CDNs' Dark Side: Security Problems in CDN-to-Origin Connections

BEHNAM SHOBIRI, Concordia University, Canada
MOHAMMAD MANNAN, Concordia University, Canada
AMR YOUSSEF, Concordia University, Canada

Content Delivery Networks (CDNs) play a vital role in today's Internet ecosystem. To reduce the latency of loading a website's content, CDNs deploy edge servers in different geographic locations. CDN providers also offer important security features including protection against DoS attacks, Web Application Firewalls (WAF), and recently, issuing and managing certificates for their customers. Many popular websites use CDNs to benefit from both the security and performance advantages. For HTTPS websites, TLS security choices may differ in the connections between end-users and a CDN (front-end or user-to-CDN), and between the CDN and the origin server (back-end or CDN-to-Origin). Modern browsers can stop/warn users if weak or insecure TLS/HTTPS options are used in the front-end connections. However, such problems in the back-end connections are not visible to browsers or end-users, and lead to serious security issues (e.g., not validating the certificate can lead to MitM attacks). In this paper, we primarily analyze TLS/HTTPS security issues in the back-end communication; such issues include inadequate certificate validation and support for vulnerable TLS configurations. We develop a test framework and investigate the back-end connection of 14 leading CDNs (including Cloudflare, Microsoft Azure, Amazon, and Fastly), where we could create an account. Surprisingly, for all the 14 CDNs, we found that the back-end TLS connections are vulnerable to security issues prevented/warned by modern browsers; examples include failing to validate the origin server's certificate, and using insecure cipher suites such as RC4, MD5, SHA-1, and even allowing plain HTTP connections to the origin. We also identified 168,795 websites in the Alexa top million that are potentially vulnerable to Man-in-the-Middle (MitM) attacks in their back-end connections regardless of the origin/CDN configurations chosen by the origin owner.

CCS Concepts: • **Security and privacy → Web protocol security**.

Additional Key Words and Phrases: Network Security, CDN, TLS

ACM Reference Format:
Behnam Shobiri, Mohammad Mannan, and Amr Youssef. 2021. CDNs' Dark Side: Security Problems in CDN-to-Origin Connections. *Digit. Threat. Res. Pract.* 111, 111, Article 111 (2021), 22 pages. https://doi.org/11111111

1 INTRODUCTION

Content Delivery Networks (CDNs) are an essential part of the Internet. Due to their globally distributed network infrastructures, high-performance computing power, and high network bandwidth, many websites, including many popular sites, are using CDNs. According to a Cisco report [14], 56% of the Internet traffic was carried by CDNs in 2017, and by 2022, Cisco predicts that CDNs will carry 72% of total Internet traffic. To reduce the latency of loading a website's content, CDNs deploy edge servers scattered across various geographic locations. Moreover, CDNs provide many different security advantages including Denial of Service (DoS) protection, Web Application Firewalls (WAF), and certificate management for their customers.

Authors' addresses: Behnam Shobiri, Behnam.shobiri@concordia.ca, Concordia University, Montreal, Canada; Mohammad Mannan, Concordia University, Montreal, Canada, m.mannan@concordia.ca; Amr Youssef, Concordia University, Montreal, Canada.

2576-5337/2021/111-ART111 $15.00
https://doi.org/11111111

When a website is using a CDN, before serving the client, the CDN's edge servers request the data through a connection to the origin server and cache the website's content. When clients request the website, they get redirected to the closest CDN edge server which returns the cached content. Therefore, instead of a straightforward connection between the client and the origin server, there is a connection between the client and the CDN edge server and another connection between the CDN and the origin server. We refer to the communication between the CDN and the origin server as the "back-end" (CDN-to-Origin) connection. Similarly, we refer to the communication between the CDN and the client as the "front-end" (user-to-CDN) connection [29]; see Figure 1.

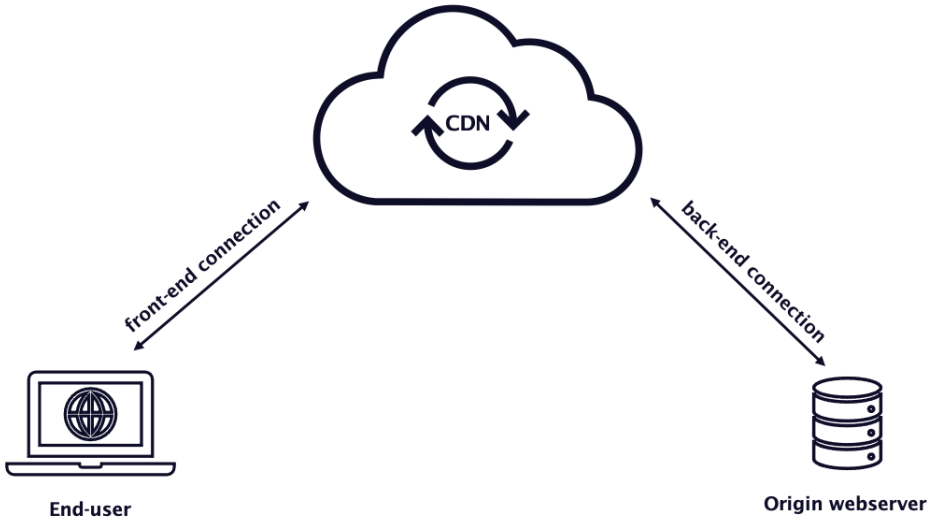

Fig. 1. The front-end connection is the connection between the client/user and CDN, and the back-end connection is between the CDN and the origin.

Since the user's browser is not involved in the back-end communication, the browser would not warn the end-user about any security vulnerabilities in back-end communication, specifically for HTTPS websites. In other words, while users think that they are connecting to a website with the best HTTPS security practice (according to the browser and certificate), users are oblivious to any security vulnerability in the back-end communication. Therefore, the back-end communication, in particular, can be a potential target for attackers. The back-end vulnerabilities can have different root causes including the lack of certificate validation, the use of weak ciphers, weak key exchange configuration, or outdated TLS version. These security vulnerabilities may lead to information leakage and impersonation against popular websites served via CDNs.

In 2014, Liang et al. [29] investigated the certificate validation process of back-end communication for five CDNs, but not the security parameters such as ciphers and key exchange configurations. While other studies [22, 23, 26] partly explore the CDN back-end IP addresses and open ports, HTTPS security of the back-end connections remains unexplored. On the other hand, past research [12, 29] proposed several front-end scanning methodologies to identify CDN-powered websites. However, such techniques cannot be relied on anymore for reasons including: recent privacy policy changes in the Whois database, and not considering the Server Name Indication (SNI) extension which has been more widely used in recent years.

The primary goal of our paper is to identify security vulnerabilities associated with CDNs' back-end communication. We deploy our test website on 14 leading CDN providers and monitor the back-end connection. In

particular, we review the supported ciphers and key exchange configurations used in the back-end connection. We also examine the CDNs' certificate validation process in the back-end connection, since in the back-end connection, the CDN acts as an HTTPS client and the origin as the HTTPS server. In addition, we check the default security configurations and available options for each CDN provider to make sure they align with the best security practice (as adopted in modern browsers). To measure how many websites are potentially affected by back-end vulnerabilities, we also conduct front-end scanning on Alexa [4] top million websites, by significantly improving past scanning methodologies [12, 29].

Contributions. We summarize our contributions as follows:

(1) We develop a test framework to evaluate security issues in CDNs' back-end (CDN-to-Origin) TLS connections. Our framework includes tests for back-end certificate validation (i.e., whether the CDN performs proper certificate validation for the origin website), the use of weak cryptographic parameters such as RC4, and weaknesses in the default CDN security configurations affecting the back-end connection. This is the first such comprehensive framework for testing TLS security in the back-end CDN connections.
(2) We use our test framework to evaluate the back-end TLS connections of 14 leading CDNs. In terms of certificate validation (tested using eight obviously malformed certificates), we found that none of the CDNs validate the origin server's certificate properly. 9/14 CDNs, including Microsoft Azure, do not properly perform the basic certificate validation tests such as self-signed, wrong Common Name, and unknown certificate issuer. The remaining 5/14 CDNs do not validate the revocation status of the origin's certificate.
(3) We develop a front-end scanning tool for identifying CDN-powered websites, and detect 168,795 websites among the top Alexa 1M sites that are using vulnerable CDNs; all these highly popular sites are possibly affected by our findings regardless of the origin/CDN configurations chosen by the origin owner. We have disclosed our results to all affected CDNs, some of which also confirmed/fixed the issues we identified.
(4) We open-sourced our back-end security evaluation framework, and our front-end scanning tool used to identify CDN-powered websites. Beside researchers, website admins can use our framework to identify and monitor security issues in a CDN's back-end connection. Our scanner can be used as a measurement tool for both security and non-security use cases.[1]

In addition to the serious certificate validation issues, our framework also reveals that 9/14 CDNs support 1024-bit and 2048-bit Diffie-Hellman (DH) prime moduli; and 3/14 CDNs support the broken RC4 cipher. Moreover, in terms of weak/insecure default settings, we found that 5/14 CDNs (including Cloudflare and Amazon CloudFront) do not use a secure connection in their back-end communication, i.e., in the Amazon CloudFront, the CDN-to-Origin connection is HTTP by default. We have disclosed all our findings to the respective CDNs, and summarize the vendor responses in Section 6.5.

2 BACKGROUND AND THREAT MODEL

In this section, we provide a brief background on HTTPS/TLS and CDNs, and discuss our threat model.

2.1 HTTPS

HTTPS provides end-to-end encrypted communication between the client and the server. HTTPS provides confidentiality, integrity, and authentication. Therefore, the connection is protected against both active and passive attackers. HTTPS relies on Transport Layer Security (TLS) protocol for security primitives. To provide authentication, TLS employs Public Key Infrastructure (PKI) and X.509 certificates. For example, in TLS 1.3, in the "Certificate Verify" message, the server provides the digital signature over the transcript hash (hash of all previous handshake messages). This massage proves that the server is the owner of the presented certificate and

[1] https://github.com/Behnam-Shobiri/CDN-Finder

ensures the integrity of the handshake messages until that point in the handshake. Although there are minor changes in different TLS versions, the previous versions follow a similar process as TLS 1.3. The client needs to authenticate the server certificate and the corresponding digital signature to avoid the MitM attack. According to TLS 1.3 RFC [35]: "The receiver of a CertificateVerify message MUST verify the signature field."

Certificate validation. Certificates bind the owner of the domain to a public key. Modern browsers validate the certificate based on the certificate chain and the certificate name. Browsers also check the revocation status of the certificate. If the certificate does not satisfy all the required validations, the browsers will show a warning/error to indicate the risk. Below, we explain each item that modern browsers use for verifying the certificate.

- *Chain validation.* A legitimate certificate must be signed by a valid root Certificate Authority (CA). Since the client has access to trusted root CAs, the client can validate the X.509 certificates and authenticate the server. However, since in most cases, the root CAs do not sign the leaf certificates, the server presents the full chain along with all the intermediate CA certificates in the handshake; thus, the client can validate the certificate chain. To validate the certificate chain, the client authenticates all the intermediate CAs to the point that a trusted CA signed the intermediate CA's certificate. If the certificate chain validating process terminates with a trusted CA, the browser accepts it as a certificate with a valid issuer.
- *Name validation.* To ensure that the presented certificate was issued for the same domain that the client intended to connect, the client browser validates the Common Name (CN) and the Subject Alternative Name (SAN) extension. The SAN extension can be used to cover subdomains individually or, it can cover multiple subdomains (by using wildcards). The SAN extension can also be used to cover multiple different domains in a single certificate. The certificate is valid for the domain if the domain is present in the SAN or CN field.
- *Revocation status.* If the private key corresponding to the certificate is compromised, the certificate should be revoked. There are a few mechanisms for certificate revocation including, Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP). The revocation mechanism may differ; however, regardless of the revocation mechanism, the client should check the revocation status of the certificate to avoid accepting revoked certificates. The owner of the certificate can ask the CA (that has issued the certificate) for revocation. For example, Let's Encrypt uses OCSP for revocation [27].

Security primitives. A TLS client starts the connection with the "Client Hello" message in which the client presents the supporting cipher suites (ordered based on the client preferences), available extensions, TLS version, etc. The TLS server chooses the best fit cipher suite and other parameters and sends them back to the client in the "Server Hello" message. To protect the confidentiality and integrity of the connection, modern browsers warn the client upon receiving weak cryptographic primitives such as weak cipher suites, broken versions of TLS/SSL, or any other known vulnerability.

During the TLS connection, the client and server employ a key exchange protocol to establish a shared secret. Depending on the TLS version, the key exchange could be based on RSA or (EC)DH(E). For example, in TLS 1.3, RSA has been deprecated due to lack of forward secrecy. One of the factors for DH security is the prime moduli that it uses. In 2015, the Logjam attack revealed that, by attacking a small number of common primes, large amounts of communications can be compromised when using Diffie-Hellman (DH) 1024-bit or smaller prime moduli [2, 37]. Nowadays, modern browsers do not support DH with 2048-bit prime moduli as well. TLS clients can also use extensions for various reasons. For example, the client can use the Server Name extension to indicate the domain name (in case a single IP address is used for multiple domains).

Server Name extension. The client must provide this extension (also known as Server Name Indication) for the HTTPS server when the server supports multiple domains (with multiple certificates) on a single IP. Using this extension, the client indicates the domain that the client wishes to connect. Subsequently, the server includes the

corresponding certificate in the TLS handshake. Thus, the server can support multiple HTTPS-enabled websites using a single IP.

2.2 CDN

A CDN is a geographically distributed infrastructure that website owners can use to reduce the access time for their website. Though CDN's primary purpose is to reduce the latency of loading the website, CDN providers also offer security services such as protection against DoS attacks and Web Application Firewall (WAF). When the user requests the CDN-powered website, the CDN's DNS server redirects the user to the best-fit edge server. The CDN load balancing system chooses the best edge server based on different metrics including, location and available resources on the edge servers [9, 22]. If the website's content is available in the chosen edge server, the edge server sends the cached content; otherwise, it fetches the content from the origin server using a separate connection and caches the content. Therefore, when other users request the same content, the edge server sends the cached content, which significantly reduces the accessing time.

To use the security advantages offered by both CDNs and HTTPS, websites delegate their domain using a DNS-based request routing mechanism such as using CNAME or using CDN's DNS server [29]. CDN edge servers need access to a valid certificate and corresponding private key for the delegated domain to complete the HTTPS handshake. Access to unencrypted data allows the edge server to filter malicious traffic using WAF, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS). Furthermore, due to the massive bandwidth and computational power of the CDN's edge servers and CDN load balancing system, CDN can mitigate DoS attacks. Moreover, since each CDN provider has a limited number of IPs, CDNs use SAN or SNI extension for their edge servers' TLS certificate. Notably, websites that use Cloudflare can create serverless [18] applications using the Cloudflare Workers [16] without using an origin server. However, this configuration was not introduced when we began our study and by the time that we finished our study, only the beta version was available. Therefore, it is not considered in our results.

2.3 Threat model

Following the Dolev–Yao model [19], we assume an on-path attacker for the back-end communication (the connection between the CDN and origin server). The attacker can alter, drop, or redirect the traffic; she can also act as a man-in-the-middle; see Figure 2. However, the attacker does not have any capabilities beyond the HTTPS/TLS attack model (such as breaking the secure encryption or issuing a valid certificate without owning the domain). Several CDN factors can facilitate the attacker to be on the back-end communication path; such factors include the inadequate number of egress IPs, high IP-churn ratio, and optional features like the origin shield (discussed more in Section 6.2).

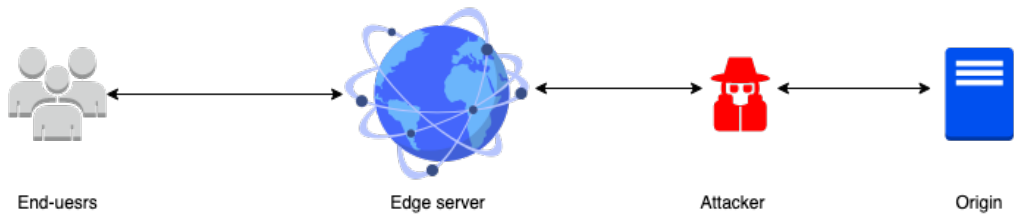

Fig. 2. Overview of the attack model. The attacker is an on-path attacker in the back-end communication.

3 RELATED WORK

Liang et al. [29] analyzed both back-end and front-end CDN connections. For their front-end connection, they conduct a measurement study for 20 known CDNs. They focused on two main issues: the TLS delegation methods and problems when a website uses CNAME or DNS as a request routing mechanism; and the deployment status (including the response code and any warning from the browser) of HTTPS for the same websites. From Alexa top 1 million, they identified 10,721 HTTPS-enabled websites that were using CDN with DNS or CNAME. They found 31.2% of these websites showed a valid certificate. Among them, 20.1% used custom certificates (custom certificates is when CDN would ask users to share their private keys), and 11.1% used a shared certificate (using the SAN extension). The other 68.8% showed an invalid certificate. For their back-end connection, they investigated 5 CDNs and their certificate validation process. 3/5 CDNs supported HTTPS, and none of them validated the origin certificate. However, they did not review the security parameters nor the default settings of the CDNs.

Cangialosi et al. [12] analyzed the prevalence of private key sharing. According to their attack model, if the owner of the IP address is not the same as the owner of the domain, it would be considered as private key sharing. For example, if the website is using a hosting provider, they would consider it as a private key sharing. Moreover, they found that many websites share their private keys with more than one hosting provider or CDN. In particular, 76.5% of the sites they scanned were sharing at least one private key with third-party hosting providers or CDNs. They realized that a small number of hosting providers and CDNs are controlling many organizations' private keys. For example, they found that if the attackers can compromise ten top hosting providers, they would control 45.3% of the domains in their scan. They did not consider the certificates that were using the SNI extension, as SNI was not widely adopted during their scan. We cannot use their methodology to determine if two different domains are owned by the same organization since the privacy policy in their dataset (such as Whois) has changed.

Guo et al. [22] showed six different attacks using CDNs, based on the fact that CDNs do not validate the ownership of the origin server. In other words, when an attacker registers a new domain in a CDN, without any validation, the CDN provides a link that points to the origin server. They used this feature and created six different attacks. They also presented a methodology to identify the inbound IP addresses of 8 CDNs. They used HTTP headers to identify the inbound IPs and extracted distinguishable features (in the HTTP header) for each CDN. In our work, we used the HTTP headers to identify websites using a CDN that still includes the distinguishable HTTP header (see Section 4.2).

Guo et al. [23] showed three different attacks to break the CDN's DoS protection. By using the discrepancy between the HTTP/2 and HTTP1, they discovered the attacker could create a specially crafted request packet that gets amplified in the back-end communication, leading to a bandwidth amplification attack. They also showed that some CDN providers start forwarding the POST request when they received the header and, the attacker could use the POST header to exhaust the back-end connection limits and create a DoS attack. Moreover, they discovered that the CDNs use a very limited number of IPs to connect to the origin. By dropping the connection for these egress IPs, the attacker can prevent most users to connect to the websites. The inadequate number of egress IPs helps the attacker (in our attack model) in the back-end to be on path-attacker for the majority of the back-end communications.

4 METHODOLOGY

We first discuss our test framework for the back-end connection. We then discuss our front-end scanning technique for identifying CDN-powered websites.

4.1 Back-end connection

In order to investigate the security of the back-end connection, we check the following factors in back-end communication: (1) the ciphers, hashing algorithms, and acceptable DH configuration that CDNs support; (2) the

origin certificate validation process by the CDN provider; and (3) the default settings and options for each CDN provider. Table 1 summarizes the tests that we performed for the back-end connection. In the rest of this section, we explain in detail how we test these three factors.

Type of the back-end test	Related tests
Certificate related	Check the origin certificate validation with the following malformed certificates: self-signed, signature mismatch, unknown issuer, wrong Common Name (CN), certificates with NULL in the CN field, fake GeoTrust Global CA, certificates with NULL in the SAN field and revoked.
Security parameters	Check the supported weak ciphers, hash algorithms, vulnerable DH groups and supported TLS versions.
Default and options	Check the default configuration and options regarding the origin certificate validation and security parameters.

Table 1. Summary of the tests performed in the CDN back-end connections.

Security parameters. To monitor the CDN's back-end communication, we create a website (with the domain name "www.cdn-tests.ga"). Our website mimics "badssl.com" [8], a web service that implements common vulnerabilities related to HTTPS that is part of the Chromium project.[2] We use badssl.com's GitHub as a reference and modify the source code to work for our domain. Following badssl.com, each subdomain in our website has a specific vulnerability. For example, dh1024.cdn-tests.ga uses DH over a 1024-bit prime group. We use our own server to have full control over the network traffic. On our server, we install the modified Docker image of badssl.com that is running on Ubuntu 18.04 LTS. To confirm our findings, we capture the traffic from the CDN to our server by running Wireshark [1] on our server. To avoid the cache-hit feature of the CDN, we clear the cache (using the CDN's admin portal). We use IP ownership information to validate that the IP address belongs to the specific CDN (at the time we connect to the website through the CDN for the first time after clearing the cache).

To discover all the acceptable configurations for each CDN, we capture the corresponding packets from the CDN provider to our server. We first check their protocol to see if the CDN provider connects to our server using HTTP instead of HTTPS. If the CDN uses HTTPS, we further analyze the TLS "Client Hello" message from the CDN provider. We observe the supported cipher suites, hashing algorithms, and TLS version that the CDN supports. We refer to the mentioned variables as security parameters.

We deploy our website and its subdomains on each CDN. Afterward, we check the subdomain as an end-user. If we can see the content of the misconfigured subdomain, it indicates that the CDN server accepts the misconfigured subdomain as a valid HTTPS configuration. Notably, when CDNs do not accept the origin configuration, they show an origin-related error page, and sometimes explicitly display the origin issue.

To test Diffie-Hellman (DH) related issues, following badssl.com, we create four subdomains that use different DH prime sizes: 480, 512, 1024, and 2048. Modern browsers detect and terminate (not as a warning) all such DH primes. To test cipher-related issues, we create subdomains that each supports a specific broken/weak cipher. Ciphers include RC4, RC4 with MD5 as a hashing algorithm, and no cipher (null). All the mentioned ciphers are considered vulnerable and modern browsers terminate the connection. Furthermore, a previous study [11] showed that attackers can break connections that use block ciphers with small block lengths including 3DES; thus, we also check 3DES.

Finally, to confirm our findings, we repeat the same experiments using badssl.com and its corresponding subdomains as the origin and verify our results. Notably, this option is not available for CDNs that use their own

[2]www.chromium.org

DNS server as a request routing mechanism (Cloudflare and ArvanCloud), since it would require changing the DNS server of badssl.com (which we do not control).

Validating the origin's certificate. If a TLS client does not authenticate the server's certificate, the TLS connection can be subjected to a man-in-the-middle attack. In the regular TLS scenario, the user's browser validates the server's certificate to make sure that the browser is connecting to the right server. However, when the end-users are connecting to a website that uses a CDN, they get redirected to the CDN edge server which returns the cached data. Thus, the user's browser can only authenticate the certificate which the CDN provider presents. In the back-end connection, the TLS client is the CDN server and the TLS server is the origin; therefore, CDN providers are expected to validate the origin server certificate. A few CDNs such as Cloudflare, StackPath, and BunnyCDN offer validating the origin certificate as an option; i.e., by default, these CDNs will perform no origin certificate validation. For all the certificate related tests, we enable this feature. We create different malformed certificates to check if the CDN is properly validating the origin server's certificate. These checks include validating the certificate chain, the certificate's Common Name (CN) and Subject Alternative Name (SAN), and the Certificate Authority (CA) that issued the certificate. We also test for NULL prefix attacks [31].

We cannot use badssl.com's certificates for our subdomains since they are not issued for our domain. However, we configure the CDNs to use the badssl.com's subdomains as the origin to confirm our findings. We also use the badssl.com's subdomain as the origin for the certificates that need special attention from a CA (badssl.com CA partners in this case). To create malformed certificates for our subdomains, we use the OpenSSL [32] library, following Waked et al. [36]. However, we cannot create all of their malformed certificates with a valid Certificate Authority (CA). Therefore, we only generate the ones that do not need a valid CA. Below, we define our malformed certificates and how we create them.

- *Self-signed certificate.* We create a certificate that is signed using its private key. With this test, we verify that CDNs avoid accepting self-signed certificates since anyone can create a self-signed certificate for any domain.
- *Signature mismatch.* We use the Let's Encrypt [28] Certificate Authority (CA) to obtain a legitimate certificate for our domain. We use our legitimate certificate and replace one byte of the certificate with a random value (at the end of the certificate). Since the signature is positioned as the last item on certificates, this creates a signature that cannot be validated using the CA's public key. Afterwards, we check the domain with a browser and verify that browser shows the signature mismatch error. With a mismatched certificate, we confirm that CDNs check the signature on the certificate.
- *Unknown issuer.* We use the OpenSSL library to create a Certificate Authority (CA). Consequently, we use the CA's private key to sign our domain's certificate. With this test, we make sure that CDNs check the issuer to be a valid root CA.
- *Wrong Common Name (CN).* We use a valid certificate that was issued by a valid CA for a different domain (that we control). Therefore, it does not have the right CN value for the domain name that presents it. With this test, we make sure that CDN providers check the CN field properly, and their validation is not just limited to the signature and issuer.
- *Fake GeoTrust Global CA.* In this experiment, we mimic the Geo Trust Global CA (a root CA). We create a CA certificate with the same Common Name (CN = GeoTrust Global CA), the Organization field (O = GeoTrust Inc.), and the Country field (C = US). We signed our fake root CA using its private key. Subsequently, we signed our website's certificate using the fake CA's private key. Using this test, we verify that CDNs do not rely only on presented values, and properly verify the root CA's signature.
- *Certificate with NULL in CN field.* NULL prefix attacks [31] were presented in Blackhat 2009. Due to the incorrect parsing of the NULL character in the CN by browsers, attackers were able to impersonate the websites without owning a certificate. We create a certificate with the NULL character in the CN field and

obtain our legitimate certificate from Let's Encrypt. We put NULL in the CN field of the certificate and register the corresponding domain name to discover if CDN providers are vulnerable to this attack.
- *Certificate with NULL in SAN field.* This malformed certificate is the same as the Certificate with the NULL character in the CN field. However, here we place the NULL character in the SAN field to perform the same attack on the domain names inside the SAN field.
- *Revoked.* We generate and revoke a valid certificate for our domain using Let's Encrypt. After a day, we check the domain with different modern browsers and verify that browsers detect the revoked certificate.

Note that, we refer to *basic certificate validation* for all the certificate-related checks, excluding the revocation test. We configure our server to present these certificates to the clients including CDNs. To ensure that the CDN fetches the new certificate, we change the HTML content and clear the CDN cache (using the CDN's admin portal). We then request the content through the CDN provider (as a normal user). If we see the new content without any origin-related error from CDN, we conclude that the CDN has not validated the certificate properly.

Default settings and extra options. When the website owner is deploying the website on a CDN, there are many options and default settings. Depending on the CDN provider, options and default settings change vastly. During our experiments, we identify the available options that would create a vulnerability for the website. If an existing option makes a website vulnerable, it should not be offered by the CDN provider in the first place. In particular, since the user's browser cannot warn the user about these risks, a vulnerable option in the back-end becomes even more severe. The default settings are also important from the security perspective. The CDN provider should have a default setting that aligns with the best security practices. If CDN providers have weak default settings, the website admins are expected to go through configurations, and change them to secure options (e.g., changing HTTP to HTTPS for the back-end connection).

4.2 Front-end connection

Feature extraction. To identify the websites that are using a CDN, we extract different features from websites. Although previous studies [22, 23] used similar features, we modify and update the parameters of their features due to various reasons including, the new extensions in certificates such as SNI, changes in privacy policies, and the elimination of some features that would leak security-sensitive information. These features are reverse DNS, HTTP headers, and CNAME (explained below). We developed Python scripts to connect to all the domains in the Alexa top one million with a 60-second timeout for each website. For each website, we extract and store all the features. Using the known features for each CDN (See Table 7, Table 8, and Table 9), we identify the domains employing the known CDNs. To identify the popular unknown CDNs, we cluster the extracted features based on their reverse DNS (top referred reverse DNS entries) and manually check them to verify that they belong to a CDN. Using this approach we identify 12 new CDN (see Table 4). To accelerate the crawling time for Alexa top million websites, we concurrently launched 20 instances of our crawler that would start from different indexes in Alexa top 1 million (on a machine with Ubuntu 18.04, i9-9900K CPU, and 32 GB of RAM). It took approximately one week to finish the scan. Below, we explain the features that we extract from each website to identify the websites that are using a CDN.

- *Reverse DNS.* Besides the typical role of the Domain Name System (DNS) that is mapping of domain to IP address, organizations can provide reverse DNS information on their DNS server. Many organizations (including some CDN providers) choose to publish the reverse DNS information to provide hints about the owner of the IP. By performing a reverse DNS query on the IP of the websites, we can distinguish the websites that are using some known CDNs [12]. For example, the domain name "www.weibo.com" would resolve to IP address 23.2.4.161. Reverse DNS for this IP would point to "a23-2-4-161.deploy.static.akamaitechnologies.com", clearly pointing to Akamai [3]. If the organization that owns the IP (in this case, the CDN provider) has published the reverse DNS information, we can identify the websites which are using known CDNs.

Furthermore, some organizations that provide both hosting and CDN services publish clear hints in their reverse DNS to distinguish these services. For instance, Amazon AWS [6] reverse DNS would point to "*.amazonaws.com". However, reverse DNS over Amazon CloudFront [5] (Amazon CDN service) IPs would point to "*.cloudfront.net". See Table 7 in the Appendix for the reverse DNS information for each CDN that publishes reverse DNS information.

- *HTTP headers.* Guo et al. [22] mentioned that CDN providers would include distinguishable headers in the HTTP connection. Although not all the CDN providers include these headers, we investigate known CDNs and check if any CDN still includes these headers. Notably, some mentioned headers in [22] are not valid anymore. In some cases, CDNs have completely removed the headers because of security reasons. For example, Fastly [21] removed the "Server" header since it was leaking information about the server software and its version, which could aid an attacker [10]. In our scan, we observe that some websites and CDNs are still displaying the version of their server software, including their OpenSSL version.[3] Table 8 in the Appendix shows the unique headers that we found for the investigated CDNs. Note that if a website is using a CDN, it does not always need to use the CDN unique header. Moreover, some CDNs such as Fastly, let users modify the headers or choose to use/not use them [20].
- *CNAME.* CDN providers use the CNAME field to redirect users from the original domain to their assigned subdomains. Since most CDN providers support the CNAME as a request routing mechanism [24], we use the website's DNS server to check if it is presenting a known CDN CNAME. Although some CDN providers announce their CNAME subdomain, to the best of our knowledge, there is no centralized dataset to find the CNAME for CDNs. We found these domains by reading CDN documentations, searching online, and manually inspecting the most referred to CNAMEs in our scan. See Appendix (Table 9) for the CNAMEs that we use in our front-end measurement.
- *TLS certificates.* Prior work [22, 29] used certificates to conduct CDN measurement studies. However, when the work in [29] was conducted (2014), SNI was not used. Cangialosi et al. [12] also mentioned that the SNI extension was not popular at the time that they conducted their study (2016); for instance, CDN providers like Akamai have just started offering SNI after Cangialosi et al. finished their scan. In contrast, nowadays, most known CDN providers support SNI. For example, Cloudflare would issue an SNI certificate for their users free of charge. Therefore, we did not use TLS certificates as a feature for our front-end scanning.

5 RESULTS

5.1 Back-end connection

In our back-end investigation, we deploy our website on as many CDNs as it was possible for us. Some CDN providers have free-trial or open to individual users at a reasonable price. However, some CDN providers (such as Akamai [3] and Imperva [25]) only support enterprise clients. Therefore, we investigate the back-end communication of 14 CDN providers that were affordable for us. Table 2 and Table 3 summarize our results. Below we categorize our findings based on the root causes.

Lack of origin server certificate validation. All of the tested CDNs have at least one shortcoming regarding the validation of origin server certificates (Table 3). The lack of certificate validation can be further categorized as not performing the basic certificate validations such as not validating the CN field or the CA that issued the certificate, and not performing the revocation check. 7 CDNs are completely missing the certificate validation process of the origin server's certificate in their back-end communication. Since these CDNs do not check the

[3]We observed that 17,162 websites are still displaying security-sensitive information including the version of their server and OpenSSL. For example, "cafebazaar.ir" a popular website in Iran that is an application store for Android devices, is using Nginx version 1.15.6. For few websites, the OpenSSL version is also visible in their HTTP headers. We recommend that these websites and CDNs follow other CDNs like Fastly and remove the version of sensitive software and libraries such as OpenSSL.

CDN	Back-end vulnerable DH	Back-end support of weak ciphers	Back-end support of weak hash	Back-end TLS version	Front-end TLS version (what end-users see)	Default setting and other problems
Cloudflare		3DES	SHA-1	Max TLS 1.3 Min TLS 1.0	TLS 1.3	Back-end: default does not validate the origin certificate
Amazon		RC4 3DES	SHA-1 MD5	Max TLS 1.3 Min SSL 3.0	TLS 1.3	Back-end: default is HTTP (not HTTPS)
KeyCDN	1024 2048		SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.3	
CDN77	2048		SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.3	
BunnyCDN	1024 2048		SHA-1	Max TLS 1.3 Min TLS 1.0	TLS 1.3	
Hostry	1024 2048		SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.2	
Microsoft Azure			SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.2	
Fastly	1024 2048	RC4, 3DES	SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.2	
Medianova	1024 2048	RC4 3DES	SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.3	
ArvanCloud	1024 2048		SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.3	
CDNSun	1024 2048		SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.2	
G-Core CDN	Not tested		SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.2	Back-end: default is HTTP (not HTTPS)
StackPath	1024 2048		SHA-1	Max TLS 1.2 Min TLS 1.0	TLS 1.3	
Azion	Not tested		SHA-1	Max TLS 1.3 Min TLS 1.0	TLS 1.3	Back-end default: change depending front-end HTTP(S). Front-end: offers no option to disable HTTP

Table 2. Summary of weak security parameters that CDNs support in their back-end communication as well as insecure default settings. G-Core and Azion CDN were not available to us at the time of our DH-related experiment. Moreover, Azion default for the back-end connection is to change the back-end HTTP(S) according to the front-end HTTP(S). Notably, to check the maximum TLS version of the front-end connection, we use the client that supports TLS 1.3. Therefore, the TLS version depends on the maximum TLS version that the CDN server supports.

origin certificate, the attackers only need to redirect the traffic to their server instead of the origin, and successfully impersonate the origin using any certificate. Azure and StackPath perform inadequate validation on the origin server certificate. Azure validates the origin server's certificate issuer, expiration, and presents an error when the origin server's certificate is invalid. However, it does not validate the CN (Common Name) field and revocation status for the origin server certificate. Thus, the most straightforward approach for the attackers to perform the MitM attack is to impersonate the origin server with a valid certificate issued for an attacker-controlled domain.

StackPath recently implemented a certificate validation feature for the origin server (missing during our initial test in November 2019). In StackPath's documentation [30, 33], it is explicitly mentioned that if this feature is enabled, the CDN would not accept the certificates issued from untrusted CAs or that have expired.

CDN provider	Basic certificate validation	Revocation validation	Comments
Cloudflare		✗	Validating the origin is an option
Amazon		✗	
KeyCDN	✗		
CDN77		✗	
BunnyCDN		✗	Validating the origin is an option
Hostry	✗		
Microsoft Azure	✗	✗	Does not validate the CN
Fastly		✗	Validating the origin is an option
Medianova	✗		
ArvanCloud	✗		
CDNSun	✗		
G-Core	✗		
StackPath	✗	✗	Validating the origin is an option and it will only stop the self-signed certificates
Azion	✗		

Table 3. Summary of origin certificate validation of CDNs (back-end communication). Notably, if the CDN does perform the basic certificate validation, we did not present the revocation test result since the attacker can perform the MitM attack in a simpler way (using the lack of basic certificate validation). ✗indicates that the CDN does not properly perform the validation.

Nonetheless, we test StackPath validation (when the origin certificate validation feature was enabled) with CA that we create (not a valid CA) and StackPath accepted the invalid certificate. We perform the same test with an expired certificate, which was also accepted. Apparently, StackPath's certificate validation feature only stops self-signed certificates.

Cloudflare, Amazon, Fastly, CDN77, BunnyCDN do not perform the revocation checks. In particular, Cloudflare mentioned that they would validate the revocation status [17]; however, we found that, even at the most secure level (that validates the origin certificate), Cloudflare accepts a revoked certificate.

Virtual upgrade and weak security parameters. We expected that CDN providers only support strong security parameters and avoid using security parameters that modern browsers do not support. The weak security parameters can be further categorized into broken ciphers (and corresponding hashing algorithms), and weak configurations of key exchange algorithms (explained below). Notably, since these vulnerabilities are in the back-end communication and the users' browser is not involved in this communication, browsers will not warn users about these weak/insecure choices.

- *Broken ciphers.* Amazon, Fastly, Medianova support RC4 as a cipher which is outdated, and modern browsers terminate any RC4 based connection (not as a warning). Amazon also supports the deprecated MD5 hash algorithm. Moreover, Amazon, Fastly, Medianova and Cloudflare support 3DES which is also deemed insecure [11].
- *Insecure key exchange configuration.* TLS uses DH (or other key exchange protocols) to establish the shared secret. Other secrets and keys are driven from the shared secret. 9/14 CDNs support weak DH configuration. DH over 1024-bit prime moduli is vulnerable to the Logjam attack [2, 37]. Nowadays browsers terminate the connection that uses 1024-bit or 2048-bit DH (not as a warning). 8 out of 9 vulnerable CDNs (except CDN77) support 1024-bit DH as well as 2048-bit.

Default settings and options. 5/14 CDNs have a vulnerable default setting. By default, Amazon CloudFront and G-Core use HTTP connections in their back-end instead of HTTPS. BunnyCDN, StackPath, and Cloudflare do not validate the origin server's certificate by default, rather origin certificate validation is an option that can be enabled. In particular, Cloudflare documentation mentions that if the origin has a valid certificate, the website should choose either the option to use HTTPS without validating the certificate (default configuration) or use HTTPS that validates the origin certificate [15]. However, since the origin has a valid certificate, not validating the origin certificate does not align with the best security practices. Concerning the vulnerable options, we found that Amazon CloudFront provides the option to support SSL3 for the origin server (not enabled by default) which is outdated, vulnerable, and not supported by the browsers. Notably, the SSL3 option was not enabled for any of our tests on Amazon CloudFront.

5.2 Front-end connection

To identify the websites that are potentially affected by our back-end findings, we identify the websites that are using vulnerable CDNs. We have extracted the features mentioned in Section 4.2 from 804,013 websites in the Alexa top million that were available to us. In our scan, we could not reach 23,345 websites. The main reason is our 60-second timeout; to accelerate our scanning process, we set a timeout for 60 seconds, and if we cannot extract all the features in less than 60 seconds, we flag the website as unreachable. In addition, some websites block automated tools. From the reachable websites that we analyzed (780,668), we found that approximately 34% use known CDNs. Cloudflare is the most popular CDN, which controls 152,070 websites (approximately 20%), followed by Amazon (58,495 websites). Notably, some CDNs such as Amazon and Google, provide both hosting services and CDN. For Amazon, we can distinguish their CDN from their other services. 10,202 websites use CloudFront (Amazon CDN) and, 48,293 websites use AWS. However, for Google, we cannot distinguish their services from each other. Table 4 shows the number of websites for each CDN. See Table 10 in the Appendix for the websites that use hosting providers.

CDN provider	**# Websites using the CDN**	**CDN provider**	**# Websites using the CDN**
Cloudflare	152070	BunnyCDN	118
ArvanCloud	1041	Azion	78
Google	79282	Medianova	13
Azure	2150	FastCDN	305
Fastly	1789	Chinacache	150
Akamai	14358	Belugacdn	38
StackPath	1077	Baidu	319
Incapsula	6511	Cdnetworks	89
Keycdn	107	Baishancloud	96
Amazon	58495 Total 10202 CDN 48293 AWS	Netlify	2756
OVH	2469	Yottaa	184
Alibaba	949	Aiscaler	21
CDN77	150	Chinanetcenter	84

Table 4. The number of websites using known CDNs in Alexa top 1 million.

6 DISCUSSION

6.1 Practical implications

Since the back-end connection is hidden from the users and cannot be validated by browsers, the users are unaware of the risk that they are taking (in the back-end connection). For example, we configure the Amazon CDN (CloudFront) to use a subdomain that only supports RC4 with MD5 as the origin over TLS 1.0. Afterwards, when we connect to the website through CDN, the browser shows AES-128-GCM over TLS 1.3. These discrepancies in the front-end and back-end communication are hidden from the end-users. Therefore, the end-users are unaware of the insecure back-end connection. Likewise, the CDN can be configured to use HTTPS in the front-end and HTTP in the back-end communication. Similarly, the end-users will be unaware of plaintext transmission of data since the browser displays the lock icon without any warning. The CDNs should expose the back-end vulnerabilities to the end-users to make them aware of the risk that they are taking. Nevertheless, none of the investigated CDNs supports such transparency for all the vulnerabilities (similar to a modern browser).

Except for certificate-related vulnerabilities, other TLS vulnerabilities can be avoided by properly configuring the security options given in a CDN account and the origin server; see Table 5. Note that we cannot determine how many origins indeed use secure configurations as this will require access to their CDN/origin configurations.

Root cause	#Websites	CDN providers	Fixable by origins
Weak DH	4,295	KeyCDN, CDN77, BunnyCDN, Hostry, Fastly, Medianova, AravanCloud, CDNSun, StackPath	Yes
Weak ciphers	164,074	Cloudflare, Amazon, Fastly, Medianova	Yes
Insecure defaults	162,350	Cloudflare, Amazon, G-Core CDN, Azion	Yes
Incomplete certificate validation	168,795	All the investigated CDNs	No

Table 5. The number of websites with possible vulnerable configurations depending on different root causes. Notably, from the 168,795 potentially vulnerable websites (due to certificate related vulnerabilities), 4,466 websites use CDNs that do not perform the basic certificate validation (the rest lack the revocation check).

Note that regardless of the origin/CDN configurations, the websites using these CDNs will be vulnerable to MitM attacks. Also, for all the investigated CDNs, in case of an option regarding the back-end connection (such as origin certificate validation), we use the most secure option available. In other words, we only report the number of websites that are vulnerable even if the website is configured with the most secure configuration. For example, the websites that are using StackPath can configure the CDN to validate the origin server certificate. However, even when this option is enabled, StackPath does not correctly perform the certificate validation (e.g., StackPath does not validate the CA that issued the certificate). Likewise, websites that are using Cloudflare can use the most secure level available for the back-end connection; however, Cloudflare does not validate the revocation status of the origin (although in their documentation, Cloudflare mentioned that they check for revocation [17]). Therefore, regardless of the origin/CDN configurations, the back-end connection remains vulnerable.

In total, we identified 168,795 websites in Alexa top million that are potentially vulnerable to MitM attacks; note that, the investigated CDNs control many more websites. Considering the 9/14 CDNs that do not perform the basic certificate validation, 4466 Alexa top-1M websites are possibly vulnerable to MitM attacks. Compromising these 4466 websites is simpler for the attacker since the CDN that these websites use does not properly perform basic certificate validation. The most popular CDN that does not validate the origin server's certificate is Microsoft Azure. Our result shows that in Alexa top million, 2150 websites use the Azure CDN, which is followed by StackPath (1077 sites) and ArvanCloud (1041 sites). In terms of failing to check revocation status of the origin

certificate, 5 CDNs (Cloudflare, Amazon, Fastly, BunnyCDN, CDN77) control 164,329 websites in Alexa top 1 million. Table 6 shows the 10 websites using CDNs that do not perform the basic certificate validation.

Alexa rank	Website	CDN
413	azure.com	Microsoft Azure
610	ultimate-guitar.com	StackPath
670	theepochtimes.com	StackPath
713	ca.gov	Microsoft Azure
737	mehrnews.com	ArvanCloud
927	magazineluiza.com.br	Azion
981	windowsazure.com	Microsoft Azure
1189	yenisafak.com	Medianova
1218	minecraft.net	Microsoft Azure
1330	isna.ir	ArvanCloud

Table 6. Top 10 websites using CDNs that do not perform the basic certificate validation (in the back-end communication).

To further illustrate how many websites would be compromised due to lack of origin certificate validation and the impact on the current web ecosystem, we assume that there is a powerful attacker that can redirect the back-end communication for any website which uses a vulnerable CDN provider. We then measure the effects based on the CDN and websites that use the CDN. In Figure 3, the number of CDNs is on the horizontal axis, whereas the percentage of websites using vulnerable CDNs (in Alexa top 1 million) is on the vertical axis. Our attacker can choose to redirect any back-end connection for a vulnerable CDN to maximize the number of compromised websites. Our results indicate that a significant number of the websites currently reside on a very small number of CDNs (as a previous study confirmed [12]). By redirecting the back-end communication for websites that use Cloudflare, the attacker compromises more than 80% of the websites that are using vulnerable CDN. Thus, if the attacker can find a way to redirect the back-end communication of the vulnerable CDNs, the result will be catastrophic. Notably, if the CDNs validate the origin certificate, the redirection of the back-end communication will not have any effect.

6.2 Likelihood of MitM attacks

Previous studies showed that the number of IPs that a CDN provider uses for connecting to the origin server is limited [22, 23]. Even when the CDN provider owns a significant number of ingress IPs, the CDN uses a small set of IPs to connect to the origin server. For example, Guo et al. [23] identified over 490,000 ingress IPs for Cloudflare; nevertheless, they only identified 242 egress IPs for the same CDN. Additionally, for most CDNs, the egress IPs are known since the origin server needs to whitelist the egress IPs in its firewall. If egress IPs are not known for the CDN, the attacker can obtain the IPs by doing the same experiment as in [23]. A limited number of known egress IPs will reduce the number of possible network paths that an attacker needs to consider to be an on-path attacker between a CDN and a target origin (or any origin server connected to a vulnerable CDN).

Guo et al. [23] also calculated the occurrence ratio (IP-churning) for each CDN provider during the 24 hours of their experiment. This ratio shows the frequency of repeatedly assigning the same egress IP for a CDN provider. Depending on the CDN provider, the occurrence ratio changes significantly. For example, MaxCDN uses only one egress IP for 96.32% of the requests [23]. Although other CDNs assign the egress IPs more evenly and randomly (less than 10% for each top assigned egress IP), by identifying the top assigned egress IPs for the CDN provider, the attacker could be on the communication's path for a significant amount of the back-end communication.

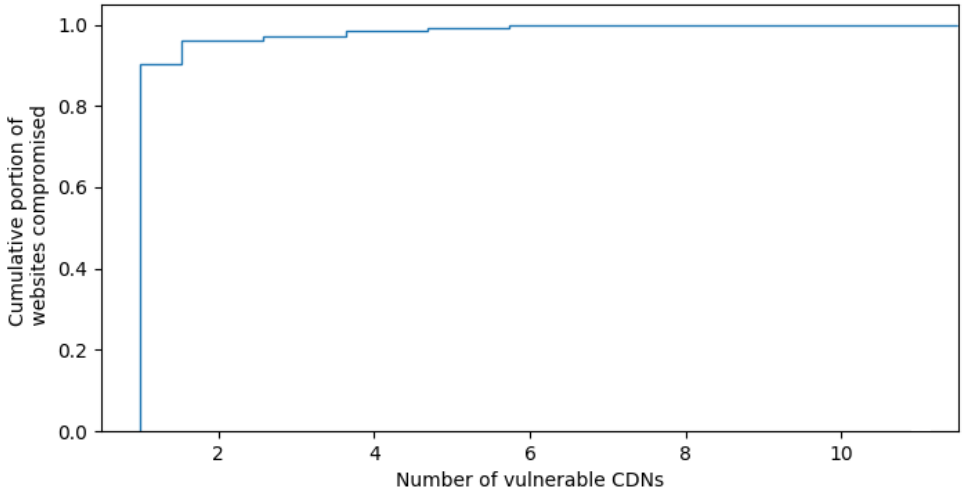

Fig. 3. Vulnerable back-end connection based on the CDN and their corresponding websites. The blue line is Alexa's top 804K websites (that was accessible to us in Alexa top million).

Figure 4 shows an overview of an on-path attacker performing a MitM attack on the most assigned IP path to the origin.

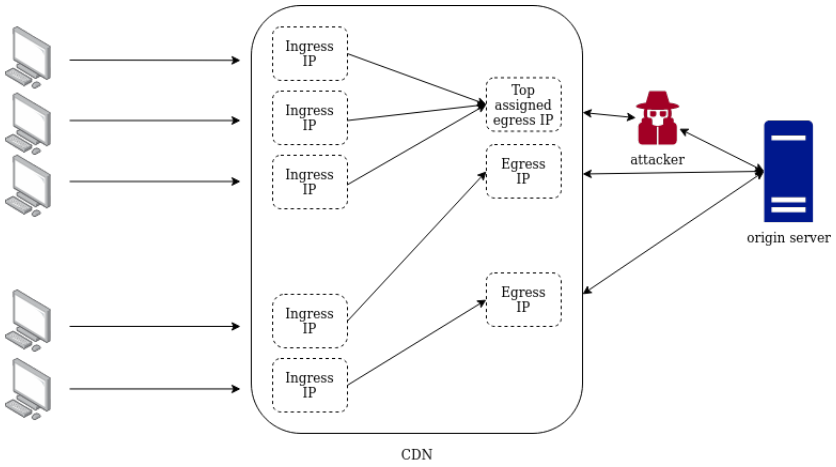

Fig. 4. The on-path attacker on the most assigned IP path to the origin server.

In addition, nowadays, many CDNs support the origin shield feature [13]. The CDN customer can use the origin shield option which is an extra layer of cache between the CDN and the origin. On one hand, the origin shield provides a better cache hit ratio, better network performance and reduce origin load [7]. On the other hand, since the origin shield is the only CDN server that connects to the origin, it can increase the chance of

the on-path attacker. Generally, even a powerful attacker cannot perform a MitM attack or DoS attack on CDN-powered websites, partly due to the geo-distributed nature of high-performance CDN servers. However, since the number of egress IPs are limited (especially when the origin shield feature is enabled), the attacker can perform these attacks in the back-end communication by rerouting the connection to the attacker-controlled server (by compromising an on-path router, proxy, middlebox, etc). For instance, in 2018, attackers stole over $150,000 by redirecting the Amazon DNS (Route53) for MyEtherWallet (an Ethereum wallet). The attacker redirected the traffic to the attacker controlled server using BGP hijacking and impersonated MyEtherWallet using a self-signed certificate [34]. It would be more destructive if the attacker attempted the attack on the back-end communication (while the CDN doesn't verify certificates) since users' browsers would not display any warning.

6.3 Mitigation

Websites are intended to be used by the end-users, and CDNs are an extra caching layer between the end-users and origin. Connection security to an intended HTTPS website is primarily evaluated by the user's browser; users are expected (and nudged in the UI) to stay away from the sites not satisfying modern browsers' security requirements. However, for CDN-powered HTTPS websites, browsers can only validate the front-end connection, unless CDN-to-origin connection security issues are exposed by the CDN. To mitigate this problem, CDNs can adopt the same security policies as modern browsers[4] for the back-end connection; thus, all the vulnerabilities as exposed by our test framework can be easily mitigated. Origin administrators should also avoid using weak cipher-suites, key exchange (on their TLS server), and insecure default settings (on their CDN account). However, they cannot do anything about insecurity introduced by CDN administrators, such as not validating origin certificates properly. Note that, incentives for improving security practices are misaligned here, which requires time and effort from the CDN administrators to implement proper security practices and convince site operators to do their part; however, strict security choices may cause CDNs to lose some of their customers who could switch to a less strict CDN. On the other hand, better security choices directly benefit website users and owners, who have no control over CDNs' choices of important security parameters.

For business purposes, CDNs may want to support all customers, including web/TLS servers that are not at par with modern standards (i.e., legacy customers). CDN operators also stressed this need during our disclosure process. However, we see no reason to support origins with invalid certificates, including the self-signed ones, since certificate-related vulnerabilities do not conflict with legacy customers. For TLS versions and cipher-suites, CDNs can be more accommodating (e.g., allowing 3DES, which is deemed to be weak, but exploiting it requires significant effort). For clearly exploitable configurations (e.g., the use of RC4, SHA-1), users should be displayed an error/warning page, which is actually done by several CDNs in our test set; e.g., Cloudflare generates an error page if the origin uses RC4, but Amazon transparently allows RC4 in the back-end connection. We believe that such error pages will eventually push the origin web admins to adopt current security best-practices. The choice between the warning and error depends on the CDN policies. Nevertheless, the CDN should expose insecure back-end to the end-users. Hiding dangerous TLS misconfigurations and bad practices may only give a false sense of security to end-users, while extending the window of opportunities for attackers. More importantly, allowing a dangerous configuration such as not validating the origin certificate to accommodate a small minority of legacy customers, may actually make other websites that fully follow current security best practices, vulnerable to attacks due to their use of a CDN.

[4]See here for the CA/Browser Forum's baseline requirements: https://cabforum.org/baseline-requirements-documents/

6.4 Limitations

Although we used the result of the front-end measurements to find new CDNs (see Section 4.2), there may be regional CDNs that we missed. Moreover, as mentioned (see Section 5.1), some CDNs only work with enterprise clients and we are not able to perform our tests on their back-end communication.

Considering our front-end scan, while extracting all the features for the websites to identify the CDNs, we realize that there are few discrepancies in our dataset. For example, the website "www.duo-wei.cn" has a CNAME "duowei12.azureedge.net" which points to Azure CDN. After manually analyzing the data in DNS for the domain, we found that "duowei12.azureedge.net" has a CNAME that points to "a1879.dscw14.akamai.net", which points to Akamai. Also, when we perform a reverse DNS query for the IP address, it points to Akamai. Therefore, in our dataset, the extracted features point to different CDNs and we categorize this domain as a website that is using both Azure (based on the first CNAME) and Akamai (when we look at reverse DNS). As a result of this discrepancy, we took a deeper look at Azure CDN. We realize that users in Azure have the option to choose their "pricing tier" from Microsoft, Akamai, and Verizon. We also detect some websites that use multiple CDNs. In this case, we count one of them (the one that we get redirected to). Nevertheless, if one of the multiple CDNs that the website uses is vulnerable, the attacker can perform a MitM attack on the vulnerable CDN. Thus, in our results, for each domain, we first determine if we can categorize the domain with HTTP headers. If not, we move to CNAME, and finally, we check the reverse DNS field. Due to this approach, we would categorize "www.duo-wei.cn" as a website that is using Azure, and not Akamai.

6.5 Responsible disclosure

We performed our evaluation between November 2019 and February 2021. We responsibly notified the vulnerable CDNs during our experiments. We could reach only 9/14 CDN security teams (the general support teams in other CDNs did not respond or connect us with their security teams). Fastly confirmed our findings relating to certificate validation and informed us that they would start working on remediation of the vulnerabilities. However, for the weak cipher suites, according to Fastly, website admins are responsible for the security parameters as site admins can control the acceptable ciphers and DH groups on their origin server. Microsoft Security Response Center (MSRC) confirmed the findings about the lack of Azure certificate validation. However, they mentioned that since exploiting the vulnerability needs the attacker to route the traffic to the attacker-controlled website, it "does not meet the bar for servicing by MSRC". However, the attacker only needs to be on-path (e.g., an ISP between the CDN and origin), and requires no other exploits. StackPath security informed us that they were aware of the vulnerabilities for the origin validation through their support channel. They also told us that their mitigation will be implemented around mid-March 2021. AWS Security team confirmed that the revocation status of the origin certificate is not performed real-time to "maintain latency and availability expectations for CloudFront customers." Regarding the support for MD5 and RC4, they also (similar to Fastly) expect their customers to disable such weak/insecure options at the origin server. Cloudflare (via HackerOne) confirmed the missing revocation check, but this threat is out of scope for them. Medianova escalated our findings to their security team; however, we did not hear anything about their fixes. G-Core did not consider our results as vulnerability and mentioned that "Cloudflare has the same issue" (which is not accurate). Hostry passed our report to their team. However, they mentioned that their CDN service has been relocated and the process is not in their control. Azion mentioned that the issue is "known in the CDN space". They also mentioned that their enterprise customers can use certificate pinning to avoid MitM attacks.

7 CONCLUSION

In this paper, we present a TLS security analysis framework for testing the communication between a CDN provider and a origin server. We unveil that none of the investigated CDNs properly validate the origin server

certificate, possibly subjecting the websites powered by these CDNs to man-in-the-middle attacks in their back-end (CDN-to-Origin) connection. Moreover, we monitor the back-end connection for any weak security parameters including ciphers and the key exchange configurations. We found that 3 CDN providers support ciphers such as RC4. We also found 9 CDNs support insecure DH key exchange that browsers do not support. Furthermore, we realize that 5 CDNs have insecure default settings including Cloudflare (the most popular CDN). We also conducted a measurement study on the front-end connection to measure the number of potentially vulnerable websites. We showed over 20% of Alexa 1M websites are using a vulnerable CDN. Therefore, all the websites that use the vulnerable CDNs (all the investigated CDNs) are potentially vulnerable in their back-end communication. Moreover, the users are oblivious to insecure back-end connections since their browsers are not involved in the back-end communication. Thus, browsers will not present any warning or terminate such weak/insecure connections; this may mislead website administrators who might also test their CDN-protected website through a browser. Our test framework can help identify these TLS/HTTPS weaknesses that are hidden from browsers, and help CDN providers and website administrators to take appropriate steps to mitigate these serious security issues.

REFERENCES

[1] Wireshark Foundation . 2021. *Wireshark · Go Deep.* Retrieved Apr, 2021 from https://www.wireshark.org/

[2] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. 2015. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 5–17. https://doi.org/10.1145/2810103.2813707

[3] Akamai Technologies. 2021. *Security, Cloud Delivery.* Retrieved Apr, 2021 from https://www.akamai.com

[4] Alexa Internet, Inc. 2021. *Keyword Research, Competitor Analysis, & Website Ranking: Alexa.* Retrieved Oct 7,2020 from http://www.alexa.com/

[5] Amazon Web Services, Inc. 2021. *Amazon cloudfront.* Retrieved Apr, 2021 from https://aws.amazon.com/cloudfront/

[6] Amazon Web Services, Inc. 2021. *Amazon Web Services (AWS) - Cloud Computing Services.* Retrieved Apr, 2021 from https://aws.amazon.com/

[7] Amazon Web Services, Inc. 2021. *Using Amazon CloudFront Origin Shield.* Retrieved Apr, 2021 from https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/origin-shield.html

[8] Badssl.com. 2021. *Badssl.com.* Retrieved Apr, 2021 from https://badssl.com

[9] A. Barbir, B. Cain, R. Nair, and O. Spatscheck. 2003. RFC3568: Known Content Network (CN) Request-Routing Mechanisms.

[10] Andrew Betts. 2018. *The headers we don't want.* Retrieved Apr, 2021 from http://www.fastly.com/blog/headers-we-dont-want

[11] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. On the Practical (In-)Security of 64-Bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 456–467. https://doi.org/10.1145/2976749.2978423

[12] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2016. Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 628–640. https://doi.org/10.1145/2976749.2978301

[13] CDN Planet. 2020. *Origin Shield - CDN Planet.* Retrieved July 2020 from https://www.cdnplanet.com/guides/origin-shield

[14] Cisco. 2017. Cisco visual networking index: forecast and methodology, 2017-2022. (2017). https://s3.amazonaws.com/media.mediapost.com/uploads/CiscoForecast.pdf

[15] Cloudflare, Inc. 2020. *End-to-end HTTPS with Cloudflare - Part 1: conceptual overview.* Retrieved April ,2021 from https://support.cloudflare.com/hc/en-us/articles/360024787372-End-to-end-HTTPS-with-Cloudflare-Part-1-conceptual-overview

[16] Cloudflare, Inc. 2021. *Cloudflare Workers®.* Retrieved April ,2021 from https://workers.cloudflare.com/

[17] Cloudflare, Inc. 2021. *Troubleshooting Cloudflare 5XX Errors.* Retrieved Apr, 2021 from https://support.cloudflare.com/hc/en-us/articles/115003011431-Troubleshooting-Cloudflare-5XX-errors#526error

[18] Cloudflare, Inc. 2021. *What is serverless computing?* Retrieved April ,2021 from https://www.cloudflare.com/en-ca/learning/serverless/what-is-serverless/

[19] D. Dolev and A. Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* 29, 2 (1983), 198–208. https://doi.org/10.1109/TIT.1983.1056650
[20] Fastly, Inc. 2018. *Adding or Modifying Headers on HTTP Requests and Responses: Fastly Help Guides.* Retrieved Apr, 2021 from docs.fastly.com/en/guides/adding-or-modifying-headers-on-http-requests-and-responses
[21] Fastly, Inc. 2021. *The edge cloud platform behind the best of the web.* Retrieved Apr, 2021 from http://www.fastly.com
[22] Run Guo, Jianjun Chen, Baojun Liu, Jia Zhang, Chao Zhang, Haixin Duan, Tao Wan, Jian Jiang, Shuang Hao, and Yaoqi Jia. 2018. Abusing CDNs for Fun and Profit: Security Issues in CDNs' Origin Validation. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. 1–10. https://doi.org/10.1109/SRDS.2018.00011
[23] Run Guo, Weizhong Li, Baojun Liu, Shuang Hao, Jia Zhang, Haixin Duan, Kaiwen Shen, Jianjun Chen, and Ying Liu. 2020. CDN Judo: Breaking the CDN DoS Protection with Itself. In *Network and Distributed System Security Symposium.* https://doi.org/10.14722/ndss.2020.24411
[24] Shuai Hao, Yubao Zhang, Haining Wang, and Angelos Stavrou. 2018. End-Users Get Maneuvered: Empirical Analysis of Redirection Hijacking in Content Delivery Networks. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 1129–1145. https://www.usenix.org/conference/usenixsecurity18/presentation/hao
[25] Imperva, Inc. 2021. *"Cyber Security Leader: Imperva, Inc.".* Retrieved Apr, 2021 from https://www.imperva.com
[26] Lin Jin, Shuai Hao, Haining Wang, and Chase Cotton. 2019. Unveil the hidden presence: Characterizing the backend interface of content delivery networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 1–11.
[27] Let's Encrypt. 2020. *Let's Encrypt, Revoking certificates.* Retrieved Apr, 2021 from https://letsencrypt.org/docs/revoking/
[28] Let's Encrypt. 2021. *Free SSL/TLS Certificates.* Retrieved Apr, 2021 from https://letsencrypt.org/
[29] Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, Tao Wan, and Jianping Wu. 2014. When HTTPS Meets CDN: A Case of Authentication in Delegated Service. In *2014 IEEE Symposium on Security and Privacy*. 67–82. https://doi.org/10.1109/SP.2014.12
[30] Tim M. 2021. *StackPath Settings: Origin SSL Validation.* Retrieved Feb 2021 from https://support.stackpath.com/hc/en-us/articles/360037197792-StackPath-Settings-Origin-SSL-Validation
[31] moxie. 2019. *moxie0/Sslsniff.* Retrieved Dec, 2019 from github.com/moxie0/sslsniff.
[32] OpenSSL Software Foundation. 2021. *OpenSSL Foundation, Inc.* Retrieved Apr, 2021 from https://www.openssl.org
[33] Yaniv Parasol. 2020. *Settings: SSL Validation.* Retrieved Feb 2021 from https://support.stackpath.com/hc/en-us/articles/360037362652-Settings-SSL-Validation-
[34] L. Poinsignon. 2018. *BGP leaks and cryptocurrencies.* Retrieved Aug,2020 from https://blog.cloudflare.com/bgp-leaks-and-crypto-currencies
[35] E. Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. *RFC* 8446 (2018), 1–160.
[36] Louis Waked, Mohammad Mannan, and Amr Youssef. 2020. The Sorry State of TLS Security in Enterprise Interception Appliances. *Digital Threats: Research and Practice* 1, 2, Article 8 (May 2020), 26 pages. https://doi.org/10.1145/3372802
[37] weakdh. 2015. *Weak Diffie-Hellman and the Logjam Attack.* Retrieved Apr, 2021 from https://weakdh.org/

A APPENDIX

In this appendix, we provide the features that were used to identify the CDNs and their unique values for each CDN (see Table 7 and Table 8 and Table 9). Moreover, we extract these features for hosting providers and present the popular hosting providers in Table 10.

CDN provider	Reverse DNS
Amazon	*.amazonaws.com *.cloudfront.net
KeyCDN	*.proinity.net
CDN77	*.cdn77.com
BunnyCDN	*.b-cdn.com
Hostry	*.hostry.com
Akamai	*.akamaitechnologies.com
Google	*.1e100.net *.googleusercontent.com
FastCDN tech	*.yourhostingaccount.com
StackPath	*.hwcdn.net

Table 7. Reverse DNS address for CDNs.

CDN provider	HTTP response (H: header)
Akamai	H: "Server: AkamaiGHost"
Cloudflare	H: "Server: Cloudflare"
ArvanCloud	H: "Server: ArvanCloud"
StackPath	H: "X-hw"
Incapsula	H: "X-Iinfo" H: "X-CDN: Incapsula"
KeyCDN	H: "Server: keycdn-engine"
Amazon	H: "Server: CloudFront"
Alibaba	H: "eagleeye-traceid" H: "Alisite-Track"
CDN77	H: "Server: CDN77-Turbo"
BunnyCDN	H: "Server: BunnyCDN-*"
Medianova	H: "Server: MNCDN-*"
ChinaCache	H: "Powered-By-ChinaCache"
Baidu	H: "Server: yunjiasu-nginx"
Baishan cloud	H: "X-Ser"
Netlify	H: "X-NF-Request-ID"
Yottaa	H: "X-Yottaa-Optimizations" H: "X-Yottaa-Metrics"
Beluga CDN	H: "BelugaCDN" H: "X-Beluga-Trace"

Table 8. Unique HTTP headers for CDNs.

CDN provider	CNAME
Cloudflare	cdn.cloudflare.net.
Amazon	cloudfront.net
KeyCDN	kxcdn.com kvcdn.com
CDN77	cdn77.com
BunnyCDN	bunnycdn.com.
Microsoft Azure	azureedge.net azurefd.net. azure.net
Fastly	fastly.net map.fastly.net global.prod.fastly.net
Medianova	mncdn.com
ArvanCloud	arvancdn.com
StackPath	hwcdn.net. stackpathcdn.com.
Azion	ha.azioncdn.net
Akamai	akamai.net akamaiedge.net edgesuite.net akamaized.net
Google	l.google.com.
Incapsula	incapdns.net
Alibaba	alibaba.com alicdn.com
FastCDN tech	fastcdn.com fastweb.com.cn
ChinaCache	chinacache.com cnccgslb.net
Beluga CDN	belugacdn.com. nucdn.net
Cdnetworks	gccdn.net cdnetworks.net cdngc.net
Verzion	alphacdn.net
Baishancloud	baishancloud.com trpcdn.net qingcdn.com bsclink.cn bsgslb.cn
Netlify	netlify.com netlifyglobalcdn.com
Yottaa	yottaa.net
Baidu	yunjiasu-cdn.net

Table 9. Unique CNAME for CDNs.

Hosting provider	# Websites using it	Hosting provider	# Websites using it
myshopify	26208	your-server	10911
sucuri	3284	linode	4375
secureserver	19120	hosting	4498
default-host	3539	web-hosting	5660
beget	7532	unifiedlayer	8739
timeweb	4478	bluehost	4721
webhostbox	3716		

Table 10. Number of websites using popular hosting provider.