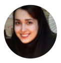
Content Delivery Network Security: A Survey

Article *in* IEEE Communications Surveys & Tutorials · June 2021

DOI: 10.1109/COMST.2021.3093492

Milad Ghaznavi
University of Waterloo
9 PUBLICATIONS **377** CITATIONS
SEE PROFILE


Elaheh Jalalpour
University of Waterloo
5 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Mohammad Salahuddin
University of Waterloo
53 PUBLICATIONS **1,877** CITATIONS
SEE PROFILE

R. Boutaba
University of Waterloo
455 PUBLICATIONS **18,873** CITATIONS
SEE PROFILE

Content Delivery Network Security: A Survey

Milad Ghaznavi*, Elaheh Jalalpour*, Mohammad A. Salahuddin*, Raouf Boutaba*, Daniel Migault†, Stere Preda†
*University of Waterloo, Waterloo, ON, Canada †Ericsson Research, Montreal, QC, Canada
*{eghaznav, ejalalpo, mohammad.salahuddin, rboutaba}@uwaterloo.ca
†{daniel.migault, stere.preda}@ericsson.com

***Abstract*—A content delivery network (CDN) is a distributed infrastructure to deliver digital contents to end users with high performance. CDNs are critical to provide and protect the availability of Internet contents. However, adversaries can not only evade the CDN protection but also weaponize CDN resources to mount more sophisticated attacks.**

In this paper, we provide the first survey on CDN security. We categorize CDN security challenges per CDN infrastructure components, discuss possible countermeasures and their effectiveness, and delineate future research directions. This paper aims to highlight the state of CDN security and identify important research challenges in this area.

I. Introduction

A content delivery network (CDN) consists of geographically distributed servers to cache and efficiently deliver Internet contents, such as HTML pages, images, and videos. CDNs are extremely popular and serve the majority of web traffic. CDNs will deliver 72% of Internet traffic by 2022 [1], [2], and the CDN market value is expected to rise from $11.76 billion in 2019 to $49.61 billion in 2025 [3].

A CDN is also subject to security attacks, such as denial of service, that affect CDN services and end user experience [4], [5]. Protecting CDNs against security attacks is critically important because these attacks cause a CDN to operate poorly and get negative public media coverage [6]–[9], which can diminish the reputation of the CDN provider resulting in significant revenue losses. A CDN must protect content from theft and loss, while preserving content availability by mitigating security attacks.

Unfortunately, adversaries not only compromise the CDN protection, but weaponize its infrastructures against end users and websites that use CDN services. For example, adversaries trick CDN caching mechanisms to generate high traffic volumes against these websites [10]. They even exploit a CDN to publicly cache end user sensitive information and then steal this information from the CDN caches [11].

Considering the critical role of CDNs in content delivery, providing an understanding of the state of CDN security is essential. In this paper, we provide a comprehensive survey on security challenges facing CDNs, along with their attack detection and mitigation approaches. The main contributions of this survey are:

- CDN security challenges: We discuss security vulnerabilities that a CDN face. We categorize CDN security challenges based on the CDN infrastructure components.
- CDN countermeasures: We discuss potential detection and mitigation approaches to counter CDN security challenges. We cover both academic and commercial approaches and argue their effectiveness and limitations.
- Research directions: We discuss opportunities to improve CDN security, as well as security challenges that confront future CDNs. Specifically, we describe the research challenges due to user-generated content, and discuss opportunities using software defined security and collaborative security among parties involved in content delivery.

To the best of our knowledge, this survey is the first to focus on CDN security. Other surveys in the literature focus on CDN architecture and infrastructure [12]–[16], collaboration in content delivery [17], [18], CDN performance [19], [20], and CDN operational algorithms [21], [22]. These surveys are oblivious to the inherent security challenges in CDNs.

We proceed as follows. In Section II, we describe CDN infrastructure and the content delivery process, and categorize CDN security challenges. This categorization facilitates the discussion of security challenges in the subsequent sections. Section III, Section IV, and Section V are dedicated to security challenges in each CDN component, i.e., edge server, request routing, and origin server, respectively. Finally, we highlight current vulnerabilities, discuss opportunities and future research directions in Section VI, and conclude this survey in Section VII.

II. Content Delivery Networks: A Primer

A CDN aims at enhancing the quality of experience in delivering digital contents to end users, while utilizing network resources more efficiently. A CDN caches contents in locations nearby end users, routes content requests to these locations, and transfers the contents to the end users [13], [23], [24].

CDN providers, content owners, and end users are the main parties involved in the content delivery process. A CDN provider manages and operates the CDN infrastructure. A content owner owns digital contents and is a customer of a CDN. Content owners delegate the delivery of their contents to CDNs. An end user consumes content using its digital devices, such as TVs, tablets, and smart phones.

A. *Content Delivery Process Overview*

A content owner places digital contents in origin servers. The CDN distributes and replicates content from origin servers into numerous edge servers (e.g., hundreds to thousands of servers). These edge servers are distributed across the Internet to provide high-capacity storage capability to cache contents in vicinity of end users [25].

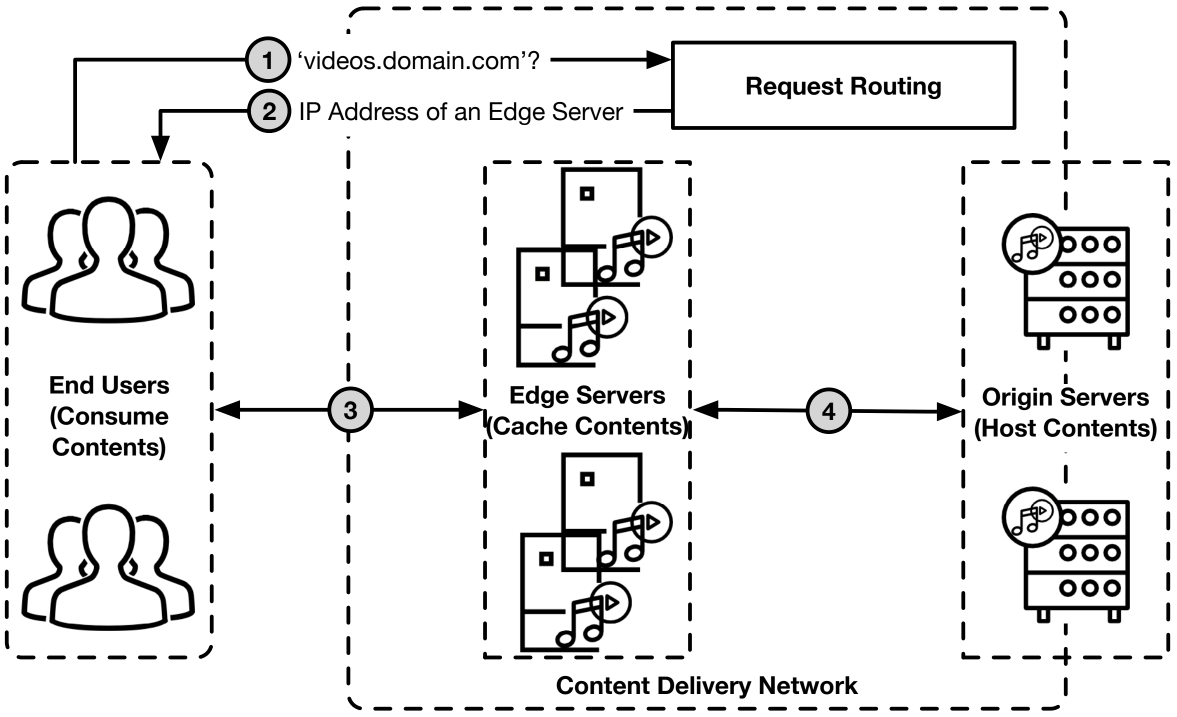

1 'videos.domain.com'?
2 IP Address of an Edge Server
Request Routing
End Users (Consume Contents)
3
Edge Servers (Cache Contents)
4
Origin Servers (Host Contents)
Content Delivery Network

Figure 1: Content Delivery Network Infrastructure. The request routing component redirects end user requests to edge servers that cache and serve contents. On cache misses, edge servers forward requests to origin servers that host contents. Origin servers are operated by either CDN providers or content owners.

Figure 1 shows content delivery through a CDN. A CDN receives and serves end user requests on behalf of distant content owners (step 1). Using a request routing mechanism, the CDN selects and redirects a legitimate request to one of its edge servers (step 2). The selected edge server performs an admission control, and if the request is accepted, the edge server delivers the content from its cache (step 3) [26]. On cache misses, an edge server retrieves and caches the content from either another edge server or an origin server (step 4).

B. Content Delivery Network Infrastructure

All CDN components, including edge servers, request routing, and origin servers, are involved in content delivery [25], [27], [28]. Next we discuss each component in more details.

1) Edge servers: Edge servers act as a protective shield to cache contents to keep down the load on origin servers. Edge servers are strategically placed in CDN points of presence, at the edge nearby end users, e.g., one or two hops away. Internet exchange points, internet service provider networks, and data centers are examples of points of presence. A point of presence can contain several edge servers.

Edge servers can also cooperate by serving each other's content requests [26]. This collaboration creates a second caching layer as the latter edge server caches responses from origin servers. Moreover, edge servers use a hierarchical caching based on popularity. They store popular contents in memory, while keeping others on disks [29], [30]. A replacement algorithm (e.g., least recently used or least frequently used) manages contents remaining in the cache [31].

There are three main models to distribute contents from origin servers to edge servers [13], [22], [28], [32]. First, a push model distributes content if its request is anticipated at an edge server [33], [34]. Second, in a pull model, an edge server fetches contents upon receiving end user requests [35], [36]. Finally, a hybrid push-pull model dynamically adapts to changing end user requests by pushing some contents proactively and pulling others reactively [37].

Edge servers run web cache proxies to implement caching. Cache proxies can easily store static contents to serve future requests, because the static contents do not change over time. In contrast, caching dynamic contents is more complicated. For example, Edge Side Includes [38] is a markup language to specify dynamic parts of a web content, allowing a cache proxy to retrieve only dynamic parts (e.g., the latest news in a company webpage) while caching static parts (e.g., the image of a company's brand). Edge servers can also run scripts to generate a set of dynamic contents based on events and inputs, such as time of day, device types, and end user locations.

2) Request routing: The request routing component monitors network condition, load on edge servers, and distributes requests among edge servers based on monitored data [25], [32], [39]–[41]. Routing a request is based on a variety of metrics, such as proximity to end users, logical distance (e.g., the number of hops), latency, jitter, and server load [27].

Request routing techniques are mostly classified into domain name system (DNS)-based routing, anycast routing, application layer routing, and combinations of these [40], [42]. DNS-based request routing utilizes the existing ubiquitous

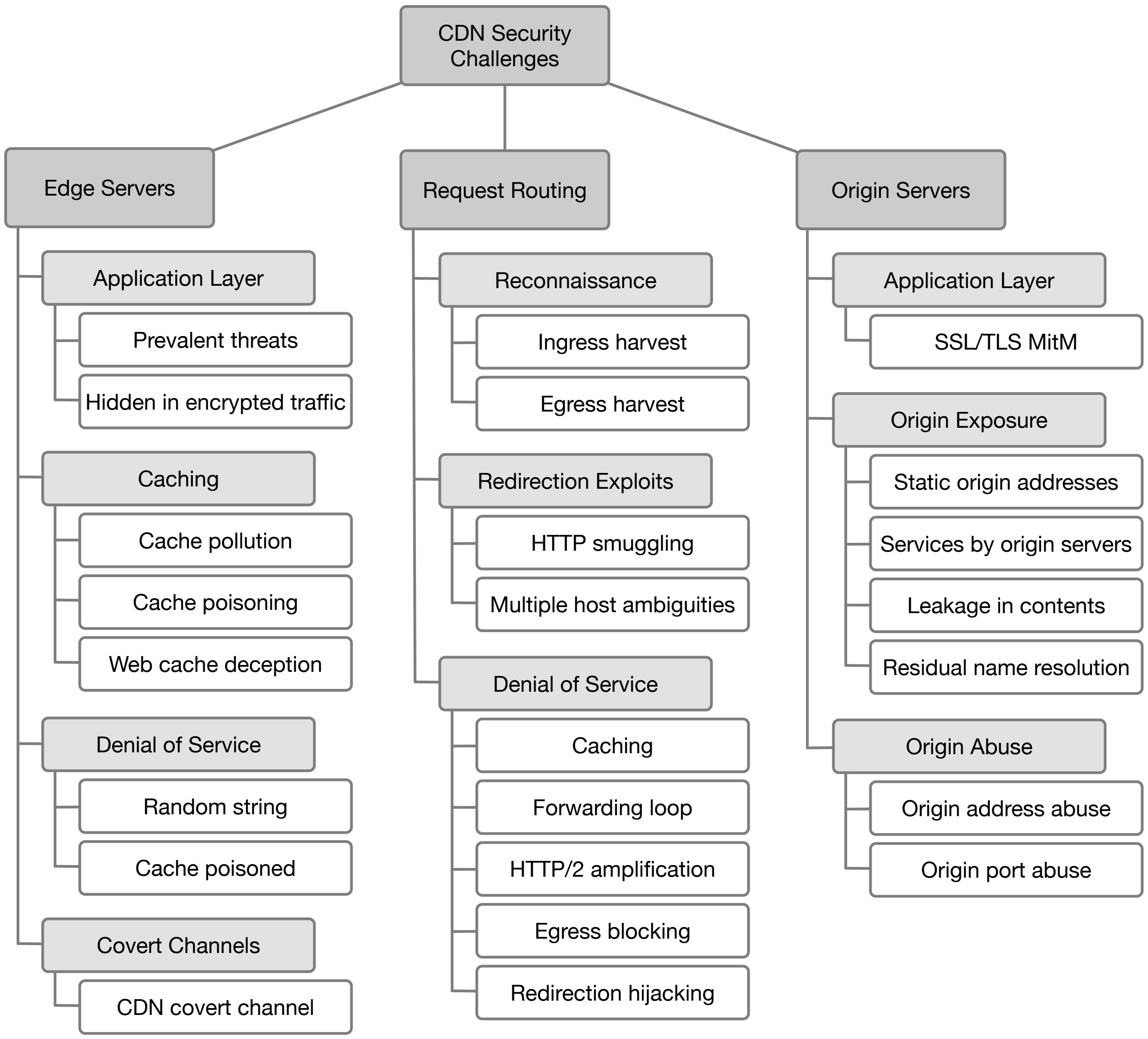


Figure 2: Categorization of CDN security challenges.

DNS services. CDNs commonly run their own name servers and maintain dynamic DNS records to balance the load among edge servers [42]–[44].

With a DNS-based routing, an end user sends a DNS request for a content's domain name (e.g., 'youtube.com') to its local name server. This name server forwards the request to a CDN's authoritative name server that responds with the IP address of an edge server. CDN authoritative name servers also perform load balancing, e.g., they resolve requests for the same domain name to the IP addresses of different edge servers.

Anycast simplifies request routing by delegating routing to the Internet. In anycast routing, the CDN authoritative name server also returns an IP address. The difference with DNS-based routing is that multiple edge servers use the same IP address, and the border gateway protocol (BGP) routes a request to its nearest edge server. Specifically, multiple edge servers in a particular geographical location announce the same IP address, and BGP selects the shortest autonomous system path to reach the nearest edge server.

For example, Open Connect [45], Netflix's CDN, uses BGP routing. Open Connect contains a suite of purpose-built physical edge servers, called Open Connect Appliances (OCAs) to deliver video contents. Open Connect deploys OCAs in ISP networks and internet exchange points, and uses Multi Exit Discriminator (MED) to prioritize OCAs over alternative BGP paths. This allows Netflix to localize its traffic as close as possible to its end users, minimizing network and geographical distances for content delivery.

An application layer request routing is based on HTTP protocol and can route requests in the granularity of content objects. Using URL rewriting, website URLs are substituted with CDN subdomains that are resolved to CDN edge servers. For example, the owner of a domain name "www.great.com" publishes associated contents with "www.great.com.cdn.net" belonging to a CDN.

CDNs may combine the above techniques to improve the accuracy and performance of request routing. For example, YouTube [46], Google [47], Akamai [48], and Microsoft use DNS-based and application-layer redirection. Bing and LinkedIn combine DNS-based and anycast routing [49], [50].

3) Origin servers: Origin servers host original contents, e.g., webpages of a website. An origin server is managed by either a content owner or a CDN provider [25], [51]. A content owner may place contents either on-site or on a cloud [52]. NOKIA Velocix [51] is a CDN that operates origin servers that are optimized for streaming, download, and caching.

Origin servers normally run web servers, e.g., NGINX and Microsoft IIS, to host and serve web contents. Even with using CDN services, origin servers still serve most requests for dynamic web contents (e.g., online social network and personal web pages). This is because dynamic contents are generated per request and based on data that a content owner cannot share with a CDN (e.g., end user information).

C. Comparison to Information Centric Networks

Information Centric Networking (ICN) [53] shifts networking from a point-to-point communication to a content centric paradigm. In this paradigm, network nodes (e.g., routers) cache contents using their names. Moreover, the network provides two primitives, similar to those of a publish/subscribe system. A content owner uses a 'publish' primitive to make contents available, and an end user requests contents using a 'subscribe' primitive [53].

ICNs and CDNs originate from different networking paradigms, although they have similarities in content caching. The ICN paradigm eliminates content addresses, where contents are published and subscribed to only by name. Furthermore, network nodes cache and deliver contents instead of edge servers. Consuming a content is no longer an end-to-end connection with a server (e.g., 'cdn.youtube.com/best-video-ever'), but rather the delivery of a named content (e.g., 'best-video-ever').

ICNs deal with unique security challenges relevant to contents rather than communication channels, and these challenges are not necessarily applicable to CDNs [54]. For example, caching contents in network nodes makes an ICN vulnerable to bogus announcements and time analysis attacks [54]. Using bogus announcements, an adversary can announce many content updates that leads to erroneous content state. Time analysis allows an adversary to violate end users privacy by deducing their request patterns.

D. Security Challenges Organization

CDNs are massive networking infrastructure across the Internet with a multi-billion dollar market [1]–[3]. This motivates us to survey CDN security challenges.

Figure 2 categorizes CDN security challenges, i.e., security attacks and vulnerabilities. We categorize the security challenges per CDN component, as discussed in Section II-B. We also use a subcategory similar to that of [55], which are representative of known attacks. We do not claim that our categorization *exclusively* divides security challenges; a security challenge and its countermeasures are possibly relevant to multiple CDN components.

Table I and Table II facilitate reading by listing the CDN terminology and acronyms used in this survey. In Table I, the first column shows the terms used throughout this survey, and the second column lists other equivalent terms from the literature. Table II provides the expanded forms of the acronyms used in this survey.

III. EDGE SERVERS SECURITY CHALLENGES

In this section, we review the security challenges and countermeasures related to edge servers. CDNs serve web requests making edge servers vulnerable to application layer attacks (cf., Section III-A). CDNs are also vulnerable to caching threats (cf., Section III-B). Moreover, adversaries send malicious requests that exploit vulnerabilities of edge servers to launch denial of service attacks (cf., Section III-C). Edge servers are also exploited as a covert communication channel to transmit sensitive information (cf., Section III-D).

A. Application Layer

Most CDNs serve web traffic, and cache web contents on edge servers. This makes edge servers prone to web application layer attacks [56]–[61].

1) Prevalent threats: Web application threats include SQL injection, cross site scripting, file inclusion, remote command execution, illegal resource access, dictionary attacks, bandwidth abuse, to name a few. They can lead to varying implications for victim organizations, such as data leakage (e.g., data exfiltration using SQL injection and cross site scripting), fraud or business malfunctioning (e.g., screen scrapping, spamming and fake accounts), and disruption due to denial of service.

Static or dynamic analyses of web application's source code can reveal security vulnerabilities [62], [63]. However, these analyses do not deter all possible threats. CoDeen [60] employs limiting and blacklisting particular requests to counter threats. It restricts HTTP POST requests due to their inherent security risks. However, this limits the flexibility of a CDN to support HTTP requests. CoDeen also enforces higher restriction on requests that match specific threat signatures. For example, it bans an end user for a day for conducting frequent HTTP login attempts. Furthermore, CoDeeN blacklists end users that are suspicious of launching vulnerability tests and dictionary attacks.

Web application firewalls (WAFs) can also defend edge servers against common web application attacks, extensively used by commercial CDNs [64]–[68]. WAFs are installed on or in front of edge servers. They perform deep inspection of web requests and responses to detect and block these attacks [69]. They configure WAFs to filter traffic based on the open web application security project (OWASP) identified threats [70], CDN specific services, and security requirements of content owners. For example, the Alibaba WAF [65] protects web applications against top ten security risks of OWASP [71]. Other WAFs, such as the Incapsula WAF [66], are designed for compliance with standards, including the "payment card industry data security standard" or "health insurance portability and accountability act standards."

Term	Equivalent terms	Description
Content	Digital content	Digital assets, e.g., web pages and images
End user	User, consumer, client, content consumer, content visitor	The party consuming content
Content Delivery Network (CDN)	Content distribution system, content distribution network	An infrastructure of servers delivering content to end users
Content owner	Content provider, CDN customer	The party (e.g., Netflix, Amazon, YouTube) owning content
CDN provider	Content distribution service provider, content service provider, CDN service provider	The party owning a CDN infrastructure (e.g., Akamai, Limelight)
Origin server	Backend server, content library server, original server, origin cache, origin	Servers containing the original content
Edge server	Surrogate, cache, proxy, replica, edge node, edge cache, content agent	The CDN servers in the proximity of end users
Content distribution	Content outsourcing, distribution system, content management subsystem	The CDN component replicating/distributing content from origin servers to edge servers
Request routing	Request redirection, content redirection, content routing, traffic routing, mapping system, rerouting	The CDN component assigning content requests to edge servers
Request	Content request	An end user's request to retrieve a content

Table I: Terminology. We use the terms listed in the first column throughout the survey. We also list the equivalent terms from the literature.

The detection of application layer threats require deep packet inspection (DPI), which comes with a high resource overhead. Some CDNs [64], [67], [68] deploy a WAF per edge server, sharing the same resources used by content delivery services. Furthermore, DPI comes with privacy concerns. If a CDN has sufficient resources to perform DPI on traffic, it can potentially surveil and discriminate traffic to its edge servers, which goes against net neutrality. Moreover, they can share information with marketing firms [72]–[74].

2) Hidden in encrypted traffic: Attackers can camouflage application layer attacks inside encrypted traffic to bypass the CDN security mechanisms and edge servers, and directly target origin servers [74]. Encrypted traffic is becoming a de facto in today's CDN ecosystem. This is excerbated by the adoption of next-generation web protocols, e.g., HTTP/2. Traffic encryption is mostly achieved via Secure Sockets Layer (SSL)/Transport Layer Security (TLS) protocol. Using SSL/TLS, end users establish an end-to-end secure connection to origin servers to access contents provided by content owners [75]. Without private keys from content owners, a CDN has limited ability to analyze encrypted traffic between end users and origin servers. In this case, a CDN must redirect traffic towards origin servers, without inspecting traffic payload.

A CDN can still analyze non-encrypted portions of encrypted traffic (e.g., the negotiation phase of SSL/TLS communication) and perform behavioral analysis on traffic statistics (e.g., flow-based features) [76]. Intrusion detection systems can detect attacks using these statistical features, and enforce mitigation workflows. For example, Amoli and Hamalainen [77] use unsupervised machine learning on real-time metrics, such as number of network flows, packets, and bytes, to detect scanning and denial of service attacks.

DPI for detection of application layer threats entail inspecting either decrypted or encrypted traffic [78]. In both scenarios, content owners must trust a CDN and share their private keys to enable traffic inspection at the edge. However, with access to the private keys, a CDN can essentially become a man-in-the-middle, making content owners vulnerable to eavesdropping, and even impersonation. We discuss this dilemma further in Section V-A.

It is difficult for a CDN to provide both effective content delivery and security services. Intercepting traffic for DPI sacrifices a degree of privacy and trust. Although there are approaches, such as fully homomorphic or functional encryption [79], [80] that allow inspection of encrypted traffic without decryption, they are impractical due to their prohibitively low performance. Indeed, it is a challenge for a CDN to provide effective security services, while preserving the privacy of content owners and end users.

B. Caching Threats

Content popularity is an important characteristic that allows for its efficient replacement on edge servers. The content

Acronym	Expanded form
BGP	Border Gateway Protocol
CCPA	California Consumer Privacy Act
CDN	Content Delivery Network
DNS	Domain Name System
DONA	Data Oriented Network Architecture
DoS	Denial of Service
DPI	Deep Packet Inspection
FTP	File Transfer Protocol
GDPR	General Data Privacy Regulation
HPACK	Header Compression for HTTP/2
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICN	Information Centric Networking
IP	Internet Protocol
ISP	Internet Service Provider
MitM	Man-in-the-Middle
NDN	Named Data Networking
NFV	Network Functions Virtualization
OWASP	Open Web Application Security Project
PII	Personally Identifiable Information
QoE	Quality of Experience
QoP	Quality of Protection
QoS	Quality of Service
SDN	Software Defined Networking
SGX	Software Guard Extension
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator
WAF	Web Application Firewall

Table II: Acronyms and their expanded forms

popularity follows a Zipf distribution with long-tail, such that there is a small set of popular contents, while most contents are seldom requested [22], [55]. Edge servers cache popular content to take advantage of the locality reference principle, i.e., a recently requested content is likely to be requested again.

Attackers target the locality of reference principle to degrade the caching performance with unpopular content [81], [82]. They also target the cache integrity of edge servers by conducting cache poisoning that replaces a legitimate cached response with a spoofed content. Until cache expiry, subsequent requests for the same content will receive the spoofed content instead of the original one [83]. Using similar techniques to cache poisoning, adversaries steal sensitive information by tricking edge servers to cache end user information [11], [84]–[86].

1) Cache pollution: Edge servers are vulnerable to cache pollution [10], [87], which results in degraded caching service with frequent cache misses. To successfully pollute an edge server's cache, an attacker must generate requests for unpopular contents, which is comparable in numbers to the requests from legitimate end users for popular contents. These attacks can deteriorate the overall network performance without flooding the network, while impacting legitimate end users and edge servers. Moreover, the detection of cache pollution is challenging, because unpopular contents that occupy the cache are inherently non-malicious.

Attackers target cache locality to degrade the CDN's quality of service (QoS). Locality refers to the tendency of an edge server to serve the same content. High locality allows an edge server to cache popular contents and serve several requests for the same content from its local cache.

Attackers affect the cache locality using two approaches. First, using a false locality strategy, an attacker continuously generates requests for the *same set* of unpopular contents, which deteriorates the cache hit ratio for popular contents. These attacks can quickly repopulate the cache with unpopular contents. Second, an attacker frequently requests a new set of unpopular contents, consequently degrading cache efficiency and impacting legitimate end users.

Cache pollution attacks entail content access patterns that are in stark difference to legitimate requests. Thus, countermeasures capitalize on metrics capturing this difference, and perform threshold based analyses to counter these attacks.

In a false locality attack, malicious end users request the same unpopular contents frequently, while a legitimate end user rarely requests the same content repeatedly in a short period of time [88]. This difference has been leveraged to develop two detection approaches, namely attacker-based detection and object-based detection [81]. In the attacker-based detection, if the number and percentage of repeated requests exceed predefined thresholds, the end user is classified as malicious. In the object-based detection, if the number of requests for a content is relatively larger compared to the number of end users requesting it, the content is deemed a false popular content and evicted from the cache. All these mitigation approaches require adjusting appropriate detection thresholds. However, finding threshold values is non-trivial.

Manivel et al. [89] detect malicious end users by tracking their *recent* history of cache hits and misses. To track the activities of end users, a mapping between end users and their content requests is maintained. If the recent cache hit ratio of an end user is below a predefined threshold, the end user is classified as an attacker. This approach emphasizes on recent behavior of an end user and avoids a permanent classification. Indeed, an end user with poor locality may have a low hit ratio, while being legitimate. Therefore, with a classification that changes over time, an end user is only subjected to a temporary misclassification.

Park et al. [90] consider the randomness of requests to identify a locality disruption attack, i.e., the lower the randomness, the higher the probability of locality disruption. This approach maintains a matrix of request statistics. An attack is detected, if the entropy of the matrix falls below a predefined threshold.

In contrast, Dang et al. [88] propose a two step approach to detect both false locality and locality disruption attacks. Firstly, the authors leverage attacker-based or object-based detection approaches to detect false locality. This allows for excluding unpopular contents from the following step.

Secondly, the locality disruption attacks are detected separately using the periodic average life time metric.

2) Cache poisoning: Kettle [91] discover an attack that misuses "unkeyed inputs" and the concept of "cache keys" to poison web caches at edge servers.

Web proxies use key value stores to track their caches. To associate a HTTP request to cached responses, they combine the values of certain HTTP headers (e.g., `HOST` and `GET`) in the request, as a cache key. However, web proxies exclude some HTTP headers (e.g., `User-Agent` and `Cookie`) from cache keys, as they can be very specific to a particular request, making them impractical for caching. These are called unkeyed inputs, which are amendable to web cache poisoning. An adversary can craft a HTTP request with arbitrary values (potentially malicious) for these headers, while the request cache key remains general enough to match future requests.

Figure 3 depicts a cache poisoning attack. First, an adversary crafts and sends a HTTP request, including harmful values for unkeyed inputs (e.g., `X-Host: badguy.com`). Second, an edge server forwards this request to an origin server. Third, the origin server replies with a poisoned response including the harmful values from the HTTP request. Specifically, origin web servers may include some HTTP request header values into their responses. An origin web server generates a poisoned response that includes the harmful values of the unkeyed inputs (e.g., an HTML page including `<script src="badguy.com/foo.js"></script>`). Fourth, the edge server caches this response for legitimate HTTP requests. Fifth, an end user sends a request that maps to the poisoned cache record. Finally, the edge server replies with a response from its poisoned cache.

This attack can be mitigated at the origin and the edge servers. Origin servers can prevent it by excluding the values of HTTP headers in responses. However, this limits the flexibility of web applications that make use of HTTP headers.

Edge servers can alleviate the impact of this attack by including more HTTP headers in cache keys. For instance, Cloudflare includes further HTTP headers, such as `X-Host`, `X-Forwarded-Host`, and `X-Forwarded-Scheme`, as part of their cache keys [92]. However, this can impact the cache performance, as including more HTTP headers increases the cache key space.

3) Web cache deception: This attack aims at stealing end user's sensitive information. An attacker tricks an end user to request a dynamic content containing sensitive information (e.g., the details of a private payment account), which is served from the origin server and cached in an edge server. The attacker later accesses this information directly from the edge server's cache [11], [84]–[86]. Akamai, Cloudflare, CloudFront, and Fastly were reported vulnerable to this attack [11].

This attack involved five steps. First, an adversary deceives an end user into requesting a non-existing URL that ends with a static content, such as 'http://www.bank.com/profile/foo.jpg' (i.e., 'foo.jpg' is a non-existing static content). Second, an edge server receives and redirects this request to an origin server. Third, the origin server generates and responds with a dynamic content as follows. The web server on the origin can be configured to be flexible in serving different paths and serves this non-existing URL by invoking a script located at 'http://www.bank.com/profile'. The script generates dynamic contents with sensitive end user information (e.g., an end user's profile information). Forth, the edge server caches the response, because the request is apparently for a static content (e.g., a 'jpg' image). Fifth, the adversary can then request the same URL, and the edge server responds with the cached content.

To overcome this attack, origin servers can be configured to appropriately respond to unexpected URLs, or force edge servers to exclude sensitive contents from caching [86]. For example, an origin web server responds with HTTP 404 for non-existing URLs. An origin server can also set the `Cache-Control` header to `No-Cache`, ensuring the edge server does not cache responses with sensitive information.

The edge servers can also avoid caching sensitive contents, by matching the response type with the requested URLs. For example, an edge server should not cache an HTML response for an an image request with a 'jpg' extension [85], [93].

C. Denial of Service

In a denial of service attack, adversaries aim to prevent legitimate end users from accessing CDN services [94]. A distributed denial of service attack employs multiple attacking entities to achieve the same goal. For example, an adversary can flood an edge server, consume its resources and prevent legitimate end users from accessing contents. These attacks disrupt CDN services causing CDN businesses to lose considerable revenue.

By taking advantage of the vulnerability of edge servers in serving dynamic contents, adversaries use a CDN to amplify their denial of service attacks against origin servers [10]. Moreover, they exploit vulnerabilities of HTTP request headers to cache poisoned contents at edge servers making original contents unavailable to legitimate end users [95]–[97].

1) Random string denial of service: Typically, edge servers forward requests for dynamic contents to origin servers, since serving these requests from the local cache is non-trivial. However, forwarding these requests to origin servers introduces a vulnerability that attackers can exploit to launch amplification denial of service attacks.

Triukose et al. [10] craft random string URLs to flood origin servers and consume the resources of a CDN. Appending a random string to a user request URL (e.g., appending '?q=rand' to 'http://www.domain.com/image.jpg') forces a cache miss on the edge server, even when the appended string might not correspond to actual dynamic contents. Consequently, over two decoupled TCP connections, the edge server fetches and delivers the missed content (e.g., 'image.jpg') from the origin server.

The content is fetched from an origin server to the edge server over the first connection, while the content is delivered from the edge server to the attacker over the second connection. The first connection is high throughput, while the throughput of the second connection is low (i.e., adjusted to

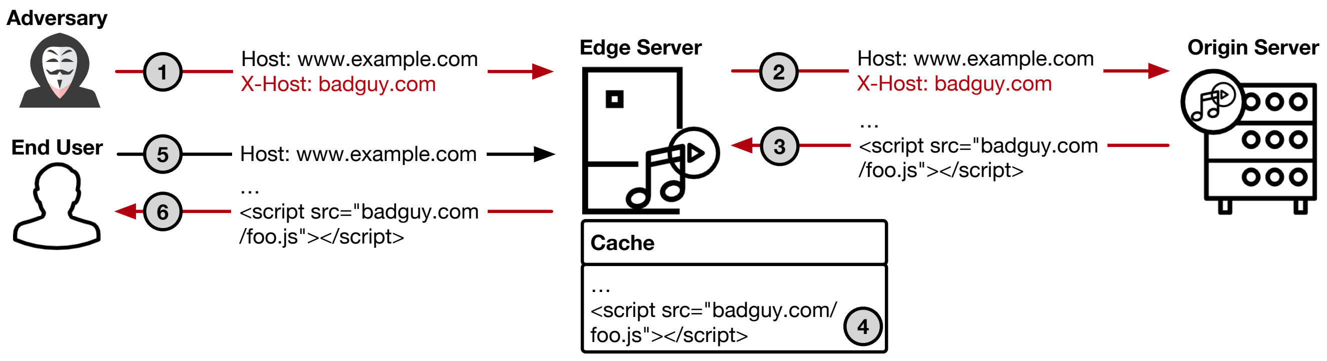


Figure 3: Cache poisoning attack. Black and red arrows show legitimate and malicious HTTP messages, respectively.

the attacker's connection). Since the attack consumes an order of magnitude more bandwidth at the origin server connection, the attacker can force edge servers to amplify the traffic to the origin servers. Although the main target of this attack are the origin servers, an attacker can also pollute caches and reduce the performance of the edge servers.

Triukose et al. [10] explore a list of mitigation techniques. In a "no string attached" defense, a dynamic content is served only by origin servers, while edge servers only serve static contents. In this case, requests for dynamic contents can either be dropped by edge servers or origin servers (i.e., requests for dynamic contents forwarded by an edge server). However, this decreases the effectiveness and flexibility of a CDN in serving dynamic requests.

Edge servers and origin servers can also collaborate to serve dynamic content. The edge servers append certain information (e.g., IP addresses of end users) to requests that are forwarded to the origin servers. The origin servers can leverage this information for threat detection and mitigation. For example, the origin servers can limit the rate of requests coming from the same IP address. This approach is effective if the CDN also manages the origin servers. Otherwise, content owners must protect their origin servers against the random string attack, without sufficient support from the CDN. Alternatively, a CDN can leverage anomaly detection mechanisms to detect abnormal rate of requests to the origin servers. However, this comes at a high performance cost.

The attack's amplification factor depends on the throughput difference between the two TCP connections. Therefore, controlling the throughput of these connections can decrease the impact of an amplification attack. A CDN can apply connection throttling, where content transfer between origin servers and edge servers is slowed down, based on the delivery progress between an edge server and an end user. The CDN can also leverage abort forwarding to stop transferring contents between an origin server and an edge server, as soon as an end user closes its connection.

2) Cache poisoned denial of service: Using a web cache poisoning attack, adversaries can cause an edge server to cache and serve error pages instead of original contents [95]–[97]. This makes contents unavailable to upcoming legitimate requests. Akamai, Azure, CDN77, Cloudflare, CloudFront, and Fastly were reported vulnerable to this attack [95].

Cache poisoning attack has four steps. First, an adversary crafts and sends an HTTP request including a malicious header that targets a victim content. Second, an edge server forwards the request to an origin server, while the malicious header remains undetected at the edge server. Third, processing the malicious header at the origin server provokes an error, and the origin server generates and replies with an error page to the edge server. Fourth, the edge server caches this error page instead of the original content. The edge server replies with this error page to recurring requests for the victim content.

There are three headers in HTTP requests that can be exploited for this attack [95], [96]. Using an oversized HTTP header, an adversary crafts a request header, which although supported by an edge server, is too large for the origin server to handle due to its configuration.

Using HTTP meta character, an adversary crafts a request header with a harmful meta character, e.g., `\n`, `\r`, and `\a`. An edge server forwards this request to an origin server, where the meta character causes an error.

Using HTTP method override, an adversary crafts `PUT` and `DELETE` requests to bypass edge servers and cause an error at an origin server. Web proxies and firewalls at edge servers commonly support only HTTP `GET` and `POST` methods, and block requests with `DELETE` and `PUT`. Some web frameworks at origin servers, e.g., Play Framework 1, circumvent this limitation by supporting override headers, such as `X-HTTP-Method-Override`. These headers enable tunnelling blocked HTTP methods. At an origin server, the web framework replaces the HTTP method of a request with the value of the override header. To conduct this attack, an adversary sends a `GET` request with an `X-HTTP-Method-Override` header set to `POST`. An edge server interprets this request as a benign `GET`, while an origin server interprets it as a `POST` after overriding the HTTP method. If the web application logic does not implement `POST` for the requested content, the origin server returns the HTTP 404 error message. Based on RFC 7231 [98], the edge server is allowed to cache this error page to serve recurring requests.

Excluding error pages from caching can completely prevent this attack. However, this can impact the performance and scalability of CDNs. An origin server can also generate response messages with `Cache-Control: no-store`, and edge servers must honor this header. Another approach is to disable caching at edge servers. Akamai, CDN77, CloudFront, CloudFront, and Fastly can be configured with this option.

WAFs deployed in front of web proxies at edge servers can also filter malicious request headers. However, WAFs in front of origin servers do not mitigate this attack, as they can still generate error pages that will be cached at edge servers. Furthermore, DPI at WAFs can also impact CDN performance.

D. Covert Channels

Covert channels allow adversaries to communicate information, while the communication remains undetected. This allows adversaries to bypass security mechanisms and misuse a CDN as a medium to secretly transmit and steal sensitive information [100].

The CDN infrastructure can be used as a covert channel. Public access to CDN services enable adversaries to recruit and exploit the communication between edge servers and origin servers to encode and transmit secret messages.

CDN covert channel: Wang et al. [99] exploit CDN edge servers and origin servers to create a covert communication channel between malicious end users and a malicious content owner using CDN services. As a proof of concept, the authors conduct the attack on CloudFront.

Figure 4 illustrates the CDN covert channel unfolding in five steps as follow. First, a malicious end user or malicious content owner collects the IP addresses of edge servers (we will discuss vulnerabilities that lead to harvesting IP addresses of edge servers in Section IV-A). Second, the malicious end user or the malicious content owner devise an information encoding scheme. Third, the collected IP addresses and the encoding scheme are broadcast to malicious end users and the malicious content owner. Fourth, the malicious end user selects an edge server and encodes a secret message into a series of penetration requests, i.e., content requests forcing an edge server to fetch from the origin server (e.g., URL 'http://malicious.org?q=dnar' ending with a random string `dnar` [10]). Finally, the malicious origin server receives and decodes the requests to reconstruct the messages.

The information encoding scheme used to encode and decode messages is based on the sending edge server. For example, as shown in Figure 4, the same request received from edge servers `1` and `2` are respectively decoded to alphabets `i` and `j` in the malicious origin server.

The information scheme used in the fifth and sixth steps require the malicious origin server to receive a penetration request from the same edge server that receives the malicious end user request. However, this may not always be the case, as a request may pass through multiple edge servers before reaching the origin server. For example, an edge server `A` that receives an end user request (i.e., first hop edge server) may forward the request to edge server `B` that forwards it to edge server `C` (i.e., last hop edge server), from which the origin server receives the request.

Moreover, in some CDNs (e.g., CloudFront and CoralCDN), an edge server `A`, in most cases, will forward a non-cached request to an edge server `B`. This static forwarding between edge servers on a cache miss, allows an attacker to devise a stable mapping between the first and last hop edge servers by sending and tracking unique requests.

A successful covert channel communication attack requires the following. First, a malicious end user is able to select and forward a penetration request to an edge server. Second, the penetration request bypasses the edge server cache. Third, a malicious end user is able to infer the mapping between the first and last hop edge servers.

If any of the above requirements fail, the attack is unsuccessful [10]. To deny the first requirement, an edge server accepts only certified requests. Therefore, an end user must first contact an intermediate server to receive a certificate token and the IP address of the selected edge server. The intermediate server also sends the certificate to the selected edge server. The end user must then provide the token with a request to the edge server, which only serves a request with an authorized token. Hence, end users cannot connect to arbitrary edge servers. However, this increases the latency of serving requests and introduces an overhead of generating and validating tokens.

To tackle the second requirement, query strings in HTTP requests can be disabled. This prevents edge servers from forwarding dynamic requests to the origin servers, thus avoiding the bypass of edge servers. This also prevents a CDN from caching dynamic content in its edge servers.

Similarly, to thwart the third requirement, a CDN can randomize the mapping between first and last hop edge servers. This undermines the information encoding scheme for the covert channel, since the mapping between the sending edge server and alphabets are no longer stable.

E. Summary

Table III summarizes the security challenges and countermeasures discussed in this section. As shown, in addition to the application layer attacks, adversaries mostly target the caching and performance of edge servers. This undermines the CDN's primary functionality to quickly and efficiently deliver contents.

There are open problems for CDNs to counter edge server security challenges due to two opposing reasons. First, these security challenges exploit the application layer, thus their detection requires DPI, which comes with a high performance overhead. The growth in traffic encryption also requires CDNs to spend more resources for DPI. Second, inspecting encrypted traffic raises privacy and confidentiality concerns. The application layer contains personally identifiable information (PII), and DPI may expose content owners and end users PII. From another perspective, if a CDN owns sufficient resources for DPI, the CDN can also surveil and discriminate traffic [72]–[74].

When inspecting traffic, CDNs must comply with privacy preserving regulations, such as General Data Privacy Regulation (GDPR) [101] and California Consumer Privacy Act (CCPA) [102]. These regulations require stricter privacy protections. They grant consumers the right to control which PII can be collected, and how this information is used. To comply with these regulations, a CDN must employ appropriate controls and measures to protect the privacy of end users and

Security challenge	Goal	Vulnerability	Countermeasures
Prevalent threats [56], [60], [61]	-	-	Source code analysis [62], [63], web traffic inspection [64]–[68]
Hidden in encrypted traffic [74], [75]	Bypass security using encrypted traffic	Limited ability to inspect end-to-end encrypted traffic	Behavioral and statistical analysis [76], [77], expressive cryptographic schemes [79], [80]
Cache pollution [87]	Reduce cache performance	Locality of reference	Attacker-based and object-based detection [81], cache miss ratio history [89], randomness check [90], and two-step detection [88]
Cache poisoning [91]	Target cache integrity	Cache keys and HTTP unkeyed inputs	Exclude HTTP request headers from responses, include more HTTP request headers in cache keys [92]
Web cache deception [11], [84]–[86]	Stealing end user information	Caching control	Appropriate response to unexpected URLs, not force caching sensitive content [86], matching response type with requested content [85], [93]
Random string denial of service [10]	Denial of service	Supporting query strings in HTTP requests	Blocking requests with query strings, connection throttling, abort forwarding [10]
Cache poisoned denial of service [95]–[97]	Denial of service	HTTP headers	Exclude error pages from caching, WAF in front of edge servers [95]–[97]
CDN covert channel [99]	Hiding information transmission	Static request routing	Certified requests, blocking requests with query strings randomize mapping between [99] first and last hop edge servers

Table III: Edge Server Security Challenges

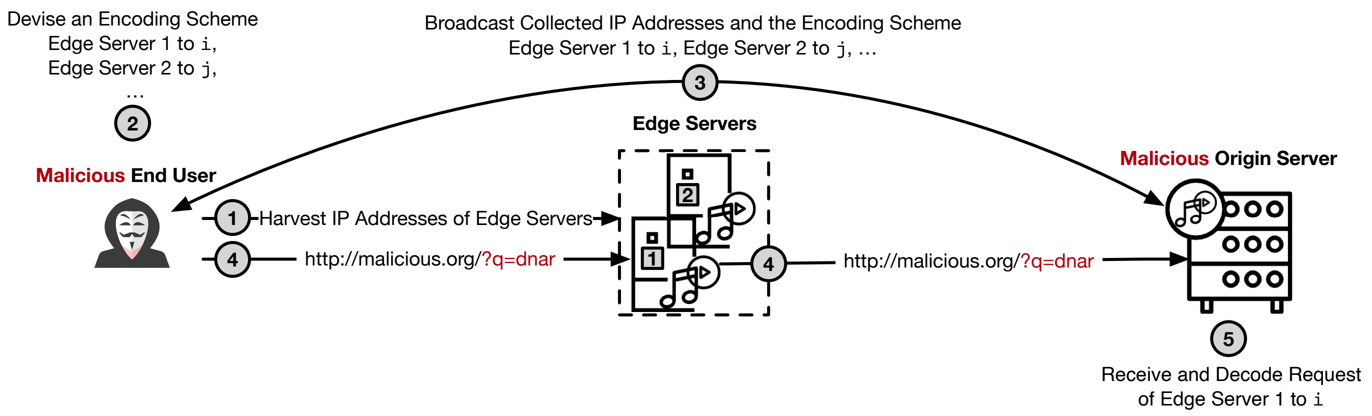

Devise an Encoding Scheme
Edge Server 1 to i,
Edge Server 2 to j,
...
2
Broadcast Collected IP Addresses and the Encoding Scheme
Edge Server 1 to i, Edge Server 2 to j, ...
3
Edge Servers
Malicious End User
Malicious Origin Server
1
Harvest IP Addresses of Edge Servers
4
http://malicious.org/?q=dnar
1
2
4
http://malicious.org/?q=dnar
5
Receive and Decode Request
of Edge Server 1 to i

Figure 4: Covert channel attack

content owners. For example, DPI must not expose end user identities, inadvertently or otherwise. Non-compliance can lead to serious repercussions for CDN providers, including financial penalty and reputation loss.

IV. Request Routing Security Challenges

In this section, we discuss security challenges and defense mechanisms pertaining to the request routing component. DNS is an integral part of CDN's request routing. Although we acknowledge that security challenges of DNS and anycast routing, e.g., DNS cache threats [103]–[105] and the BGP hijacking [9], [106], [107] are also applicable to CDNs, we refer readers to earlier research on these attacks [106], [108]–[113]. Instead, we focus on security challenges specific to CDNs.

We discuss how adversaries exploit the request routing component to collect sensitive information regarding a CDN, e.g., the IP addresses of edge servers, and use this information to launch sophisticated attacks against other CDN components (cf., Section IV-A). Adversaries tamper with the request routing component to redirect end user requests to their own desired addresses (cf., Section IV-B). They also overwhelm this component to prevent end users from accessing services provided by the CDN and content owners (cf., Section IV-C).

A. Reconnaissance

Adversaries engage with the request routing component using reconnaissance attacks to gather information about contents, content owners, and CDNs. They exploit the collected information for further active attacks, e.g., denial of service attacks, phishing, and cache poisoning.

Adversaries are particularly interested in collecting CDN IP addresses using reliable approaches [114]–[116]. CDNs own IP address ranges and distribute them among their edge servers. CDNs arrange their edge servers in two cache layers using the IP addresses as ingress and egress addresses. Edge servers at the first cache layer use the ingress IP addresses for their communications with end users, and edge servers at the second cache layer use egress IP addresses to communicate with origin servers [116].

Adversaries are interested in reliable approaches to collect the IP addresses of edge servers. A corollary is that some CDNs publish their IP ranges [117]–[119], allowing adversaries to target these addresses directly. However, there are CDNs that do not publish their IP addresses. Another approach is to query WHOIS for the history of IP addresses of a CDN. However, WHOIS can be incomplete or stale, because GDPR has prevented collection and storage of IP addresses [101].

1) Ingress harvest: A harvesting attack aims to collect the ingress IP addresses that edge servers use to communicate with end users. By knowing the IP address of an edge server, an adversary can launch denial of service against this edge server. This attack exploits randomness of assigning end user requests to edge servers to collect more ingress IP addresses [114]. This attack is not an effective reconnaissance mission for a CDN that routes requests based on anycast, because an adversary is unable to collect specific IP addresses as all edge servers have the same IP address.

In this attack, multiple insiders cooperate to collect the IP addresses of edge servers. They send content requests and process the request routing responses to collect and share the IP addresses of edge servers.

The success rate of this attack depends on the number of discovered IP addresses. Theoretically, the number of requests to harvest n edge servers is $O(n \log(n))$ [114]. However, the number of edge servers is unknown in practice, preventing an attacker from estimating a reasonable number of requests. Therefore, an attacker must carefully try a range of IP addresses without being detected by intrusion detection systems that monitor and track requests.

Venkatesan et al. [114] propose two countermeasures to combat harvesting attacks, namely bind-split and proactive proxy migration. The bind-split strategy maintains a mapping of end users to edge servers. An end user is bound to an edge server, to limit the number of edge servers that an insider can harvest [114]. This binding holds, unless the edge server is under attack. In this case, the edge server under attack is shutdown and its end users are equally split among two new spawned edge servers. If the edge server under attack was serving only a single end user, this end user is a malicious insider. However, this approach may lead to an unbalanced load distribution among edge servers, since each end user is tightly bound to an edge server.

To improve load distribution among the edge servers, proactive proxy migration strategy periodically shuffles the end user-edge server bindings, even when no edge server is under attack [114]. However, this strategy may also suffer from performance overhead, due to periodic proxy migrations.

2) Egress harvest: Another type of request routing harvesting collects egress IP addresses that edge servers use to communicate with origin servers [115], [116]. Using this information, an adversary can disrupt a CDN's communication with the origin. For example, the adversary achieves this goal using a crossfire attack [120], i.e., a link flood denial of service attack that uses thousands of bots to send traffic flows that together flood network links connected to the discovered egress IP address.

To collect egress IP addresses, an adversary must cause cache misses at edge servers. The reason is that edge servers communicate through egress IP addresses to fetch contents from origin servers only when cache misses happen.

An adversary conducts this attack by acting as a malicious end user and a malicious content owner. The adversary runs an origin server and registers it to receive delivery services from a target CDN. Controlling the origin server allows an adversary to collect CDN egress IP addresses by monitoring connections coming from edge servers. The adversary crafts HTTP requests with random strings resulting in cache misses at edge servers (cf., Section III-C1). The edge servers forward requests to the malicious origin server where the malicious origin server collects the egress IP addresses.

Mitigating this attack is non-trivial because all the attack activities are considered legitimate. A CDN alleviates the attack severity by limiting requests with query strings, as discussed in the random string denial of service in Section III-C1. However, as also mentioned before, doing so reduces the flexibility of a CDN to support dynamic requests and cache their responses for legitimate websites.

B. Redirection Exploits

Adversaries exploit the vulnerabilities of request routing based HTTP, to bypass CDN security mechanisms and redirect end user requests to their intended destinations.

The discrepancy in HTTP implementations to interpret a request is at the heart of a series of threats in HTTP request routing. A CDN uses multiple HTTP entities, e.g., WAFs, reverse proxies, and web servers to process HTTP requests. When these entities are on a data stream path, they may have different interpretations of the same HTTP request. Adversaries exploit these discrepancies to launch HTTP smuggling, multiple hosts ambiguities [121], and denial of service attacks [122]. Alibaba, Cloudflare, Cloudfront, and Level3 are among CDNs reported to be vulnerable to these attacks [121]. We discuss the denial of service attacks in Section IV-C.

1) HTTP smuggling: Adversaries use HTTP smuggling [121] to bypass traffic filtering and launch cross site scripting and session hijacking attacks. To conduct a HTTP

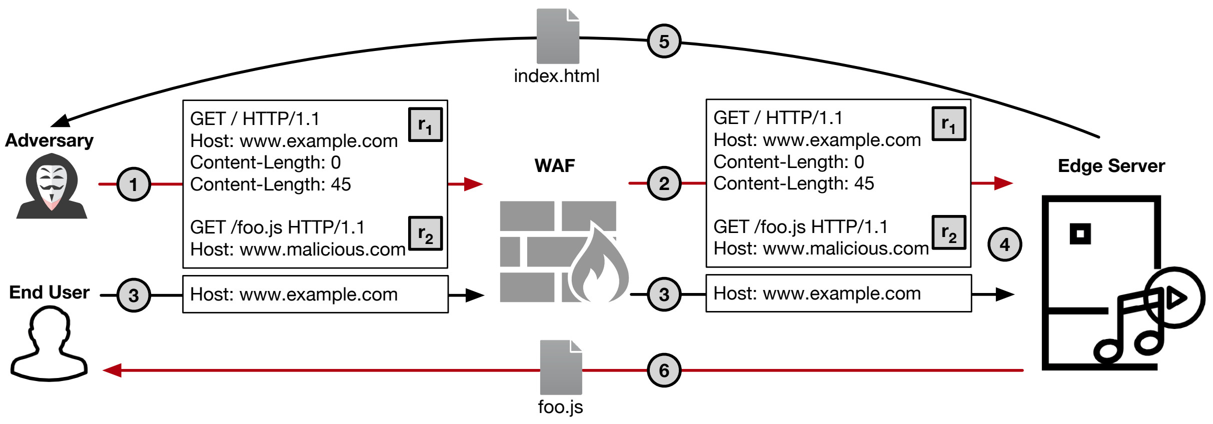


Figure 5: HTTP smuggling attack

smuggling, an adversary sends multiple HTTP requests that pass through multiple HTTP entities. These requests are crafted such that the HTTP entities, due to inconsistency, observe different sets of requests. In this way, the attacker “smuggles” an HTTP request to an entity, while the other entities are unaware of this request.

Figure 5 shows an example of HTTP smuggling attack, where a WAF intercepts requests going to CDN edge servers. First, an adversary crafts and sends a malicious HTTP request that actually consists of two requests, r_1 for ‘www.example.com’, and r_2 pointing to a malicious script ‘bad.js’ on ‘www.malicious.com’. The malicious request contains two ‘Content-Length’ headers that causes inconsistent interpretations at the WAF and the edge server. Second, the WAF prioritizes ‘Content-Length: 45’ leading to interpreting r_2 as the body of r_1, and forwards them as a single request to the edge server. This allows the attacker to bypass the WAF filter even when ‘www.malicious.com’ is in the WAF’s blacklist. Third, an end user requests ‘www.example.com’, and the WAF forwards this request to the edge server. Fourth, the edge server prioritizes ‘Content-Length: 0’, and interprets the malicious request as two separate requests. Fifth, the edge server replies with ‘index.html’ to the adversary in response to r_1. Sixth, the edge server replies to the end user with ‘foo.js’ from ‘malicious.com’ in response to r_2.

To counter the above challenge, HTTP entities must use stricter parsing semantics for requests. Intercepting requests with a WAF that correctly applies strict parsing rules is a useful defense mechanism. Other solutions include HTTPS communication and requiring the termination client sessions after each request.

HTTP entities can avoid the threat by complying with RFC 7230 [123]. However, it is unrealistic to expect that all existing implementations will change and comply with the standard.

Another approach with lesser implementation changes is to use a firewall that enforces the RFC specifications. This firewall can be placed before any HTTP entity that does not comply with the standard. The firewall will intercept traffic before it reaches other HTTP entities, and either reject or raise an alert about invalid HTTP requests.

2) Multiple host ambiguities: Adversaries exploit the inconsistencies in interpreting the `Host` header field of HTTP 1.1 requests to conduct multiple host ambiguities attack, which allows an adversary to bypass security mechanisms and launch other attacks, such as cache poisoning. `Host` header is used for virtual hosting that allow multiple domain names to be hosted on a single IP address. It identifies the domain name and optionally a TCP port on the web server (e.g., `Host: www.domain1.com:8080`).

There are three types of adversely crafted requests [121]. The first is an HTTP request with multiple `Host` headers. Although RFC 7230 [123] dictates that a request with multiple `Host` headers must be rejected, existing implementations still process the request by selecting one of the `Host` headers. Different implementations may have varying preferences in selecting a host among multiple `Host` headers of a single HTTP request.

The second type misuses space characters inside a `Host` header to cause inconsistencies. HTTP entities interpret a `Host` header with space characters as multiple `Host` headers. For example, if an entity prefers the first host, this entity might consider the space preceding the first header as an unknown `Host` header. On the other hand, if an entity prefers the last host, the preceding space does not affect its `Host` header selection.

The third type of crafted HTTP requests include absolute URIs. RFC 7230 [123] instructs web servers to accept an absolute URI, and to favor the hostname with an absolute URI over a `Host` header. The RFC also specifies that the hostname in the absolute URI and `Host` header must be unique. However, some entities do not identify the hostname of an absolute URI, while others do not check the identity of an absolute URI domain name and `Host` header.

The countermeasures used for HTTP smuggling are also applicable for multiple host ambiguities.

C. *Denial of Service*

Adversaries take advantage of vulnerabilities in HTTP and DNS, to reduce the performance of request routing and launch application layer denial of service attacks. In this section, we discuss five attacks that exploit HTTP and DNS vulnerabilities to conduct denial of service.

1) Forwarding loop: This attack misuses the `Host` header field, similar to the multiple host ambiguities attack, to exhaust resources of one or more CDNs [122], [124]. An edge server can be configured to forward a request based on the `Host` header field. An attacker can exploit this request redirection to force CDNs to process a request repetitively, and even indefinitely, leading to a denial of service.

An adversary conducts a HTTP forwarding loop attack as follows. The adversary rents and configures edge servers with forwarding rules based on `HOST` headers. Then, the adversary crafts HTTP requests with malicious `HOST` headers that trigger the forwarding rules on edge servers and cause indefinite forwarding loops across the edge servers.

Chen et al. [122] identify four kinds of forwarding loop attacks: (i) self loop within an edge server (ii) intra CDN loop between multiple edge servers within a CDN, (iii) inter CDN loop across multiple CDNs, and (iv) CDN dam flooding that can cause faster loops against CDNs supporting HTTP streaming.

There are three countermeasures against the forwarding loop attacks [122], [125]. First, a CDN can rate limit or filter requests with `HOST` headers to mitigate a forwarding loop attack. A CDN monitors HTTP headers and applies rate limiting based on per source IP address or per end user. If the monitored traffic exceeds a predefined threshold, the CDN can reject or downgrade subsequent requests from the same IP address or end user. Filtering can be applied on forwarding destinations, e.g., a CDN filters a request if it must be forwarded to another CDN [122].

Second, a loop can be detected by tracking the servers that have already processed a HTTP request. HTTP 1.1 standard specifies the `Via` header in a HTTP request to indicate web proxies that have processed this HTTP request [123]. In a CDN, each edge server that forwards a HTTP request, appends its identifier (e.g., its host name). Thus, the `Via` header field contains a list of recipient edge servers and an edge server can detect a loop by observing its own identifier in this list. To prevent long loops, a request is also rejected if the number of servers listed in the `Via` header crosses a given threshold. CDNs must prevent removing and forging `Via` headers to ensure the effectiveness of this defense mechanism. However, some web servers have traditionally used the `Via` header for other purposes that conflict with loop detection. Particularly, some web servers disable certain features in presence of `Via` headers, which can cause performance degradation. For example, NGINX does not compress responses to requests with `Via` headers. In addition, `Via` header can expose internal information of CDNs, e.g., the host names of edge servers.

Finally, Akamai, Fastly, and Cloudflare have collaborated to standardize the HTTP `CDN-Loop` header [125] to remedy the performance issues of using the `Via` header for loop detection. Web servers can use the `CDN-Loop` header for the purpose of loop detection, while applying `Via` headers for traditional purposes without the above performance penalties. This header works similar to `Via` header. Each edge server appends its CDN identifier before forwarding a request, and the edge server detects a loop if its CDN identifier is found in a HTTP request. `CDN-Loop` header is effective only if used by all CDNs. A CDN that does not implement this header remains an attack vector against other CDNs, even if they have implemented this standard [125].

2) HTTP/2 amplification: Guo et al. [115] exploit CDNs' support of HTTP/2 to launch an amplification denial of service attack against the origin. Some CDNs support only HTTP/2 for connections at the edge between end users and edge servers, while all origin connections between edge servers and origin servers are HTTP/1.1, even when origin servers support HTTP/2 [115]. They convert HTTP/2 to HTTP/1.1 and back, for HTTP/2 connections at the edge. Adversaries can abuse this HTTP/2-HTTP/1.1 conversion to conduct a bandwidth amplification denial of service attack against origin servers. Cloudfront, Cloudflare, CDNSun, Fastly, KeyCDN, and MaxCDN are vulnerable to this attack.

HTTP/2 improves HTTP/1.1 bandwidth usage using connection multiplexing and HPACK compression algorithm. Specifically, HTTP/2 uses multiple concurrent bidirectional streams within a single connection to reduce unnecessary TCP handshakes, and full request and response multiplexing. Moreover, HTTP/2 HPACK compresses HTTP messages to serve a web page. Rendering a modern web page can generate hundreds of HTTP requests with redundant HTTP headers. HPACK maintains an index table to track HTTP headers and substitutes redundant large headers in a HTTP message with much smaller table indices.

Adversaries can also abuse this HTTP/2-HTTP/1.1 conversion to conduct a bandwidth amplification denial of service attack against origin servers. HTTP/1.1 connections consume many times more bandwidth than their associated HTTP/2 connections. This allows adversaries to craft HTTP requests resulting in an amplification factor of 166.

This threat arises since CDNs do not support HTTP/2 on the CDN-origin side. CDNs can eliminate this threat by running the same version of HTTP for both communication sides.

3) Slow pre-POST: Adversaries can abuse the approach of some CDNs in forwarding POST requests to conduct a slow HTTP denial of service attack against the origin [115]. CDNs including Cloudfront, Fastly, and MaxCDN have a pre-post forwarding behavior where they forward a HTTP POST request upon receiving the POST header before receiving the entire POST body. They perform this behavior for POST forwarding of both HTTP/1.1 and HTTP/2 to speed up serving POST requests.

A slow HTTP attack exploits HTTP behavior in keeping a TCP connection open until the data is entirely delivered. HTTP behaves the same for different stages of a request flow making these stages vulnerable to a attack. For example, a slow POST attack sends POST body at a very slow rate. The slow HTTP attack opens massive number of connections and keeps them open as long as possible. This exhausts the number of concurrent connections at the server side.

This attack involves two steps. First, an adversary crafts and sends massive number of large POST requests, e.g., thousands of requests. The adversary keeps the request connections open by sending each request body very slowly, e.g., for 300 seconds. Second, an edge server opens a connection with an origin server to forward a received POST request from the adversary. It opens the connection before receiving the entire POST body. The edge server also keeps its connections with the origin server open during forwarding of the POST bodies. Large number of connections from the edge server exhausts the origin server resources.

This attack is stealthy and hard to detect, as crafted requests are legitimate, and an origin server receives connections from CDN edge servers that are considered non-malicious. However, the attack can be mitigated at the origin and at the edge. Origin servers can use a shorter timeout to receive POST bodies, while the edge servers can mitigate the attack by storing a request completely before forwarding it [115]. Cloudflare implements this countermeasure at its edge servers.

4) Egress blocking: Adversaries can threaten content availability by exploiting the low IP churning rate of CDNs for their communications with the origin. Some CDNs assign predictable egress IP addresses, for example MaxCDN uses a single egress IP address for 96% of its traffic with an origin server [115]. An adversary can collect these egress IP addresses using harvesting attacks, discussed in Section IV-A2, and make the contents of this origin server unavailable by disrupting the CDN communications with the origin using these egress IP addresses.

Caching at edge servers motivates this low egress IP churning rate, i.e., the frequency of a CDN using the same egress IP address. Edge servers commonly have a heavier traffic load at the ingress compared with that of the egress. They batch incoming requests and forward them to origin servers using few egress IP addresses.

The feasibility of disrupting CDN egress communication depends on the geographical locations of edge servers. An adversary can use crossfire attack [120], where the adversary programs thousands of bots to flood network links connected to discovered egress IP addresses. The bots must be within close proximity of the egress IP addresses. The links can be found using common tools, such as `traceroute` [120].

Another approach is to exploit on-path appliances, e.g., middleboxes and routers to block the CDN's egress traffic. For example, China's great firewall inspects HTTP flows for sensitive words, e.g., `ultrasurf` and blocks the source IP address of TCP connections carrying these HTTP flows for 90 seconds [115], [126]. The adversary can exploit this to temporary block egress IP addresses within China. The adversary crafts and sends HTTP requests with sensitive words, and edge servers use egress IP addresses to forward these requests to the origin that is outside China. The great firewall will inspect the request and block the egress IP addresses.

CDNs can mitigate this attack using a more unpredictable assignment strategy of egress IP addresses, e.g., by assigning more egress IP addresses to an origin server and churn them more frequently. Doing so prevents this attack from exploiting the CDN's strategy that uses predictable egress IP address assignment with a few IP addresses per origin server [115]. However, this countermeasure can reduce caching performance because distributing request forwarding across more egress edge servers reduces aggregation probabilities. Moreover, this also allows adversaries to conduct more effective reconnaissance, i.e., they can collect more CDN IP addresses.

5) Redirection hijacking: Adversaries can insert crafted DNS records on name servers using DNS cache poisoning [104], [127] to redirect end user requests to an edge server. A flood of these requests overwhelms the victim edge server. Sixteen popular CDNs, including Cloudflare, Akamai, and Limelight were identified to be vulnerable to this attack [42].

An adversary can even nullify hijacked requests by sending these requests to offline edge servers. To do so, the adversary collects the IP addresses of edge servers (e.g., using harvesting attacks discussed in Section IV-A1), tracks their liveliness (e.g., using `ping`), and poisons name servers with records pointing to the offline edge servers.

The DNS cache poisoning exploits the authentication vulnerability of the DNS name resolution to insert bogus DNS records to a name server's cache. Until poisoned records expire, the adversaries can redirect end user requests to the IP addresses. If a resolving name server receives a DNS query and does not have a cached response, it asks another name server. The former name server uses a 16 bit transaction identifier to authenticate replies from the latter name server. If an attacker guesses the value of this identifier and responds faster than the former name server, the attacker can poison the resolving name server's cache.

There are two DNS cache poisoning attacks, Kaminsky attack [128] and exploiting long DNS responses [127]. Kaminsky attack [128] increases the chance of correctly selecting a transaction identifier, by forcing a victim name server to resolve numerous random domain names. An attacker launches a name server responsible to resolve a domain name controlled by the attacker (e.g., `bad.com`). When a victim name server queries the malicious name server to resolve the domain name, the malicious name server replies with a response that refers the victim to random names within a popular domain name (e.g., `1.trump.com`, `2.trump.com`, `3.trump.com`). The victim generates new DNS queries to resolve these domain names, and the malicious name server sends a flood of fake DNS replies with random transaction identifiers. Each DNS reply is an opportunity for the attacker to poison the victim's cache.

The fragmentation of long DNS responses can be exploited to poison a name server's cache [127]. Adversaries craft DNS queries that force a name server to return long responses that are a few thousand bytes in length. With the typical 1500 bytes for the maximum transmission unit, these responses are fragmented into two IP packets. The first fragment contains authentication parameters, while the second contains the result of name resolution. Hence, an adversary can inject malicious IP addresses in the second fragment. On the receiving end, two fragments are reassembled only if they contain matching `IP Identifiers`. Therefore, the attacker must correctly guess this 16-bit `IP Identifier` for the second fragment. The authors show that on average around a 1000 guesses lead to a successful attack.

DNS cache poisoning is the heart of the request redirection hijacking threat. Preventing DNS cache poisoning remedies this threat as well. To alleviate this threat, request remapping at edge servers can be used [42]. Next we discuss three defense mechanisms against DNS cache poisoning, then we discuss request remapping.

Challenge-response is a widely used approach to harden name servers against DNS cache poisoning [129]. The resolver randomizes the values of some fields in requests, and corresponding responses must contain the same values for these fields. These fields include transaction identifier, source ports, and name server addresses [130], [131]. By matching randomized fields, the resolver can validate the legitimacy of received responses. However, this approach does not completely eliminate DNS cache poisoning, as these fields are plaintext and attackers can easily craft bogus responses [132].

To protect name servers against cache poisoning, the IETF standardized the DNS Security Extension (DNSSEC) protocol [133], [134]. DNSSEC establishes a chain of trust among name servers, which allows them to authenticate the DNS responses. Responses must be digitally signed (i.e., using public key encryption) to guarantee that DNS responses are from legitimate name servers. For example, when a name server `server` resolves `sub.domain.com`, the top level domain name servers responsible for `.com` help `server` verify the responses from the `domain.com` name servers, which in turn assists `server` in authenticating the responses of `sub.domain.com`. Finally, the root name servers assist `server` in authenticating the `.com` responses. A root signing ceremony for DNSSEC [135] guarantees the authenticity and integrity of the root name servers.

Although some CDNs [43], [44] support DNSSEC, CDNs have not widely used DNSSEC due to its high performance overhead [42]. A CDN must dynamically update DNS records for load balancing based on real-time conditions of the network and the edge servers. The high computational cost of signing dynamic DNS records make DNSSEC an unpractical solution to secure DNS-based request routing [42].

Wildcard secure DNS [136] utilizes wildcard domain names to harden a name server against attacks, which is more compatible with existing DNS services. A name server includes a random string in its DNS query to increase the entropy of its DNS queries, thus making valid DNS responses difficult to guess. Specifically, a wildcard domain (e.g., `*.www.dmn.com`) includes an '`*`', which can be replaced with a combination of characters. Before forwarding a DNS query, a name server inserts a random string (e.g., `rand`) instead of '`*`' to the queries domain name (e.g., `rand.www.dmn.com`). A valid response must include this random string, making it difficult to craft fake responses. To poison a corresponding DNS record, an attacker must guess the random string in addition to the transaction identifier.

Edge servers also can perform request remapping to alleviate request redirection hijacking threat [42]. In this approach, edge servers only serve end users with acceptable round-trip time. On receiving a request, an edge server measures the round-trip time. If the measured round-trip time is high, the edge server initiates a remapping of the request to another edge server via a specific reply (e.g., HTTP status code 3xx). On receiving this reply, the end user must query the request routing component again, which assigns this request to another edge server. This request remapping can protect edge servers from being overwhelmed due to redirection hijacking. However, it results in additional latency in serving requests, due to the overhead of measuring the round-trip time. Moreover, it does not eliminate the threat of nullifying end users' requests.

Another approach is to divert and absorb traffic [137]. CDNs, such as Akamai and Incapsula divert traffic to their scrubbing centers, where over-provisioned resources absorb the traffic load. Furthermore, the huge infrastructure of CDNs at the edge can be used to absorb traffic close to its origin. More information on defense mechanisms against reflection attacks can be found in [137]–[139].

Limiting DNS responses in rate or size is another mitigation approach. Paul Vixie [140] propose DNS response rate limiting, where authoritative name servers limit the rate of DNS responses for the same query to the same IP block. However, this approach can also affect legitimate DNS queries [141]. By limiting the size, a DNS response cannot exceed a specific size, which impacts the ability of responding to DNS `ANY` queries. Nevertheless, some CDN providers are willing to deprecate `ANY` queries, as they can be exploited to launch DoS attacks [142].

D. Summary

Table IV summarizes the security challenges and countermeasures discussed in this section. As shown in Table IV, adversaries exploit the request routing for reconnaissance, bypassing CDN security mechanisms, and launching denial of service attacks. They exploit vulnerabilities of request routing implementations based on DNS and HTTP.

Request routing mechanisms are easily exploitable, while effective countermeasures are hard to implement. For example, legitimate requests can be easily misused to harvest CDN IP addresses, while a CDN must disable query strings for mitigation, affecting its flexibility in serving dynamic requests. Moreover, standardization can remedy several security challenges due to HTTP vulnerabilities if all involved entities implement them. For example, all CDNs should collaboratively implement one of `Via` or `CDN-Loop` headers to eliminate forwarding loop attacks.

Finally, mitigating a security challenge may introduce vulnerability for another security threat. A CDN can mitigate egress blocking with higher churn rates in assigning egress IP addresses. However, this introduces reconnaissance opportunities for adversaries to harvest more CDN IP addresses.

V. Origin Server Security Challenges

In this section, we discuss the origin server security challenges and countermeasures. We start with the application layer threats (cf., Section V-A). Hiding or isolating the IP addresses of origin servers is essential, since attackers can directly target origin servers with exposed addresses. We focus

Security challenge	Goal	Vulnerability	Countermeasures
Ingress harvest [114]	Reconnaissance	The randomness of request routing	Bind-split, proactive proxy migration [114]
Egress harvest [115], [116]	Reconnaissance	Supporting query strings in HTTP requests	Blocking requests with query strings [10]
HTTP smuggling [121]	Security bypassing	Discrepancy of HTTP entities in request interpreting	Filtering invalid HTTP requests, using HTTPS, terminating client sessions after each request [121], modify HTTP entities to comply with RFC 7230 [123]
Multiple host ambiguities [121]	Security bypassing	Discrepancy of HTTP entities in interpreting `Host` headers	Filtering invalid HTTP requests, using HTTPS, terminating client sessions after each request [121], modify HTTP entities to comply with RFC 7230 [123]
HTTP forwarding loop [122], [124]	Denial of service	HTTP `Host` header	Filtering or rate limiting on requests with `HOST` headers [122], using HTTP `Via` header [123] or HTTP `CDN-Loop` header [125]
HTTP/2 amplification [115]	Denial of service	HTTP/2-HTTP/1.1 conversion in a CDN	Using the same HTTP version at both end user and origin sides [115]
Slow pre-POST [115]	Denial of service	Pre-POST behavior and HTTP slow connections	Using a shorter timeout to receive POST bodies at origin servers, storing a request completely before forwarding at edge servers [115]
Egress blocking [115]	Denial of service	Low churn rate in assigning egress IP addresses	Unpredictable strategies for egress IP address assignment [115]
Redirection hijacking [42]	Denial of service	DNS cache poisoning vulnerability	Response field values must match randomized request field values [129]–[131], DNSSEC [133], [134], wildcard DNS [136], request remapping [42]

Table IV: Request Routing Security Challenges

on vulnerabilities that can lead to the exposure of origin server IP addresses (cf., Section V-B). Finally, we discuss how the configuration options of origin servers can be exploited to circumvent restrictions that content owners enforce for content delivery (cf., Section V-C).

A. Application Layer Threats

Web authentication depends on SSL/TLS. SSL/TLS is coupled with a public key infrastructure to provide a straightforward authentication semantic—Alice has a certificate binding Bob to a public key, and the remote entity must be Bob with proof of knowledge regarding Bob's private key. SSL/TLS use a similar semantic to ensure confidentiality of messages exchanged between Alice and Bob. Both authentication and confidentiality are provided based on the assumption that Bob is the only entity with knowledge of Bob's private key.

SSL/TLS man-in-the-middle: SSL/TLS and CDNs do not blend well together. Having edge servers in the middle of SSL/TLS connections between end users and origin servers, break the end-to-end confidentiality. Nevertheless, for a CDN to inspect and act as a proxy for SSL/TLS connections, content owners often share their private keys with CDNs [143], [144]. However, sharing of private keys violate the fundamental security principle of keeping them secret. Furthermore, with access to private keys an edge server essentially becomes a man-in-the-middle, making content owners vulnerable to eavesdropping and tampering of SSL/TLS connections, or even impersonation.

SSL/TLS protocols enable two parties to establish a secure end-to-end connection. In SSL/TLS communication, the client and server end points (e.g., end user's browser and origin server) perform a SSL/TLS handshake using asymmetric encryption, authenticate each other and establish a session key. The session key is symmetric and is used to encrypt messages exchanged between the client and server endpoints. The confidentiality is ensured assuming that no third party can decrypt the encrypted messages, without access to the private and session keys of the client and server.

Existing measures to counter this threat include approaches that do not enforce the sharing of private keys for SSL/TLS communication. Some of them require a content owner to share an unencrypted content with a CDN. Others dictate specific application design, which comes at the cost of complex application development.

Approaches that do not require sharing of private keys [145]–[147], capitalize on the fact that a private key is only required during the SSL/TLS handshake to establish a session key. There onwards, the session key is used to encrypt the rest of the communication. Thus, a CDN is still able to intervene secure SSL/TLS connections without accessing the private keys of content owners.

To implement this, Cloudflare Keyless SSL [145] involves an additional key server at the origin. When an end user connects to an edge server, it sends a secret to the key server. This secret is encrypted using the content owner's public key. The edge server connects to the key server and authenticates using its certificate, and forwards the secret over an encrypted channel. Upon receiving the secret, the key server replies to the edge server with the decrypted secret. Using this secret, both the end user and the edge server derive the same session key. In Cloudflare Keyless SSL, the end user can trust the edge server, as the key server is the only party that has the private key to decrypt the secret. While the edge server can only obtain the decrypted secret, after it has been authenticated by the key server.

Cloudflare Keyless SSL eliminates the need for sharing private keys with a CDN. However, a CDN can still learn session keys to eavesdrop SSL/TLS connections or impersonate content owners. Phoenix [148] uses enclaves, private regions of memory, in a trusted execution environment provided by Intel software guard extension (SGX) [149]. This enables a content owner to rely on an untrusted edge server for establishing TLS connections, without exposing private and session keys.

Phoenix establishes TLS connections from an enclave, where both the running code and private keys are protected from software attacks. The integrity and confidentiality of the code and data are ensured, even if adversaries have full control over an underlying platform. Another enclave, called the provisioning enclave, on an edge server retrieves private keys from an origin server over a TLS connection, using Let's Encrypt [150]. Since the private keys are stored in an enclave, the CDN does not have access to them.

Phoenix relies on Intel SGX, which is vulnerable to side channel attacks [149], i.e., an untrusted CDN can still obtain private or session keys. However, a side channel attack requires physical access to edge servers. Moreover, the content owner must trust the manufacturer of the processor that have access to the keys. Nevertheless, the consequences of sharing private keys with a third party manufacturer in Phoenix, can be more severe in comparison to sharing session keys in KeyLess SSL.

Another approach is to allow end users and edge servers to establish a SSL/TLS connection for content transmission, without involving the origin servers. Specifically, content owners provide services that enable end users to authenticate edge servers [144]. At the origin, content owners run DNSSEC with DNS TLS authentication records to pin the certificates of content owners and the CDN. An end user can query these records to authenticate an edge server. Thus, content owners do not need to share their private keys with a CDN. However, they must share their unencrypted contents with the CDN.

For untrusted CDNs, Stickler [151] and CDN-on-demand [152] facilitate content owners to store encrypted content with CDNs. In this case, edge servers cache contents that are signed by the content owner. Using a script that is provided by a content owner, an end user can retrieve, authenticate, and decrypt contents. For scalability, an origin server provides only the script and does not take part in delivering contents. This solution addresses the issue of sharing private keys and unencrypted contents. However, modifying the application may require additional development effort to accordingly update the script.

B. *Origin Exposure*

A CDN acts as a protection shield for origin servers by absorbing many security threats against the origin. However, adversaries can bypass this protection and directly target origin servers using their IP addresses [153], [154]. In this section, we discuss security challenges that lead to the leakage of origin server IP addresses.

1) Static origin addresses: Using static IP addresses for origin servers make them vulnerable to exposure. When origin servers are temporary exposed to end users, e.g., while a CDN pauses its service for maintenance [153], [154], an adversary can collect and retain their IP addresses. Moreover, there are organizations [155], [156] that maintain databases of DNS records. Adversaries can search these databases for a history of origin domain names and their IP addresses, and exploit static IP addresses.

Changing the IP addresses of origin servers and DNS records can mitigate this attack. When a CDN shields origin servers, DNS records must be changed to CDN addresses, and new DNS records must not expose the origin addresses [157]. Origin IP addresses should be changed to differ from the IP history collected before being protected by a CDN or during a CDN's temporal inactivity.

2) Services by origin servers: The IP addresses of services that are directly provided by origin servers can expose the origin addresses. Typically, origin servers directly serve services, such as mail, FTP, and SSH [154], without using a CDN. Hence, adversaries can collect origin addresses from DNS records of these services (e.g., `MX` records that refer to mail services). Content owners also use hidden sub-domains for some services, such as SSH (e.g., `ssh.owner.com`). Using a dictionary attack, an adversary can guess and query hidden sub-domains to collect origin IP addresses.

This vulnerability can be alleviated using port forwarding [158]. For services that are directly served by origin servers, content owners can use edge servers as proxies that receive and forward requests. The DNS records associated to these services must also point to the edge servers. Consequently, edge servers first receive requests for these services and forward them to origin servers without exposing origin IP addresses [154].

Another approach leveraged by origin servers is to only serve requests coming from trusted addresses. To implement this, content owners deploy firewalls to inspect and whitelist incoming traffic to origin servers.

3) Leakage in contents: Web contents and Pingback services can leak the origin addresses. Vulnerable web contents include configuration files, verbose pages, and log files. Developers may also unintentionally leave the IP addresses of content owners in HTML files.

An attacker can exploit Pingback services to collect origin addresses. Using pingback, content owners check the validity of a third party link to their contents. Upon receiving the notification of a third party link, an origin server initiates a connection with the third party to check the validity of this link [154]. An attacker can extract the IP address of the origin server from this incoming connection.

To prevent such information leakage, sensitive and source-code files must be inspected to ensure that they do not expose the origin server's information. Furthermore, the access to sensitive files should be limited. To mitigate the Pingback vulnerability, Pingback requests can be dropped at the edge, e.g., using a WAF [159], or origin servers can disable the Pingback service [160].

4) Residual name resolution: Origin addresses can be leaked during dynamic changes, when content owners switch between pausing or termination of their services with CDNs [153]. For services that are active, name servers operated by a CDN redirect content requests to the CDN's edge servers. If the content delivery service is paused or terminated, the name servers still retain some relevant DNS records. These DNS records resolve requests to the origin IP addresses, instead of edge servers, because the CDN no longer serves the requests.

A CDN can eliminate this threat if its name servers stop responding to content requests for paused or terminated delivery services. However, if content owners decide to serve contents from their own origin servers, this mitigation can cause temporary disruptions. This is because other name servers across the Internet cache DNS records, with relatively long time-to-live, that still point to the name servers of the old CDN.

Some CDNs allow a content owner to configure origin domain names [124], [161]. Before terminating a service, the content owner configures the old CDN to redirect requests to a domain name belonging to the new CDN. Thus, requests are redirected to the new CDN, while no origin IP addresses reside in the name servers of the old CDN.

C. *Origin Abuse*

CDNs support a convenient set of options that enable content owners to configure the content delivery process. For example, the content owners can specify the content origins (e.g., the domain names, IP addresses, and port numbers of origin servers), caching and forwarding policies. However, several CDNs do not validate the specified configuration (e.g., content owner being the actual owner of the origin servers), allowing content owners with malicious intent to configure and abuse CDNs [124], [161].

1) Origin address abuse: An adversary can use configuration options for domain names and IP addresses of origin servers, to circumvent restrictions that content owners or certain geographical locations enforce for contents. For example, some content owners (e.g., Pandora Media and Netflix) enforce geographical restrictions in providing their services. These include serving contents only in some countries and providing unique sets of content in different geographical locations. Moreover, in certain geographical locations, some sensitive contents are restricted.

An adversary, acting as a malicious content owner, can bypass these restrictions by exploiting configuration options for domain names and IP addresses of origin servers. The adversary can configure edge servers to query origin servers with restricted contents. Despite enforced restrictions, edge servers can serve restricted contents to end users as they have access to these contents on origin servers.

CDNs can eliminate this vulnerability by validating the ownership of origin servers. For example in origin pinning mitigation [124], a CDN requires a content owner to provide both the IP and domain name of the origin server, and upload a special file to it. The CDN continuously monitors this IP-domain pair and the existence of the uploaded file. It can terminate the service and demand a new validation process in case of an abnormality. However, the usability of origin pinning for content owners require in-depth investigation. Moreover, it may open up new vulnerabilities and security threats, due to uploading of files to origin servers.

2) Origin port abuse: The configuration of port numbers for origin servers allow an adversary to launch stealthy port scan and denial of service attacks against legitimate origin servers. An adversary can configure a CDN to scan ports of origin servers. In case of errors, the CDN generates error responses that disclose the status of the scanned ports. Moreover, the CDN can be configured to conduct a denial of service using edge servers that open concurrent connections to an origin server. The origin server becomes unresponsive if the number of connections exceed its limit.

The mitigation approaches for origin address abuse are also applicable against port abuse. In addition, whitelisting of the origin port numbers can alleviate this threat.

D. *Summary*

Table V highlights the security challenges and countermeasures of this section. From the origin side, CDNs can be considered a man-in-the-middle that can break the end-to-end confidentiality of content transmission. Adversaries also collect origin IP addresses from different information sources to bypass CDNs and directly target origin servers. Moreover, malicious content owners exploit origin configuration options, provided by CDN, to bypass geographical content restrictions, conduct port scanning and denial of service attacks.

Delivering SSL/TLS traffic through CDNs is still a dilemma. Content owners must trust CDNs and share their private

Security challenge	Goal	Vulnerability	Countermeasures
SSL/TLS man-in-the-middle [143], [144]	Impersonation, connection tampering and eavesdropping	Content owners sharing private keys with a CDN	Keyless SSL [145], SSL/TLS connection enclaves [148], Stickler [151], and CDN-on-demand [152]
Static origin addresses [153], [154]	Collecting origin IP addresses to bypass CDNs	IP history and exposures during CDN maintenance	Changing origin IP addresses on-demand [157]
Services by origin servers [154]	Collecting origin IP addresses to bypass CDNs	Serving services, such as SSH and FTP at the origin	Port forwarding at the edge [154], [158], connection whitelisting at the origin
Leakage in contents [154]	Collecting origin IP addresses to bypass CDNs	Explicit data in static web files, open access to sensitive web files, and Pingback exposure	Inspection of sensitive files, restricting access to sensitive files, blocking Pingback service [159], [160]
Residual name resolution [153]	Collecting origin IP addresses to bypass CDNs	Residual origin IP addresses in CDN name servers	No DNS response for paused or terminated services, DNS resolution to origin domain names instead of origin IP addresses [124], [161]
Origin address abuse [124]	Bypassing geographical content restrictions	No validation of origin configurations	Validating origin ownership using origin pinning [124]
Origin port abuse [124]	Stealthy port scan, denial of service	No validation of origin configurations	Validating origin ownership using origin pinning [124], port whitelisting

Table V: Origin Server Security Challenges

keys, or rely on mechanisms, such as KeyLess SSL, which still enable a CDN to impersonate and tamper end-to-end encrypted traffic with end users. On the other hand, measures to counter origin exposure and origin abuse are successful. Good programming and design practices can eliminate or effectively mitigate origin exposure and abuse.

VI. Future Research Directions

We discuss current obstacles and vulnerabilities, and outline opportunities and future research directions in this section. We begin with emerging contents that challenge the current CDN operation in Section VI-A. In Section VI-B, we present opportunities and challenges of software defined security, a new paradigm in CDN security. Finally, we discuss opportunities and challenges of security collaboration between the parties involved in content delivery in Section VI-C.

A. Securing Emerging User Generated Contents

The volume and diversity of contents are increasing, making the protection of CDNs even more challenging. Live video delivery and user generated live contents, e.g., Skype and video recordings from mobile devices, are generated and consumed at the edge [162], [163]. Mobile end users will grow to 13.1 billion by 2023, and this growth changes the volume and diversity of video content in mobile networks [164].

Delivering live video stream is challenging because it is generated and consumed in real time. In the HTTP-based live streaming, video is broken and encoded into multiple chunks (e.g., 2 to 10 seconds long). The receiver end user fetches each video chunk independently using a HTTP GET request. Live video streams are latency sensitive, making them an attractive target for denial of service attacks.

CDNs will serve a dynamic population of viewers, while they are not typically optimized for such long tailed contents. Social cascading and recommender promotions can lead to sudden flash crowds [162]. It also makes distinguishing denial of service attacks from flash crowds more difficult. Indeed, unexpected flash crowds due to user generated contents are hard to predict. The detection and mitigation of such events demand for rich information about the network and geographical distribution of end users.

The unique characteristics of mobile networks, e.g., battery usage of mobile devices and low bandwidth, make the protection of mobile content delivery more challenging, where the resource efficiency of security mechanisms become a compulsion.

B. Software Defined Security

Software Defined Networking (SDN) and Network Functions Virtualization (NFV) are transforming computer networks. SDN decouples the network control and data planes, where a logically centralized controller programs network switches to route data packets. NFV decouples network functions, e.g., firewalls and proxies, from underlying hardware and implements them in software, e.g., containers and virtual machines, that run on commodity hardware.

SDN and NFV enable more effective and flexible security solutions compared to traditional approaches based on scrubbing centers (i.e., dedicated locations with physical scrubber servers for traffic inspection) and hardware solutions. SDN and NFV pave the way for low overhead network monitoring, more flexible attack detection, and elastic deployment of security functions, to name a few [165]–[171].

1) Secure edge computing: Software defined security is more effective in securing CDNs. Edge servers are deployed in points of presence that are typically collocation facilities with limited power and expensive physical space [169], [172], [173]. ISP providers deploy micro datacenters (i.e., racks of commodity servers), radio access networks [174], and edge clouds (i.e., cloud infrastructures in close proximity to end users) in their networks to deliver contents and provide infrastructure as a service to third parties [174]–[176]. High capital expenditures for scarce physical space at the edge incent minimizing hardware and promote softwarization. This calls for lightweight, on-demand security functions, and their management and orchestration at the CDN edge.

2) Quality of Protection vs. Quality of Service: Some security mechanisms require high computational resources, while security functions and delivery services can be co-located, i.e., share the same physical edge server resources. For example, DNSSEC has not been widely adopted due to high computational costs, as discussed in Section IV-C5. Moreover, Cloudflare deploys firewall to inspect traffic at the same server running web proxies to serve content [177].

Sharing resources for content delivery and security purposes entail an important tradeoff between the quality of protection (QoP) [178] and QoS. Security configurations, such as the number of rules in a firewall impact QoP. On the other hand, QoS is impacted by the overhead on traffic, such as increased delay due to security functions, making less resources available for content delivery.

In deploying security services, CDNs must balance QoP and QoS requirements. For example, with on-demand deployment of security services, more resources can be dedicated to content delivery in the absence of threats [179]. Moreover, applying heavyweight inspection on all traffic increases QoP, while decreasing QoS by introducing high latency. To reduce such overheads, heavyweight inspection can be applied only to suspicious traffic [179].

C. Collaborative Security

With a lack of adequate collaboration among parties involved in content delivery, vital intelligence to mitigate security challenges is scattered across multiple parties that perform solitary security actions. Attackers exploit this limitation to use one party's resources to amplify and reflect attack traffic that overwhelm the resources of other parties (e.g., random string denial of service and forwarding loop attacks discussed in Section III-C1 and Section IV-C1, respectively).

1) Collaboration of CDNs and content owners: A CDN and content owners can collaborate to more effectively detect and mitigate attacks that exploit interactions between the CDN and origin servers. This enables countermeasures that are jointly performed at edge servers and origin servers.

For example, the detection of a random string attack is easier at the origin than at the edge, while mitigation can be performed more effectively at the edge. Origin servers can detect this attack and inform edge servers to block malicious IP addresses at the edge.

2) Collaboration of a CDN and ISP providers: With ISPs collaboration, the CDN defense can be even pushed closer to attack sources [180]. CDNs can inform an ISP network to filter malicious traffic at its ingress points. This not only reduces resource usages at the ISP premises, but also saves those of the CDN and content owners. ISP networks also have access to the edge information (e.g., real time network load information and end user locations), which a CDN might not have access to. A CDN can perform more effective protection by accessing this information.

CDNs and ISPs have collaborated to improve content delivery. However, collaborative mechanisms are needed to improve security. CDNs have deployed their servers in ISP networks (e.g., Google Global Cache and Netflix OpenConnect [45], [181]). Moreover, there are communication channels allowing ISPs and CDNs to exchange request routing recommendations [182]–[185]. A CDN and network service providers can also collaborate to jointly manage the network using SDN abstractions [186].

3) Collaboration of CDNs: Multiple CDNs can collaborate to bring together their scattered knowledge and resources for more effective countermeasures against common security threats. Smaller CDNs have collaborated to compete with major CDNs, such as Akamai and Cloudflare [187], [188]. For example, a federated CDN, e.g., EdgeCast, combines CDNs of multiple network service providers to deliver contents through multiple ISP networks [189]–[191]. Moreover, Content Delivery Interaction defines solutions to issues arising in collaboration, e.g., in delivery pricing, consistent service level agreements, and monitoring [187], [188].

Improving security requires collaboration in different levels from standardization to operation. As discussed in Section IV-C1, major CDNs have recently collaborated in a standardization effort to mitigate the common threat of forwarding loop attacks [125]. However, it is the first step, and mitigation is effective only if all CDNs implement this extension.

4) Collaboration incentivisation: A party involved in content delivery embraces collaboration that is in its benefit. For example, protecting edge servers incentivise a CDN to collaborate with another CDN in mitigating a forwarding loop attack that targets both CDNs. However, another party may have no obvious incentives to collaborate with a CDN, while the collaboration is still in benefit of the CDN. For example, an end user might not be interested to peer with another end user to deliver content, while this could alleviate the load of a CDN that is under denial of service attack.

Incentivisation strategies are required to motivate other parties to collaborate with a CDN to improve CDN security. Blockchain is promising to create secure incentivisation. Blockchains have been used to create monetary incentives in content delivery, where a peer proves delivery of content and receives payment in cryptocurrency from content owners and CDN providers [192].

VII. Conclusion

CDNs provide an infrastructure to deliver Internet contents to end users and ensure content availability. However, CDNs are vulnerable to security threats that affect CDN services and end user experience. Adversaries can also weaponize CDN resources to launch more sophisticated attacks against end users and origin servers.

This paper provides a comprehensive survey of CDN security. Specifically, we categorized CDN security challenges based on its infrastructure components, discussed their attack detection and mitigation approaches, and presented our insights and promising directions for future research. We hope that this survey will provide a better understanding of CDN security challenges and pave the way for future research in this direction.

References

[1] T. Barnett, S. Jain, U. Andra, and T. Khurana, *Cisco visual networking index (VNI) complete forecast update, 2017–2022*, https://bit.ly/2KvbhWL, 2018.

[2] Cisco Systems, *Cisco visual networking index: Forecast and trends, 2017–2022 – white paper*, https://davidellis.ca/wp-content/uploads/2019/05/cisco-vni-feb2019.pdf, 2019.

[3] Mordor Intelligence, *Content delivery network (cdn) market - growth, trends, and forecast (2020 - 2025)*, https://www.mordorintelligence.com/industry-reports/content-delivery-market, 2019.

[4] Cloudflare, *How cloud are's architecture can scale to stop the largest attacks*, https://bit.ly/2VpvXpC, 2017.

[5] Akamai, *Akamai's [state of the internet] / security q3 2016 report*, https://bit.ly/2Kn2N44.

[6] C. Cimpanu, *Cpdos attack can poison cdns to deliver error pages instead of legitimate sites*, https://zd.net/34Pturk, 2019.

[7] Cimpanu, Catalin, *Web cache deception attacks still impact websites with substantial user populations*, https://zd.net/2KlyydI, 2019.

[8] A. TECHNEWS, *Cpdos attack can poison cdns to deliver error pages instead of legitimate sites*, https://bit.ly/2VmMhr1, 2019.

[9] Z. Doffman, *Russia and china 'hijack' your internet traffic: Here's what you do*, https://bit.ly/3brjcAn, 2020.

[10] S. Triukose, Z. Al-Qudah, and M. Rabinovich, "Content delivery networks: Protection or threat?", in *European Symposium on Research in Computer Security*, Springer, 2009, pp. 371–389.

[11] O. Gil, *Web cache deception attack*, https://bit.ly/3bvPb2s, 2017.

[12] A. Vakali and G. Pallis, "Content delivery networks: Status and trends", *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, Nov. 2003.

[13] M. PATHAN, "A taxonomy of cdns", *Content Delivery Netowrks, LNEE*, vol. 9, 2008.

[14] Z. Lu, Y. Wang, and Y. R. Yang, "An analysis and comparison of cdn-p2p-hybrid content delivery system and model", *JOURNAL OF COMMUNICATIONS (JCM)*, 2012.

[15] N. Anjum, D. Karamshuk, M. Shikh-Bahaei, and N. Sastry, "Survey on peer-assisted content delivery networks", *Comput. Netw.*, vol. 116, no. C, pp. 79–95, Apr. 2017.

[16] B. Zolfaghari, G. Srivastava, S. Roy, H. R. Nemati, F. Afghah, T. Koshiba, A. Razi, K. Bibak, P. Mitra, and B. K. Rai, "Content delivery networks: State of the art, trends, and future roadmap", *ACM Comput. Surv.*, vol. 53, no. 2, Apr. 2020.

[17] B. Frank, I. Poese, G. Smaragdakis, A. Feldmann, B. M. Maggs, S. Uhlig, V. Aggarwal, and F. Schneider, "Collaboration opportunities for content delivery and network infrastructures", *ACM SIGCOMM ebook on Recent Advances in Networking*, vol. 1, Aug. 2013.

[18] Q. Jia, R. Xie, T. Huang, J. Liu, and Y. Liu, "The collaboration for content delivery and network infrastructures: A survey", *IEEE Access*, vol. 5, pp. 18 088–18 106, 2017.

[19] A. Passarella, "A survey on content-centric technologies for the current internet: Cdn and p2p solutions", *Computer Communications*, vol. 35, no. 1, pp. 1–32, 2012.

[20] G. Peng, "CDN: content distribution network", *CoRR*, vol. cs.NI/0411069, 2004.

[21] J. Sahoo, M. A. Salahuddin, R. Glitho, H. Elbiaze, and W. Ajib, "A survey on replica server placement algorithms for content delivery networks", *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 1002–1026, 2016.

[22] M. A. Salahuddin, J. Sahoo, R. Glitho, H. Elbiaze, and W. Ajib, "A survey on content placement algorithms for cloud-based content delivery networks", *IEEE Access*, vol. 6, pp. 91–114, 2018.

[23] G. Carofiglio, G. Morabito, L. Muscariello, I. Solis, and M. Varvello, "From content delivery today to information centric networking", *Computer Networks*, vol. 57, no. 16, pp. 3116–3127, 2013, Information Centric Networking.

[24] I. Lazar and W. Terrill, "Exploring content delivery networking", *IT Professional*, vol. 3, no. 4, pp. 47–49, Jul. 2001.

[25] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications", *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010.

[26] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative caching: Using remote client memory to improve file system performance", in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '94, Monterey, California: USENIX Association, 1994.

[27] N. Bartolini, E. Casalicchio, and S. Tucci, "A walk through content delivery networks", in *Performance Tools and Applications to Networked Systems: Revised*

Tutorial Lectures. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–25.

[28] E. Kosonen, "Video content delivery over the internet; videosisällön jakelu internetin välityksellä", en, G2 Pro gradu, diplomityö, Aalto University School of Electrical Engineering, 2016, pp. 49+9.

[29] Apache, *Traffic server components*, https://bit.ly/3boWbhf, 2020.

[30] NGINX, *Cache placement strategies for nginx and nginx plus*, https://bit.ly/2VokoyH, 2016.

[31] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies", *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, Dec. 2003.

[32] M. H. Kabir, E. G. Manning, and G. C. Shoja, "Request-routing trends and techniques in content distribution network", in *Proceedings of the International Conference on Computing and Information Technologies (ICCIT02)*, 2002.

[33] F. Wang, J. Liu, M. Chen, and H. Wang, "Migration towards cloud-assisted live media streaming", *IEEE/ACM Transactions on networking*, vol. 24, no. 1, pp. 272–282, 2014.

[34] M. A. Salahuddin, H. Elbiaze, W. Ajib, and R. Glitho, "Social network analysis inspired content placement with QoS in cloud based content delivery networks", in *IEEE Global Communications Conference*, 2015, pp. 1–6.

[35] C. Papagianni, A. Leivadeas, and S. Papavassiliou, "A cloud-oriented content delivery network paradigm: Modeling and assessment", *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 287–300, 2013.

[36] N. Carlsson, D. Eager, A. Gopinathan, and Z. Li, "Caching and optimized request routing in cloud-based content delivery systems", *Performance Evaluation*, vol. 79, pp. 38–55, 2014.

[37] X. Guan and B.-Y. Choi, "Push or pull? toward optimal content delivery using cloud storage", *Journal of Network and Computer Applications*, vol. 40, pp. 234–243, 2014.

[38] M. Tsimelzon, B. Weihl, J. Chung, D. Frantz, J. Brasso, C. Newton, M. Hale, L. Jacobs, and C. O'Connell, *Esi language specification 1.0. world wide web consortium (w3c)*, https://www.w3.org/TR/esi-lang, 2016.

[39] T. Li and R. Atkinson, "Intermediate system to intermediate system (is-is) cryptographic authentication", RFC Editor, RFC 3567, Jul. 2003, pp. 1–6.

[40] A. Barbir, B. Cain, R. Nair, and O. Spatscheck, "Known content network (cn) request-routing mechanisms", RFC Editor, RFC 3568, Jul. 2003, pp. 1–19.

[41] C.-S. Yang and M.-Y. Luo, "Efficient support for content-based routing in web server clusters", in *Proceedings of the 2Nd Conference on USENIX Symposium on Internet Technologies and Systems - Volume 2*, ser. USITS'99, Boulder, Colorado: USENIX Association, 1999, pp. 20–20.

[42] S. Hao, Y. Zhang, H. Wang, and A. Stavrou, "End-users get maneuvered: Empirical analysis of redirection hijacking in content delivery networks", in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, 2018, pp. 1129–1145.

[43] Cloudflare, *Universal dnssec: Secure your domain against dns vulnerabilities, for free.* https://bit.ly/3aqV9QK,

[44] Akamai, *What is dnssec?*, https://bit.ly/34P9wx4,

[45] Netflix, *Netflix open connect*, https://openconnect.netflix.com.

[46] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting video server selection strategies in the youtube cdn", in *2011 31st International Conference on Distributed Computing Systems*, Jun. 2011, pp. 248–257.

[47] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of google's serving infrastructure", in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13, Barcelona, Spain: ACM, 2013, pp. 313–326.

[48] Akamai, *Request router: A software-based routing and redirection service within a comprehensive cdn solution*, https://goo.gl/mMPV69, 2014.

[49] A. Flavel, P. Mani, D. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev, "Fastroute: A scalable load-aware anycast routing architecture for modern cdns", in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA: USENIX Association, 2015, pp. 381–394.

[50] I. Bermudez, S. Traverso, M. Mellia, and M. Munafò, "Exploring the cloud from passive measurements: The amazon aws case", in *2013 Proceedings IEEE INFOCOM*, Apr. 2013, pp. 230–234.

[51] Velocix, *Velocix content delivery network*, https://velocix.com.

[52] L. Networks, *Origin storage: The next level of delivery optimization*, https://bit.ly/2VLc8b1, 2014.

[53] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: Seeing the forest for the trees", in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X, Cambridge, Massachusetts: Association for Computing Machinery, 2011.

[54] E. G. AbdAllah, H. S. Hassanein, and M. Zulkernine, "A survey of security attacks in information-centric networking", *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1441–1454, 2015.

[55] R. Tourani, S. Misra, T. Mick, and G. Panwar, "Security, privacy, and access control in information-centric networking: A survey", *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 566–600, Jan. 2018.

[56] R. Johari and P. Sharma, "A survey on web application vulnerabilities (sqlia, xss) exploitation and security engine for sql injection", in *2012 International*

Conference on Communication Systems and Network Technologies, May 2012, pp. 453–458.

[57] V. K. Malviya, S. Saurav, and A. Gupta, "On security issues in web applications through cross site scripting (xss)", in *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, vol. 1, Dec. 2013, pp. 583–588.

[58] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song, "Towards a formal foundation of web security", in *2010 23rd IEEE Computer Security Foundations Symposium*, Jul. 2010, pp. 290–304.

[59] J. Fonseca, N. Seixas, M. Vieira, and H. Madeira, "Analysis of field data on web security vulnerabilities", *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 2, pp. 89–100, Mar. 2014.

[60] L. Wang, K. S. Park, R. Pang, V. Pai, and L. Peterson, "Reliability and security in the codeen content distribution network", in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '04, Boston, MA: USENIX Association, 2004, p. 14.

[61] A. Cruz and A. Singh, *Cloudflare's protection against a new remote code execution vulnerability (cve-2019-16759) in vbulletin*, https://bit.ly/3dBc6e8.

[62] O. Tripp, M. Pistoia, P. Cousot, R. Cousot, and S. Guarnieri, "Andromeda: Accurate and scalable security analysis of web applications", in *Fundamental Approaches to Software Engineering*, V. Cortellessa and D. Varró, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 210–225.

[63] IBM, *Ibm security appscan standard: Scan and analyze results*, https://ibm.co/2RTB1jI, 2020.

[64] Limelight, *Limelight web application firewall*, https://bit.ly/2VIzY7f.

[65] B. Cohen, *Alibaba cloud mobile security service is an online mobile application security service that protects applications from potential risks, threats and vulnerabilities.* https://bit.ly/3ml8Oie.

[66] Imperva, *Web application firewall (waf) — application security — incapsula*, https://bit.ly/3cD0GoT.

[67] Fastly, *Ddos mitigation and protection*, https://bit.ly/2VITu3I.

[68] Imperva, *Imperva incapsula ddos protection*, https://bit.ly/3brfcj8, 2020.

[69] S. Prandl, M. Lazarescu, and D.-S. Pham, "A study of web application firewall solutions", in *Information Systems Security*, S. Jajoda and C. Mazumdar, Eds., Cham: Springer International Publishing, 2015, pp. 501–510.

[70] O. Foundation, *Owasp*, https://www.owasp.org/.

[71] OWASP, *Owasp top ten*, https://owasp.org/www-project-top-ten/, Accessed: 2020-04-02.

[72] R. Bendrath and M. Mueller, "The end of the net as we know it? deep packet inspection and internet governance", *New Media & Society*, vol. 13, no. 7, pp. 1142–1160, 2011.

[73] N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, N. Weaver, and V. Paxson, "Beyond the radio: Illuminating the higher layers of mobile networks", in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '15, Florence, Italy: Association for Computing Machinery, 2015, pp. 375–387.

[74] J. Jarmoc and D. Unit, "Ssl/tls interception proxies and transitive trust", *Black Hat Europe*, 2012.

[75] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "s" in https", in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14, Sydney, Australia: Association for Computing Machinery, 2014, pp. 133–140.

[76] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis", *Netw.*, vol. 25, no. 5, pp. 355–374, Sep. 2015.

[77] P. V. Amoli and T. Hämäläinen, "A real time unsupervised nids for detecting unknown and encrypted network attacks in high speed network", in *2013 IEEE International Workshop on Measurements Networking (M N)*, Oct. 2013, pp. 149–154.

[78] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic", *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 213–226, Aug. 2015.

[79] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable garbled circuits and succinct functional encryption", in *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '13, Palo Alto, California, USA: Association for Computing Machinery, 2013, pp. 555–564.

[80] C. Gentry, "Fully homomorphic encryption using ideal lattices", in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '09, Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 169–178.

[81] Y. Gao, L. Deng, A. Kuzmanovic, and Y. Chen, "Internet cache pollution attacks and countermeasures", in *Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, ser. ICNP '06, Washington, DC, USA: IEEE Computer Society, 2006, pp. 54–64.

[82] M. Aiello, M. Mongelli, and G. Papaleo, "Basic classifiers for dns tunneling detection", in *2013 IEEE Symposium on Computers and Communications (ISCC)*, Jul. 2013, pp. 000 880–000 885.

[83] A. Klein, "Web cache poisoning attacks", in *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2011, pp. 1373–1373.

[84] S. A. Mirheidari, S. Arshad, K. Onarlioglu, B. Crispo, E. Kirda, and W. Robertson, "Cached and confused: Web cache deception in the wild", in *In Proceedings of the 2020 Network and Distributed System Security Symposium, NDSS*, 2020.

[85] K.-H. Cheung, *Web cache deception attack revisited*, https://goo.gl/KHdeaj, 2018.

[86] B. Brown, *On web cache deception attacks*, https://goo.gl/HSeYNg, 2017.

[87] I. Mubarok, K. Lee, S. Lee, and H. Lee, "Lightweight resource management for ddos traffic isolation in a cloud environment", in *ICT Systems Security and Privacy Protection: 29th IFIP TC 11 International Conference, SEC 2014, Marrakech, Morocco, June 2-4, 2014. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 44–51.

[88] L. Deng, Y. Gao, Y. Chen, and A. Kuzmanovic, "Pollution attacks and defenses for internet caching systems", *Computer Networks*, vol. 52, no. 5, pp. 935–956, 2008.

[89] V. Manivel, M. Ahamad, and H. Venkateswaran, "Attack resistant cache replacement for survivable services", in *Proceedings of the 2003 ACM Workshop on Survivable and Self-regenerative Systems: In Association with 10th ACM Conference on Computer and Communications Security*, ser. SSRS '03, Fairfax, VA: ACM, 2003, pp. 64–71.

[90] H. Park, I. Widjaja, and H. Lee, "Detection of cache pollution attacks using randomness checks", in *2012 IEEE International Conference on Communications (ICC)*, Jun. 2012, pp. 1096–1100.

[91] J. Kettle, *Practical web cache poisoning*, https://bit.ly/39nCTab.

[92] J. Levine, *How cloudflare protects customers from cache poisoning*, https://bit.ly/2JzopKd, 2018.

[93] J. Liebow-Feeser, *Understanding our cache and the web cache deception attack*, https://bit.ly/3cbmw3n, 2017.

[94] M. McDowell, *Understanding denial-of-service attacks*, https://goo.gl/WCfNMT, 2013.

[95] H. V. Nguyen, L. L. Iacono, and H. Federrath, "Your cache has fallen: Cache-poisoned denial-of-service attack", in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 1915–1936.

[96] Akamai, *Cpdos poisoning attack*, https://bit.ly/2vEqcu4, 2019.

[97] R. Lalkaka, *Cloudflare response to cpdos exploits*, https://bit.ly/2U4Tnzy, 2019.

[98] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Semantics and content", RFC Editor, RFC 7231, Jun. 2014, pp. 1–101.

[99] Y. Wang, Y. Shen, X. Jiao, T. Zhang, X. Si, A. Salem, and J. Liu, "Exploiting content delivery networks for covert channel communications", *Computer Communications*, vol. 99, pp. 84–92, 2017.

[100] Y. Desmedt, "Covert channels", in *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2011, pp. 265–266.

[101] E. Union, *General data protection regulation*, https://gdpr-info.eu, 2016.

[102] California Legislature, State of California, *Ab-375 privacy: Personal information: Businesses (2017-2018)*, https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375, 2018.

[103] L. Grangeia, "Dns cache snooping", Independent, Tech. Rep., 2004.

[104] S. Son and V. Shmatikov, "The hitchhiker's guide to dns cache poisoning", in *Security and Privacy in Communication Networks*, S. Jajodia and J. Zhou, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 466–483.

[105] Imperva, *Dns flood*, https://bit.ly/2XS9Qtx.

[106] M. Lepinski and S. Kent, "An infrastructure to support secure internet routing", RFC Editor, RFC 6480, Feb. 2012, pp. 1–24.

[107] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (s-bgp)", *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 582–592, 2000.

[108] F. Zou, S. Zhang, B. Pei, L. Pan, L. Li, and J. Li, "Survey on domain name system security", in *2016 IEEE First International Conference on Data Science in Cyberspace (DSC)*, Jun. 2016, pp. 602–607.

[109] H. Shulman and M. Waidner, "Towards security of internet naming infrastructure", in *Computer Security – ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*. Cham: Springer International Publishing, 2015, pp. 3–22.

[110] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, "A survey of naming and routing in information-centric networks", *IEEE Communications Magazine*, vol. 50, no. 12, pp. 44–53, Dec. 2012.

[111] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford, "A survey of bgp security issues and solutions", *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, 2010.

[112] G. Huston, M. Rossi, and G. Armitage, "Securing bgp — a literature survey", *IEEE Communications Surveys Tutorials*, vol. 13, no. 2, pp. 199–222, 2011.

[113] A. Mitseva, A. Panchenko, and T. Engel, "The state of affairs in bgp security: A survey of attacks and defenses", *Computer Communications*, vol. 124, pp. 45–60, 2018.

[114] S. Venkatesan, M. Albanese, K. Amin, S. Jajodia, and M. Wright, "A moving target defense approach to mitigate ddos attacks against proxy-based architectures", in *2016 IEEE Conference on Communications and Network Security (CNS)*, 2016, pp. 198–206.

[115] R. Guo, W. Li, B. Liu, S. Hao, J. Zhang, H. Duan, K. Sheng, J. Chen, and Y. Liu, "Cdn judo: Breaking the cdn dos protection with itself", in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, Internet Society, 2020.

[116] L. Jin, S. Hao, H. Wang, and C. Cotton, "Unveil the hidden presence: Characterizing the backend interface of content delivery networks", in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, 2019, pp. 1–11.

[117] Fastly, *Fastly ip address ranges*, https://api.fastly.com/public-ip-list, 2020.

[118] Cloudflare, *Cloudflare ip ranges*, https://www.cloudflare.com/ips/, 2020.

[119] Cloudfront, *Cloudfront ip address ranges*, https://ip-ranges.amazonaws.com/ip-ranges.json, 2020.

[120] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack", in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 127–141.

[121] J. Chen, J. Jiang, H. Duan, N. Weaver, T. Wan, and V. Paxson, "Host of troubles: Multiple host ambiguities in http implementations", in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Vienna, Austria: ACM, 2016, pp. 1516–1527.

[122] J. Chen, J. Jiang, X. Zheng, H. Duan, J. Liang, K. Lik, T. Wan, and V. Paxson, "Forwarding loop attacks in content delivery networks", in *the 23st Annual Network and Distributed System Security Symposium*, 2016.

[123] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Message syntax and routing", RFC Editor, RFC 7230, Jun. 2014, http://www.rfc-editor.org/rfc/rfc7230.txt.

[124] R. Guo, J. Chen, B. Liu, J. Zhang, C. Zhang, H. Duan, T. Wan, J. Jiang, S. Hao, and Y. Jia, "Abusing cdns for fun and profit: Security issues in cdns' origin validation", in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2018, pp. 1–10.

[125] S. Ludin, M. Nottingham, and N. Sullivan, "Loop detection in content delivery networks (cdns)", RFC Editor, RFC 8586, Apr. 2019, pp. 1–6.

[126] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, "Your state is not mine: A closer look at evading stateful internet censorship", in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17, London, United Kingdom: Association for Computing Machinery, 2017, pp. 114–127.

[127] A. Herzberg and H. Shulman, "Fragmentation considered poisonous, or: One-domain-to-rule-them-all.org", in *2013 IEEE Conference on Communications and Network Security (CNS)*, Oct. 2013, pp. 224–232.

[128] N. Alexiou, S. Basagiannis, P. Katsaros, T. Dashpande, and S. A. Smolka, "Formal analysis of the kaminsky dns cache-poisoning attack using probabilistic model checking", in *2010 IEEE 12th International Symposium on High Assurance Systems Engineering*, Nov. 2010, pp. 94–103.

[129] A. Hubert and R. van Mook, "Measures for making dns more resilient against forged answers", RFC Editor, RFC 5452, Jan. 2009, pp. 1–18.

[130] R. Elz and R. Bush, "Clarifications to the dns specification", RFC Editor, RFC 2181, Jul. 1997.

[131] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, "Increased dns forgery resistance through 0x20-bit encoding: Security via leet queries", in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08, Alexandria, Virginia, USA: ACM, 2008, pp. 211–222.

[132] H. Shulman and M. Waidner, "Fragmentation considered leaking: Port inference for dns poisoning", in *Applied Cryptography and Network Security*, I. Boureanu, P. Owesarski, and S. Vaudenay, Eds., Cham: Springer International Publishing, 2014, pp. 531–548.

[133] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Dns security introduction and requirements", RFC Editor, RFC 4033, May 2005, http://www.rfc-editor.org/rfc/rfc4033.txt.

[134] A. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Protocol modifications for the dns security extensions", RFC Editor, RFC 4035, Mar. 2005, pp. 1–53.

[135] I. A. N. Authority, *Root ksk ceremonies*, https://www.iana.org/dnssec/ceremonies, 2020.

[136] R. Perdisci, M. Antonakakis, X. Luo, and W. Lee, "Wsec dns: Protecting recursive dns resolvers from poisoning attacks", in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, Jun. 2009, pp. 3–12.

[137] M. Geva, A. Herzberg, and Y. Gev, "Bandwidth distributed denial of service: Attacks and defenses", *IEEE Security Privacy*, vol. 12, no. 1, pp. 54–61, Jan. 2014.

[138] F. J. Ryba, M. Orlinski, M. Wählisch, C. Rossow, and T. C. Schmidt, "Amplification and drdos attack defense - A survey and new perspectives", *CoRR*, vol. abs/1505.07892, 2015.

[139] M. Aiello, M. Mongelli, and G. Papaleo, "Dns tunneling detection through statistical fingerprints of protocol messages and machine learning", *Int. J. Commun. Syst.*, vol. 28, no. 14, pp. 1987–2002, Sep. 2015.

[140] E. Winstead, "DNS response rate limiting", in *LISA 2014*, Seattle, WA: USENIX Association, Nov. 2014.

[141] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "Dnssec and its potential for ddos attacks: A comprehensive measurement study", in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14, Vancouver, BC, Canada: ACM, 2014, pp. 449–460.

[142] M. Majkowski and O. Guomundsson, *Deprecating the dns any meta-query type*, https://bit.ly/2VmrdRn, 2015.

[143] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "Measurement and analysis of private key sharing in the https ecosystem", in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Vienna, Austria: Association for Computing Machinery, 2016, pp. 628–640.

[144] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When https meets cdn: A case of authentication in delegated service", in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 67–82.

[145] N. Sullivan, *Keyless ssl: The nitty gritty technical details*, https://bit.ly/2VJBvKl, 2014.

[146] D. Migault, "LURK Protocol for TLS/DTLS1.2 version 1.0", Internet Engineering Task Force, Internet-Draft draft-mglt-lurk-tls-01, Mar. 2017, Work in Progress, 30 pp.

[147] D. Migault, K. J. Ma, R. Salz, S. Mishra, and O. G. de Dios, "LURK TLS/DTLS Use Cases", Internet Engi-

neering Task Force, Internet-Draft draft-mglt-lurk-tls-use-cases-02, Jun. 2016, Work in Progress, 13 pp.

[148] S. Herwig, C. Garman, and D. Levin, "Achieving keyless cdns with conclaves", in *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA: USENIX Association, Aug. 2020.

[149] V. Costan and S. Devadas, "Intel sgx explained", *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[150] LetsEncrypt, *Let's encrypt is a free, automated, and open certificate authority.* https://letsencrypt.org.

[151] A. Levy, H. Corrigan-Gibbs, and D. Boneh, "Stickler: Defending Against Malicious CDNs in an Unmodified Browser", *ArXiv e-prints*, Jun. 2015. arXiv: 1506.04110 [cs.CR].

[152] Y. Gilad, A. Herzberg, M. Sudkovitch, and M. Goberman, "Cdn-on-demand: An affordable ddos defense via untrusted clouds", in *2016 Network and Distributed Systems Symposium*, 2016.

[153] L. Jin, S. Hao, H. Wang, and C. Cotton, "Your remnant tells secret: Residual resolution in ddos protection services", in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2018, pp. 362–373.

[154] T. Vissers, T. Van Goethem, W. Joosen, and N. Nikiforakis, "Maneuvering around clouds: Bypassing cloud-based security providers", in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, Denver, Colorado, USA: ACM, 2015, pp. 1530–1541.

[155] SecurityTrails, *The world's largest repository of historical dns data*, https://securitytrails.com/dns-trails.

[156] C. DNS, *Dns history - largest archive of dns records - domain history*, https://completedns.com/dns-history.

[157] Q. Jia, H. Wang, D. Fleck, F. Li, A. Stavrou, and W. Powell, "Catch me if you can: A cloud-enabled ddos defense", in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Jun. 2014, pp. 264–275.

[158] T. Ylonen and C. Lonvick, "The secure shell (ssh) connection protocol", RFC Editor, RFC 4254, Jan. 2006, pp. 1–24.

[159] T. Butler, *Analysis of a wordpress pingback ddos attack*, https://bit.ly/2VL9OAP, 2016.

[160] G. Shatz, *Wordpress default leaves millions of sites exploitable for ddos attacks*, https://bit.ly/2VIRQz4, 2013.

[161] M. Prince, *Introducing cname flattening: Rfc-compliant cnames at a domain's root*, https://bit.ly/2XFiPz8, 2014.

[162] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, "Practical, real-time centralized control for cdn-based live video delivery", in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, London, United Kingdom: ACM, 2015, pp. 311–324.

[163] Q. Fan, H. Yin, G. Min, P. Yang, Y. Luo, Y. Lyu, H. Huang, and L. Jiao, "Video delivery networks: Challenges, solutions and future directions", *Computers and Electrical Engineering*, vol. 66, pp. 332–341, 2018.

[164] Cisco, *Cisco annual internet report (2018–2023) white paper*, https://bit.ly/2VOW486, 2020.

[165] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement", in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, Oct. 2015, pp. 255–260.

[166] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud", *CoRR*, vol. abs/1305.0209, 2013. arXiv: 1305.0209.

[167] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic ddos defense", in *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX Association, 2015, pp. 817–832.

[168] E. Jalalpour, M. Ghaznavi, D. Migault, S. Preda, M. Pourzandi, and R. Boutaba, "A security orchestration system for cdn edge servers", in *2018 IEEE Conference on Network Softwarization (NetSoft)*, Jun. 2018.

[169] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, "Engineering egress with edge fabric: Steering oceans of content to the world", in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, Los Angeles, CA, USA: ACM, 2017, pp. 418–431.

[170] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering", in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, Los Angeles, CA, USA: ACM, 2017, pp. 432–445.

[171] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow", in *IEEE Local Computer Network Conference*, Oct. 2010, pp. 408–415.

[172] J. T. Araujo, L. Saino, L. Buytenhek, and R. Landa, "Balancing on the edge: Transport affinity without network state", in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, Renton, WA: USENIX Association, 2018, pp. 111–124.

[173] zayo, *Datacenter and collocation*, https://bit.ly/34TCH1K, 2020.

[174] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g", *ETSI White Paper*, 2015.

[175] S. Kuenzer, A. A. Ivanov, F. Manco, J. J. Mendes, Y. Volchkov, F. Schmidt, K. Yasukata, M. Honda, and F. Huici, "Unikernels everywhere: The case for elastic cdns", in *VEE*, 2017.

[176] L. Hardesty, *At&t integrated cloud to include 105 data centers by year's end*, https://bit.ly/2KksH8D, 2016.

[177] J. Graham-Cumming, *No scrubs: The architecture that made unmetered mitigation possible*, https://bit.ly/2W1Y5OF, 2017.

[178] Y. Sun and A. Kumar, "Quality-of-Protection (QoP): A Quantitative Methodology to Grade Security Services", in *International Conference on Distributed Computing Systems Workshops*, 2008, pp. 394–399.

[179] E. Jalalpour, M. Ghaznavi, D. Migault, S. Preda, M. Pourzandi, and R. Boutaba, "Dynamic security orchestration for cdn edge-servers", in *2018 IEEE Conference on Network Softwarization (NetSoft)*, Jun. 2018.

[180] J. François, I. Aib, and R. Boutaba, "Firecol: A collaborative protection network for the detection of flooding ddos attacks", *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1828–1841, Dec. 2012.

[181] Google, *Google global cache program*, http://ggcadmin.google.com/ggc, 2020.

[182] I. Poese, B. Frank, B. Ager, G. Smaragdakis, and A. Feldmann, "Improving content delivery using provider-aided distance information", in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10, Melbourne, Australia: ACM, 2010, pp. 22–34.

[183] R. Alimi, R. Penno, Y. Yang, S. Kiesel, S. Previdi, W. Roome, S. Shalunov, and R. Woundy, "Application-layer traffic optimization (alto) protocol", RFC Editor, RFC 7285, Sep. 2014, pp. 1–91.

[184] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, "P4p: Provider portal for applications", in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08, Seattle, WA, USA: Association for Computing Machinery, 2008, pp. 351–362.

[185] E. Pujol, I. Poese, J. Zerwas, G. Smaragdakis, and A. Feldmann, "Steering hyper-giants' traffic at scale", in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19, Orlando, Florida: Association for Computing Machinery, 2019, pp. 82–95.

[186] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "Sdx: A software defined internet exchange", in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14, Chicago, Illinois, USA: ACM, 2014, pp. 551–562.

[187] B. Niven-Jenkins, F. L. Faucheur, and N. Bitar, "Content distribution network interconnection (cdni) problem statement", RFC Editor, RFC 6707, Sep. 2012, pp. 1–32.

[188] J. Seedorf, J. Peterson, S. Previdi, R. van Brandenburg, and K. Ma, "Content delivery network interconnection (cdni) request routing: Footprint and capabilities semantics", RFC Editor, RFC 8008, Dec. 2016, pp. 1–31.

[189] J. Yao, H. Zhou, J. Luo, X. Liu, and H. Guan, "Comic: Cost optimization for internet content multihoming", *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 7, pp. 1851–1860, Jul. 2015.

[190] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian, "Optimizing cost and performance for content multihoming", in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12, Helsinki, Finland: ACM, 2012, pp. 371–382.

[191] J. Broberg, R. Buyya, and Z. Tari, "Metacdn: Harnessing 'storage clouds' for high performance content delivery", *Journal of Network and Computer Applications*, vol. 32, no. 5, pp. 1012–1022, 2009, Next Generation Content Networks.

[192] P. Goyal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "Secure incentivization for decentralized content delivery", in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, Renton, WA: USENIX Association, Jul. 2019.

Milad Ghaznavi received the PhD degree in computer science from the University of Waterloo, Canada in 2020. His research interests include distributed systems and computer networks. He is a recipient of David R. Cheriton Graduate Scholarship at the University of Waterloo.

Elaheh Jalalpour graduated from Master's of Computer Science at the University of Waterloo in 2018. Her research interest includes applied machine learning. She received Microsoft Azure Champ Prize at the University of Toronto in 2019. She was a finalist as a member of the SafeTrip team in Microsoft Imagine Cup in 2019.

Mohammad A. Salahuddin received the M.Sc. and Ph.D. degrees in Computer Science from Western Michigan University in 2003 and 2014, respectively. He was a Postdoctoral Researcher with the Université du Québec à Montréal and University of Waterloo, and a Visiting Scientist with Concordia University. He is currently a Research Assistant Professor of Computer Science at the University of Waterloo. His research interests include the Internet of Things, content delivery networks, network softwarization, network security, machine learning, and cognitive network management. He serves as a TPC member for international conferences and is a reviewer for various journals and magazines.

Raouf Boutaba received the M.Sc. and Ph.D. degrees in computer science from Sorbonne University in 1990 and 1994, respectively. He is currently a University Chair Professor and the Director of the David R. Cheriton School of Computer science at the University of Waterloo (Canada). He also holds an INRIA International Chair in France. He is the founding Editor-in-Chief of the IEEE Transactions on Network and Service Management (2007-2010) and the current Editor-in-Chief of the IEEE Journal on Selected Areas in Communications. He is a fellow of the IEEE, the Engineering Institute of Canada, the Canadian Academy of Engineering, and the Royal Society of Canada.

Daniel Migault is a member of the Ericsson Research Security team. He is actively involved in standardizing security protocols at the IETF. He received the PhD degree in Telecom and Security from TELECOM SudParis, France in 2012.

Stere Preda is a Senior Researcher in Security at Ericsson, Canada. He received his PhD in computer science from TELECOM Bretagne, France. His current research focus is Network Function Virtualization (NFV) security. He is an active contributor to ETSI NFV security standardization.