Temporal CDN-Convex Lens: A CDN-Assisted Practical Pulsing DDoS Attack

Run Guo[1], Jianjun Chen[1,2]*, Yihang Wang [1], Keran Mu[1],
Baojun Liu[1,2], Xiang Li[1], Chao Zhang[1,2], Haixin Duan[1,2,3], Jianping Wu[1,2]
[1]Tsinghua University. [2]Zhongguancun Laboratory. [3]QI-ANXIN Technology Research Institute.

Abstract

As one cornerstone of Internet infrastructure, Content Delivery Networks (CDNs) work as a globally distributed proxy platform between clients and websites, providing the functionalities of speeding up content delivery, offloading web traffic, and DDoS protection. In this paper, however, we reveal that the inherent nature of the CDN forwarding network can be exploited to compromise service availability.

We present a new class of pulsing denial of service attack, named CDN-Convex attack. We explore the possibility of exploiting the CDN infrastructure as a converging lens, and *concentrating* low-rate attacking requests into short, high-bandwidth pulse waves, resulting in a pulsing DDoS attack to saturate the targeted TCP services periodically. Through real-world experiments on five leading CDN vendors, we demonstrate that the CDN-Convex attack is practical and flexible. We show that attackers can use it to achieve peak bandwidths over *1000 times* greater than their upload bandwidth, seriously degrading the performance and availability of target services. Following the responsible disclosure policy, we report our attack details to all affected CDN vendors and propose possible mitigation solutions.

1 Introduction

Denial of Service (DoS) attack is considered one of the most popular security threats in cyberspace [32]. Adversaries perform DoS attacks to consume critical resources of a network or a web server, and prevent or severely degrade online services to serve legitimate Internet users. Despite significant efforts have been devoted to research prevention techniques to mitigate flooding traffic, detecting and blocking DoS attacks is an ever-changing cat-and-mouse game between service providers and attackers. As a result of the constant confrontation and evolution, more sophisticated DoS strategies are proposed.

Among these strategies, the *Pulsing DoS attack*, consisting of a series of *short-lived bursts* that occur in clockwork-like succession, serves as a new type of *low-rate* DDoS attack, but it is more powerful and could cause more damages [36, 40]. Actually, the pulsing DoS attack has been studied for years and presented impressive impacts [33, 36, 40, 41, 49, 51]. First, the pulsing DoS attack can severely degrade TCP throughput and lead to a repeated TCP re-transmission timeout [36, 40]. Second, due to the fundamental susceptibility of bottleneck resources within the service or protocol, the security community believes that any application with limited resources is vulnerable to pulsing DoS attack [53]. Last, compared with a conventional volume-based DoS attack that floods the victim with a huge number of connections, a pulsing DoS attack only generates a suite of low-volume sequential requests, consequently which is more efficient and stealthier.

Research Gap. However, traditional pulsing DoS attack generally relies on large-scale botnets, which requires attackers to compromise a number of devices (e.g., IoT devices) [5, 58]. Moreover, these compromised bot nodes can be of varying bandwidth and synchronization. As a result, previous studies demonstrated that it is difficult to coordinate attacking requests from different bots to synchronize as a bursting pulse at the victim server [33, 40, 41]. One notable study proposed the temporal lensing attack and employed Open DNS servers to reflect and generate pulsing DoS traffic [49]. However, the work focuses on UDP amplification and can only achieve a lensing bandwidth gain of about 10, thus the real-world impact of the attack is strictly limited. In addition, CDN infrastructure can also absorb attacking requests and disrupt intermittent pulses, invalidating the above attacks.

Key Observations. In this study, we observe that CDN infrastructure works as a large-scale proxy network, which employs *extremely massive* edge servers around the world. Besides, almost all mainstream CDN vendors *lack sufficient validation* of customer-supplied origin [21]. Thus, a malicious CDN customer can drive millions of globally distributed edge servers launching TCP requests to an arbitrary domain name or IP address. And more importantly, those edge servers are generally located at network backbones [1, 10], which could provide various and *stable network latency*. Therefore, we believe that *the prevalence of CDN edge servers provides an excellent opportunity to be exploited as a "botnet" by an attacker to perform pulsing wave DoS attacks*.

Our Study. Inspired by the above key observations, we propose a CDN-assisted pulsing DDoS attack, which is entitled the *CDN-Convex* attack. Empowered by CDN-Convex, an adversary can periodically saturate target TCP services with concentrated attacking requests that burst in a short period.

*Corresponding author: jianjun@tsinghua.edu.cn

The core concept of our attack is to exploit existing CDN infrastructure (stable proxy network) to *concentrate* attacking requests in time. Global distributed CDN edge servers provide a wide range of attacker-CDN-victim network paths with stable network latency. It allows an adversary to schedule the arrival of attack requests. By first sending requests to paths with longer delays, and then sending requests to those with shorter delays, an adversary can manipulate different requests arriving at the victim server simultaneously within a small time window. In this way, CDN infrastructure works like a convex lens to converge the low-rate attacking requests into short, high-rate pulse waves, resulting in bandwidth concentration effects at the victim.

Despite the above idea being straightforward, in the real world, it is a non-trivial task to achieve a high bandwidth concentration ratio. In this study, we further proposed three advanced techniques to improve the temporal convergence of the basic attack.

The first technique (CDN-Cascading Convex) chains multiple CDN vendors together, the second technique (DNS-holdon Convex) manipulates DNS resolution, and the third technique (Request-pending Convex) leverages incomplete HTTP requests, all aiming to obtain a wider range of time range to fire more attacking requests in one pulsing period. We also discuss the applicability of leveraging the IP fragmentation mechanism to further let the CDN withhold more attacking requests.

Evaluation Results. To evaluate the practical feasibility of our proposed attacks, we select five leading CDN vendors (Akamai, Azure, CloudFront, Cloudflare, and Fastly) and deploy our web server to set up a controlled environment. Throughout this study, we take the utmost care of potential ethical issues. First, the victim's website is registered by ourself. Second, to avoid collateral damage on CDN platforms, we only utilized dozens of CDN nodes to demonstrate the effectiveness of CDN-Convex attack. Actually, we collect millions of available CDN nodes (Table 2 in Section 4.1), and all these nodes can be theoretically employed to conduct CDN-Convex attack.

Our experiments show that it is practical and flexible to exploit CDN infrastructure as a convex lens to temporally converging attacking requests. Moreover, the effectiveness of CDN-Convex attack is also impressive. Using the most serious attacking technique of the proposed three, we obtain a real-world bandwidth concentration ratio of 1526.9, a 72.82KBps attacker-side bandwidth can achieve a total of 108.58MBps victim-side bandwidth, resulting in a pulsing DDoS attack. Although we only take advantage of 64 CDN nodes in our controlled experiments, we present that CDN-Convex attack can severely saturate the target TCP service and damage the targeted website's availability and performance (Section 5).

Contributions. We make the following contributions in this paper.

- *A Novel Attack.* We propose a new class of pulsing DDoS attacks. It exploits global distributed edge servers provided by CDN platforms and converges low-rate attacking requests as a series of high-bandwidth pulses at the victim.
- *Real-world Evaluations.* By performing real-world controlled experiments on five leading CDN vendors, we demonstrate that CDN-Convex attack can reach a peak bandwidth concentration ratio of over 1000, and undermine the performance and availability of target services.
- *Mitigation and Responsible Disclosure.* We present approaches to mitigate the proposed attack and responsibly report vulnerabilities to CDN vendors.

2 Background

2.1 CDN Overview

CDN infrastructure is designed to improve the end-to-end network performance and reduce its origin website servers' workload burden. To enable a CDN service, a website owner needs to delegate its domain name to a CDN vendor. And then, leveraging the flexibility of domain resolution, the CDN vendor adopts the request-routing mechanism and redirects web requests from global Internet end-users to different CDN nodes. Such nodes are located in globally distributed counties. As a result, CDN significantly improves online services' availability by caching and serving customer sites' web resources at a vast number of CDN nodes.

Figure 1 illustrates how a CDN works. To access a CDN-hosting website (origin server), the client needs to resolve the website's domain name to IP addresses first. With the mechanism of domain delegation, the IP addresses (red ones) are actually provided by CDN's authoritative name servers. These IP addresses are generally referred to as *front-end addresses* [30]. The client then establishes a connection with one of the provided front-end addresses to obtain web resources. As for the CDN infrastructure, a CDN node would process the connection and check whether the requested resource is cached. If the web resource is not cached, the CDN node fetches the requested content from the origin server. The IP addresses used to establish with origin server are referred to as *back-end addresses*. Finally, once the CDN node receives the web resource, it transfers the content to the client.

In conclusion, the CDN distributes a massive number of nodes globally, to efficiently redirect the web requests of a CDN-powered website to the closest CDN node instead of directly visiting the origin website server. Moreover, the CDN nodes are generally deployed in the Internet backbones, to provide fast and stable access.

Besides improving the network latency, CDN vendors also provide security-related services for the website servers, such as DDoS protection services. As the widespread DDoS flooding attack is becoming more and more powerful, it is insufficient to merely rely on on-site DoS protection mechanisms. CDN infrastructure becomes an ideal place to absorb DDoS attacks by distributing malicious traffic across many nodes located in different data centers. According to public reports, Akamai has 275,000 servers located in 136 countries [1], and the nodes of Cloudflare are distributed in more than 100 countries [10].

Currently, CDN infrastructure has become one of the cornerstones of the Internet, as more than 38.98% of Alexa Top 100K websites are hosting on the CDN platforms now [29]. In this paper, we also demonstrate that the inherent nature of CDN architecture may be exploited by an attacker to launch a powerful pulsing DoS attack against its hosting customers.

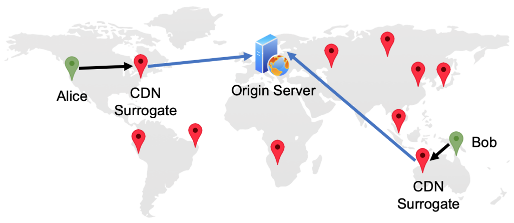


Figure 1: CDN's global distribution and Geo-location based request routing.

2.2 Pulsing DDoS Attacks

Typically, in the starting stage, a common brute-force flooding DDoS attack looks like a steadily growing stream of malicious traffic. Nevertheless, such attacks have been demonstrated to be feasibly detected and mitigated [2, 4, 32]. Continuously evolving, several sophisticated DDoS strategies were proposed. Among these strategies, the *Pulsing DDoS attack*, consisting of a series of short-lived bursts that occur in clockwork-like succession, serves as a new type of low-rate DoS attack, but it enables the attacker to efficiently utilize his attacking bandwidth [33, 49, 51].

Figure 2 illustrates the concept of the pulsing DDoS attack. Unlike traditional traffic flooding attacks, an attacker controlling a botnet can *strategically* generate *a small number* of well-crafted requests (or packets) to the victim. And later, these attacking requests result in intermittent pulses at the victim server with certain periodicity.

Some previous works discussed how to use pulsing requests to trigger the bottleneck of resources within the victim server, which presents impressive experimental applicability [33, 41, 49, 51, 52]. In a word, compared with the flooding DDoS attacks, the Pulsing DDoS attacks are considered to be more *efficient*.

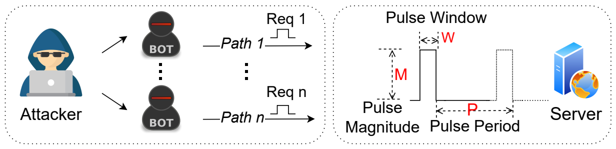


Figure 2: Concept of a pulsing DDoS attack.

However, the pulsing DDoS attack is generally launched from botnets [27, 60]. Unfortunately, the infected devices among a botnet can be of varying network bandwidth and time synchronization, invalidating the efficiency of a pulsing DDoS attack [45]. In addition, for websites hosted and protected by a CDN vendor, the CDN infrastructure is able to absorb attacking requests and disrupt intermittent pulses from the botnet without affecting the availability of online service. Actually, previous research work has demonstrated the difficulty of finding an ideal attacking platform to coordinate the malicious requests as a pulse wave [46].

3 CDN-Convex Attack Overview

As an Internet infrastructure, on the one hand, the CDN provides geo-distributed nodes, facilitating the availability of online services for widespread users and mitigating the traditional DDoS flooding attack to the origin server. However, on the other hand, we find that the CDN could be exploited as a converging lens to launch a novel and powerful pulsing DoS attack against target services.

In this section, we first introduce the threat model of the CDN-Convex attack, then we present the concept and the strategy of the attack.

3.1 Threat Model

In this paper, we assume that an adversary has two limited capabilities. First, the attacker can craft legitimate requests like a benign end-user to geographically distributed front-end addresses *at a slow rate*. Instead of employing a botnet with the capable bandwidth to saturate the victim, the CDN-Convex attack can be initiated from a vantage point, like an IoT device with a slow bandwidth; Second, the attacker can register accounts on CDN providers. The current majority of CDN vendors, presumably for competitive reasons, offer *free* or *free-trial* services to potential customers (and thus for attackers) without strong identity verification [9], thus the attack can be launched anonymously with little cost.

For the victims, any website or TCP service could be affected. Today's CDN operation is overly loose in customer-controlled forwarding policy and lacks origin validation [20] . A malicious CDN customer can configure CDN edge servers to forward traffic to an arbitrary domain name or IP address even if he/she does not own it.

Based on the above threat model, we propose a novel and practical CDN-Convex attack by exploiting the CDN platform as a convex lens to converge attacking requests into short, burst pulse waves (could be within tens of milliseconds). Those pulse waves could be exploited to attack either network infrastructure (e.g., temporally saturate the bandwidth of victims or network paths), TCP implementations at end-host, or web applications with bottleneck resources.

3.2 Concept of the CDN-Convex Attack

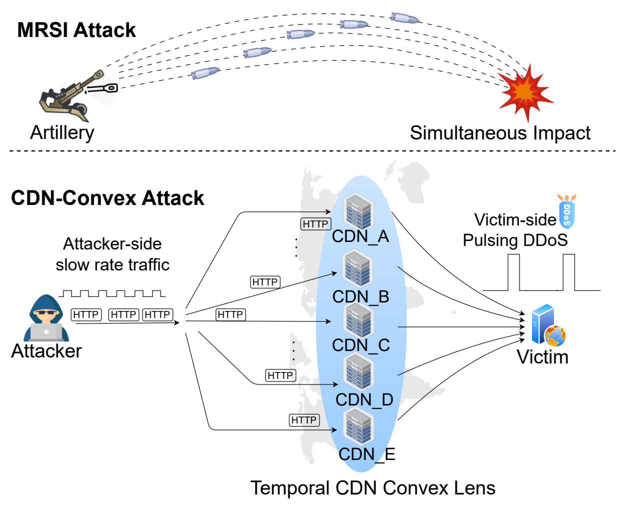


Figure 3: Concept of the CDN-Convex attack.

The core concept of CDN-Convex attack is analogous to the military tactic "Multiple Round Simultaneous Impact (MRSI)" [57]. Single artillery fires multiple shells by varying the angle and propellant charges, so all shells arrive at the same target simultaneously. This is possible because different shells travel different trajectories to fly to a given target. By leveraging this technique, artillery can fire more shells to arrive at the victim in one period of time than it can send in that period.

In this study, we observe that the global accessing mechanism of the CDN platform makes up a request-forwarding platform with various latencies in a wide range. Specifically, the CDN-Convex attack takes advantage of two CDN features. First, the globally distributed CDN nodes can be accessed directly and lack sufficient origin validation. Typically, when a CDN node receives an HTTP request, it inspects the "Host" header to verify whether the requested website is a registered CDN customer website and then forwards the request to the customer-supplied origin. Therefore, this provides an opportunity for an attacker to drive millions of CDN edge services to any TCP services by sending HTTP requests directly to a node's IP address with a CDN-hosting website's domain name in the "Host" header field.

Second, the globally distributed CDN nodes provide a wide range of stable accessing latency. When we send requests from one vantage point (like one laptop) to access the website, the global CDN nodes provide us with many different network paths to the origin server, consequently with a wide range of different network latencies.

Based on the two above characteristics, we present the CDN-Convex attack, as in Figure 3. An attacker can schedule sending in such a way that the attacker first sends requests to the path with the longer latency and then sends requests to those with shorter latency. In this way, an attacker utilizes the various network delays provided by the distributed CDN nodes to facilitate the requests to converge as a series of intermittent pulses at the victim, periodically saturating the victim-side bandwidth as a pulsing DDoS attack.

Compared with previous CDN-related flooding DoS attack [21, 37], we take a new angle on CDN proxy networks, and use them to temporally *concentrate* the arrival of requests at the victim, much like a lens focuses light. Our attack inherits the advantages of traditional pulse DDoS attacks, which are stealthy, efficient, and can cause severe damage at a low cost.

3.3 Strategy of the CDN-Convex Attack

Nevertheless, it is not straightforward to perform a CDN-Convex attack described above. There are myriad crucial technical challenges to conquer. Significantly, given the attacker's bandwidth is limited, it's a non-trivial task to craft and synchronize a maximum number of requests and make them arrive at the origin server simultaneously, to achieve a high bandwidth concentration at the victim.

We identified four attacking techniques to achieve the CDN-Convex, including one basic attack and three enhanced techniques, as follows:

(1) **Basic CDN-Convex Attack,** which constitutes the basic form of CDN-Convex. We measure CDN edge servers and analyze their latency to determine the attack path through CDN to the victim. Then we schedule sending to create maximal lensing from these paths.

(2) **CDN-Cascading Convex Attack.** We cascade multiple CDN vendors together to extend the network transmitting path, thus allowing a wider range of time to fire more attacking requests in one pulsing period.

(3) **DNS-holdon Convex Attack.** We improve the attack by leveraging DNS resolution in edge servers to converge more attacking requests at the victim.

(4) **Request-pending Convex Attack.** We enhance the attack further by carefully crafting and scheduling incomplete HTTP requests to make edge servers hold as many attacking requests as possible before bursting out, which can significantly increase damage.

We evaluate our techniques on five leading CDN vendors (Akamai, Azure, CloudFront, Cloudflare and Fastly) and demonstrate the feasibility of the CDN convex lens. These five CDN vendors are often studied in previous works as well [9, 22, 29, 30, 38]. According to the statistics of CDN market sharing [15, 24], these five CDN vendors are the top players in the market [24]. Besides, we also find that more than 43.4% of Alexa Top 10,000 websites host on these five CDN vendors with our experiments. Table 1 summarizes the CDN vendors vulnerable to the four attack techniques.

Table 1: CDN vendors vulnerable to the CDN-Convex attacks.

	Akamai [†]	Azure	CloudFront	Cloudflare	Fastly
Basic CDN-Convex Attack	✓	✓	✓	✓	✓
CDN-Cascading Convex	✓	✓	✓	✓	✓
DNS-holdon Convex	✓	✓			
Request-pending Convex	✓	✓	✓	✓	✓

[†] Akamai only provides service to enterprise customers, experiments are implemented through Microsoft provided Akamai service subscription.

4 CDN-Convex Attack Techniques

In this section, we will present the four attacking techniques in detail, along with measurements and experiments to assess them.

4.1 Basic CDN-Convex Attack

The Basic CDN-Convex attack can be decomposed into the following steps, which we will develop in detail:

(*i*) collecting as many CDN edge servers as possible, to launch plenty of requests from different front-end addresses with various network delays.

(*ii*) measuring the latencies of the CDN network to determine the attacking path through CDN to the victim.

(*iii*) bypassing the unfriendly CDN cache mechanism, to force the CDN node to request the origin server rather than handling the request by itself.

(*iv*) conducting the attack by sending requests in line with latencies to different attacking paths, these requests are temporally converged by *the CDN convex lens*, leading to a pulsing DDoS attack.

4.1.1 Collecting CDN edge servers

With as many available CDN edge servers as possible, the attacker can send requests to different geo-located IP addresses with various latencies to exploit the CDN as a convex lens.

We use two methods to collect CDN edge servers on the Internet: 1) resolving the CDN-hosting websites through open DNS resolvers [35, 44]; 2) fingerprinting CDN servers by inspecting HTTP headers [20, 28].

Table 2: Number of CDN edge servers being collected.

Vendor	OpenDNS Resolving		HTTP Fingerprinting	
	# IP	# Cities	# IP	# Cities
Akamai	604,810	808	**2,092,719**	673
Azure	6,014	61	**323,528**	103
CloudFront	98,644	31	**726,543**	78
Cloudflare	19,693	154	**717,256**	292
Fastly	7,037	56	**97,190**	58

The result is shown in Table 2, we can see that these millions of CDN edge servers from different CDN vendors all are the candidates which may be employed in a CDN-Convex attack.

4.1.2 Configuring CDN origin

To verify whether a 3-rd party TCP service can be targeted in the CDN-Convex attack, we examined the CDN validation policy on customer-supplied origin, as shown in Figure 3. All CDNs allow the attacker to configure the origin to be any 3rd-party TCP service or website by setting their domain name or IP. Although 4 out of 5 CDN vendors put a limit on the "origin TCP port", we can cascade multiple CDN together to remove the TCP port number limit, as in the form of "attacker → CloudFront → Fastly → origin" (Section 4.2).

Table 3: The CDN "origin" option can be configured as the domain or IP address of any 3rd party TCP service.

	Akamai	Azure	CloudFront	Cloudflare	Fastly
Configure Domain	✓	✓	✓	✓	✓
Configure IP	✓	✓	✓	✓	✓
TCP Port Range	Subset [†]	Subset [†]	Subset [‡]	Subset [‖]	Unspecified [§]

[†] https://learn.microsoft.com/en-us/previous-versions/azure/mt757337(v=azure.100)
[‡] https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/distribution-web-values-specify.html
[‖] https://developers.cloudflare.com/fundamentals/get-started/reference/network-ports/
[§] Unspecified: We haven't find any limit on TCP port range.

4.1.3 Measuring CDN network latency

With sizable CDN nodes, we could request each of those nodes to measure latencies.Theoretically, each node creates an attacker→CDN→victim attacking path with one specific transmission latency. Therefore, to conduct a CDN-Convex attack and obtain stable pulsing traffic at the target website, *we should send the requests in precise sequence according to the network latency.*

In the following, we measure the latency of different CDN-provided network paths in step 1 and choose stable attacking paths in step 2 (with stable path latency).

Step 1: Estimating the request latency. We use two methods to measure latencies depending on whether the victim is a website or other TCP services.

Measurement method 1. If the victim is a website, we can use HTTP requests to measure the latency. The advantage of this method is that we do not have to know the victim's IP. For example, when the victim is a website hosted on the CDN already, the CDN hides the IP address of the website server that can not be attacked directly.

Firstly, we send a request to establish a TCP connection with the CDN node, accordingly, the CDN node will establish a connection to the victim. Then, we send a measurement HTTP request to the node and the request would be transmitted within the existing TCP connection, no more TCP connection is initiated.

As shown in Figure 4, $T_{Forward}$ is the CDN's internal forwarding latency, and T_s is the processing latency of origin server. T_{TTFB} (Time to First Byte) is the pure HTTP request-reply time, and is composed as:

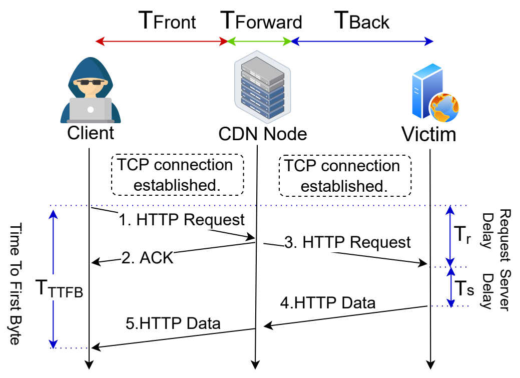


Figure 4: Time diagram of a CDN-forwarded request.

$$T_{\text{TTFB}} = 2 * T_{Front} + 2 * T_{Forward} + 2 * T_{Back} + T_s \quad (1)$$

The `Request Latency` T_r is calculated from the HTTP request packet sending time to the time when the HTTP request packet arrives at the origin as the following:

$$T_r = T_{Front} + T_{Forward} + T_{Back} \quad (2)$$

Compared equation 1 with equation 2, T_r can be calculated as 1/2 of T_{TTFB}, with the error:

$$\begin{aligned} T_{\text{ERR}} &= (2 * T_{Front} + 2 * T_{Forward} + 2 * T_{Back} + T_s) * \frac{1}{2} - \\ &\quad (T_{Front} + T_{Forward} + T_{Back}) \quad (3) \\ &= \frac{1}{2} T_s \end{aligned}$$

If T_{ERR} can be minimized, then a more accurate `Request Latency` T_r can be obtained. T_s depends on the specific victim-side processing time, which varies according to the type of requested resource and the website's loading. To minimize the origin-side processing time T_s, the attacker can send requests of *small resources or non-exist resources* to reduce the origin-side processing time T_s. Moreover, the HTTP "`HEAD`" requests can also be used to reduce origin-side processing time.

Thus, we can conclude the equation 4, and use this it to calculate `Request Latency` T_r for further attacks:

$$T_r \approx \frac{1}{2} * T_{TTFB} \quad (4)$$

Measurement method 2. If the victim is a TCP service, we should know the victim's IP address. Measuring latencies between two Internet end hosts is a well-studied problem [17, 19,43]. We can use "Ping" method to measure the "Attacker → CDN" latency $\boldsymbol{T}_{Front}$, and use the "DNS King" method [19] to estimate the "CDN → Victim" latency $\boldsymbol{T}_{Back}$. $T_{Forward}$ is the CDN internal forwarding latency which the CDN endeavors to minimize, and it's normally stable within milliseconds through measurements.

The network instability could also have a negative impact on the attack, such as anycast routing [7]. However, previous research [56] shows that for 99% of measured Internet anycast routes are stable for hours, days, or longer. To minimize those negative effects, an attacker can measure the latencies of different paths for multiple rounds before the actual attack to choose the stable ones, as follow.

Step 2: Choosing stable attacking paths. To choose usable attacking paths, we measure the T_r of the five CDN vendors. *Abundant stable attacking paths.* To characterize each Client-CDN-Origin path's delay stability, we calculate the *STD* (standard deviation) value of 30 measured delays in each path.

Results are shown in Figure 5. CDN vendors provide a wide range of latencies from tens of milliseconds to more than 1 second. However, the path stability of CDN vendors differs. First, from the axis Y of Figure 5a, latencies of most Akamai-provided paths have a standard deviation within 0.1 seconds, which shows that Akamai provides the most stable path latency. Although Fastly provides a wide range of latencies to almost 3 seconds in Figure 5e, the standard deviation of latencies shows that Fastly-provided paths are unstable compared with other CDN vendors.

However, by multiplying the *STD* ratio with the millions of CDN IP addresses we found in Table 2, we can see that *more than hundreds of thousands of stable paths can be chosen to send attacking requests.*

4.1.4 Bypassing CDN Cache

To make successive pulse waves at the victim, the existence of the CDN cache may stop attacking requests being forwarded to the victim. When targeting a non-website TCP service, the victim will not return a cache-able HTTP response, so the CDN cache mechanism has no impact on the attack. When targeting a website, the victim may return cache-able HTTP responses. To ensure the attacking requests arrive at the victim rather than hitting the CDN cache, we need to detour *the CDN cache mechanism.*

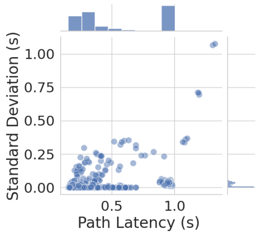
After exploring the CDN forwarding strategies and working mechanism exhaustively, we conclude several delicate approaches to *bypass the CDN cache mechanism* and confirm them in the five CDN vendors as listed in Table 4.

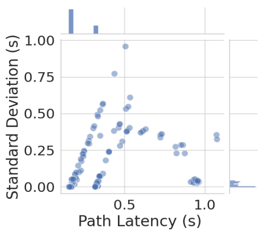
CDN vendors normally cache resources based on HTTP-related parameters, like file extensions [12] or URL path [42], thus, an attacker can exploit these specific rules to bypass the CDN cache. Besides, we also find the five CDN vendors just forward any requests with a WebSocket handshake header ("`Upgrade: websocket\n Connection: Upgrade`") to the victim without a further inspection.

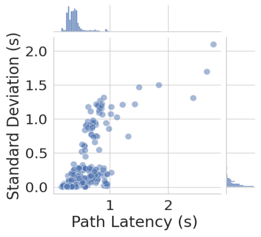
Table 4: Techniques to bypass CDN cache mechanism.

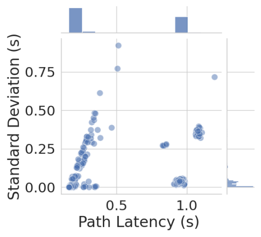
	Akamai	Azure	CloudFront	Cloudflare	Fastly
Dynamic Resouces	✔	✔	✔	✔	✔
Random URL	✔	✔	✔	✔	✔
Query Parameters	✔	✔	✔	✔	✔
HTTP POST/PUT	✔	✔	✔	✔	✔
WebSocket handshake	✔	✔	✔	✔	✔

Besides, CloudFront and Fastly even provide "CachingDisabled" configuration to disable its CDN caching mecha-


(a) Akamai


(b) Azure


(c) CloudFront


(d) Cloudflare

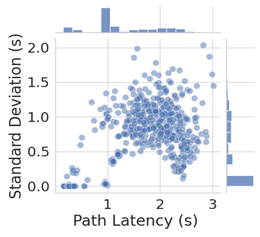

(e) Fastly

Figure 5: Latency Distribution of CDN-provided Paths.

nism [13,16], possibly for the reason to help websites to locate caching faults in the early CDN deployment phase. Hence, an attacker can also use these configurations to directly bypass the CDN cache mechanism.

4.1.5 Attack experiments

We conducted experiments through real-world CDN platforms to evaluate how CDNs converge web requests into the actual pulsing traffic. To avoid raising ethical problems, we performed our experiments through a limited number of CDN IP addresses and just sent hundreds of attacking requests to our own origin server. Consequently, we believe our experiments will not cause any collateral damage on the CDN platform and other CDN-hosting websites.

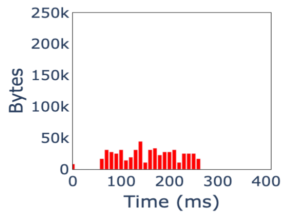
Experiments Setup. We registered the service of five CDN vendors and set up a TCP service as the victim on a cloud VPS (2.5GHz/512MB/100Mbps) located in Germany. Then, from a vantage point(as the attacker) located on a cloud VPS (2.5GHz/512MB/100Mbps) in Singapore.

We sent an 8KB HTTP/1.1 request to 64 global CDN IP addresses according to the measured latencies, aiming to make a pulse of 64 requests at our origin server. We captured requests at our origin server and analyze the `Bandwidth Concentration Ratio` of all these requests. The metric provides a basis for determining how much extra bandwidth can be gained from the attack, the ratio is defined as follows, and we sample the bandwidth of attacker and victim both in the unit of 10ms in this study.

$$Bandwidth\ Concentration\ Ratio = \frac{Victim\ maximum\ bandwidth}{Attacker\ bandwidth} \quad (5)$$

Results. Figure 6 illustrates the bandwidth concentration of the Basic CDN-Convex attack on Akamai. By sending a total of 512KB (an 8KB request to each of 64 paths) slow rate traffic, we obtained a victim-side pulse, with a high victim-side peak bandwidth (about 5.46 times of attacker-side peak bandwidth).

Above all, in the Basic CDN-Convex attack across five CDN vendors, we achieve the peak bandwidth concentration ratio as in Table 5. We observe that, for CDNs that provide stable network latency in a wide range, like Akamai, we can achieve better peak bandwidth concentration. For CDNs that


(a) Akamai (Attacker-side bandwidth)

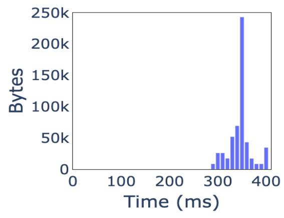

(b) Akamai (Victim-side bandwidth)

Figure 6: Basic CDN-Convex attack on Akamai. The attacking requests converged as a pulse at the victim, with a 5.46 times bandwidth concentration.

provide relatively unstable paths during our experiments, like Fastly, the attacking requests are not well converged at the victim.

We further analyzed the experiment results to understand the reasons causing the low concentration ratio, and identified two affecting factors: 1) the distribution of the latencies of the selected path is uneven. We only randomly selected 64 paths with stable network latencies, but the distribution of those latencies is not even. Some selected paths have roughly the same latencies, thus some requests are sent out at nearly the same time, which makes a large attacker-side peak bandwidth and thus a lower concentration factor according to Equation 5; 2) the negative impact from network instability is not fully reduced. For each randomly chosen CDN node, we measure its path latency several times to filter the stable one, thus it cannot fully reduce the impact of the network jitter. As the second factor is hard to eliminate, a real attacker can improve the attack by gathering more stable paths with even latencies. The attacker can measure and filter tens of thousands of CDN nodes to obtain a list of stable CDN paths with a larger time difference and even distribution, then the attacker can send out requests sequentially instead of bursting out requests to different paths nearly at the same time. In this way, this will make a smaller attacker-side peak bandwidth and thus a bigger concentration factor. Besides, these impacts can also be reduced with advanced techniques revealed later, which can obtain a wider time range to send attacking requests and result in a higher concentration ratio.

Table 5: Bandwidth concentration ratio in basic CDN-Convex attack

	Akamai	Azure	CloudFront	Cloudflare	Fastly
Bandwidth Concentration	5.46	4.66	6.42	3.73	1.49

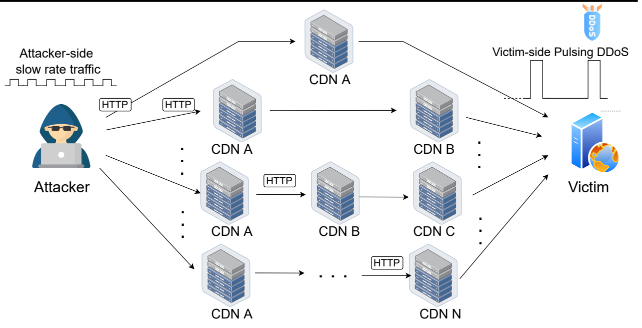


Figure 7: CDN-Cascading Convex attack: One or more CDN platforms are cascaded, which enlarges the time range to send more requests.

4.2 CDN-Cascading Convex Attack

As previously analyzed, when the attacker has a wider range of time to send requests, a higher request pulsing wave can be converged at the victim, like a larger ordinary convex lens can focus more rays of light. However, from previous measurements, CDN-provided paths are generally within a time range of hundreds of milliseconds.

In order to achieve better traffic concentration, we present the CDN-Cascading Convex attack. The concept of CDN-Cascading Convex attack is shown in Figure 7. We can cascade two or more CDN platforms in a chain to extend the attacking path, path latencies are also extended, as shown in 8. This will allow the attacker to have a wider and more even distribution of time range to send more attacking requests in one pulsing period, resulting in higher bandwidth pulsing wave at the victim. Theoretically, the maximum request accumulation time can be extended as:

$$RequestAccumulationTime = Latency(CDN\_A + CDN\_B + \ldots + CDN\_N) \quad (6)$$

4.2.1 Factors Affecting the Attack

However, there are two technical factors that affect the attack: 1) whether different CDN platforms are cascadable; 2) whether attacking paths are stable.

Cascadable CDN Platforms. A necessary condition for cascading two or more CDN platforms in a chain is that all involved CDN platforms must forward the request in such a way that the next-level CDN treats it as a benign request and continues the forwarding.

Whether the next-level CDN accepts the forwarded requests depends on the “Host:” header. If the front-level CDN platform support modifying the “Host:” header field of the forwarded requests as the domain name we registered the service on the next-level CDN platform, then the requests can be accepted and forwarded by the next-level CDN platform.

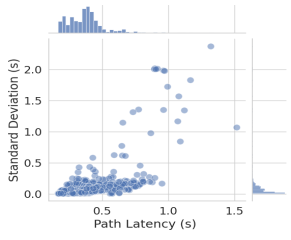
Our measurements show that four out of five CDN vendors can be cascaded in any level of the CDN chain, as Cloudflare does not support modifying the “Host:” header field as the “origin” domain name, we can only cascade Cloudflare as the last-level in a CDN-chain.

Table 6: Modifiable of “Host:” header field on CDNs

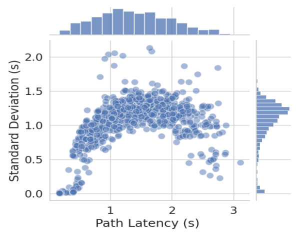
	Akamai	Azure	CloudFront	Cloudflare	Fastly
Origin Domain	✓	✓	✓	N/A	✓
Any Domain	N/A	✓	N/A	N/A	✓

Stability of the Attacking Paths. When more CDN platforms are cascaded in a chain, we can obtain a wider latency range of attacking paths, but it may bring in instability of the path.

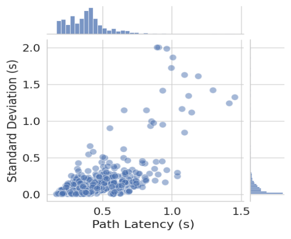
To understand this, we cascade these five CDN vendors in the four combinations, including “CloudFront → Akamai”, “CloudFront → Fastly → Azure”, “CloudFront → Fastly → Cloudflare”, “CloudFront → Fastly → CloudFront”. The results are shown in Figure 8. Compared with Basic CDN-Convex attack with one CDN platform (Figure 5), three combinations have a wider range of latency, as shown in Figure 8a, 8c, 8d. However, in one combination (Figure 8b), more instability of path latency is introduced.



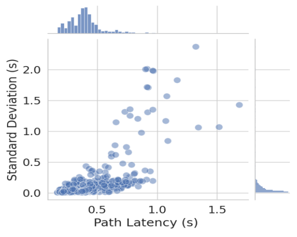
(a) CloudFront→Akamai



(b) CloudFront→Fastly→Azure



(c) CloudFront→Fastly→Cloudflare



(d) CloudFront→Fastly→CloudFront

Figure 8: Latency Distribution of CDN-cascaded Paths.

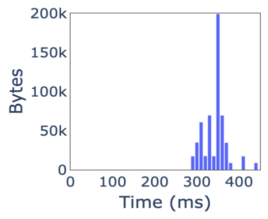
4.2.2 Experiments

As in Figure 8, the cascaded-CDN in the form of CloudFront→Akamai and CloudFront→Fastly →CloudFront are more stable, so we evaluate the CDN-Cascading Convex attack in these two combinations. The attacker and the victim are the same ones in Section 4.1.5.

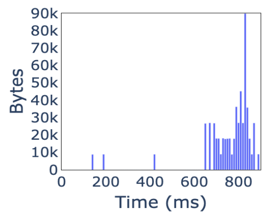
First, with 64 CDN nodes from Akamai and CloudFront→Akamai, we sent each node an HTTP/1.1 request of 8KB and achieved a maximum bandwidth concentration ratio of 8.52 (Table 7), and a pulsing time mostly within 90ms (Figure 9a). In the cascaded CloudFront→Fastly →CloudFront, we obtained a concentration ratio of 7.19, and a pulsing time mostly within 200ms (Figure 9b), which exhibited the negative impact of the unstable Fastly nodes.

Table 7: Bandwidth concentration ratio in CDN-Cascading Convex attack

	CloudFront→Akamai	CloudFront→Fastly→CloudFront
Bandwidth Concentration	8.52	7.19



(a) CloudFront → Akamai



(b) CloudFront → Fastly → CloudFront

Figure 9: Victim-side bandwidth in CDN-Cascading Convex attack.

4.3 DNS-holdon Convex Attack

DNS-holdon Convex attack can further leverage DNS resolution to make CDN converge more requests at victim, as shown in Figure 8. DNS-holdon Convex attack exploits the CDN feature that when CDNs forward a request to the customer-configured domain name, CDNs need to resolve the IP address first upon receiving the request. Our measurement shows that all five CDN vendors support configuring a domain name as the forwarding destination (the victim).

To launch DNS-holdon Convex attack, an attacker can first configure the CDN's "origin" option to be a domain name under his control. Then, he sends requests to different CDN nodes, these nodes will resolve the configured domain name first by sending DNS queries to the attacker's DNS authoritative server. Therefore, the attacker can hold these DNS queries for a while and then schedule the time to send DNS replies according to the latencies of attack paths. Therefore, DNS resolution provides the attacker another opportunity to extend the time range further and send more attacking requests in one pulsing period.

After sending the DNS resolving queries, CDN waits for a reply until the DNS resolving timeout occurs. Thus the attacker has to reply to each DNS query before the CDN node's

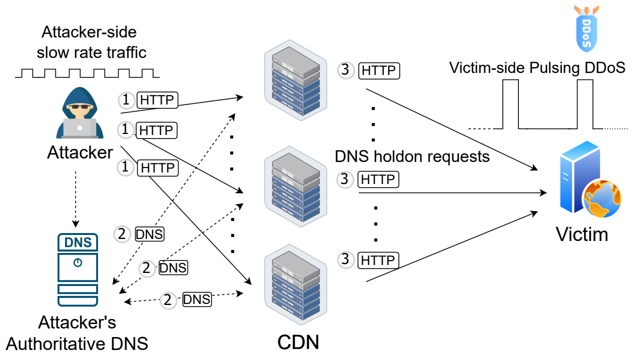


Figure 10: Concept of DNS-holdon Convex attack. Step 1: The attacker sends requests to CDN nodes. Step 2: CDN resolves the origin domain while requests being heldon. Step 3: The attacker replies to the DNS queries, on which the CDN starts up the temporal convergence of requests at the victim.

resolving timeout. Instinctively, before the DNS resolving timeout, the attacker can reply with a DNS A record. Therefore, all previous HTTP requests are being held on in this DNS resolving process.

We also try to leverage the DNS CNAME-chain method to extend the above DNS resolving timeout. Specifically, the attacker first returns a DNS CNAME record before the CDN resolving timeout; then the CDN node will ask to resolve the returned CNAME. In this way, by successively returning DNS CNAME records before DNS resolving timeout. In this way, theoretically, the attacker can extend the request accumulation time to a fair wider range as follows.

$$RequestAccumulationTime = Depth(CNAME\_Chain) \times DNS\_Timeout \quad (7)$$

4.3.1 Factors Affecting the Attack

The applicability and efficiency of DNS-holdon Convex attack depend on the CDN's specific DNS resolving behaviors. Therefore, we identified the following affecting factors.

Table 8: DNS resolving behaviors on CDNs.

	Akamai	Azure	CloudFront	Cloudflare	Fastly
Resolver	per-node	per-node	per-center	per-center	per-center
RR Timeout [†]	≈ 5s	≈ 4.3s	≈ 5s	≈ 10s	≈ 4.2s
RR Chain Depth [‡]	>16	> 16	12	> 16	> 16
T_Resolve Limit	<5s	<5s	< 8s [*]	< 10s [*]	<5s [$]

[†] The timeout when the CDN resolves a DNS resource record.
[‡] The depth of allowed CNAME chain when the CDN resolves a DNS resolver record.
[$] Fastly always pre-resolves the origin, which invalidates the attack.
[*] Per-center DNS resolving invalidates the attack.

DNS Resolving Behaviors. We found two DNS resolving behaviors that may affect DNS-holdon Convex attack: 1) resolver sharing; 2) resolving timeout.

First, an attacker would expect that CDN nodes on the different attacking paths will not interfere with each other on DNS resolution. Ideally, CDNs should have an independent

DNS resolver on each node. Our measurements show that two CDN vendors do not share DNS resolution results (via common DNS caches or DNS resolvers), and meet the requirement of per-node form, as shown in Table 8. However, CloudFront and Cloudflare have DNS resolvers shared on per-center form, Fastly always pre-resolves the victim. Therefore, we can not preciously hold on each request through a different DNS reply, which makes the attack invalidated.

Second, DNS timeout can also affect the efficacy of DNS-holdon Convex attack. Our measurements show that all CDN vendors have a DNS RR(Resource Record) timeout of over 4 seconds, and they support the DNS CNAME chain in a depth of more than 12. However, we find that all CDN vendors also apply a limit on the DNS resolving timeout, with a maximum value of about 5 seconds. So, the request accumulation time can be extended within about 5 seconds in total.

DNS TTL=0 Obedience Behaviors. The DNS resolver caching behavior can also affect DNS-holdon Convex attack. To make periodic pulse waves at the victim, the attack requires each CDN node to issue a DNS query upon each attacking request. To address this requirement, the attacker can set the authoritative DNS server to return DNS responses with "TTL=0". TTL (Time To Live) of a DNS Resource Record (RR) is defined to represent the maximum number of seconds that a record can be used before it must be discarded.

However, we found the "TTL=0" DNS RR is not always strictly obeyed. Our measurements show that Akamai has a minimum DNS cache time (TTL=60) even if we set TTL to 0 in DNS records, while the other four CDN vendors normally respect the "TTL=0" value in DNS replies. We note that Akamai TTL behavior wouldn't prevent the DNS-holdon Convex attack, because the attacker can simply wait for the DNS cache expiration (e.g., 60s) before making the next pulse wave.

Table 9: DNS TTL=0 obedience on CDNs.

	Akamai	Azure	CloudFront	Cloudflare	Fastly
Obedient	✗	✓	✓	✓	✓
Minimum TTL	60	0	0	0	0

4.3.2 Experiments

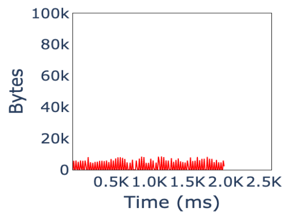
Table 10: Bandwidth concentration ratio in DNS-holdon Convex attack

	Akamai	Azure	CloudFront	Cloudflare	Fastly
Bandwidth Concentration	17.40	4.14	N/A	N/A	N/A

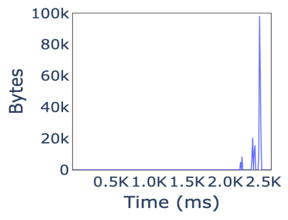
† N/A: Not applicable.

Due to the sharing issue of DNS resolvers, the DNS-holdon Convex is applicable on Akamai and Azure. Thus, with the same attacker and victim in Section 4.1.5, we experiment on Akamai and Azure.

First, we send 8KB requests to 64 CDN nodes of Akamai and Azure. In the authoritative DNS server, upon receiving the first DNS query, we withhold the first and following DNS replies for two seconds. Finally, we sent out the holdon DNS responses in line with the different path latencies, and we obtain the results as in Table 10. Results show that the DNS-holdon Convex can converge web requests into short pulse on both CDNs, achieving a better bandwidth concentration ratio for Akamai (17.4 times) and a lower concentration for Azure, because Akamai DNS is more stable.



(a) Akamai(Attacker-side bandwidth)



(b) Akamai (Victim-side bandwidth)

Figure 11: DNS-holdon Convex Experiment on Akamai.

4.4 Request-pending Convex Attack

In the Request-pending Convex, we use segmented HTTP requests to make CDN edge servers hold as many requests as possible before bursting out, as shown in Figure 12.

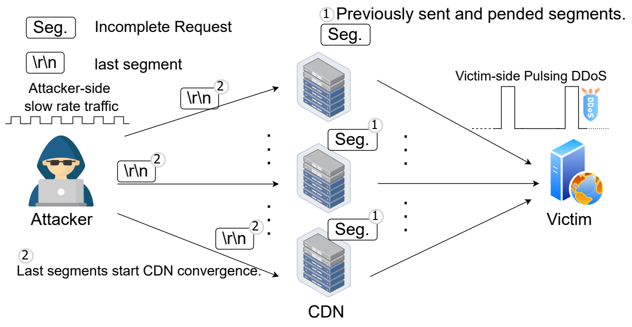

Seg.
Incomplete Request
\r\n
last segment
Attacker-side slow rate traffic
① Previously sent and pended segments.
Victim-side Pulsing DDoS
Attacker
Victim
CDN
② Last segments start CDN convergence.

Figure 12: Concept of Request-pending Convex. Step 1, the attacker sends segmented requests to CDN nodes, and CDN will pend these incomplete HTTP requests. Step 2, the attacker schedules to send the last segment of previous requests, on which the CDN starts up the temporal convergence of requests at the victim.

Request-pending Convex attack exploits the fact that, upon receiving incomplete HTTP requests, some CDNs will hold them and do not forward them to the origin immediately until the requests are completely received. Thus, an attacker can use incomplete requests to make CDNs hold as much attacking traffic as possible before busting out, which allows the attacker to have a wider time to send more attacking requests in a pulsing period.

The attacker can send just 1 byte in each segmented request, and deliberately hold on for some time. Then before the HTTP session timeout, the attacker sends the following one byte. Finally, the attacker sends the last part (like "\r\n") of previous incomplete HTTP requests. By controlling the sending time of the last segments, the attacker can let the CDN converge much more requests to the victim. In this way, the number of segmented requests equals the number of bytes allowed in each CDN platform. Considering the timeout of an HTTP session, the attacker obtains a much wider range to time window to send requests:

$$RequestAccumulationTime = Max\_Size(Request) \times HTTP\_Timeout \quad (8)$$

Whether the CDN pends the following segments or immediately forwards incomplete requests back to the victim depends on the CDN's specific behaviors. Hence, we measure and evaluate these factors as follows.

4.4.1 Factors Affecting the Attack

We found that there are two categories of CDNs forwarding behaviors that can be exploited for Request-pending Convex: 1) header pending, which CDNs forward requests upon HTTP headers are completely received; 2) body pending, which CDNs forward requests upon HTTP body are completely received. Hence, we measure and evaluate these factors as follows.

Header-only Request Pending. The attacker serially sends segmented head-only HTTP requests to a CDN node, if the CDN node pends these incomplete segments until the last segment, then the Request-pending Convex attack is applicable.

As in Table 11, experiments show that five CDN vendors all pend the segmented header-only requests. For Fastly, an attacker can send a maximum of 64KB header-only request, with a timeout of *600 seconds*. In other words, the attacker has 600 seconds to send more attacking requests.

Table 11: Header-only Request-pending behaviors on CDNs.

	Akamai	Azure	CloudFront	Cloudflare	Fastly
GET Header Pending	✔	✔	✔	✔	✔
POST Header Pending	✔	✔	✔	✔	✔
Header Timeout	≈ 24s	≈ 124s	≈ 30s	≈ 15s	≈ 600s
Header Size Limit	32KB	28KB	24KB	32KB	64KB

BODY Request Pending. The attacker serially sends the HTTP requests with the segmented body to a CDN node, if the CDN node pends all incomplete segments until the last segment, then the Request-pending Convex attack is applicable.

As in Table 12, experiments show that three CDN vendors pend the segmented requests of POST Body. In HTTP protocol, the body of a request can be sent in the way of "Content-Length" or "Transfer-Encoding", and we explore both across CDN platforms. Experiments show that an attacker can use the CDN to withhold an 80KB or even 400MB POST message. Further, we found that all CDNs support a timeout of over ten seconds, especially Azure even supports *1600 seconds*, and Cloudflare supports >*5400 seconds*. Those CDN behaviors allow attackers to have a much wider time to withhold a huge amount of attacking traffic in CDN edge servers before bursting out, resulting in a high traffic concentration pulsing wave at the victim, as demonstrated below.

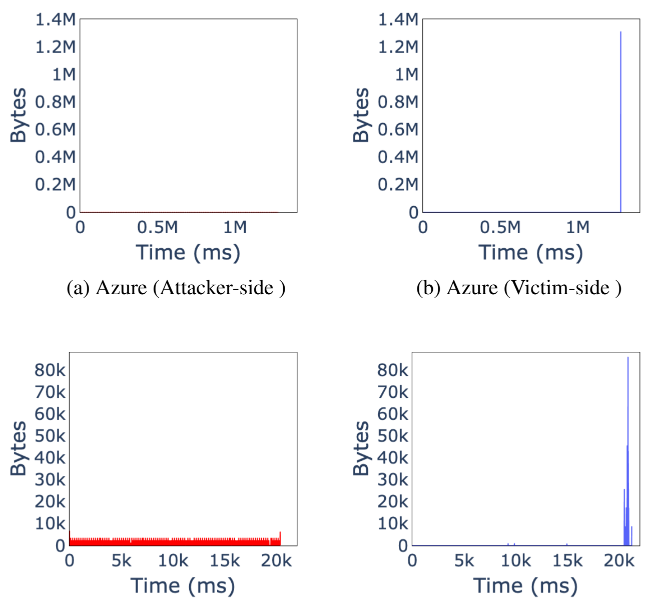
Table 12: POST BODY pending behaviors on CDNs.

	Akamai	Azure	CloudFront	Cloudflare	Fastly
CL Pending [†]	N/A[*]	✔	N/A	✔	N/A
TE Pending [‡]	N/A	✔	N/A	✔	N/A
CL Timeout	≈ 120s	≈ 65s	≈ 36s	≈ 15s	≈ 52s
TE Timeout	≈ 16s	≈ 1600s	≈ 12s	> 3600s	≈ 16s
CL Forwarding Threshold	N/A	80KB	N/A	100MB	N/A
TE Forwarding Threshold	N/A	80KB	N/A	> 400MB	N/A

[†] CL (Content-Length), we use "Content-Length" to send segmented requests.
[‡] TE (Transfer-Encoding), we use "Transfer-Encoding" to send segmented requests.
[*] Not applicable. Akamai, CloudFront, and Fastly would forward every byte of the POST body request immediately to the origin server.

4.4.2 Experiments

With the same attacker and victim configuration in Section 4.1.5, we experiment with five CDN vendors.



(a) Azure (Attacker-side) (b) Azure (Victim-side)
(c) CloudFront (Attacker-side) (d) CloudFront (Victim-side)

Figure 13: Request-pending Convex Experiments on Azure and CloudFront.

To compare the effectiveness of the attack across CDN vendors, we send requests to 64 CDN nodes to evaluate the effectiveness of the attack. For the GET header pending technique, the size of requests is 8KB, and the accumulation time (how long the attacker is allowed to send requests) is set according to the header timeout in Table 11. For the POST body pending technique, we slowly send requests of 80KB size to 64 Azure nodes in 1500 seconds, and we also send requests of 80KB size to 64 Cloudflare nodes in 5400 seconds, according to Table 12.

Table 13: Bandwidth concentration ratio in Request-pending Convex attack

	Akamai	**Azure**	**CloudFront**	**Cloudflare**	**Fastly**
Accumulation Time	24s	1500s	20s	5400s	512s
Technique	GET-Header	POST-BODY-TE	GET-Header	POST-BODY-TE	GET-Header
Request Size	8KB	80KB	8KB	80KB	8KB
Bandwidth Amplification Ratio	**146.38**	**4842.69**	**31.30**	**1786.37**	**988.48**

The results are shown in Table 13, we sample the bandwidth of both attacker and victim in the unit of 10ms, and compare the bandwidth concentration ratio. With the GET header pending technique, we obtain a concentration ratio of 31.30 on CloudFront (Figure 13a and Figure 13d). With the POST BODY pending technique, we obtain a ratio of 4842.69 on Azure (ref Figure 13a and Figure 13b). Cloudflare has a much longer time of 5400s to converge requests, however, the latency instability introduced by its anycast-based request routing applies a negative impact on its effectiveness.

4.5 Other CDN-Convex Techniques

We also found other forwarding mechanisms like IP-layer fragmentation could also be utilized to improve the CDN-Convex attack. An attacker actively splits an attacking request into IP fragments and sends them to CDN nodes while reserving one of them as the final signal to let the CDN start the temporal convergence of requests.

This IP fragmentation technique can bring in two benefits: 1) kernel-level reassembling of IP fragments further enlarges the time window which allows the CDN to withhold more attacking traffic; 2) IP fragments can be sent out disordered, that the last sent IP fragment can be any part of the request, for example, the attacker can send the first start line of an HTTP request as the last IP segment to schedule the final temporal convergence of requests at the victim. Therefore, even if the CDN does not withhold incomplete requests in the application layer, like CloudFront and Fastly do not withhold the POST Body in Table 12, IP fragmentation can help to bypass this CDN-side constraint.

We also explore the maximum allowable time that the kernel of different CDN servers will withhold IP fragments in the memory before dropping. In a Linux system, the IPv4 fragmentation timeout is specified in ipfrag_timeout, which defaults to 30 seconds, while the IPv6 fragmentation timeout is configured in ip6frag_timeout, which defaults to 60 seconds [34]. Table 14 shows the five CDNs' reassembling timeout of IP fragments.

In addition to the above techniques, an attacker may also use TCP-layer segmentation techniques to delay the requests.

Table 14: Maximum time of IP fragments kept on CDNs

	Akamai	**Azure**	**CloudFront**	**Cloudflare**	**Fastly**
IPv4 Fragmentation Timeout	≈30	≈30	≈30	≈15	≈10
IPv6 Fragmentation Timeout	≈60	≈60	≈30	≈15	≈6

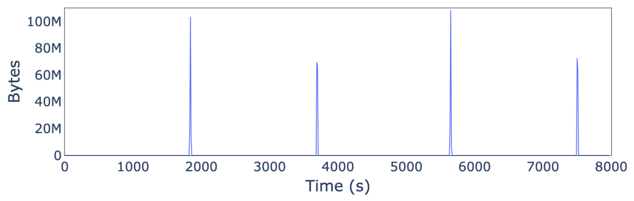


Figure 14: Victim-side Bandwidth Concentration through Request-pending Convex

Future work can explore those techniques and extend our study to other Internet middle-boxes that can cache and forward.

5 Real-world Evaluation

In order to evaluate how the CDN-Convex attack degrades the availability and network latency of online services, we conduct two classes of controlled experiments.

Exp1. Attack a TCP Service. In the first experiment, we take the DNS zone transfer service as an example to demonstrate the effectiveness of CDN-Convex attack against TCP services. We first set up a DNS master server (2.5GHz/512MB/1000Mbps/BIND 9.18.1) as the victim (to target TCP DNS zone transfer), meanwhile, there is a DNS slave server which periodically queries the DNS master server for zone transfer in TCP protocol.

Accordingly, we use Cloudflare to launch a Request-pending Convex against the victim at an 1800 seconds period. During the attack, requests of 2MB are sent in 1800 seconds to 64 Cloudflare nodes, we observe an average 72.82KBps attacker-side traffic results to a 108.58MBps (Figure 14) on the website (1526.9 times).

Consequently, when the bandwidth of the DNS master server is periodically saturated just in time when the DNS slave server sends the DNS query, the DNS slave server is also starved, demonstrating the effectiveness of a CDN-Convex attack against a TCP service.

Instinctively, the CDN-Convex attack can also be launched against other well-known TCP services, such as SSH, and SMTP, although these TCP services will drop requests after receiving and inspecting, the bandwidth is still exhausted. Considering the massive resources empowered by the CDN platform, we think the CDN-Convex attack can severely degrade or even damage the availability of the targeted TCP service.

Exp2. Attack a Website. Besides the peak bandwidth exhaustion against the TCP service, moreover, the CDN-Convex attack is especially harmful to websites. When the pulsing requests arrive at the website concurrently requesting for an internal bottleneck resource (CPU, Memory, Logic Queue, etc.), the pulse waves can periodically saturate the bottleneck resources, resulting in much more severe DDoS damage.

We launch a Request-pending Convex against a website (2.5GHz/512MB/1000Mbps/Apache 2.4.52) hosted on Cloudflare. In the attack, we sent 64 POST pending requests (2MB of each) as slow rate traffic in 1800 seconds, resulting in a peak 103.55MBbps victim-side traffic on the website. Under attack, the targeted website server is directly out of service, as the "apache" process is killed by the operating system ("Out of memory: Killed process apache2").

The above experiments show that the CDN-Convex attack is ideally matching the "moments to go down, hours to recover" attacking philosophy [60]. Worse, the stealthy pulsing nature of a CDN-Convex attack can also help it to escape DDoS detection.

6 Mitigation

The root cause of CDN-Convex attack is the inherent nature of CDN-introduced global latency distribution. Combined with the weakness to direct the CDN traffic to a 3rd-party service, CDN can be abused as a pulsing DDoS platform. To defend the attack, we propose the following mitigations from the view of the CDN.

- *Enforcing the origin validation*. Currently all five CDNs we tested do not validate the ownership of customer-supplied origin. We recommend CDNs enforce this validation to avoid abuse. For example, CDNs can require their customers to upload a special file to the origin website and continuously probe this special file's existence. Any abnormality will make CDNs terminate the service and relaunch a new validation process, we call this process the *Origin Pinning*.
- *Monitoring traffic globally*. A CDN provider may perform more fine-grained traffic monitoring and filter suspicious high-bandwidth pulses at edge servers. However, an attacker can bypass this defense by sending attacking requests to globally distributed nodes of different CDN platforms. Thus, to prevent CDN-Convex, a CDN vendor not only has to apply the defense within itself, but it also needs cooperation between CDN platforms, which may require data synchronization and statistical analysis of all related CDN platforms.
- *Standardizing a unified header field to expose client IP address*. The CDN-Convex attack originated from one vantage point, theoretically, the CDN and the victim can detect the attack from the CDN-forwarded client (attacker) IP address. However, we note that currently CDN vendors use different header fields to expose the client IP, as shown in Table 15, and other cascaded CDN can filter or reset these header fields. We suggest CDN vendors standardize a unified header that can not be reset or filtered even in CDN-cascaded techniques.

Table 15: CDN-forwarded client IP header field

	Akamai	Azure	CloudFront	Cloudflare	Fastly
Header	True-Client-IP	X-Azure-ClientIP	X-Forwarded-For	CF-Connecting-IP	Fastly-Client-IP

- *Avoid pending HTTP requests and reduce timeout.* Among our four attacking techniques, Request-pending Convex causes the most severe damage. CDN could prevent this attack by avoiding pending HTTP requests. For example, CDN servers can start relaying a request or response to its next hop immediately after receiving its initial chunk, rather than waiting for the complete content. We also recommend CDNs reduce request pending timeout to minimize the impact of Request-pending Convex. As IP fragmentation is hard to avoid, CDNs can reduce the fragmentation timeout to minimize the impact. However, this cannot stop attackers from exploiting the other three techniques.

We also discuss the following countermeasures from the view of network and victim.

- *In-network detection for bandwidth-targeted pulse-wave attack*. Generally, the CDN-Convex techniques can be exploited for two kinds of DDoS attack: 1) pulsing DDoS attack against network infrastructure. This attack aims to temporally saturate the bandwidth of victims or network paths, which could be detected by network devices. For example, recent researches use a network-wide view of the whole attacking traffic to mitigate flooding DDoS in seconds-level [39, 59, 61]. And the ACC-Turbo research uses programmable switches to detect the sub-second short-lived traffic burst [3], thus can detect the bandwidth-targeted pulse-wave attack; 2) pulsing DDoS attack against server-side implementations. This attack aims to exploit the weakness of server-side implementations rather than the network, for example, web applications with bottleneck resources, or TCP implementations. Thus it would be more challenging for network devices to detect such attacks because it requires additional knowledge of various server implementations for detection.
- *IP-based filtering to protect non-CDN customers*. When the victim is not any CDN customer, he can apply IP-based filtering to drop the traffic from all CDN egress IPs to mitigate the attack. He can construct and maintain those IP lists by using some CDNs' publicly available IP ranges [11], monitoring BGP updates of a CDN's ASes, or observing when being under attack.

Ethical Consideration. We take the utmost care of potential ethical issues. First, we performed controlled attacks against our website server, with a limited number of requests and bandwidth consumption. Thus, we believe our experiments have limited impacts on the regular operations of CDN and network path. Second, CDNs encourage security tests through bug bounty programs, and our experiments strictly adhere to their program rules.

Responsible Disclosure. We contacted all tested CDNs to responsibly disclose our findings, unveiling experiment details and reproducing procedures. Cloudflare thanked us for our report and provided a reward of $500. Akamai and CloudFront thanked our report and stated that they appreciated this type of collaboration and responsible disclosure. Azure and Fastly thanked our reports but have no further comments to date.

7 Related Work

CDN Security. CDN infrastructure serves as a cornerstone of the Internet. Therefore, finding the vulnerabilities of CDN architecture and attacking CDN hosting origin servers are always hot research topics in the field of network security.

Previous studies found various vulnerable implementations of configuration and validation existing in CDN vendors. For example, Chen *et al.* identified the inconsistency of HTTP request processing strategies between CDN providers and proposed a CDN forwarding-loop attack [9] that may lead to CDN services being unavailable. Due to the conflicts between man-in-the-middle architecture and end-to-end encryption, CDN exposed serious security risks in certificate management, such as sharing private key [38]. Some CDN providers even did not examine DNS resolutions' correctness, resulting in exploiting by crafted malicious DNS responses [20, 22].

Some research works also find the technique to reveal the origin servers' IP addresses and bypass the CDN security protection mechanisms. For instance, if the origin server terminates the CDN services or switches to another CDN provider, DNS resolution flaws of the nameserver could leak the IP addresses of origin [29]. Further, [55] discussed several origin-exposing attack vectors that could be used to uncover the less protected origin server behind the CDN platform.

Recently, through exploiting the good reputation of CDN providers and their IP addresses, some attacks are designed to circumvent Internet censorship [23, 62], launch cache poisoning [8] or cache deception attacks [42]. In other words, the CDN infrastructure is at risk of progressive abuse.

In summary, compared with previous CDN security work, we discover a new architectural weakness in CDN infrastructure to launch DDoS attacks, given that CDNs are widely employed to prevent DDoS attacks. And previous CDN researches [18, 25, 37] focus on amplifying requests in size to launch flooding DDoS attacks, while we are the first to use CDN to launch pulse-wave DDoS attacks by concentrating requests in time.

Pulsing DDoS Attack. The pulsing DDoS attack has been studied for years. As a low-rate DDoS attack, it can bypass traditional sampling-counter-based DoS protection mechanisms [26, 41].

Most previous studies focus on the eventual effects of pulsing attacks to degrade the TCP connection in 2003, [36] introduced the concept of shrew DoS attack. The attack exploits the weakness of TCP re-transmissions or congestion control when the timeout occurred, and results in periodical congestion at the victim side [36, 40]. Following, [41] also demonstrated that pulsing waves could trigger web applications' bottleneck. Due to the complex dependencies of Internet web resources, the pulsing attack could easily create resource saturation and starve legitimate requests [51].

However, launching a pulsing DDoS attack is a non-trivial task, the attacker is usually required to employ a considerable number of compromised or exploitable devices. Due to the inherent weakness of complex coordination and synchronization, the pulsing attacks generated from botnets are more likely to be detected, filtered, or degraded to less effective [31, 46, 48]. In 2015, Rasti et.al. proposed a DNS reflection pulsing DDoS attack by utilizing open DNS resolvers existing in the wild [49]. However, they focus on studying UDP-based amplification which requires source IP address spoofing and the maximal bandwidth concentration factor is only 10, significantly limiting the real-world impact of the attack. Inspired by [49], this paper demonstrates that an attacker could exploit the CDN infrastructure to launch pulsing DDoS attacks against its hosting websites. Interestingly, the CDN security protection mechanisms are broken by their inherent nature, known as the geographically distributed edge servers.

Previous work discovered various amplification DoS attacking vectors like NTP server [14, 50], memcached server [54], DNS server [47], and TCP middleboxes [6] to amplify the attacker's traffic to launch DDoS attacks. Instead, our work focuses on concentrating on the arrival of attacking traffic at the victim. This approach is orthogonal to traditional amplification attacks.

In summary, in terms of pulse-wave DDoS research, we propose new techniques to launch TCP-based pulsing attacks by exploiting CDNs, while most previous pulse-wave DDoS studies focus on the stealthy and efficiency of pulse-wave attacks to degrade the target service except Rasti's work [49]. Our work is inspired by the research of Rasti et.al. [49] but differs from them in two aspects. First, they study UDP-based amplification and requires source IP address spoofing, while our work focus on TCP-based pulsing attack and do not have this requirement; Second, our work can achieve much higher concentration factors than theirs (1500x vs 10x), by proposing multiple advanced concentration techniques.

8 Conclusion

We present the CDN-Convex attack in this work, which uses the CDN-introduced delay distribution to launch a pulsing DDoS attack against a 3-rd party TCP service. With real-world experiments to present the applicability and the severity of this CDN-assisted pulsing DDoS attack, our work unveils a practical CDN-based pulsing DDoS attacking platform, which completes the previous theoretical studies. The fundamental cause of the attack arises from the global geographical distribution of the CDN and the fact that CDN vendors are still striving to deploy more edge servers globally to speed up website access. These can also make the attack more severe and efficient. We hope our study can help CDN vendors bolster their DDoS defense with proposed mitigation.

Acknowledgement

We sincerely thank all anonymous reviewers and our shepherd for their valuable comments to improve the paper, especially the suggestion of the IP fragmentation attacking technique. We also thank Jia Zhang, Yi Guo, Shuang Hao, Kaiwen Shen, Yao Wang, and Xiarun Chen for their help on the paper. This work is in part supported by the National Key Research and Development Program of China (2021YFB2701000), National Natural Science Foundation of China (grant #62272265, #61972224), and Beijing National Research Center for Information Science and Technology (BNRist) under Grant BNR2022RC01006.

References

[1] Akamai. Facts & figures. https://www.akamai.com/us/en/about/facts-figures.jsp.

[2] Akamai. State of the internet / security: Ddos and application attacks. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/2018-state-of-the-internet-security-a-year-in-review.pdf.

[3] Albert Gran Alcoz, Martin Strohmeier, Vincent Lenders, and Laurent Vanbever. Aggregate-based congestion control for pulse-wave ddos defense. In Fernando Kuipers and Ariel Orda, editors, *SIGCOMM '22: ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, August 22 - 26, 2022*, pages 693–706. ACM, 2022.

[4] Esraa Alomari, Selvakumar Manickam, B. B. Gupta, Shankar Karuppayah, and Rafeef Alfaris. Botnet-based distributed denial of service (ddos) attacks on web servers: Classification and art. *CoRR*, abs/1208.0403, 2012.

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1093–1110. USENIX Association, 2017.

[6] Kevin Bock, Abdulrahman Alaraj, Yair Fax, Kyle Hurley, Eric Wustrow, and Dave Levin. Weaponizing middleboxes for {TCP} reflected amplification. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3345–3361, 2021.

[7] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of an anycast CDN. In Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring, editors, *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, pages 531–537. ACM, 2015.

[8] Jianjun Chen, Jian Jiang, Hai-Xin Duan, Nicholas Weaver, Tao Wan, and Vern Paxson. Host of troubles: Multiple host ambiguities in HTTP implementations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1516–1527. ACM, 2016.

[9] Jianjun Chen, Xiaofeng Zheng, Hai-Xin Duan, Jinjin Liang, Jian Jiang, Kang Li, Tao Wan, and Vern Paxson. Forwarding-loop attacks in content delivery networks. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[10] Cloudflare. The cloudflare global anycast network. https://www.cloudflare.com/network/.

[11] Cloudflare. Ip ranges. https://www.cloudflare.com/ips/.

[12] Cloudflare. Understanding cloudflare's cdn. https://support.cloudflare.com/hc/en-us/articles/200172516-Which-file-extensions-does-CloudFlare-cache-for-static-content-#h_51422705-42d0-450d-8eb1-5321dcadb5bc.

[13] CloudFront. Understanding the managed cache policies. https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/using-managed-cache-policies.html.

[14] Jakub Czyz, Michael Kallitsis, Manaf Gharaibeh, Christos Papadopoulos, Michael Bailey, and Manish Karir. Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 435–448, 2014.

[15] datanyze.com. Cdn market share. https://www.datanyze.com/market-share/cdn.

[16] Fastly. Controlling caching. https://docs.fastly.com/en/guides/controlling-caching.

[17] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. Idmaps: A global

internet host distance estimation service. *IEEE/ACM Transactions On Networking*, 9(5):525–540, 2001.

[18] Milad Ghaznavi, Elaheh Jalalpour, Mohammad A. Salahuddin, Raouf Boutaba, Daniel Migault, and Stere Preda. Content delivery network security: A survey. *IEEE Commun. Surv. Tutorials*, 23(4):2166–2190, 2021.

[19] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. King: estimating latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Internet Measurement Workshop, IMW 2002, Marseille, France, November 6-8, 2002*, pages 5–18. ACM, 2002.

[20] Run Guo, Jianjun Chen, Baojun Liu, Jia Zhang, Chao Zhang, Hai-Xin Duan, Tao Wan, Jian Jiang, Shuang Hao, and Yaoqi Jia. Abusing cdns for fun and profit: Security issues in cdns' origin validation. In *37th IEEE Symposium on Reliable Distributed Systems, SRDS 2018, Salvador, Brazil, October 2-5, 2018*, pages 1–10. IEEE Computer Society, 2018.

[21] Run Guo, Weizhong Li, Baojun Liu, Shuang Hao, Jia Zhang, Haixin Duan, Kaiwen Shen, Jianjun Chen, and Ying Liu. Cdn judo: Breaking the cdn dos protection with itself. In *27rd Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 21-24, 2020*. The Internet Society, 2020.

[22] Shuai Hao, Yubao Zhang, Haining Wang, and Angelos Stavrou. End-users get maneuvered: Empirical analysis of redirection hijacking in content delivery networks. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1129–1145. USENIX Association, 2018.

[23] John Holowczak and Amir Houmansadr. Cachebrowser: Bypassing chinese censorship without proxies using cached content. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 70–83. ACM, 2015.

[24] intricately.com. 2019 cdn market report. `https://cdn2.hubspot.net/hubfs/4238862/2019%20Intricately%20CDN%20Market%20Report.pdf`.

[25] Bahruz Jabiyev, Steven Sprecher, Anthony Gavazzi, Tommaso Innocenti, Kaan Onarlioglu, and Engin Kirda. FRAMESHIFTER: security implications of http/2-to-http/1 conversion anomalies. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 1061–1075. USENIX Association, 2022.

[26] Hossein Hadian Jazi, Hugo Gonzalez, Natalia Stakhanova, and Ali A. Ghorbani. Detecting http-based application layer dos attacks on web servers in the presence of sampling. *Comput. Networks*, 121:25–36, 2017.

[27] Jie Ji. Reflective and short-burst ddos attacks harnessed to knock down the targets in ukraine. `https://nsfocusglobal.com/reflective-and-short-burst-ddos-attacks-harnessed-to-knock-down-the-targets-in-ukraine/`.

[28] Jian Jiang, Jia Zhang, Hai-xin Duan, Kang Li, and Wu Liu. Analysis and measurement of zone dependency in the domain name system. In *2018 IEEE International Conference on Communications, ICC 2018, Kansas City, MO, USA, May 20-24, 2018*, pages 1–7. IEEE, 2018.

[29] Lin Jin, Shuai Hao, Haining Wang, and Chase Cotton. Your remnant tells secret: Residual resolution in ddos protection services. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*, pages 362–373. IEEE Computer Society, 2018.

[30] Lin Jin, Shuai Hao, Haining Wang, and Chase Cotton. Unveil the hidden presence: Characterizing the backend interface of content delivery networks. In *27th IEEE International Conference on Network Protocols, ICNP 2019, Chicago, IL, USA, October 8-10, 2019*, pages 1–11. IEEE, 2019.

[31] Anestis Karasaridis, Brian Rexroad, and David A. Hoeflin. Wide-scale botnet detection and characterization. In Niels Provos, editor, *First Workshop on Hot Topics in Understanding Botnets, HotBots'07, Cambridge, MA, USA, April 10, 2007*. USENIX Association, 2007.

[32] Kasperksy. Ddos attacks in q1 2019. `https://securelist.com/ddos-report-q1-2019/90792/`.

[33] Yu-Ming Ke, Chih-Wei Chen, Hsu-Chun Hsiao, Adrian Perrig, and Vyas Sekar. CICADAS: congesting the internet with coordinated and decentralized pulsating attacks. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, pages 699–710. ACM, 2016.

[34] Linux Kernel. Ip sysctl. `https://docs.kernel.org/networking/ip-sysctl.html`.

[35] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. Going wild: Large-scale classification of open DNS resolvers. In *Proceedings of the 2015 ACM Internet Measurement Conference,*

IMC 2015, Tokyo, Japan, October 28-30, 2015, pages 355–368. ACM, 2015.

[36] Aleksandar Kuzmanovic and Edward W. Knightly. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 25-29, 2003, Karlsruhe, Germany*, pages 75–86. ACM, 2003.

[37] Weizhong Li, Kaiwen Shen, Run Guo, Baojun Liu, Jia Zhang, Haixin Duan, Shuang Hao, Xiarun Chen, and Yao Wang. Cdn backfired: Amplification attacks based on http range requests. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Luxembourg City, Luxembourg, June 25-28, 2020*. IEEE Computer Society, 2020.

[38] Jinjin Liang, Jian Jiang, Hai-Xin Duan, Kang Li, Tao Wan, and Jianping Wu. When HTTPS meets CDN: A case of authentication in delegated service. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 67–82. IEEE Computer Society, 2014.

[39] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 3829–3846. USENIX Association, 2021.

[40] Xiapu Luo and Rocky K. C. Chang. On a new class of pulsing denial-of-service attacks and the defense. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005, San Diego, California, USA*. The Internet Society, 2005.

[41] Gabriel Maciá-Fernández, Jesús E. Díaz-Verdejo, Pedro Garcia-Teodoro, and Francisco de Toro-Negro. Lordas: A low-rate dos attack against application servers. In *Critical Information Infrastructures Security, Second International Workshop, CRITIS 2007, Málaga, Spain, October 3-5, 2007. Revised Papers*, volume 5141 of *Lecture Notes in Computer Science*, pages 197–209. Springer, 2007.

[42] Seyed Ali Mirheidari, Sajjad Arshad, Kaan Onarlioglu, Bruno Crispo, Engin Kirda, and William Robertson. Cached and confused: Web cache deception in the wild. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 665–682. USENIX Association, 2020.

[43] TS Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 170–179. IEEE, 2002.

[44] Jeman Park, Aminollah Khormali, Manar Mohaisen, and Aziz Mohaisen. Where are you taking me? behavioral analysis of open DNS resolvers. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*, pages 493–504. IEEE, 2019.

[45] Jeman Park, Manar Mohaisen, DaeHun Nyang, and Aziz Mohaisen. Assessing the effectiveness of pulsing denial of service attacks under realistic network synchronization assumptions. *Comput. Networks*, 173:107146, 2020.

[46] Jeman Park, DaeHun Nyang, and Aziz Mohaisen. Timing is almost everything: Realistic evaluation of the very short intermittent ddos attacks. In *16th Annual Conference on Privacy, Security and Trust, PST 2018, Belfast, Northern Ireland, Uk, August 28-30, 2018*, pages 1–10. IEEE Computer Society, 2018.

[47] Vern Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *ACM SIGCOMM Computer Communication Review*, 31(3):38–47, 2001.

[48] Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader, and Elmar Gerhards-Padilla. A comprehensive measurement study of domain generating malware. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 263–278. USENIX Association, 2016.

[49] Ryan Rasti, Mukul Murthy, Nicholas Weaver, and Vern Paxson. Temporal lensing and its application in pulsing denial-of-service attacks. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 187–198. IEEE Computer Society, 2015.

[50] Christian Rossow. Amplification hell: Revisiting network protocols for ddos abuse. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[51] Huasong Shan, Qingyang Wang, and Calton Pu. Tail attacks on web applications. In *Proceedings of the 2017*

ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pages 1725–1739. ACM, 2017.

[52] Huasong Shan, Qingyang Wang, and Qiben Yan. Very short intermittent ddos attacks in an unsaturated system. In *Security and Privacy in Communication Networks - 13th International Conference, SecureComm 2017, Niagara Falls, ON, Canada, October 22-25, 2017, Proceedings*, volume 238 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 45–66. Springer, 2017.

[53] Lumin Shi. Two decades of ddos attacks and defenses. https://cs.uoregon.edu/area-exam/two-decades-ddos-attacks-and-defenses.

[54] Kulvinder Singh and Ajit Singh. Memcached ddos exploits: operations, vulnerabilities, preventions and mitigations. In *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, pages 171–179. IEEE, 2018.

[55] Thomas Vissers, Tom van Goethem, Wouter Joosen, and Nick Nikiforakis. Maneuvering around clouds: Bypassing cloud-based security providers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1530–1541. ACM, 2015.

[56] Lan Wei and John S. Heidemann. Does anycast hang up on you? In *Network Traffic Measurement and Analysis Conference, TMA 2017, Dublin, Ireland, June 21-23, 2017*, pages 1–9. IEEE, 2017.

[57] Wikipedia. Multiple round simultaneous impact. https://en.wikipedia.org/wiki/Artillery#Multiple_round_simultaneous_impact.

[58] Zhijun Wu, Wenjing Li, Liang Liu, and Meng Yue. Low-rate dos attacks, detection, defense, and challenges: A survey. *IEEE Access*, 8:43920–43943, 2020.

[59] Jiarong Xing, Wenqing Wu, and Ang Chen. Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 3865–3881. USENIX Association, 2021.

[60] Igal Zeifman. Attackers use ddos pulses to pin down multiple targets. https://www.imperva.com/blog/pulse-wave-ddos-pins-down-multiple-targets/.

[61] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qi Li, Mingwei Xu, and Jianping Wu. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[62] Hadi Zolfaghari and Amir Houmansadr. Practical censorship evasion leveraging content delivery networks. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1715–1726. ACM, 2016.