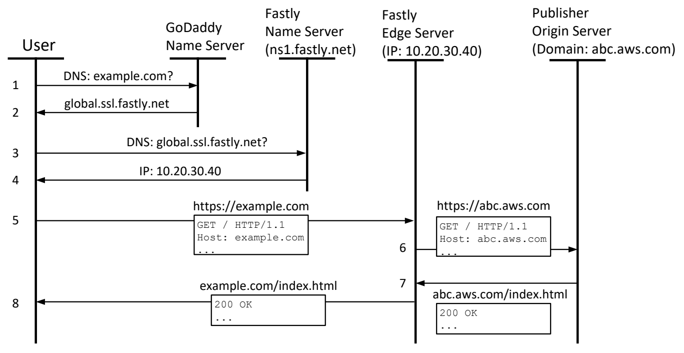
We assume the front domain is allowed by the censor, and the target domain is blocked. The property of the shadow domain varies in different circumstances, and we provide a detailed explanation for this in later sections.

3.1.3 Objective

The objective of the censored user is to access the target domain without being identified and blocked by the censor.

3.2 How does CDN Resolve Domain Names

The key idea of domain shadowing is to "repackage" and "rebrand" the response from a blocked domain into a new response that appears to belong to an allowed domain. In



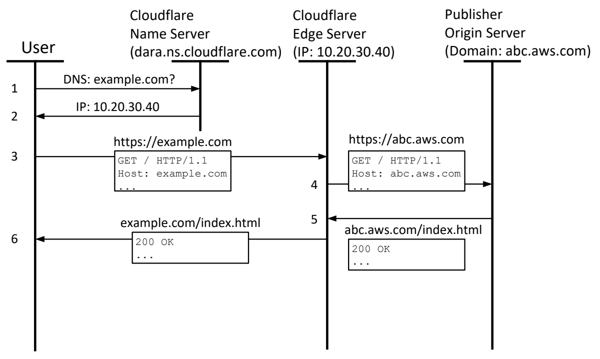
```
GET / HTTP/1.1
Host: example.com
...
```

```
GET / HTTP/1.1
Host: abc.aws.com
...
```

```
200 OK
...
```

```
200 OK
...
```

(a) DNS resolution by Fastly.


User
Cloudflare Name Server (dara.ns.cloudflare.com)
Cloudflare Edge Server (IP: 10.20.30.40)
Publisher Origin Server (Domain: abc.aws.com)
1
DNS: example.com?
2
IP: 10.20.30.40
3
https://example.com
https://abc.aws.com
4
5
example.com/index.html
abc.aws.com/index.html
6

```
GET / HTTP/1.1
Host: example.com
...
```

```
GET / HTTP/1.1
Host: abc.aws.com
...
```

```
200 OK
...
```

```
200 OK
...
```

(b) DNS resolution by Cloudflare.

Figure 2: Essential steps of DNS resolution by Flastly and Cloudflare, some steps are omitted for clarity reason.

order to comprehend the mechanism of domain shadowing, it is essential first to explain how domain names are resolved and translated when a CDN is involved.

Acting as a reverse proxy, CDN hides the back-end domain and presents only the front-end domain to the public. CDNs typically take two approaches to accomplish the name translation, which are presented in Figure 2. We make the following assumptions to facilitate the illustration: assume the publisher's origin server is hosted on Amazon Web Service (AWS) and assigned with a canonical name `abc.aws.com`, and the publisher wants to advertise the web service using the domain `example.com`, which is owned by the publisher and hosted on *GoDaddy's* [19] name server.

Figure 2a presents the name translation procedure adopted by most CDNs, and we use Fastly as a specific example in the following explanation. To use Fastly's service, the publisher will first log into his/her Fastly account, and set `example.com` as the front-end, and `abc.aws.com` as back-end. Then, the publisher will create a new CNAME record in GoDaddy's name server, which resolves the domain `example.com` to a fixed domain `global.ssl.fastly.net`. The name resolution of `example.com` follows the steps presented in Figure 2a.

Besides Fastly, many other CDNs also take the same approach with slight differences. For example, StackPath [39] will create a unique domain name, such as `j1s5u3d4.stackpathcdn.com`, for each front-end and back-end binding, instead of using `global.ssl.fastly.net` as a universal domain for all domain bindings.

Figure 2b demonstrates Cloudflare's [8] approach. Cloudflare itself hosts top-level domain (TLD) name servers that can directly resolve top-level domains. To use Cloudflare's CDN service, the publisher needs to switch his/her name server (from GoDaddy's) to Cloudflare's name servers. Consequently, the steps 2 & 3 in Figure 2a are skipped in Figure 2b, because Cloudflare's name server can directly resolve the top-level domain `example.com`.

We remind the reader to pay particular attention to the last four steps in the above two figures, which differ from a regular DNS name resolution. Using Figure 2a as an example, when the request to `https://example.com` arrives at a Fastly's edge server (step 5), the edge server will not redirect the user to `https://abc.aws.com`. Instead, it will directly fetch the document from the origin server (step 6 & 7) and use this document to respond to the request to `https://example.com` (step 8). During this process, the user only perceives that they are communicating with `example.com`, while the name translation took place only within the CDN and is completely hidden from the outside world.

In the following subsections, we present the operations of domain shadowing being used along, and together with domain fronting.

3.3 Domain Shadowing (*Ds*)

Domain shadowing takes advantage of the fact that when the domain binding is created, the CDN allows arbitrary domains to be set as the back-end. As a result, a user can freely bind a front-end domain to any back-end domain. To access a prohibited domain within a censored area, a censored user only needs to take the following steps.

1. The user registers a new domain, `shadow.com`, which is allowed because the censor applies a blocklist rather than a whitelist.

2. The user subscribes to a CDN service that is accessible, but the CDN itself is not censored.

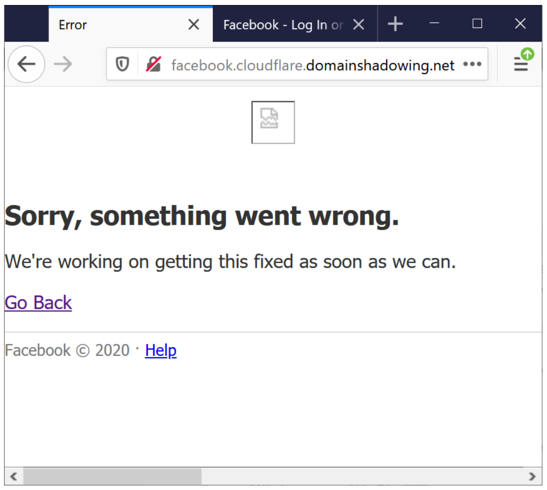
3. The user binds the shadow domain to the target domain in the CDN service by setting the shadow domain as the front-end and the target domain as the back-end.

4. The user creates a rule in his/her CDN account to rewrite the `Host` header of incoming requests from `Host:shadow.com` to `Host:target.com`. This is an essential step since otherwise, the origin server of `target.com` will receive an unrecognized `Host` header and reject the request.

5. Finally, to access the target domain, the user sends a request to `https://shadow.com` within the censored area.

The request will be sent to the CDN, which will rewrite the `Host` header and forwards the request to `target.com`. After the response is received from `target.com`, the CDN will return the response to the user under the name of `https://shadow.com`.

During this process, the censor will only see the user connect to the CDN using HTTPS and request resources from `shadow.com`, and thus will not block the traffic.



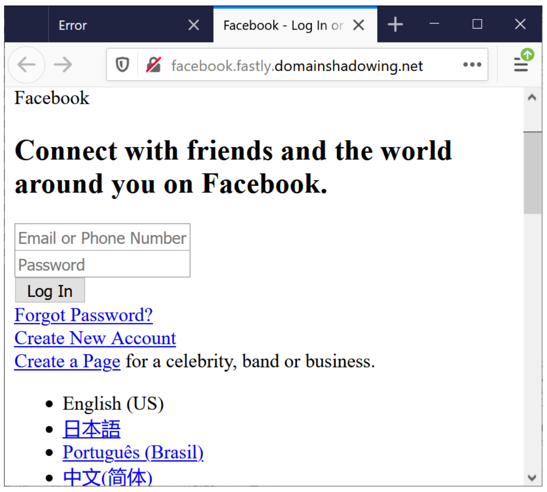
Sorry, something went wrong.

We're working on getting this fixed as soon as we can.

Go Back

Facebook © 2020 · Help

(a) *Ds* with Cloudflare.



Facebook

Connect with friends and the world around you on Facebook.

Email or Phone Number
Password
Log In
Forgot Password?
Create New Account
Create a Page for a celebrity, band or business.

- English (US)
- 日本語
- Português (Brasil)
- 中文(简体)

(b) *Ds* with Fastlyz.

Figure 3: Domain shadowing using Cloudflare and Fastly.

To prove the validity of the idea, we registered the domain `domainshadowing.net`, and created accounts on both Cloudflare and Fastly. We set `www.facebook.com` as the back-end to both CDNs, and `facebook.cloudflare.domainshadowing.net` and `facebook.fastly.domainshadowing.net` as the front-end for Cloudflare and Fastly, respectively. With the free-tier account, `Host` header rewriting is allowed by Fastly but not but Cloudflare.

Then, we visited the two shadow domains by firstly connecting to a rented HTTP proxy located in a heavily censored country known by its strict censorship policies and advanced technologies, who explicitly blocks access to `www.facebook.com`. The results of visiting the two domains are presented in Figure 3a and Figure 3b.

Figure 3a shows an error page, which is because we were unable to rewrite the `Host` header in Cloudflare. Therefore, the `Host` header stayed as `Host: facebook.cloudflare.domainshadowing.net` when the request arrived at Facebook's origin server, who did not recognize this header value and returned an error page. Nevertheless, the "Facebook © 2020" copyright mark at the bottom of the page suggests we have successfully circumvented censorship and accessed Facebook's server.

Figure 3b shows a more promising result, where the Facebook login and sign-up section (only part of the page is displayed due to space limitation) were all successfully loaded into the browser. The reason for the shabby layout is because many CSS style sheets were hosted on another domain `static.xx.fbcdn.net`, which is also blocked and can not be directly accessed by the browser.

3.4 Domain Fronting and Shadowing (*DfDs*)

Remind that domain fronting achieves censorship circumvention by connecting to an allowed domain while setting the `Host` header to be a prohibited one on the same CDN. By doing this, domain fronting prevents the censor from knowing the real front-end the user is requesting. Domain fronting's limitation lies in that it can only access domains that use the same CDN on which the front domain is hosted.

On the contrary, domain shadowing achieves censorship evasion by creating a domain binding on the CDN, and using this binding to access a blocked domain that can be hosted on any CDN, or even not using CDN at all. However, domain shadowing must be operated "under the radar" of the censor, because otherwise, it can be easily blocked by blocking access to the shadow domain.

Interestingly, we find domain fronting and domain shadowing each tackles one-half of the CDN operation, and thus can be integrated to achieve a more robust blocking-resistance. The scheme of corroborating domain fronting and shadowing, namely the *DfDs*, is depicted in Figure 4.

To use *DfDs*, the user must choose a CDN that supports domain fronting; however, it needs not to be the CDN that hosts the target domain. Then, the user registers to this CDN and creates a binding between the shadow domain and the target domain. Lastly, the user selects a domain on this CDN allowed by the censor and uses it as the front domain. Essentially, in the *DfDs* setting, the front domain, the shadow domain, and the target domain are three distinctive domains.

In order to access the target domain, the user will initiate an

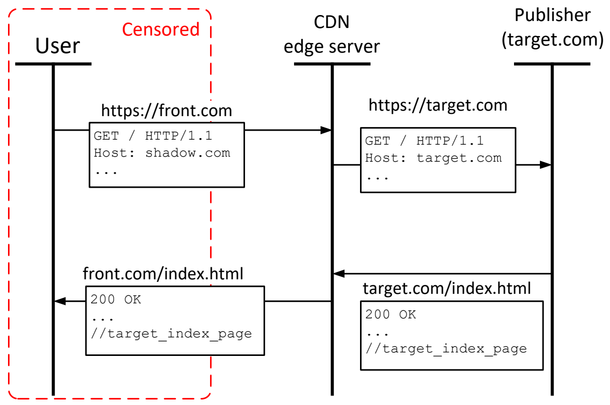


Figure 4: Domain fronting + Domain shadowing (*DfDs*).

HTTPS request to the front domain but set the `Host` header to the shadow domain, such that the request can penetrate censorship and reach the CDN. When the request arrives at the CDN, the CDN forwards the request to the shadow domain according to the `Host` header, follows the domain binding, and sends the request to the target domain. On the reverse path, the document returned from the target domain will be repackaged into a response that appears to be replying to the front domain, and stealthily returned to the user.

3.5 Enhanced *DfDs* (*DfDs*++)

We argue that *DfDs* already has strong resistance against most censorship techniques. However, during our experiments, we found a feature that can make *DfDs* even more stealthy. We nickname it as *DfDs*++, i.e., the evolved version of *DfDs*.

In particular, we found that many CDNs do not check the ownership of the front-end domain either. For instance, in Fastly, it is possible to claim existing domains as the front-end, as long as they have not been claimed by other users. For instance, we have successfully set `cmu.edu` (the domain of Carnegie Mellon University) as the front-end in our account. The attempts to set other domains, such as `apple.com` or `microsoft.com`, were rejected with a notice that these domains were "already taken by another customer". This may imply these domains have been claimed by other users, or Fastly has blocklisted these domains from being used. More interestingly, we found that even *non-existent* domains are acceptable. For instance, we have successfully set `5f4dcc3b5aa765d61d8327deb882cf99.com`, the MD5 value of the word "password", as the front-end in Fastly, and connected it to `www.facebook.com`.

This feature implies that the user does not even need to register a shadow domain to use *DfDs*++. The user can register a CDN account and claim a random domain that has not been claimed as the shadow domain and bind it with the target domain. By doing this, the shadow domain only resides within the scope of the CDN, and will not be known to the public because it never existed.

4 Implementation

In the previous section, we have demonstrated how to use domain shadowing to send a single request to a blocked domain within a censored area. However, a typical webpage nowadays contains a number of subresources that are heavily interdependent, which must be properly disentangled such that the webpage can be properly displayed. In this section, we present our experimental implementation of domain shadowing as a Firefox extension based on Fastly's service.

We begin this section by listing the major technical challenges encountered during the development, and then move on to present the implemented system and show its capability.

4.1 Technical Challenges

4.1.1 Subresources from Multiple Domains

Nowadays, a webpage contains different types of subresources, such as CSS style sheets, JavaScripts, and images, which may be hosted on domains different from the main document. Therefore, to properly display a web page, the browser needs to access multiple domains that may also be blocked. As a result, the user must create multiple domain bindings such that all the resources can be successfully fetched. Depending on the specific target domain, the workload of this task may vary significantly. For instance, as we have tested a few domains, a user's Facebook front page only contains resources from less than ten domains, while the main page of `cnn.com` contains resources from more than 70 domains. However, the user does not need to register separate shadow domains for each target domain. Instead, the user can use subdomains of a single shadow domain, e.g., using `www.facebook.com.shadow.com` as the front-end of `www.facebook.com`.

The task of creating multiple domain bindings, fortunately, can be fully automated using APIs or SDKs provided by the CDN provider, which can be integrated into the browser extension and reliefs the user from manual configurations.

4.1.2 CORS and CSP

A bigger hurdle that needs to be overcome is the sharing of resources among different domains, i.e., cross-origin resource sharing (CORS). Based on the *same origin policy (SOP)* [5] enforced on modern browsers, a domain can use the `Access-Control-Allow-Origin (ACAO)` response header to inform the browser of the domains that can access its resources. For instance, `static.xx.fbcdn.net` can allow `www.facebook.com` to access its resources by including the response header `Access-Control-Allow-Origin: www.facebook.com`.

However, since domain shadowing transforms these two domains into `static.xx.fbcdn.net.shadow.com` and `www.facebook.com.shadow.com`, the original `ACAO` header value must be changed to `Access-Control-Allow-Origin: www.facebook.com.shadow.com` instead. This task can be handled by the browser extension, which can modify such header values before the browser sees them.

Similar to the CORS issue, the `Content-Security-Policy` is another response header that specifies certain domains that can conduct cross-domain actions that bears security risks, such as allowing/disallowing the webpage being framed. In order to enforce the same security policy when domain shadowing is used, these domains must also be transformed into the new domain with `.shadow.com` as the suffix, which can be done by the browser extension similarly.

4.1.3 Cookie Management

Usually, cookies [4] are automatically managed by the browser based on the domain of the visited URL. In the case of domain shadowing, however, the browser is unable to manage cookies correctly because of the domain name transformation. Instead, the cookie management must be handled by the extension to set and read cross-domain cookies.

Specifically, an origin server sets cookies to a browser by appending the `Set-Cookie` response header. The `Set-Cookie` header contains an optional `Domain` field, if such field is omitted, the cookie will be set to the host of the current root domain; otherwise, the cookie will be set according to the specified domain by the `Domain` field [10].

Assuming a `Set-Cookie` header is received from `a.com.shadow.com`. If this header includes `Domain=a.com`, the extension can directly set the cookie according to the specified domain `a.com`; on the other hand, if the `Domain` is unspecified, the extension must retrieve the host from the current domain, i.e., `a.com.shadow.com`, remove the suffix `.shadow.com`, and then set the cookie based on the original domain. Similarly, when a request is sent, the extension will intercept the request to `a.com.shadow.com`, and append the correct cookie that belongs to `a.com`.

4.1.4 Limitations

In this section, we have listed a few major technical challenges that we have encountered during our experiments. Essentially, the root cause of all these issues is that the browser sees the front domain or the shadow domain in the address bar, while the web document is actually fetched from the target domain. As demonstrated above, our main approach to address these challenges is to let the browser extension make proper modifications before the document is processed and rendered by the browser. These manipulations, however, may give rise to a series of security risks, which will be discussed in Section 7.

We denote that the challenges listed here are only representative major issues that are common across all webpage loading, which are by no means comprehensive since the relationship among a webpage's subresources heavily depends on the specific implementation. We will open-source our implementation so non-typical issues can be identified and solved by the community incrementally.

4.2 Domain Shadowing Automation

We implemented *DfDs* as a Firefox extension based on Fastly's CDN service. The extension automates procedures, including setting the front and back-end and creating the `Host` header rewriting rule. This extension hides the complex tasks from the user: all that is required from the user is for them to register for a Fastly account, obtain the API authentication key and enter it into the extension.

4.2.1 Fastly's Web APIs

Fastly provides comprehensive APIs [14] that can be used by its customer to create new and configure existing services. To use the web API, the user will send HTTP(S) requests to Fastly's entry point `https://api.fastly.com`, where the `POST`, `PUT` and `GET` methods are used to create, modify, and retrieve information of a specific configuration. Each request must contain a `Fastly-Key` request header, which is a 32-character token to identify and authenticate the user.

4.2.2 Automation of Domain Shadowing

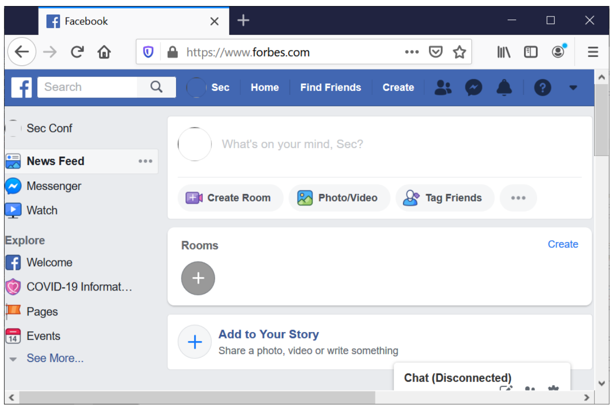


Figure 5: Accessing Facebook using *DfDs*.

After enabled, the extension will intercept every request issued by the browser, inspect the domain of the URL, and create a new binding using the API if a domain is being visited for the first time. The extension also keeps local storage of the domains that have been configured such that repeating requests will not trigger new bindings being created. Then, the extension will modify the `Host` header (to `shadow.com`), and redirect the request to either the front domain (if using *DfDs*) or the shadow domain (if using *Ds* only).

As a case study, we selected `www.forbes.com`, a reputable business news media, as the front domain, which uses Fastly's service and is accessible from the experimented country. We use our registered domain `domainshadowing.net` as the shadow domain and set `www.facebook.com` as the target domain. Figure 5 demonstrates the result of visiting the target domain within the censored country. The figure shows that the web page and all sub-resources have been appropriately loaded by the browser. The cookies were also correctly handled as the website has been successfully logged in. Note that the address bar of the browser displays the front domain, i.e., `www.forbes.com`, instead of the target domain.

In the following three sections, we thoroughly discuss domain shadowing's advantages and limitations. Our discussion will focus on three aspects: *usability*, the convenient level for an ordinary user to use domain shadowing; *blocking-resistance*, domain shadowing's strength in resisting existing and future censorship techniques; and *security impacts*, the security impact brought to the user, the CDN provider, and the publisher, and possible approaches to minimize such impacts.

5 Usability

5.1 Possible Choice of CDN Providers

We choose six CDNs and discuss their possibility to be used for domain shadowing. These CDNs include: Google Cloud CDN, AWS Cloudfront, Microsoft Azure CDN, Fastly, Cloudflare, and StackPath (formally know as MaxCDN). We chose these six CDNs because the first three are the most representative cloud service providers where CDN is provided as one among many other services on their cloud platform, while the last three are well-known CDN providers that take a considerable share in North America's CDN market [32]. This shortlist is arguably insufficient and may be biased; however, we denote that our main focus is to propose the idea of domain shadowing and demonstrate its viability. Furthermore, because of their significant market share, we believe they are reasonably representative of all other CDN services.

5.2 Technical Barrier

By and large, successfully deploying domain shadowing requires the user to be able to install the browser extension and properly configure the CDN. While extension installation is intuitive, the complexity of CDN configuration varies. In the following of this section, we discuss the technical barriers exposed to the user by using the six CDNs for domain shadowing.

Essentially, the operation of domain shadowing relies on three tasks: setting the front-end domain, setting the back-end domain, and rewriting the `Host` header. In Table 1, we survey and present if a CDN supports API or SDK configuration of these three tasks. If a task can be finished using API or SDK rather than manually, it can be accomplished by the browser extension and thus relieved from the user.

	Set front-end	Set back-end	`Host` rewriting
Cloud CDN	✓(fixed IP)	✓	✓
Cloudfront	✓(fixed subdomain)	✓	✓(default)
Azure CDN	✓(fixed subdomain)	✓	✓(default)
Cloudflare	✓	✓	✗(limited)
Fastly	✓	✓	✓
StackPath	✓	✓	✓

Table 1: CDN support of configuration automation.

As shown in the table, all six CDNs provide API for such configuration, while Google Cloud CDN and AWS Cloudfront also provide SDK to further ease the task.

Setting front-end. We found that all three dedicated CDN providers allow the user to directly set the front-end, but the three cloud service providers act differently. Specifically, Cloudfront and Azure CDN will assign a front-end to a user-configured back-end, which is a subdomain of the provider's root domain and cannot be freely modified by the user, e.g., Azure CDN assigns `facebook.azureedge.net` where the user can only rename the first section, and Cloudfront assigns `a1jfp0jyfnb0xd.cloudfront.net` where user cannot change any section of the domain. On the other hand, Google Cloud CDN directly generates an IP address to the user-configured back-end. In such cases, the user does not need to register a shadow domain, and can directly use the CDN-assigned domain as the shadow domain instead.

Setting back-end. All six CDNs allow the user to freely set any domain as the back-end domain.

Rewriting `Host` header. Cloudflare limits the `Host` rewriting function to enterprise-tier users only, making it an infeasible option for domain shadowing. All other five CDNs allow `Host` rewriting. Furthermore, when a back-end domain is configured by the user, Cloudfront and Azure will directly set the `Host` header to be the back-end domain by default.

Bootstrapping effort. As discussed above, except for Cloudflare, domain shadowing can be automated on all other five CDNs. However, before API/SDK can be used for such automation, the user must first log into the CDN account, allow API/SKD operations, and create user credentials, which can impose certain technical difficulties to an ordinary user. Such tasks, on the other hand, are standardized and thus can be well-documented for the user's reference.

5.3 Accessibility and Cost

In Table 2, we list a few examples to compare the cost of using CDNs and running an HTTP proxy hosted using Virtual Private Services (VPS), which is another popular censorship evasion solution. We divide Table 2 into two parts. In the

Provider	CDN Cost	VM-proxy Cost (US Easts region)	Payment
Google	- Ranging $0.08 - $0.20 per GB for the first 10 TB, depending on the origin server's location. - $300 free credit newly registered account.	- VM hosting: $0.075462 per hour, 2 vCPU nd 8 GB memory. - Network traffic: $0.12 per GB for 0 - 1 TB (excluding China and Australia)	Card, PayPal.
AWS	- Ranging $0.085 - $0.17 per GB for the first 10TB, depending on the origin server's location. - 50GB free per month for the first year.	- VM hosting: $0.0047 per hour, 2 vCPU and 0.5 GB memory. - Network traffic: First 1 GB free per month, $0.09 per GB for 1 GB - 10 TB.	Card.
Microsoft	- Ranging $0.081 - $0.233 per GB for first the 10 TB, depending on the origin server's location.	- VM hosting: $0.0021 per hour, 1 vCPU and 0.5 GB memory. - Network traffic: First 5 GB free per month, $0.085 per GB for 5GB - 1 TB.	Card.
Fastly	- Free developer account without explicit data usage limit. - Ranging $0.12 - $0.28 per GB for first 10TB, $50 minimal per month. $50 credit newly registered account.	Not available.	Card.
Stackpath	- $10 per month for up to 1TB.	Not available.	Card, PayPal.
BelugaCDN	- $5 for first 200GB, $0.0008 per GB overage.	Not available.	Card, Google Pay.
KeyCDN	- $0.04 - $0.11 per GB for up to 10 TB.	Not available.	Card, PayPal.
CDNSun	- $0.04 - $0.159 per GB.	Not available.	Card, PayPal, Wire Transfer.
Accu Web Hosting	Not available.	- VM hosting: $5 per month, 1 vCPU and 1 GM memory. - Network traffic: first 150 GB free, $2 per 50 GB overage.	Card, PayPal.
DreamHost	Not available.	- VM hosting: $15 per month, 1 GB memory, vCPU unspecified. - Network traffic: unlimited.	Card.
Hostinger	Not available.	- VM hosting: $9.95 per month, 1 vCPU, 1 GM memory. - Network traffic: 1 TB.	Card, PayPal, Google Pay, Digital currency.

Table 2: Cost comparison of using CDN vs. using virtual hosting.

top part, i.e., the top 5 rows, we compare the more reputable service providers.

As shown in the top part of Table 2, besides StackPath that charges a fixed monthly fee, the other four CDNs charge based on data usage, and the price of which varies depending on the location of the origin server. Generally, servers located in North America and Europe have the lowest charge, and Asia has the highest, and the cost of domain shadowing will be close to the lower boundary since many of the censored websites, such as Facebook and Twitter, are based in North America.

On the other hand, the cost of the VPS approach comprises two parts: the cost to rent the service, and the cost to send and receive data. The VPS renting charge varies depending on the hardware configuration and we only demonstrate the price of the lowest configuration in Table 2.

We are also aware that other than these reputable CDN and VPS service providers, there are also many smaller scale and less expensive options. In the bottom part of Table 2, i.e., the lower 6 rows, we compare the cost of using such smaller scale service providers. Note that we were unable to conduct a comprehensive comparison because there are many such service providers on the Internet. Instead, we searched the term "affordable VPS" and "affordable CDN" on Google, and randomly selected three of each type from the first returned page. We also denote that, for the three CDN providers, we did not actually subscribe to their service and verify they can be used for domain shadowing, mainly because they do not offer free accounts and require up-front payment to use their service (e.g., KeyCDN requires the user pay a minimum amount of $50 to start using their service). However, we did check their API documentation to ensure that all the actions necessary for domain shadowing are supported through APIs.

In Figure 6, we visualized Table 2 and plotted the cost of using these CDNs and VPSs. We set the data usage in the figure to be 500 GB per month, which we believe is more than enough for regular Internet users. From this figure, we can observe that the cost of using domain shadowing is generally comparable to that of using VPS.

Another barrier the user may encounter is the potential payment issue: the user in the censored country may not have a valid western credit card to pay for a CDN service. To this end, we also investigated the payment method accepted by all the CDN and VPS service providers and listed them in the last column in Table 2. As shown by the table, all three smaller scale CDNs accept either PayPal or Google Pay, giving the user the option to pay the service with foreign currencies other than U.S. dollars. Google, Amazon, and Microsoft, on the other hand, provide services in many countries across the world and accept different currencies by themselves. Furthermore, it is also noteworthy that because domain shadowing does not require the target domain and the shadow domain to be on the same CDN, the user can freely choose *any* CDN, such as local or regional ones that the user can easily subscribe to, as long as it is accessible by the user and is not being censored.

5.4 Performance

Compared with directly visiting a website, domain shadowing added two more steps into the procedure, which are: when a domain is being visited for the first time, the user must create a new domain binding at the CDN; after such binding has been created, the browser will fetch a document from a CDN's edge server instead of from the origin server.

The time spent to create the domain binding varies among CDN providers. During our experiment, we found that ded-

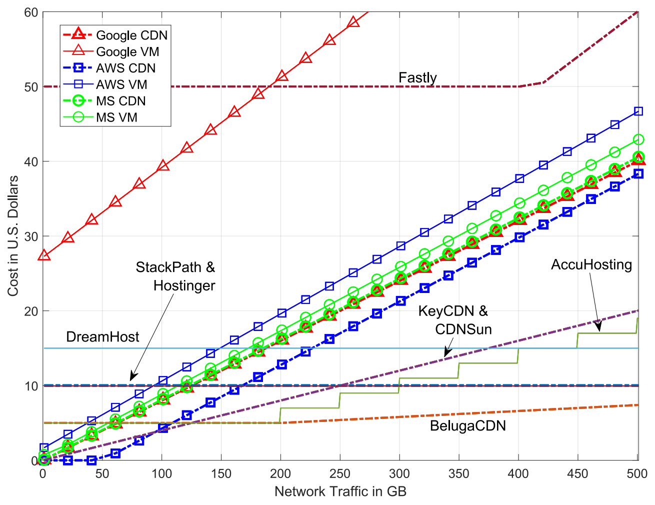


Figure 6: Monthly cost comparison of using CDN and Virtual hosting for web browsing.

icated CDN providers, such as Fastly and StackPath, take a shorter time (less than 10 seconds) to deploy newly created binding, while Cloud service providers take longer (over 10 seconds but merely exceed 20 seconds). This may be the result of the complicated dependencies among different cloud services on the cloud platform. For instance, the user must also configure a load balancer on Google Cloud to cooperate with the Cloud CDN service. This being considered, domain shadowing is better suited for users with relatively steady browsing habits, i.e., the user frequently visits the same websites instead of always browsing new ones.

On the other hand, the time spent to fetch web documents from the CDN could be even shorter than the time spent fetching documents directly from the origin server, because in the former case, the user will connect to a close-by edge server, and then obtain the document via CDN's high-speed infrastructure. In order to validate our hypothesis, we compare domain shadowing's performance with other censorship circumvention tools, as well as directly fetching from the origin server.

Our experiment includes two common and representative censorship circumvention tools/techniques: *Psiphone* [36], which is a popular free VPN service; and a *TinyProxy* [40], a lightweight HTTP proxy application, which runs on an Ubuntu instance hosted on AWS EC2. We configured Psiphone to connect to the "best performance" endpoint, and created two instances (t3a.nano with 2 vCPU and 0.5 GB memory, and t3a.2xlarge with 8 vCPU and 32 GB memory) on the AWS's datacenter region that is closest to us, to test the impact of different hardware. We did not include Tor because censorship circumvention is not its main purpose.

In the experiment, we evaluated the delay of fetching the document `tools.ietf.org/html/rfc2616` directly, using domain shadowing based on the five CDNs, and the above-

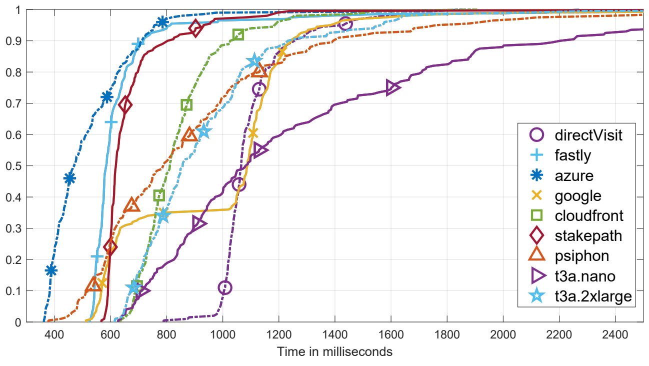


Figure 7: CDF of delays to fetch a web document using different methods.

mentioned censorship circumvention tools. We chose the *RFC 2616* document without any particular reason but only because it is a document that we access quite frequently, using which can also achieve fairness since it is not served by any of the 5 CDN providers. The domain `tools.ietf.org` resolves to three IP addresses, and during our experiment, we have made sure all methods connect to the same IP `4.31.198.61` for fairness. It is noteworthy that this IP address belongs to *Level 3*, an Internet Service Provider (ISP) that also provides CDN services. Because we were unable to tell if the domain only uses it as the ISP or also uses its CDN service, we are not certain if all the methods are actually fetching the document from the same physical server (could be from different edge servers if Level 3 CDN are used). However, we argue this is a typical and representative situation. Specifically, because of the prevalent usage of web caches nowadays, it is very likely that a target domain that the user wants to access has already been served by some type of web caches (not necessarily a CDN).

The experiment was conducted in our research lab located in an uncensored country. We did not conduct the experiment from a censored country because, first, we did not have access to any vantage point in a censored country; second and more importantly, even if we had such vantage points, the result would not be more representative, because the delay of a web request highly depends on the relative location of the user and the web server, the infrastructure of the user and the webserver's ISPs, and the relationship between them. Therefore, this experiment only demonstrates a "snapshot" of domain shadowing's performance relative to other methods.

Using each method, we fetched the document 200 times and recorded the delay of each request. We disabled the caching function for all the CDNs so each request would reach the origin server. We also throttled the request rate to avoid overwhelming the origin server. In Figure 7, we demonstrate the CDF plot of the delays of each method. As shown, all the five CDNs beat directly fetching the document, where Azure, Fastly, and StackPath cost only less than half of the delay. The behavior of Google Cloud CDN is a bit strange, and we reckon this may be caused by our request being sent to two different edge servers due to load balancing, which takes an obviously different time to obtain the document. We can also find the Psiphon case converges slowly, indicating the delay distribution is less concentrated. Lastly, user-configured HTTP(S) proxy still underperforms compared to domain shadowing, even runs on powerful hardware (the t3a.2xlarge instance).

5.5 Reliability and Trustworthiness

In this paper, we view the CDN as a trusted infrastructure rather than a third-party that actively participates in the operation, who will not intercept, inspect, and react to, users' traffic. However, it is noteworthy to point out that the CDN is at the vantage point of intercepting all the users' traffic, including HTTPS connections. Therefore, a malicious CDN is able to inspect and tamper with the traffic between the user and the target domain without being identified. In the case of an untrusted CDN, the user can use domain shadowing as a tunnel to evade censorship and add extra-layers to achieve privacy and security. For instance, the user can set the back-end to be a Tor bridge and encrypt all the traffic exchanged between the user and the bridge, making the CDN unable to see any plain text, which is similar to the current implementation of Tor Meek [34].

5.6 Summary

In this section, we have discussed the usability of domain shadowing from various perspectives. To summarize, domain shadowing may *not* be a censorship evasion solution for all censored users. The use of domain shadowing requires the user to have reasonable knowledge of domain and CDN account registering and the capability to afford a certain amount of charge, possibly in foreign currencies. On the other hand, for those users who fulfill these requirements, they can enjoy all the benefits such as being able to visit any websites with better delay performance than many other solutions.

6 Blocking-Resistance

This section discusses domain shadowing's blocking-resistance from two aspects: how can it resist existing censorship techniques, and how can it stay ahead of the potential arm-race once it is publicly known.

6.1 Against Existing Censorship Techniques

6.1.1 DNS Interference

When being used alone, domain shadowing is susceptible to DNS interference once the censor knows the shadow domain. Therefore, domain shadowing is more suitable to be used privately by the user, such that the shadow domain stays "under the radar" of the censor.

6.1.2 Active HTTPS Probing

In the case of an aggressive censor, however, privately using the shadow domain may still be insufficient. For instance, a censor may "traceback" to unknown HTTPS traffic, i.e., if the censor sees HTTPS traffic that connects to an unknown domain, it may send requests to the domain by itself and inspect the response. To resist such aggressive censors, the user can add a simple credential to authenticate himself/herself. For instance, the user can specify a rule at the CDN that only if a specific custom header presents in the request, the CDN will fetch the document from the target domain; otherwise, the CDN will connect to an allowed domain instead, or simply does not respond at all. The user can communicate with the CDN by appending the header, while the censor is unable to get the same response without such knowledge.

6.1.3 IP Blocking

Domain shadowing's resilience against IP blocking is even more potent than that of domain fronting. Specifically, in the case of domain fronting, if the blocking of a particular domain is of paramount significance to the censor, the censor can still choose to completely block access to the specific CDN that hosts that domain. In the case of domain shadowing, however, in order to block access to a domain, the censor must block access to *all CDNs* that allow domain shadowing, resulting in much more significant collateral damage compared with domain fronting.

6.1.4 Deep Packet Inspection

Deep packet inspection relies on the censor being able to inspect the content of the packets transmitted between the user and the edge server, and it is safe to assume domain shadowing is not susceptible to DPI unless the censor is strong enough to break HTTPS. Note that in this paper, we assume direct connections between the user and the CDN, and do not consider the cases where the censor applies middleboxes to intercept HTTPS connections [11].

6.2 Potential Moves by the CDN

It is likely that once being publicly known, domain shadowing will face the same fate as domain fronting, i.e., the CDN provider may receive pressure from the censor to disable it. We hereby discuss possible moves taken by the CDN to disable domain shadowing.

Essentially, domain shadowing's success relies on the following three indispensable steps: setting the front-end, setting the back-end, and rewriting the `Host` header. Among the three, the CDN is unlikely to put any limitation on the front-end besides disabling *DfDs++*, since the front-end is legitimately owned by the user. In the following subsections, we will focus on discussing possible moves a CDN may take on the back-end and the `Host` header.

6.2.1 Limiting the Back-end Domain

Technically, the CDN can limit the user to set an arbitrary domain as the back-end to disable domain shadowing. Such a move, however, comes with considerable damage to the CDN itself for the following reasons.

First of all, CDN provides name translation services similar to the DNS. In a DNS CNAME record, the owner of the domain `example.com` can freely set this domain to be the alias of any other domain. For instance, the owner can create CNAME records such as:

`CNAME  example.com  www.facebook.com`,

such that a request sent to `example.com` will be resolved to `www.facebook.com`. The name server has neither interest nor capability to verify the relationship between the two domains. To this end, the CDN should also follow the same logic and allow arbitrary back-end domains.

In fact, the open back-end is an indispensable feature that the CDN (and a DNS name server) *must* allow. In today's world wide web (WWW), it is very common for a website to outsource part of its service to a third-party. One typical example is customer service. For instance, the website `example.com` wants to outsource its customer service, such as creating and handling service tickets, to the third-party service provider *Zendesk* [50]. For this to work, Zendesk will create a custom domain, `example.zendesk.com`, as the entry point to handle `example.com`'s customer service request. In the meanwhile, `example.com` will create a subdomain `customer-service.example.com` and point it to `example.zendesk.com` using a CNAME record [50]. As a result, a customer will enjoy the customer service provided by Zendesk while perceiving that they stay on the `example.com` domain. Therefore, a CDN should allow a user to set a back-end domain that they do not own.

An alternative option for the CDN is to verify the "legitimacy" of the back-end usage, which, however, is laborious at best, because such verification may require both `example.com` and `zendesk.com` to submit certain types of proof, which may lower the CDN's customers' satisfaction and negatively impact its business.

Another solution could be for the CDN to disallow "popular" websites, such as `www.facebook.com`, being set as the back-end. This, however, is also problematic. For the first, the definition of "popularity" is vague. While high-profile websites such as Facebook and Twitter are undoubtedly popular, these may not be the websites that the censored user wants to visit. For the second, such limitation is not impossible to bypass. For example, if the CDN only use keyword-filtering of the back-end, the user may create a public CNAME:

`CNAME  fb.example.com  www.facebook.com`,

and set `fb.example.com` as the back-end instead. The CDN is unable to find it unless it stretches further to inspect the DNS resolution result.

6.2.2 Limiting the `Host` Header Rewriting

Like the previous case, the rewriting of the `Host` header is also an indispensable feature to a CDN that can not be simply disabled. Some cloud services, such as AWS S3 bucket, expect a specific format of the `Host` header of the incoming request, i.e., `<bucket-id>.s3.amazonzws.com`. Therefore, if a user uses CDN to connect the domain `example.com` to an S3 bucket [16], the `Host` header must be rewritten to the proper format such that the request can be served.

A possible move for the CDN could be to limit the `Host` rewriting function available only to a certain level of users, as adopted by Cloudflare (only available to enterprise users). However, such a move will affect all the CDN's existing customers and also negatively affect its business.

6.2.3 Anomaly Detection

Another move of the CDN could be monitoring all its users' account activity to detect "abnormal" actions, such as more than usual API requests, or an extraordinary number of domain bindings. Such "anomaly detection", however, is also possible to circumvent. An intuitive countermeasure could be splitting such activities among multiple CDN accounts to bring these metrics back to "normal".

To summarize, we denote that domain shadowing is fundamentally different from domain fronting. The operation of domain fronting is based on an untightened implementation: the CDN does not check the consistency between the SNI and the `Host` header. Therefore, a CDN can easily disable domain fronting by enforcing the match, and such action does not bring much damage to the CDN itself. On the other hand, domain shadowing utilized a legitimate feature of the CDN, and any change of this feature will result in considerable negative impacts to the CDN's business. Thus, a CDN must weigh these factors when facing pressure from a censor.

6.3 Potential Moves by the Censor

In general, the censor can detect an evasion based on three traces: what the user talks about (content), whom the user talks to (destination), and how the user talks (behavior). Since access to the CDN is allowed and HTTPS is used, the censor is unable to infer information using the first two approaches. The behavior-based approach, however, is not as unreliable. We discuss possible techniques in the following subsections.

6.3.1 Website Fingerprinting

Website Fingerprinting [6, 7, 12] is a new censorship technique, which relies on inspecting the traffic pattern, such as the number and size of packets sent/received by a user, rather than the content, to identify suspicious activities. To generate the fingerprint of a webpage, the censor first needs to identify the packets that belong to a single webpage browsing activity. This task can usually be done with common traffic analysis techniques, such as grouping packets which belong to one TCP session/port or a single IP address.

To evade website fingerprinting, the user can split requests and send them to multiple shadow domains. Using *Ds*, the user will need to use multiple shadow domains, for instance, `shadow1.com` and `shadow2.com`, and let both binds to the same target domain. To be more stealthy, these shadow domains can be configured on different CDNs. Using *DfDs* or *DfDs*++, the user only needs to find multiple front domains to split the requests while setting the `Host` header of all requests to the same shadow domain. In both cases, because these requests are destined to different domains and even different CDNs, it will be difficult, if not impossible, for the censor to group packets that belong to one webpage browsing and generate valid fingerprints.

6.3.2 Anomaly Detection

The censor may also monitor a user's activity to detect anomalies. For example, the censor may find the user communicates with a CDN's API entry point with higher-than-usual frequency, or the user is always visiting a single (front or shadow) domain. Such anomaly detection, however, is fragile and mostly possible to bypass. For instance, the user may choose to communicate with the API entry point less frequently, willingly sacrificing usability to trade for stealthiness. The user can also use domain fronting or domain shadowing to hide the communication to the API entry point. The user can also register multiple shadow domains, choose multiple front domains, and even use multiple CDNs to "normalize" the "abnormal" behavior.

6.3.3 Completely Blocking a CDN

Although unlikely, it is still possible for the censor to completely block access to a CDN if the blockage of a domain on this CDN outweighs the benefit of allowing all other domains on the same CDN, especially if the CDN is smaller and bears less collateral damage. However, we argue that blocking any particular CDN, regardless of being large or small, will not disable domain shadowing, since the user can easily switch to other CDNs that still allows domain shadowing. Essentially, domain shadowing cannot be blocked unless the censor is willing to block access to *all* the CDNs that allow it.

6.4 Ethical Considerations

Seemingly, domain shadowing is a "hack" on CDNs that may be detrimental to the CDN or the publisher. In fact, the damage, if any, is more operational than technical.

Technically, domain shadowing uses a legitimate feature essential for CDN operation, which does not incur material damage to the CDN other than bringing unintended traffic. In fact, we found this fact was disclosed to major CDN providers back in 2018 by [21]. For the publisher, using CDN to fetch a web document from its origin server has no difference from doing the same via a VPN endpoint or an HTTP proxy.

Operationally, however, a CDN that is knowingly supporting domain shadowing may face pressure from the censor, which may negatively impact its business, similar to what has happened to domain fronting. Moreover, since CDN intercepts the HTTPS traffic between the user and the publisher, a malicious CDN may passively spy or actively tamper such traffic and cause damage to both the user and the publisher. Therefore, we deem the user should be informed of this risk prior to using domain shadowing to transmit sensitive data.

7 Security Impacts

This section discusses domain shadowing's security impacts to the CDN, the publisher, and the user.

7.1 The Open Front-end and Back-end

At first glance, domain shadowing uses the open back-end to "impersonate" the target domain and modify its content on the fly, which may negatively impact the target domain, such as phishing or defamation. However, we emphasize the operation of domain shadowing requires the cooperation between the browser extension and the CDN. For example, if a user directly visits `facebook.com.shadow.com` without the extension , they will likely see partially-displayed webpage (due to CORS) and broken session handling (due to cookies not being properly handled). Therefore, domain shadowing is infeasible for malicious purposes such as setting a phishing website.

Furthermore, while it is possible to use domain shadowing to impersonate a static webpage, such impact is brought by the Internet's open nature and would still exist without using CDNs. For instance, a simple CNAME record:

```
CNAME    example.com    www.facebook.com
```

will let a user enter `example.com` in the address bar but be presented with the content from `www.facebook.com`. To modify the content on-the-fly, the owner of `example.com` can run a proxy server instead of a web server, which relays the requests (sent to `example.com`) to `www.facebook.com`. To this end, there are many websites that explicitly provide this so-called "website rehosting" service. We discuss more details of web rehosting and related security issues later in this section.

Regarding the open front-end (which is used by *DfDs++*), we are unable to see any benefit for a CDN to allow a user to set an arbitrary domain as the front-end. Therefore, we reckon it is an untightened implementation similar to domain fronting. On the other hand, however, this implementation does not incur any security impact to the CDN or the public. Using Fastly as an example, assuming the domain `a.com` does not use Fastly's service, but a user claims `a.com` as the front-end in Fastly. Because `a.com`'s owner controls its name server, a regular HTTP(S) request to `a.com` will be resolved to the IP address of `a.com`'s actual origin server instead of a Fastly's edge server. Therefore, under normal cases, an HTTP(S) request with `Host: a.com` will not reach Fastly's edge server at all, and this configured front-end will never receive any request.

7.2 Manipulation of the SOP and Cookies

Compared to the CDN and the publisher, the user is more likely to suffer from security risks because domain shadowing completely disturbs the established same-origin policy. Similar security issues have been comprehensively discussed in [47], where the authors studied the *website rehosting* services and discovered many practical security issues. Web rehosting services allow a user to access a domain by visiting the rehosting service provider's domain. For instance, assuming the rehosting service provider's domain is `rehosting.com`, a user can access Facebook via visiting `rehosting.com?url=https://www.facebook.com`. And a malicious website, say `evil.com`, when being visited by the same rehosting service, i.e., `rehosting.com?url=https://evil.com`, will be able to access credentials of all other sites because they all belong to the same origin. The authors of [47] identified five attack vectors regarding the rehosting service, and we list them below.

To facilitate the following illustration, we refer the websites `rehosting.com?url=https://example.com` and `rehosting.com?url=https://evil.com` as the *victim site* and *malicious site*, respectively.

Persistent Man-in-the-Middle: where the malicious site can register a *service worker* [48] in the user's browser and use it to intercept the traffic of the entire domain `rehosting.com`, including the victim site.

Privilege Abuse: where the privileges that have been granted to the victim site, such as accessing the user's location, can be accessed by the malicious site since they belong to the same origin.

Credential Theft: where the malicious site can exploit the auto-fill feature of the browser's password manager to steal the user name and password saved for the victim site.

History Theft: where the malicious site can use the *localStorage* API to access the local data stored by the victim site, and based on which to infer the user's browsing history.

Session Hijacking and Injection: where the malicious website can access the cookies set by the victim website and hijack the user's session.

And finally, although not mentioned in [47], the *CSRF attacks* [33] can also be a practical threat: if both websites are transformed to be under the same origin, conventional countermeasures such as CSRF token is no longer effective.

Domain shadowing by itself is not susceptible to any of the above attacks, because when being used alone, two target domains stay to be two separate domains even after being transformed, such as `static.xx.fbcdn.net.shadow.com` and `facebook.com.shadow.com`. Therefore, if one target domain does not allow access from another target domain, such restriction is still enforced by the browser for these two shadow domains. On the other hand, since *DfDs* and *DfDs++* transforms all target domains into a single front/shadow domain, they are susceptible to all of the above attacks but the *session hijacking* (since the cookie is handled by the extension, a malicious webpage cannot access cross-domain cookies).

Persistent MITM and *credential theft* can be prevented by using the private browsing mode (on Firefox and Edge [47]) since service worker and password manager are disabled. LocalStroage is also deleted on closing the browser. However, to the best of our knowledge, the *privilege abuse* and the plain *CSRF attack* can not be prevented. Therefore, the user of *DfDs* and *DfDs++* must be informed of the potential risks and be judicious to choose the website to visit.

A better solution that can eradicate all these security risks and address all the technical challenges in Section 4.1 is to "deceive" the browser. Specifically, we can let the browser "perceive" it is communicating with the target domain, but after a request is processed by the browser, we intercept and redirect it to the front or shadow domainbut instead. Essentially, this technique uses *Ds* as a tunnel rather than directly loading the target domain "as" the shadow domain. Such functions, based on our knowledge, is beyond the capability of a browser extension, which necessitates a heavily customized browser, and we denote this as one of our future directions.

8 Related Works

We discuss and compare other common censorship evasion techniques in this section.

1. Proxy-based Evasion Systems. A very common censorship evasion technique is running a proxy outside the censored area to relay the traffic between the censored user and the prohibited websites. Typical examples of proxy-based systems include virtual private network (VPN) [29, 31, 42], Phiphon [36], Tor [35], and HTTP/HTTPS proxies [45]. Most proxy-based systems, however, use static IPs, which can be trivially blocked once the IP is known by the censor. The user can also self-run a proxy server on a cloud service and leverage the cloud service's dynamic IP address. An obvious downside of this approach is the user must have an always-running instance to host the proxy, which incurs higher costs as well as maintenance complexity, as shown in Table 2.

The use of third-party proxies also incurs trust issues. Because many proxies are privately run, their configurations are not transparent to the user. Therefore, although the connection between the user and the proxy is secured, the user can not prevent the proxy from intercepting or tampering with the data exchanged between the user and the proxy [41,49]. On the other hand, although the CDN can intercept users' HTTPS connections and is theoretically more dangerous, many CDN services are provide by reputable cloud service providers, such as Google, Amazon, and Microsoft, and thus we deem CDNs are relatively more trustworthy than small proxies.

2. CDN Browser. Amir Houmansadr, et al. also studied leveraging CDNs for censorship evasion and proposed a tool named the *CDN browser* [23,52], where the authors found if the user knows the IP address of an edge server located outside of the censored area, they can skip the DNS query step and directly send HTTPS requests to this IP address, and use the `Host` header to indicate the domain. Because edge servers' IP addresses are usually dynamic and frequently changing, blocking CDN browser will result in the same collateral damage as domain fronting. However, the CDN browser's implementation requires the user to locally host a DNS server, which is beyond an ordinary user's skillset. Further, it is still possible to block CDN browser: the censor or the CDN can block an HTTPS connections that do not have a valid domain name, but only an IP address, as the SNI [9].

3. Protocol Tunneling. There are also several works that propose to tunnel traffic using protocols designed for other purposes. For instance, Sweet [51] proposes to encapsulate HTTP traffic inside email messages, while Castle [22] and Rook [44] attempt to use online games as a cover to tunnel secret traffic. These implementations usually suffer from lower QoS due to the overhead incurred by protocol translation.

9 Conclusion

In this paper, we proposed *domain shadowing*, a novel technique leveraging CDNs for censorship evasion. We demonstrated that domain shadowing is an effective technique that can resist most known censorship techniques, and is difficult to disable. We implemented domain shadowing as a Firefox extension based on Fastly's service and demonstrated its capability. Further, we thoroughly discussed domain shadowing's benefits and limitations, and envisioned its future moves. Our work paves the way for a fully-fledged censorship evasion system based on this novel technique.

References

[1] Giuseppe Aceto and Antonio Pescapé. Internet censorship detection: A survey. *Computer Networks*, 83:381–421, 2015.

[2] Amazon. Enhanced domain protections for amazon cloudfront requests. https://aws.amazon.com/blogs/security/enhanced-domain-protections-for-amazon-cloudfront-requests/, 2020.

[3] Arstechnica. Google disables domain fronting capability used to evade censors. https://arstechnica.com/information-technology/2018/04/google-disables-domain-fronting-capability-used-to-evade-censors/, 2020.

[4] Adam Barth. Http state management mechanism. 2011.

[5] Adam Barth, Collin Jackson, and Ian Hickson. The web origin concept. Technical report, RFC 6454, December, 2011.

[6] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238, 2014.

[7] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 605–616, 2012.

[8] Cloudflare. Cloudflare. https://www.cloudflare.com/, 2020.

[9] Cloudflare. Error 1003 access denied: Direct ip access not allowed. https://support.cloudflare.com/hc/en-us/articles/360029779472-Troubleshooting-Cloudflare-1XXX-errors#error1003, 2020.

[10] MDN Web Docs. Set-cookie. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie, 2020.

[11] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. The security impact of https interception. In *NDSS*, 2017.

[12] 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 *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[13] Donald Eastlake et al. Transport layer security (tls) extensions: Extension definitions. Technical report, RFC 6066, January, 2011.

[14] Fastly. Api reference. https://developer.fastly.com/reference/api/, 2020.

[15] Fastly. Fastly. https://www.fastly.com/, 2020.

[16] Fastly. Specifying an override host. https://docs.fastly.com/en/guides/specifying-an-override-host, 2020.

[17] R Fielding, Mark Nottingham, and J Reschke. Hypertext transfer protocol (http/1.1): Caching. *IETF standardization, RFC 7234*, 2014.

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

[19] GoDaddy. Godaddy. https://aws.amazon.com/cloudfront/, 2020.

[20] Luigi Grimaudo, Marco Mellia, Elena Baralis, and Ram Keralapura. Select: Self-learning classifier for internet traffic. *IEEE Transactions on Network and Service Management*, 11(2):144–157, 2014.

[21] Run Guo, Jianjun Chen, Baojun Liu, Jia Zhang, Chao Zhang, Haixin Duan, Tao Wan, Jian Jiang, Shuang Hao, and Yaoqi Jia. Abusing cdns for fun and profit: Security issues in cdns' origin validation. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 1–10. IEEE, 2018.

[22] Bridger Hahn, Rishab Nithyanand, Phillipa Gill, and Rob Johnson. Games without frontiers: Investigating video games as a covert channel. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 63–77. IEEE, 2016.

[23] John Holowczak and Amir Houmansadr. Cachebrowser: Bypassing chinese censorship without proxies using cached content. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 70–83, 2015.

[24] Lantern. lantern. https://lantern.io/, 2020.

[25] Christopher S Leberknight, Mung Chiang, Harold Vincent Poor, and Felix Wong. A taxonomy of internet censorship and anti-censorship. In *Fifth International Conference on Fun with Algorithms*, 2010.

[26] Philip Levis. The collateral damage of internet censorship by dns injection. *ACM SIGCOMM CCR*, 42(3), 2012.

[27] Marketsandmarkets. Content delivery network market global forecast to 2024. https://www.marketsandmarkets.com/Market-Reports/content-delivery-networks-cdn-market-657.html, 2020.

[28] Mybroadband.co.za. How telegram and signal used domain fronting to beat censors. https://mybroadband.co.za/news/security/259019-how-telegram-and-signal-used-domain-fronting-to-beat-censors.html, 2020.

[29] Daiyuu Nobori and Yasushi Shinjo. Vpn gate: A volunteer-organized public vpn relay system with blocking resistance for bypassing government censorship firewalls. In *11th USENIX Symposium on Networked Systems Design and Implementation NSDI 14*, pages 229–241, 2014.

[30] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 2010.

[31] Vasile C Perta, Marco V Barbera, Gareth Tyson, Hamed Haddadi, and Alessandro Mei. A glance through the vpn looking glass: Ipv6 leakage and dns hijacking in commercial vpn clients. *Proceedings on Privacy Enhancing Technologies*, 2015(1):77–91, 2015.

[32] CDN Planet. Content delivery networks. https://www.cdnplanet.com/cdns/, 2020.

[33] PortSwigger. Cross-site request forgery (csrf). https://portswigger.net/web-security/csrf, 2020.

[34] Tor Project. Tor meek. https://trac.torproject.org/projects/tor/wiki/doc/meek, 2020.

[35] Tor Project. Tor project. https://www.torproject.org/, 2020.

[36] Psiphon. Psiphon. https://psiphon.ca/, 2020.

[37] Will Reese. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2, 2008.

[38] Signal. A letter from amazon. https://signal.org/blog/looking-back-on-the-front/, 2020.

[39] Stackpath. Stackpath maxcdn. https://www.stackpath.com/maxcdn/, 2020.

[40] Tinyproxy. Tinyproxy. http://tinyproxy.github.io/, 2020.

[41] Giorgos Tsirantonakis, Panagiotis Ilia, Sotiris Ioannidis, Elias Athanasopoulos, and Michalis Polychronakis. A large-scale analysis of content modification by open http proxies. In *NDSS*, 2018.

[42] Ramachandran Venkateswaran. Virtual private networks. *IEEE potentials*, 20(1):11–15, 2001.

[43] John-Paul Verkamp and Minaxi Gupta. Inferring mechanics of web censorship around the world. In *FOCI*, 2012.

[44] Paul Vines and Tadayoshi Kohno. Rook: Using video games as a low-bandwidth censorship resistant communication platform. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, pages 75–84, 2015.

[45] VPNmentor. Best proxy services of 2020. https://www.vpnmentor.com/blog/top-proxy-services/, 2020.

[46] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V Krishnamurthy. Your state is not mine: a closer look at evading stateful internet censorship. In *Proceedings of the 2017 ACM IMC*, pages 114–127, 2017.

[47] Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, and Tatsuya Mori. Melting pot of origins: Compromising the intermediary web services that rehost websites. In *Proceedings of the Network and Distributed System Security Symposium*, 2020.

[48] MDN web docs. Service worker api. https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API, 2020.

[49] Philipp Winter, Richard Kower, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled onions: Exposing malicious tor exit relays. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 304–331. Springer, 2014.

[50] zendesk. Host mapping - changing the url of your help center. https://support.zendesk.com/hc/en-us/articles/203664356-Host-mapping-Changing-the-URL-of-your-Help-Center, 2020.

[51] Wenxuan Zhou, Amir Houmansadr, Matthew Caesar, and Nikita Borisov. Sweet: Serving the web by exploiting email tunnels. *arXiv preprint arXiv:1211.3191*, 13, 2012.

[52] Hadi Zolfaghari and Amir Houmansadr. Practical censorship evasion leveraging content delivery networks. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1715–1726, 2016.