Forwarding-Loop Attacks in Content Delivery Networks

Jianjun Chen*†‡, Jian Jiang§, Xiaofeng Zheng*†‡, Haixin Duan†‡¶,
Jinjin Liang*†‡, Kang Li‖, Tao Wan**, Vern Paxson§¶,

*Department of Computer Science and Technology, Tsinghua University
†Institute for Network Science and Cyberspace, Tsinghua University
‡Tsinghua National Laboratory for Information Science and Technology
{chenjj13, zhengxf12, liangjj09}@mails.tsinghua.edu.cn, duanhx@tsinghua.edu.cn
§University of California, Berkeley jiangjian@berkeley.edu
¶International Computer Science Institute vern@icir.org
‖Department of Computer Science, University of Georgia kangli@cs.uga.edu
**Huawei Canada tao.wan@huawei.com

***Abstract*—We describe how malicious customers can attack the availability of Content Delivery Networks (CDNs) by creating forwarding loops inside one CDN or across multiple CDNs. Such forwarding loops cause one request to be processed repeatedly or even indefinitely, resulting in undesired resource consumption and potential Denial-of-Service attacks. To evaluate the practicality of such forwarding-loop attacks, we examined 16 popular CDN providers and found all of them are vulnerable to some form of such attacks. While some CDNs appear to be aware of this threat and have adopted specific forwarding-loop detection mechanisms, we discovered that they can all be bypassed with new attack techniques. Although conceptually simple, a comprehensive defense requires collaboration among all CDNs. Given that hurdle, we also discuss other mitigations that individual CDN can implement immediately. At a higher level, our work underscores the hazards that can arise when a networked system provides users with control over forwarding, particularly in a context that lacks a single point of administrative control.**

I. Introduction

Content Delivery Networks (CDNs) are widely used in the Internet to improve the performance, scalability and security of websites. A CDN enhances performance for its customers' websites by redirecting web requests from browsers to geographically distributed CDN surrogate nodes. A surrogate serves the content directly if cached, or forwards requests to the origin site otherwise. To improve availability, surrogates absorb distributed denial-of-service (DDoS) attacks by distributing the attack traffic across many data centers. Some CDN providers also provide WAF (Web Application Firewall) services to normalize traffic and filter intrusions to their customer's web sites.

NDSS '16, 21-24 February 2016, San Diego, CA, USA

http://dx.doi.org/10.14722/ndss.2016.23442

In this work we present "forwarding-loop" attacks, which allow malicious CDN customers to attack CDN availability by creating looping requests within a single CDN or across multiple CDNs. Forwarding-loop attacks allow attackers to massively consume CDN resources by building up a large number of requests (or responses) circling between CDN nodes. The impact can become more severe in the (common) case where attackers can manipulate DNS records to dynamically control a loop's IP-level routing on a fine-grained basis.

Although many CDN providers have internal mechanisms (such as appending custom HTTP headers like CloudFlare's CF-Connecting-IP [19]) to detect repeated requests when they circle back, we find that an attacker can bypass such defense mechanisms by using features offered by some other CDNs to filter HTTP headers. Our experiments with 16 commercial CDNs show that all of them are vulnerable to forwarding-loop attacks, even with their existing defense mechanisms.

We also examine the threat of stealthy forwarding-loop attacks. In the *Dam Flooding Attack*, an attacker secretly and gradually accumulates a large number of pending CDN requests over a lengthy period (hours). They then trigger a huge volume of cascading traffic by suddenly providing bandwidth-consuming responses and controlling all responses to arrive simultaneously. Worse, we find that internal CDN features—such as automatic server probing (Azure China), forwarding retries (Akamai and CloudFront), and proactive decompression of `gzip`'d responses (Akamai, Baidu and CloudFlare)—can amplify the DoS effect of forwarding-loop attacks and further exacerbate the load on the CDN.

Overall, we make the following contributions:

1) We describe forwarding-loop attacks that broadly threaten CDN providers. Our study shows that the amplification attacks are severe and can consume a huge volume of resources in commercial CDNs at low cost. The attacks can potentially undermine the security provided by CDNs, which are usually considered robust against DoS attacks.

2) We performed controlled tests on 16 popular CDN providers to verify the practicality of such attacks. Although some CDN providers implemented defense mechanisms for mitigating forwarding loops, we show that those defenses can be bypassed.
3) We present the Dam Flooding attack, a highly damaging type of forwarding-loop attack.
4) We propose four approaches to preventing or mitigating forwarding-loop attacks, and discuss their advantages and limitations.

We organize the rest of this paper as follows. Section II describes CDN operation, especially forwarding and filtering techniques. In Section III we present various forwarding-loop attacks and analyze the factors affecting them. We also described our experiments to construct loops within and across CDNs, and the "Dam Flooding" attack leveraging streaming HTTP responses. We discuss possible defenses to prevent or mitigate forwarding-loop attacks in Section IV and related research regarding forwarding loops and CDN security in Section V. We conclude in Section VI.

II. Background

CDNs are distributed systems with large numbers of servers deployed across the Internet. Initially created to improve website performance and scalability, many CDNs also provide security features such as DDoS protection and Web Application Firewalls (WAFs) for websites. CDNs have evolved to become important Internet infrastructure. For example, the leading CDN provider Akamai claims that it alone delivers 15–30% of all Web traffic [1].

Web access involving a CDN includes two steps: first, a user's request is directed to a CDN server geographically close to the user; second, the CDN server obtains the content for the responding to the request. The first step is called request routing [2]. Commonly used request-routing techniques include URL rewriting and DNS-based request routing [2]. URL rewriting requires website owners to change website URLs to use CDN-assigned subdomains that resolve to CDN servers. DNS-based request routing instead works by changing the DNS resolution of website domains, either directly mapping to CDN server IP addresses, or using CNAMEs to chain to CDN subdomains. These request-routing techniques usually determine the selection of edge (entry) server, but users can also override a CDN's selection by directly connecting to a desired edge server using its IP address rather than hostname [22]. Users can obtain CDN IP addresses by resolving CDN subdomains via public platforms such as PlanetLab [20]. We verified that this technique for overriding a CDN's selection works for all CDNs in our study.

The second step, *i.e.,* how the CDN server obtains the requested content, also has two different modes: push and pull. In the push mode, website owners upload their content to the CDN's servers in advance. In the pull mode, the CDN server works as a reverse proxy with caching. It firstly tries to respond from the local cache. In the case of a cache miss, it forwards the request to the original website to retrieve the content. Most CDNs support both modes. The vulnerability we examine in this paper only occurs when using pull mode. In pull mode, the cache hit ratio becomes an important indicator for measuring

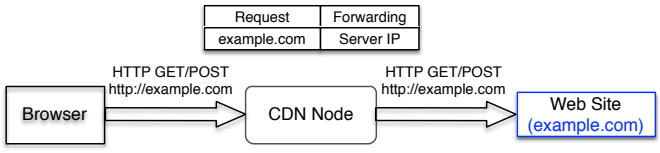

Fig. 1. Normal CDN forwarding behavior.

a CDN's performance. The higher the ratio, the more requests that the CDN answers out of its cache, significantly reducing response latency and as well as the load imposed on the origin website. However, a user can force requests to come from the origin website instead of the CDN's cache. First, adding a no-cache request header will make the CDN re-validate the response from the origin server [6]. Second, POST requests usually will write through to the origin server [7], [16]. In addition, most CDNs provide ways for customers to configure the CDN to not cache certain URLs.

Typical commercial CDNs usually have massive bandwidth and computational resources distributed around the Internet, making them much more resilient to DDoS attacks than most of websites. DDoS traffic targeting a CDN-protected website will be directed to CDN servers distributed across data centers with ample bandwidth, and then absorbed or blocked before arriving the original website. Indeed, capacity for mitigating DDoS attacks has become a "selling point" for today's commercial CDNs.

Many CDNs also provide an additional security service called content filtering, or WAF. A WAF applies a set of rules to each HTTP request and response. Generally, these rules cover common attacks such as cross-site scripting (XSS) and SQL injection. By customizing WAF rules, customers can have their CDNs examine HTTP requests and filter out some suspicious traffic before it reaches the origin website.

III. Forwarding-Loop Attacks

Malicious customers of CDNs can deliberately manipulate the forwarding process (in the pull mode) to create forwarding loops inside CDNs. Forwarding loops can cause CDNs to process one client request repetitively or even indefinitely. The consequent amplification effect allows malicious customers to launch, with little resources and cost, resource-consuming DoS attacks against CDNs.

In general, as shown in Figure 1, before a CDN node forwards an HTTP request from a client, it checks the Host header of the request to look up any customer-specified forwarding destination. The node then connects to the forwarding destination and relays the request. In the benign case, the forwarding destination returns a response that is further relayed by the CDN node to the client. However, if an attacker intentionally configures the forwarding destination to point to another CDN node, the forwarding process can continue, and might eventually form a loop. Figure 2 illustrates a conceptual view of a forwarding loop between three CDN nodes. Note that the three nodes could be distributed either within a single CDN or across different CDNs.

We have identified four approaches to create forwarding loops: 1) *self loop*, which loops within a single CDN node;

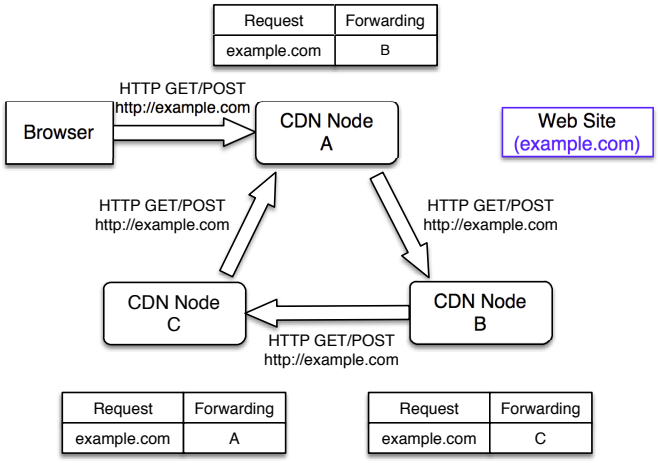

Fig. 2. A conceptual view of a CDN forwarding loop created by manipulating forwarding configuration: see Section III-B through Section III-E for the detailed mechanisms for constructing forwarding loops.

TABLE I. VULNERABILITY OF THE MEASURED CDNS TO FOUR TYPES OF FORWARDING-LOOP ATTACKS. *("Likely" refers to inference from indirect evidence.)*

	Self-Loop	Intra-CDN loop	Inter-CDN loop	Dam Flooding
Akamai			✓	✓
Alibaba			✓	✓
Azure (China)	✓	✓	✓	✓
Baidu			✓	✓
CDN77		✓	✓	✓
CDNlion		✓	✓	✓
CDN.net		✓	✓	✓
CDNsun		✓	✓	✓
CloudFlare			✓	✓
CloudFront			✓	✓
Fastly			✓	✓
Incapsula			✓	✓
KeyCDN	Likely	✓	✓	✓
Level3			✓	✓
MaxCDN	Likely	✓	✓	✓
Tencent			✓	✓

2) *intra-CDN loop*, which loops around multiple nodes of one CDN provider; 3) *inter-CDN loop*, which loops across multiple CDNs; and 4) *CDN Dam Flooding*, which couples forwarding-loop attacks with timely controlled HTTP responses to significantly increase damage.

We gathered a list of popular CDNs[1] and signed up with those (or their resellers) that provide free or free-trial accounts without strong customer identity verification (except Alibaba and Tencent, per Section III-G). We then measured various aspects of the CDNs using our testing accounts. This approach enables us to measure 16 popular CDNs around the world. We found all of them vulnerable to some form of forwarding-loop attacks. Table I presents the 16 CDNs and their vulnerability to the four types of attack. While most CDNs can defend against the first attack, little more than half can defend against the second, and none can defeat the last two.

We chose to measure the 16 CDNs that provide free or free-trial accounts without strong identify verification to emphasize the fact that forwarding-loop attacks can be launched anonymously and with little cost. Rigorous customer authentication can help raise the bar, but it does not suffice to prevent forwarding-loop attacks. We further discuss the issue of anonymity and cost in Section III-G.

As we will present in detail, the root cause of forwarding-loop attacks is that CDN customers have flexible control over their forwarding configuration, and CDNs lack sufficient defensive mechanisms to ensure that these configurations—especially across multiple customers or multiple CDNs—will not cause requests to be processed repeatedly. The fact that CDN customers can override edge-server selection of CDNs (as explained in Section II) further enables forwarding-loops attacks on any CDN public IP address or data center that an attacker seeks to target. Moreover, we identified a number of factors that affect the efficacy of forwarding-loop attacks. In the following sections, we first discuss how these factors interact with forwarding-loop attacks and vary across CDN implementations. We then present detailed mechanisms for the four attacks, along with measurements and experiments to assess them.

A. *Factors affecting Forwarding Loops*

Modification of the Host header. The Host header of a request plays a key role in the forwarding process, as well as in forwarding-loop creation. A necessary condition for a request to create a forwarding loop is that all involved CDN nodes must forward the request in such a way that the successor node treats it as a benign request, and continues the forwarding.

Whether the successor node accepts the forwarded request depends on the Host header. We can classify forwarding loops into two categories based on whether the Host header changes during the forwarding loop. Figure 2 shows the first category: a request is issued for the original domain of the website, and when forwarded by a CDN node, its Host header does not change, thus not affecting acceptance and further forwarding of the forwarded request.

Another type of forwarding loop has a changing Host header. Our measurements show that CDN nodes can change the Host header to reflect the forwarding destination, depending on the request-routing mechanism and the form of the forwarding destination. Table II presents detailed results. We note that forwarding loops are feasible as long as all involved nodes keep a valid domain name in the Host header, but can be prevented by simply modifying the Host header to an IP address, because we find that no CDN accepts requests with an IP address in the Host header. As shown in Table II, this case only occurs at KeyCDN and MaxCDN when the request is

[1]Most from http://www.cdnplanet.com/cdns/.

TABLE II. HOST MODIFICATION BEHAVIORS. *("N/A" indicates that the feature is either not available for testing due to our account's limitations, or not applicable.)*

	Request with CDN Subdomain		Request with Customer Domain	
	Forwarding to IP	Forwarding to Domain	Forwarding to IP	Forwarding to Domain
Akamai	N/A		Configurable	
Alibaba	Configurable		Configurable	
Azure (China)	N/A		Request Domain	
Baidu	N/A		Request Domain	
CDN77	Request Domain	Forwarding Domain	Request Domain	Forwarding Domain
CDNlion	Request Domain	Forwarding Domain	Request Domain	Forwarding Domain
CDN.net	Request Domain	Forwarding Domain	Request Domain	Forwarding Domain
CDNsun	Request Domain	Forwarding Domain	Request Domain	Forwarding Domain
CloudFlare	N/A		Request Domain	
CloudFront	N/A	Request Domain	N/A	Forwarding Domain
Fastly	N/A		Request Domain	
Incapsula	N/A		Request Domain	N/A
KeyCDN	**Forwarding IP**	Forwarding Domain	Request Domain	Forwarding Domain
Level3	N/A		N/A	Request Domain
MaxCDN	**Forwarding IP**	Forwarding Domain	Configurable	
Tencent	N/A		Request Domain	

issued with the CDN's subdomain and the form of forwarding destination is IP address. In all other cases we could test, the feasibility of forwarding loops is not affected.

Modification of other header fields. CDNs also vary regarding changing other header fields when forwarding a request. Such behaviors, summarized in Table III, affect the efficacy of forwarding loops.

We first find that 9 CDNs depend on standard or self-defined headers to detect forwarding loops. We measured these results by connecting our origin server and, separately, each commercial CDN node in a loop. If requests in the loop always stop in a short time unless we remove a certain header or set the value of certain header on our origin server, then we deduce that the CDN uses the header for loop detection. We find that Akamai and Tencent add Akamai-Origin-Hop and X-Daa-Tunnel headers with integer values that count forwarded hops. These appear to restrict forwarding to maximum values of 12 and 6, respectively. Alibaba, CloudFront and Level3 append standard Via headers with the server's hostname. They also check for the presence of certain strings within any existing Via header to detect loops. Fastly also appends a self-defined header Fastly-FF with its hostname, and rejects a request if its hostname already appears in the header value. Incapsula adds a new header, Incapsula-Proxy-ID, with the ID set to its internal identifier, basing loop detection on the presence of this header. Baidu and CloudFlare servers append their IP addresses to the X-Forwarded-For header, and also add the self-defined header CF-Connecting-IP (Baidu confirmed that they have a partnership with CloudFlare, which CloudFlare later announced). Baidu and CloudFlare servers reject a request if its IP address already appears in the X-Forwarded-For header, or given the presence of a CF-Connecting-IP header.

TABLE IV. HEADER SIZE LIMITATION (SINGLE/ALL HEADERS)

Vendor	Limitation	Vendor	Limitation
Akamai	16KB/16KB	CloudFlare	32KB/92KB
Alibaba	32KB/64KB	CloudFront	24KB/24KB
Azure (China)	20KB/20KB	Fastly	64KB/64KB
Baidu	32KB/92KB	Incapsula	25KB/>1600KB
CDN77	16KB/64KB	KeyCDN	8KB/32KB
CDNlion	16KB/64KB	Level3	9KB/12KB
CDN.net	16KB/64KB	MaxCDN	32KB/156KB
CDNsun	16KB/64KB	Tencent	6KB/6KB

We also find that all CDNs except KeyCDN, MaxCDN, and Tencent increase the header size whenever forwarding a request, usually by adding or appending header fields like Via or X-Forwarded-For, although not necessarily using these fields for loop detection. This behavior causes forwarding loops to eventually stop, because all CDNs implement bounds on the header size of acceptable requests. If in each round of a forwarding loop, the header size of the request increases, then the loop will break when the header size exceeds the bound at any node. Table IV summarizes the header size limitations of different CDNs.

Several CDNs reset the value of certain header fields instead of appending on them. CDN77, CDN.net, CDNlion and CDNSun reset the Via header, and KeyCDN resets the X-

TABLE III. HEADER (EXCEPT HOST) MODIFICATION BEHAVIORS

	Size Increase	Loop Detection	Reset	Filtering
Akamai	Via, X-Forwarded-For	Akamai-Origin-Hop		
Alibaba	Via, X-Forwarded-For	Via		
Azure (China)	X-Forwarded-For			
Baidu	X-Forwarded-For	X-Forwarded-For, CF-Connecting-IP		
CDN77	X-Forwarded-For		Via	
CDNlion	X-Forwarded-For		Via	
CDN.net	X-Forwarded-For		Via	
CDNsun	X-Forwarded-For		Via	
CloudFlare	X-Forwarded-For	X-Forwarded-For, CF-Connecting-IP		
CloudFront	Via, X-Forwarded-For	Via		
Fastly	Fastly-FF, X-Varnish	Fastly-FF		Non-self-defined
Incapsula	Incap-Proxy-ID, X-Forwarded-For	Incap-Proxy-ID		
KeyCDN			X-Forwarded-For	
Level3	Via, X-Forwarded-For	Via		
MaxCDN				Any header
Tencent		X-Daa-Tunnel		

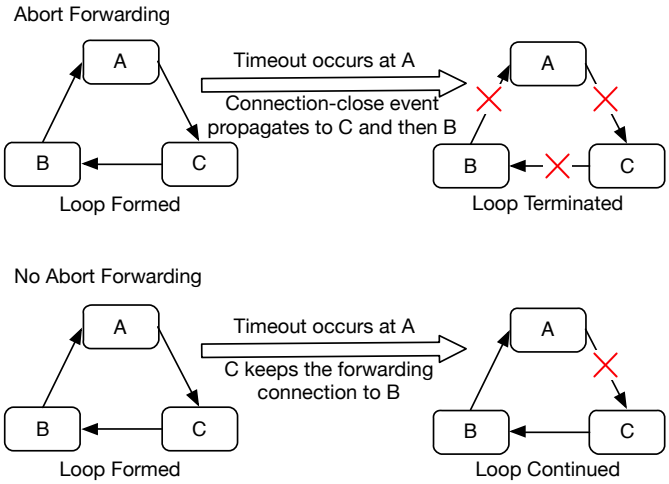

Fig. 3. The differences between abort forwarding and no abort forwarding.

Forwarded-For header to its own IP address. As we shall see, these behaviors cause undesirable interactions that increase the efficacy of forwarding-loop attacks.

Fastly and MaxCDN support WAFs that allow customer-defined rules to remove HTTP headers in requests [5] [14]. According to our measurements, Fastly prevents removal of the headers added by its own servers, while MaxCDN does not appear to impose any such limitation.

Handling timeouts. After forwarding a request to its destination, a CDN node waits for a response until a timeout occurs. Table V shows the timeout periods we measured, ranging from 60 seconds to 900 seconds.

TABLE V. FORWARDING TIMEOUTS AND THE ADOPTION OF ABORT FORWARDING.

	Forwarding Timeout (second)	Abort Forwarding
Akamai	240	
Alibaba	60	✓
Azure (China)	900	✓
Baidu	100	✓
CDN77	60	
CDNlion	60	
CDN.net	60	
CDNsun	60	
CloudFlare	100	✓
CloudFront	90	✓
Fastly	configurable (max 75)	
Incapsula	360	✓
KeyCDN	60	✓
Level3	60	
MaxCDN	60	✓
Tencent	10	✓

When a timeout occurs at a node in a forwarding loop, the node closes the corresponding connection to its successor. This closing action triggers a *client-side connection close* event at its successor node. If the successor node reacts by *abort forwarding*, *i.e.,* closing the corresponding forwarding

TABLE VI. DNS RESOLUTION BEHAVIORS.

	DNS Cache (resolver)	Minimum TTL (second)
Akamai	per CDN node	≈ 60
Alibaba	per data center	≈ 60
Azure (China)	per CDN node	≈ 0
Baidu	per CDN node	≈ 60
CDN77	Google Public DNS	≈ 0
CDNlion	Google Public DNS	≈ 0
CDN.net	Google Public DNS	≈ 0
CDNsun	Google Public DNS	≈ 0
CloudFlare	per data center	≈ 0
CloudFront	per data center	≈ 0
Fastly	per CDN node	≈ 0
Incapsula	N/A	N/A
KeyCDN	Google Public DNS	≈ 0
Level3	per CDN node	≈ 5
MaxCDN	per CDN node	≈ 0
Tencent	per data center	≈ 0

TABLE VII. SUPPORT OF HTTP STREAMING.

	Request Streaming	Response Streaming
Akamai	✓	✓
Alibaba	✓	✓
Azure (China)	✓	✓
Baidu		✓
CDN77		✓
CDNlion		✓
CDN.net		✓
CDNsun		✓
CloudFlare		✓
CloudFront	✓	✓
Fastly	✓	✓
Incapsula	✓	✓
KeyCDN		✓
Level3	✓	✓
MaxCDN		✓
Tencent		✓

connection, the close event will further propagate to the next node, and so forth. In a forwarding loop, abort forwarding propagates faster than request forwarding, because a client-side connection close event occurs immediately after receiving a single FIN packet, while receiving and forwarding a request requires many more packets. If all CDN nodes involved in a forwarding loop adopt abort forwarding, then the reaction triggered by the timeout will eventually catch up with the request forwarding to stop the loop. In this way, the life-cycle of the forwarding loop is bounded by the minimum timeout of the nodes plus the time for the abort event to catch up to the forwarding. However, if one node does not implement abort forwarding, the abort propagation will stop at that node; consequently, the request continues to circulate among all nodes in the loop.

Figure 3 illustrates how a timeout event is propagated to stop a forwarding loop if all nodes in the loop implement abort forwarding; and, in comparison, how a timeout event is locally limited if not all nodes support abort forwarding.

As shown in Table V, numerous CDNs do not adopt abort forwarding when a client-side connection closes.

DNS resolution behaviors. Per Table II, 15 out of 16 measured CDNs (except Incapsula) support using domain names as forwarding destinations. For these CDNs, an attacker can change the DNS records of the forwarding domains to control an ongoing forwarding loop dynamically, *e.g.,* switching the loop from one set of IP addresses to another set of IP addresses.

Our measurements show that, in general, CDNs do not share DNS results (via common caches or resolvers) across their servers or data centers. They also respect the time-to-live (TTL) value in DNS responses. Per Table VI, of the 15 CDNs supporting the use of domain names as forwarding destinations, 6 (Akamai, Azure China, Baidu, Fastly, Level3 and MaxCDN) deploy independent DNS resolvers on each node. Alibaba, CloudFlare, CloudFront and Tencent have one or more DNS resolvers shared per data center. The others use Google Public DNS, which deploys different instances across geographical locations using anycast. Among the 15 CDNs we could measure, Akamai, Alibaba, and Baidu's DNS resolvers set a minimum TTL of 60 seconds; Level3's resolver can set a minimum TTL of 5 sec; the other CDNs appear to respect TTL values in DNS responses even when set to zero (no caching). This latter behavior allows an attacker to dynamically reroute an ongoing loop in a fine-grained and timely manner.

We note that self-loops, intra-CDN loops, and inter-CDN loops do not require dynamic rerouting via DNS, but its availability provides additional flexibility for attackers to create and control forwarding loops. For dam-flooding attacks, this feature is required in the flooding phase to change the forwarding destination (Per Section III-E).

Non-streaming versus streaming. Attackers expect that forwarding loops should not only last indefinitely, but also propagate data as quickly as possible in order to consume maximal resources. One important factor related to the speed of a forwarding loop is whether a CDN supports HTTP streaming. HTTP streaming is a feature of HTTP 1.1 enabled by announcing a `Transfer-Encoding: chunked` header instead of the `Content-Length` header. It provides a persistent connection to transmit dynamically generated content on demand without knowing the content length in advance. For forwarding loops, a streaming-compatible CDN node will start relaying a request or response to its next hop immediately after receiving its initial chunk, rather than waiting for the complete content. This makes the loop circulate faster. In order to initiate a forwarding loop with HTTP streaming, all involved nodes must support this feature. Our measurements show that while 9 out of the 16 CDNs do not accept HTTP streaming in requests,

all support streaming responses (Table VII).

Figure 4 presents how non-streaming and streaming loops generate different traffic patterns. We presume the path between two nodes A and B is symmetric with network latency l, and the request (or response) circling around A and B requires time t to fully transmit. Assuming the data is always transmitted at full bandwidth, both the non-streaming and the streaming loops generate square waves along (each direction of) the path between A and B, with the same pulse height representing the full bandwidth, and the same pulse width reflecting t. Yet, the periods of the two waves (*i.e.,* the round-trip times of the two loops) are different. While the square wave generated by the non-streaming loop has a period of $2 \times (t + l)$, the square wave caused by the streaming loop has a period of $2 \times l$.

As the streaming loop runs faster, it keeps the path busier (in both directions). If the data is large enough so that $t \geq 2 \times l$, then the neighboring traffic pulses caused by the streaming loop overlap, which means that the path is fully occupied. In practice, overlaps of two or more rounds of a streaming loop could also result in higher traffic peaks than that of a non-streaming loop, because the data transmission between two successive nodes might not be able to utilize all available bandwidth due to factors such as TCP's congestion control.

We conducted a local experiment to verify our analysis. We set up two Nginx 1.8.0 servers, both connected to the same Ethernet. To simulate an Internet environment, we used the `tc` Traffic Control tool to add 125 ms of network latency for each server. In this setting, the full bandwidth is 100Mb/s and the network latency is 250ms. We first sent a single 500KB POST request to create a streaming loop between the two servers by configuring their Nginx instances to disable request body buffering. We then repeated the procedure with a non-streaming loop setting.

Figure 5 shows the traffic in one direction generated via non-streaming and streaming loops. As expected, the non-streaming loop generates a periodic wave; each distinct pulse represents one round of the loop; the peaks near 179KB/s. In comparison, multiple rounds of the streaming loop overlap because the time needed to transmit the request is much higher than the network latency (with the effect of TCP slow start), resulting in a curve without distinct pulses and much higher peaks (about 443KB/s).

B. Self-Loop

Self-loops occur when requests are forwarded circularly within a single CDN node. The attack is simple to mount: the attacker only needs to specify the forwarding destination of their domain as the loopback address (*i.e.,* 127.0.0.1), or the IP address of a given CDN node. Yet self-loops can be particularly damaging, because the circulation happens without network latency, potentially consuming resources very quickly.

We found that 13 out of 16 CDNs' web interfaces accept the loopback address or the IP address of their nodes as forwarding destinations. Baidu and CloudFlare however do not allow such forwarding destinations. CloudFront further rejects specifying forwarding destinations using any raw IP address or "localhost". It is worth noting that merely enforcing a blacklist

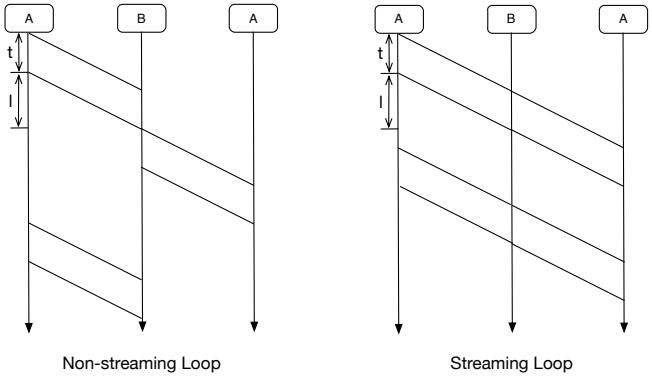

Fig. 4. The difference between non-streaming and streaming loops.

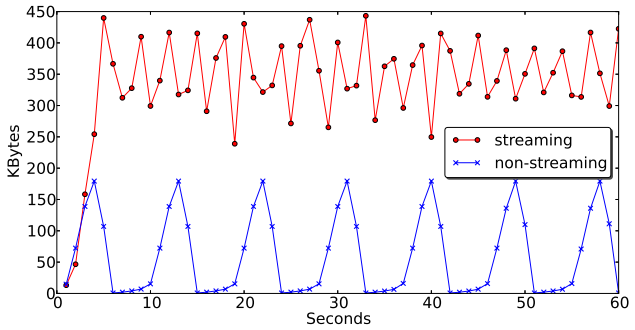

Fig. 5. Traffic generated by a single request (500KB) in a non-streaming versus a streaming loop.

of loopback and internal IP addresses at the web interface does not suffice to defend against self-loops. For CDNs supporting domain names as forwarding destination, the attacker can use this feature to bypass blacklists implemented at the web interface. For example, CloudFlare allows specifying a CNAME domain for the forwarding destination, enabling an attacker to later change the resolution to the loopback address or a CloudFlare IP address.

We also tested three popular open-source reverse proxies that are commonly used by commercial CDNs: Squid, Nginx, and Varnish. Both Nginx and Varnish by default allow self-loops, and we also could not find any option or popular extension for loop-prevention. Squid prevents loops by adding a Via header to forwarded requests and rejecting incoming requests that contain the same hostname in its Via header. This defense is similar with those of the 9 loop-aware CDNs presented in Table III.

Testing the feasibility of self-loop attacks on commercial CDNs requires care to avoid potentially inducing considerable damage. The 9 loop-aware CDNs are not vulnerable to this attack, while the other 7 CDNs are likely vulnerable. Among the 7 CDNs, 5 (Azure China, CDN77, CDNlion, CDN.net, CDNsun) have size-increasing headers, per Table IV. For these CDNs, we found a technique to infer some further information.

We first send a request to one CDN node with a size exceeding its maximum value, and record the corresponding response (*e.g.,* 400 Bad Request—Request Header Or

Cookie Too Large). Next, we send another self-loop request to the same node but slightly smaller (200 bytes less) than the size limit. Doing so ensures that if the CDN is vulnerable to self-loops, the crafted request can at most only loop a few times before reaching the header size limitation. If for both requests we observe the same response indicating an excessive request size, we can infer that the CDN is vulnerable to self-loop attacks. Otherwise, we conclude that the CDN prohibits request forwarding to the loopback or self address. Using this technique to test the 5 CDNs, we find only Azure (China) is vulnerable to self-loop attacks.

The remaining two CDNs (KeyCDN and MaxCDN) are still likely vulnerable to self-loop attacks.

Experiments. We conducted two local experiments using a Linux machine running an Nginx server to understand the potential consequences of self-loops. We first tested with the default configuration of Nginx, finding that request-forwarding to a loopback address circulated 511 times in ≈ 0.1 seconds before returning the error response 400 Request Header Or Cookie Too Large. By default Nginx limits the size of a single header to not exceed 8KB. When forwarding a request, Nginx appends its address in a X-Forwarded-For header, causing the header size to increase. We then removed the header size limitation and conducted the experiment again. This time the self-loop ran 28,231 times in 5 seconds, ultimately returning a 504 error because the loop had exhausted all of the source ports available for loopback connections. (The Linux kernel's default port range for a user-space application spans 32,768–61,000.)

As presented in Table III and Table IV, Azure (China) is vulnerable to self-loop attacks and increases the header size when forwarding a request; therefore, it is subject to the case demonstrated in the first experiment. Self-loops on KeyCDN and MaxCDN, which do not increase the header size per Table IV, likely behave like the second experiment; that is, they could exhaust all source ports of localhost before a timeout occurs (60 seconds, per Table V).

C. Intra-CDN Loops

Attackers can also create forwarding loops across multiple nodes within a single CDN. As mentioned above, 15 CDNs (all except Incapsula) allow customers to use domain names as forwarding destinations. When forwarding a request to a domain, 10 of the 15 CDNs (except Azure China, Baidu, CloudFlare, Fastly and Tencent) change the Host header to reflect the forwarding domain. For each of these CDNs, attackers can create forwarding loops across multiple nodes by chaining multiple attacking accounts using multiple forwarding domains. For example, they can set up account A_1 forwarding domain D_1 to domain D_2, account A_2 forwarding domain D_2 to domain D_3, and so forth. Account A_n closes the loop by forwarding domain D_n to domain D_1. This creates a loop across n domains, which can further be mapped to different CDN nodes.

Attackers can also create loops across multiple CDN nodes by dynamically changing forwarding destinations using DNS. As shown in Table VI, for the 15 CDNs supporting domain names for forwarding, none of these CDNs share a global DNS cache. Thus, different CDN nodes will independently resolve an attacker's forwarding domain. Attackers can create loops between two nodes A and B of the same CDN by controlling the DNS resolution of their forwarding domains so that queries from A are provided with the IP address B, and vice versa. Depending on how a CDN manages its DNS resolutions, the attacker might need to select A and B from different data centers or regions.

That said, we note that this attack does not affect the 9 CDNs that employ loop-detection headers.

Experiments. We conducted a proof-of-concept experiment on MaxCDN. We used two MaxCDN nodes plus one VPS (Virtual Private Server) under our control, employing the second strategy described above to form a three-node loop. The VPS acts as a transparent HTTP proxy to collect data and minimize harm. We also added a 0.6 seconds delay at our HTTP proxy to slow down the loop speed to ensure that the experiment did not cause significant real-world damage. We ran the experiment for 60 seconds and received 59 requests at our VPS for only one request we sent out.

D. Inter-CDN Loops

If attackers extend the multiple-node forwarding loop to span multiple CDNs, they can evade the protection of loop-detection headers to attack all 16 CDNs. This approach works by chaining loop-aware CDNs with other CDNs that disrupt the loop-detection headers.

As presented in Section III-A and Table III, Fastly and MaxCDN provide customer-defined header filtering. The header filtering feature of Fastly does not facilitate evading loop detection because Fastly adds a non-filterable loop detection header. However, including MaxCDN in a chain enables disrupting all loop-detection headers because it provides unlimited header filtering. I.e., attackers only need to add one MaxCDN node in their forwarding loops to attack even loop-aware CDNs.

The behavior of resetting headers also enables evasions of loop detection. As shown in Table III, CDN77, CDNlion, CDN.net and CDNsun reset Via, a standard header used by Alibaba, CloudFront and Level3 to detect forwarding loops. Therefore, attackers can mount a forwarding loop between any one node from the former 4 CDNs and nodes from the latter 3 CDNs.

Another use of header filtering and header resetting is to counter the effect of increasing header size so that the life-cycle of a forwarding loop escapes the bound normally imposed by header-size limitations. For example, we can form a loop among one CloudFront node, one CDN77 node and one KeyCDN node. The CDN77 node resets the Via header used by CloudFront for loop detection. The KeyCDN node resets the X-Forwarded-For header, which appears to be the only header whose size would otherwise steadily increase by CloudFront and CDN77. KeyCDN itself does not detect loops, nor does it increase header sizes. In addition, because CDN77 does not adopt abort forwarding, forwarding timeouts will not terminate the forwarding loop. In principle, such loops could last indefinitely.

Experiments. We created a forwarding loop among 4 systems: CloudFlare, CDN77, MaxCDN and a server under our

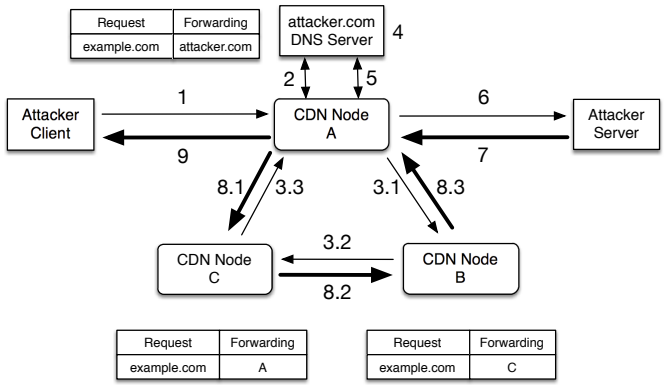

Fig. 6. The process of a CDN Dam Flooding Attack: 1) the attacker client sends a request to A; 2) A queries `attacker.com` to forward the request, and is directed to B; 3) the request circulates across B, C, and back to A; 4) the attacker points `attacker.com` to his server; 5) the next DNS query for `attacker.com` from A is mapped to the attacker's server; 6) A forwards the request to the attacker's server; 7) the attacker's server replies with a streaming response; 8) the streaming response flows through C and B, then back to A, repeating many times; 9) A finally relays the response to the attacker client.

control. We configured MaxCDN to delete any header that detects loops or increases header size. We use the CDN77 node's no-abort-forwarding to counter the effect of forwarding timeouts. Again, we used our server as a transparent HTTP proxy with a delay of 0.6 seconds to collect data and limit the resource load imposed by the loop. We initiated the loop using a single request; it lasted more than 5 hours, passing 17,266 requests through our server. When the loop finally stopped, a 522 error was received, indicating that CloudFlare could not connect to our server. Our server also received many retransmitted TCP packets, from which we infer that the loop ceased because of network connectivity issues between CloudFlare and our server.

E. The CDN Dam Flooding Attack

As presented in Section III-A, HTTP streaming makes forwarding-loop attacks more potent by enabling them to "fill the pipe" with traffic. However, for the attacks discussed above, Azure (China) is the only applicable target for streaming loops, because it is the only CDN that both supports streaming requests and does not deploy loop detection (per Table III and Table VII). Since all CDNs we examined support HTTP streaming for *responses*, we can extend the attacks by employing responses rather than requests to create streaming loops.

We call this attack "CDN Dam Flooding" because it involves two phases analogous to the filling and flooding of a dam. Figure 6 shows how the attack works. In the filling phase, the attacker launches a number of forwarding loops via the strategies described in Section III-C or Section III-D, using domain names as forwarding destinations. In the flooding phase, the attacker changes the resolution of these names to direct the forwarding destinations to a server of the attacker's that replies to incoming requests with a large file transmitted using HTTP streaming. For each forwarding loop, a streaming response flows along the CDN nodes in reverse order, for multiple rounds, until reaching a broken connection caused by a forwarding timeout at some CDN node, or the client

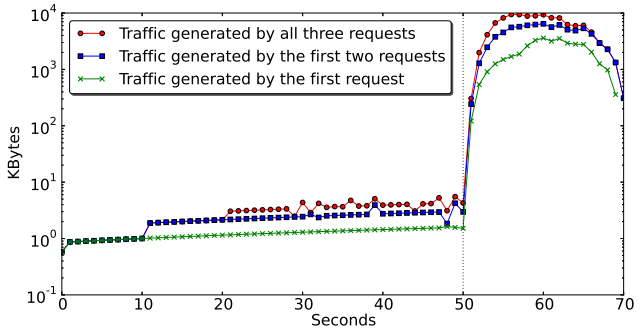

Fig. 7. Traffic generated by three forwarding loops in a "dam flooding" attack. The flooding event occurs at 50 seconds).

that initiated the loop. While the filling phase itself generates some attacking traffic, the flooding phase *bursts* the traffic by utilizing HTTP streaming with large and continuous chunks, with the impact of each chunk magnified by the number of turns it makes around the forwarding loop. The attacker can also coordinate DNS resolution to flood all filled forwarding loops simultaneously. The overlap of multiple streaming loops serves to enlarge the traffic burst.

We note that dynamically changing forwarding destinations using DNS is not a necessary condition to create streaming loops. For instance, in the example given in Section III-C, instead of chaining the attacking account A_n back to A_1, the attacker can alter the entry for A_n to point to their server, so that a request becomes forwarded to his server after n hops between CDN nodes, with a streaming response then fed back along the flow in reverse. That said, using DNS provides the attacker with more control on how and when to "flood" the filled loops.

Experiments. To assess the efficacy of this attack in practice without unduly stressing a production CDN, we set up our own VPS as the victim CDN node, imposing a strict traffic limitation of no more than 100Mb/s. On CDN77, we configure the forwarding destination to a domain under our control. On our VPS, we configure the forwarding destination to a CDN77 IP address. In this way, we create a forwarding loop between our VPS and the CDN77 node. Note that our VPS has the abort-forwarding feature, does not support request streaming, but does support response streaming.

In the filling phase, we respond to DNS queries with the IP address of our VPS and then send a single 366-byte request to the CDN77 node 3 times spaced 10 seconds apart. Thus, the attack uses a total of three small initial requests. We then wait for the three requests to loop between the CDN77 node and our CDN node for 30 seconds.

In the flooding phase, we change the DNS replies to direct the three loops to our web server. Our server replies to any request with a 1 MB file, sent using HTTP streaming.

Figure 7 shows the HTTP traffic on our VPS during the filling and flooding phases. The burst attack lasts in total for about 69 seconds. During the first phase, the three forwarding loops slowly increase the traffic volume from zero to 7 KB/s over 50 seconds. In the second phase, the traffic volume immediately peaks, reaching about 9.2MB/s. While we as the

attacker sent out three requests and three responses totaling about 3MB traffic, our VPS as a victim received about 224MB, an amplification factor of 74.

Combining with `gzip` bombs. This attack can be substantially enhanced if the attacker incorporates `gzip` bombs. In step 7 of Figure 6, the attacker needs to send a large response to the CDN as quickly as possible to increase the peak burst of the attack. `gzip` bombs, which are small compressed files easy to transport across a network, can help a great deal to achieve this goal. When unpacked by a CDN, they balloon into extremely large output.

A key factor of this attack is whether CDNs will decompress `gzip`'d responses. To assess this, we conducted a measurement of the 16 commercial CDNs. First, our client sent a request to the CDN indicating that it does not accept `gzip`-encoded HTTP replies. Next, our original server returned a `gzip`'d response. If the client receives decompressed content, this means that the CDN will decompress `gzip`'d responses. We found that 3 (Akamai, Baidu and CloudFlare) out of the 16 CDNs will decompress `gzip`'d responses for clients that do not support "`gzip`" encoding.

Although only 3 CDNs can be exploited by `gzip` bombs, we emphasize that adding one `gzip`-decompressing node into a loop suffices to attack all involved nodes with the effect of `gzip` bombs, even if the other nodes do not support `gzip` decompression. For example, in the scenario of Figure 6, even if the three nodes A, B, C do not support `gzip` decompression, the attacker can direct step 6 to a `gzip`-decompressing node, which forwards the request to the attack server and is fed a `gzip` bomb in return. The `gzip`-decompressing node then forwards the large unpacked response to node A, where it further loops among the three nodes.

To estimate the maximum amplification factor a `gzip` bomb can provide, we performed a simple local experiment. We first use `dd` to generate a 100GB file containing only the character '1'. We then compressed it using `gzip` with compression level 9, yielding a 96.2MB file, reflecting a compression ratio of 1,064. The compression ratio serves as an extra amplification factor (in addition to the number of times that the response loops) to significantly enlarge the attack traffic.

With Baidu's permission, we used the Baidu CDN to conduct two experiments to verify the feasibility and the efficacy of dam flooding attacks with `gzip` bombs. We set up two local CDN servers using Nginx; created a forwarding loop between them; and set their network latency to 200 ms. In the filling phase, we sent a single GET request into the loop. After 10 seconds, we pointed the forwarding destination to our web server, sited behind Baidu. In the first experiment, our server replied to the request with a uncompressed 1MB file consisting of a single repeated character. We then repeated the procedure with a 1KB file reflecting a `gzip`'d version of the previous 1MB file.

In the first experiment, the 1MB response looped 16 times, with the traffic received at one local server totaling 16.6MB, an amplification factor of approximately 17. In comparison, in the second experiment the 1KB response looped 17 times, and at one server induced a total traffic volume of 17.7MB. This results in an amplification factor of approximately 17,000—1,000 times that of the first experiment.

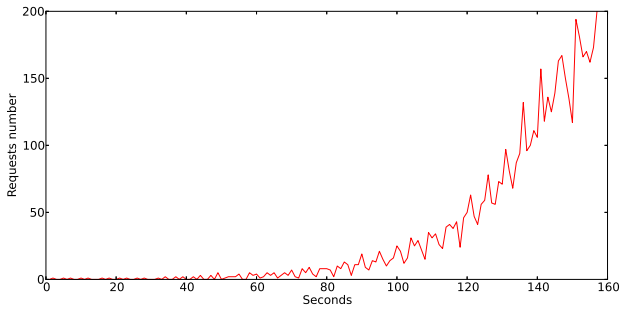

Fig. 8. Traffic generated by one request due to CloudFront's retransmission.

F. Other CDN Quirks

We also observed two rare behaviors that can further enhance the efficacy of forwarding-loop attacks.

Aggressive active probing. We found that Azure (China) proactively and frequently issues HTTP requests to forwarding destinations, presumably for availability testing. We configured a forwarding destination on Azure (China) and monitored for 36 hours. In total we received 106,764 requests from 69 different IP addresses. This behavior—if intended rather than a bug or misconfiguration—would allow attackers to generate forwarding loops without even using an initiator.

Forwarding retries. We also found that when the origin does not give a response in certain time, CloudFront and Akamai will *retransmit* requests to the origin websites. Upon receiving a request from a client, a CloudFront server forwards the request to its forwarding destination. If it does not receive a response, the server then retransmits the request twice, 30 seconds and 60 seconds after first sending it, respectively, before returning a timeout error to the client after 90 seconds. Akamai servers also retry one time at 120 seconds before a final timeout at 240 seconds. In forwarding-loop attacks, each request retransmission kicks off a new loop. In addition, even if the server closes the previous forwarding connection before issuing a retransmission, the original loop will still continue if any node in the loop does not implement abort forwarding. Together, these behaviors can make the number of loops increase exponentially.

To examine these possibilities, we created a forwarding loop between a CloudFront server and our HTTP forwarder. Our forwarder did not support abort forwarding or request streaming. We sent a single request (376 bytes) to the CloudFront server and captured HTTP traffic at our forwarder. After 156 seconds, we manually stopped the loop by killing the process of our forwarder, to avoid adversely affecting the CloudFront platform. Figure 8 shows the results. We see that the number of requests starts to increase at 30 seconds and does so much quickly every subsequent 30 seconds, reaching 200 at the end of the experiment. During the experiment, our forwarder received a total of 3,096 requests sent by the CloudFront server, even though we only sent one request.

TABLE VIII. CDN REGISTRATION REQUIREMENTS AND COST.

	Register Requirements	Price	Anonymity
Akamai	Email address Credit card	Free trial	✓
Alibaba	Email address Phone number Bank card	Free trial	✓
Azure (China)	Email address Phone number	Free trial (1 CNY)	✓
Baidu	Email address	Free service	✓
CDN77	Email address	Free trial	✓
CDNlion	Email address	Free trial	✓
CDN.net	Email address	Free trial	✓
CDNsun	Email address	Free trial	✓
CloudFlare	Email address	Free service	✓
CloudFront	Email address Credit card	Free trial	✓
Fastly	Email address	Free service	✓
Incapsula	Email address	Free service	✓
KeyCDN	Email address	Free trial	✓
Level3	Email address	Free trial	✓
MaxCDN	Email address	Free trial	✓
Tencent	Email address Phone number Bank card	Free trial	✓

G. Anonymity and Cost

One may argue that these attacks cannot be launched in the real world because of the associated costs and risk of exposing the attacker's identity. However, CDN providers, presumably for competitive reasons, provide much convenience for their prospective customers (and thus for attackers). Table VIII shows the registration information required to begin using the free or free-trial services of the CDN providers in our study. 11 out of 16 CDNs require only a valid email address. Akamai and CloudFront CDNs require valid credit cards (could be gift cards, or stolen), Azure (China) requires valid phone number (could be anonymous). Alibaba and Tencent require users to verify their identity through a valid bank card, which takes an attacker more effort to keep anonymous.

H. Disclosure and Response

We attempted to contact all 16 CDN vendors. For 4 CDNs (CDNlion, CDN.net, CDNsun and KeyCDN), we could not find specific security contacts, and our messages to the general email addresses found on their websites or WHOIS information did not receive any reply. For the other 12 CDNs, we were able to provide detailed report to their security contacts, and 9 replied (all but Incapsula, Level3 and MaxCDN). In addition, Verizon (EdgeCast) contacted us to discuss the problem after learning of this issue from one of their clients, even though we did not include their service in our study because they do not offer anonymous customer accounts. We also reported the problem to CNCERT/CC and the CERT coordination center (CERT/CC) through the HackerOne platform.[2] Below we summarize the discussions.

CloudFlare: acknowledged our report and particularly thanked us for reporting the problem of `gzip` bombs. They also actively discussed with us the potential consequences and possible defenses, and suggested that we report the problem to CERT/CC for coordinated disclosure.

Baidu: was interested in the attacks and had an in-depth discussions with us about the specifics. In particular, they stated that they have seen a few real-world cases of forwarding-loop attacks, which led them to add a self-defined loop detection header.[3] However, they did not foresee that interactions among CDNs could re-enable this attack.

Alibaba: discussed with us about the details of the attacks and their potential consequences. They chose monitoring and rate-limiting to mitigate the problem.

Tencent: evaluated the problem as a high-risk vulnerability. They stated that they view it as indeed a problem for the CDN industry, and they would internally assess how to defend against it. They thanked us for our report and provided a reward of $\approx$ \$300.

Fastly: acknowledged and discussed our report with us. They stated that both no-abort-forwarding and HTTP Streaming provide desirable performance properties, allowing them to optimize customer traffic. To defend against inter-CDN loops, they suggest that a unified, standard loop-detection header holds the most promise, and are evaluating how to best contribute to such an effort. In the mean-time, they are also evaluating how to improve their existing loop-detection mechanisms, given the knowledge of other CDN practices. They thanked us and offered several T-shirts as a token of gratitude.

CDN77: thanked us for our report and informed us that they will change their system to not reset Via. They also said that no-abort-forwarding is an important performance feature for their CDN, so they are inclined to keep it. To defend against forwarding loops, they are considering implementing a constraint on forwarding destinations to mitigate intra-CDN loops. They are also willing to cooperate with other CDN providers to define a unified loop-detection header for mitigating inter-CDN loops.

Akamai, Azure (China) and CloudFront: acknowledged our report, but provided no further comment to date.

Verizon (EdgeCast): stated that this problem is valid and can be a great danger to CDNs and the Internet in general. They are also interested in working with other CDNs to define a unified loop-detection header.

IV. POSSIBLE DEFENSES AND MITIGATIONS

Unifying and standardizing loop-detection header. As we have presented, forwarding-loop attacks within one CDN

[2] http://hackerone.com/cert

[3] This happened before Baidu's partnering with CloudFlare.

can be completely defeated with loop-detection headers, a simple and clean solution. However, even if all CDNs adopt loop detection headers, the issue of forwarding loops across CDNs will remain if any CDN unintentionally provide ways for attackers to strip the loop-detection headers of other CDNs.

We therefore suggest that CDNs should agree upon a unified loop-detection header, and prohibit disruptive operations on it. A possible candidate would be the Via header, which the current standard already requires nodes to add when forwarding/proxying HTTP requests [8]. The standard also states that proxies "SHOULD NOT" tamper with entries in the Via header set by different organizations.

A number of the CDN vendors with whom we discussed the attacks view this approach as the most desirable solution, and agreed that all CDNs should comply with the standard and not disrupt the Via header. CloudFlare is implementing a loop-detection mechanism using Via.

While this approach is conceptually simple, it needs considerable coordination efforts to be implemented and enforced. It also requires ongoing compliance testing to ensure prompt detection of gaps in deployment. In that light, CDNs should also consider immediately adoptable mitigations, as follows.

Obfuscating self-defined loop-detection headers. A lightweight mitigation is to implement a self-defined loop detection header in a way that resists stripping by "bad actors" (attackers setting up particular forwarding paths or rules). One approach would be to obfuscate the header by generating its name via encrypting a mix of a certain keyword and a random nonce, which is verifiable by decrypting and validating the presence of the keyword.

Such headers will resist stripping by regular-expression like WAF rules because the attacker will not know how to specify the header's name. We have implemented this mitigation based on Nginx 1.8.0. However, it will not help if a CDN provides whitelist-based WAF rules (only propagate headers that match a specified set).

Monitoring and rate-limiting. Another mitigation CDNs could implement is some form of rate-limiting. For example, a CDN could monitor traffic volume or concurrent connections per source IP address or per customer, rejecting or downgrading subsequent requests from the same source/customer once their activities exceed pre-defined threshold. In particular, a gracefully downgrading approach that differentiates requests forming forwarding loops and those of legitimate clients is to respond to potentially problematic requests with a 302 informing the initiator to try again later. While a normal client will usually follow the redirection automatically, measurements of our implementation confirm that this approach suffices to terminate forwarding loops because all CDNs we tested merely relay the 302 response back, rather than following the redirection.

CloudFlare informed us that they have implemented a limit on concurrent connections per source IP address, and a performance downgrade similar to the returning-with-302 strategy once the source exceeds the threshold. However, they expressed concerns with the "greylisting" vulnerability that this strategy introduces: attackers triggering the threshold on IP addresses of one CDN to affect other customers chaining that CDN to CloudFlare. In general, a more fine-grained policy such as per-account rate-limiting could avoid this problem.

However, it is worth noting that any form of rate-limiting can be evaded by sufficient planning by attackers. In the extreme case, a forwarding-loop attack could be launched so that attacking traffic comes from different IP addresses and attributed to different (bogus) customer accounts. Also, the returning-with-302 strategy will not work if the major attacking traffic comes from responses using the dam-flooding attack. Nevertheless, monitoring and rate-limiting could substantially raise the operational overhead of forwarding-loop attacks.

Constraint on forwarding destination. Another possible mitigation is to enforce a blacklist-like policy on forwarding destinations. For example, a CDN can reject a request if its forwarding destination belongs to another CDN. Such constraints could also be implemented with finer-grained conditions. In CloudFlare's response to us, they mentioned not accepting a request if it comes from a CDN and goes to another. CDN77 also expressed interest in implementing blacklist-based mitigations. The downside of this approach is that it requires considerable efforts to maintain an accurate list of CDN IP addresses. It also discourages benign customers from chaining multiple CDNs, which has real-world utility [4].

V. Related work

CDN loop attacks and their prevention. The only material we know of that studied the problem of forwarding loops in CDNs is a blog post from the OpenCDN team [17]. They mention approaches for constructing loops in CDNs that lack loop-detection capabilities. Our work contributes further in this regard in that we broadly explore the possibilities of such attacks, and expand their scope via self-loops, evading loop detection of one CDN by abusing features of other CDNs, construction of the dam flooding attack, and comprehensive measurement of how forwarding-loop attacks could work in real world.

Some publications discuss detecting internal forwarding loops inside a single CDN. Yao proposed a "Hop Counter" HTTP header to detect forwarding loops [23]. CoralCDN prevents internal loops by checking the "User-Agent" header [9]. However, these approaches do not consider that the undesired interactions among CDNs can provide opportunities to evade such defenses.

The Content Distribution Network Interconnection (CDNI) working group of the IETF works on standardizing how multiple CDNs can cooperate with each other [11], [18]. They have considered addressing potential loops in the request-routing process that determines the appropriate edge server using HTTP redirection or DNS CNAMEs among multiple CDNs [3]. However, they have yet to consider the problem of forwarding loops, which could occur when the edge server forwards the request to the original website. Our suggestion of unifying and standardizing on an HTTP header for forwarding-loop detection appears to fit within their scope.

Other CDN security issues. Prior work has examined other types of attacks, and associated defenses, relevant to CDNs. Triukose et al. proposed an attack that abuses the no-abort-forwarding of Akamai and Limelight to launch DoS

attacks on their customers [22]. This behavior is also related to the effects of forwarding-loop attacks, and our measurements show that Akamai, among other CDNs, still uses no-abort-forwarding, which is vulnerable to Triukose et al.'s attack, and makes forwarding-loop attacks more effective, although Fastly and CDN77 explained that this is intended for performance consideration. Su et al. discussed several Akamai implementation considerations that attackers could exploit to degrade streaming services [21]. Lesniewski-Laas et al. proposed a solution called "SSL splitting" to protect the integrity of data served by untrusted proxies [10]. Michalakis et al. also studied the problem of content integrity in untrusted peer-to-peer CDNs, and developed a system to ensure such integrity [15]. Levy et al. presented a system called "Stickler" to help website publishers to guarantee the integrity of web content served to end users through CDNs [12]. Liang et al. investigated the authentication problem of deploying HTTPS in CDNs [13].

VI. Conclusion

We have presented how malicious customers can launch forwarding-loop attacks against CDNs, along with a comprehensive study of their practicality in the real world. The key issue is that features of one CDN may have unintentional and undesired interactions that can disrupt another CDN's internal loop-prevention mechanisms. We believe that forwarding-loop attacks could pose severe threats to CDNs' availability, and hope that our work will provide insight into those issues and help CDNs fully understand them. In the short term, we suggest that CDNs adopt one or more of the mitigation mechanisms discussed in the paper. In the longer term, we hope our work will motivate CDN vendors to address the root cause of the problem, and possibly other potential problems caused by the lack of coordination among CDNs.

Finally, at a higher level our work underscores the hazards that can arise when a networked system provides users with control over forwarding—particularly in a context that lacks a single point of administrative control, and thus allows forwarding manipulation by leveraging inconsistencies among policies and technical mechanisms used by different networking providers.

VII. Acknowledgments

We especially thank Jie Ma, Jinghui Feng, Tingting Li and Haoting from Baidu's CDN team for valuable discussions and authorization to test on their CDN platform. We also gratefully thank Nick Sullivan from CloudFlare, Daniel McCarney from Fastly, Tomas Kvasnicka from CDN77, Amir Khakpour from Verizon (EdgeCast), and Hanqing Wu from Alibaba for their helpful comments. We also thank the anonymous reviewers, and Zhou Li, Jianwei Zhuge, Kun Yang, Kun Du, Huiming Liu, Wei Liu, and Qin Chen for suggestions and feedback. This work was funded by Tsinghua National Laboratory for Information Science and Technology (TNList) Academic Exchange Foundation, National Natural Science Foundation of China (grant #: 61472215) and was also partially supported by the US National Science Foundation under grant CNS-1237265, and by generous support from Google and IBM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the funding agencies.

References

[1] Akamai, "Facts & Figures ," http://www.akamai.com/html/about/facts_figures.htm, 2015, [Accessed Aug. 2015].

[2] A. Barbir, B. Cain, R. Nair, and O. Spatscheck, "Known Content Network (CN) Request-Routing Mechanisms," *IETF RFC 3568*, 2003.

[3] T. Choi, Y. Seo, D. Kim, J. Lee, J. Koo, J. Shinn, and K. Park, "CDNi Request Routing Redirection with Loop Prevention," http://tools.ietf.org/html/draft-choi-cdni-req-routing-redir-loop-prevention-01, 2013, [Accessed Aug. 2015].

[4] CloudFlare, "Content Delivery Network: We've built the next-generation CDN," https://www.cloudflare.com/features-cdn, [Accessed Aug. 2015].

[5] Fastly, "Adding or modifying headers on HTTP requests and responses," https://docs.fastly.com/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses, [Accessed Aug. 2015].

[6] R. Fielding, M. Nottingham, and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching," *IETF RFC 7234*, 2014.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer ProtocolHTTP/1.1," *IETF RFC 2616*, 1999.

[8] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," *IETF RFC 7230*, 2014.

[9] M. J. Freedman, "Experiences with CoralCDN: a five-year operational view," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2010.

[10] C. Lesniewski-Laas and M. F. Kaashoek, "SSL Splitting: Securely Serving Data from Untrusted Caches," *Computer Networks*, vol. 48, no. 5, pp. 763–779, 2005.

[11] K. Leung and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements," *IETF RFC 7337*, 2014.

[12] A. Levy, H. Corrigan-Gibbs, and D. Boneh, "Stickler: Defending Against Malicious CDNs in an Unmodified Browser," in *WEB 2.0 SECURITY & PRIVACY*. IEEE, 2015.

[13] 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 *Proceedings of the 35th IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, May 2014.

[14] MaxCDN, "EdgeRules Features," https://www.maxcdn.com/one/tutorial/edgerules-features/, [Accessed Aug. 2015].

[15] N. Michalakis, R. Soulé, and R. Grimm, "Ensuring Content Integrity for Untrusted Peer-to-Peer Content Distribution Networks," in *Proceedings of the 4th USENIX conference on Networked systems design & implementation (NSDI)*. USENIX Association, 2007, pp. 11–11.

[16] M. Nottingham, "Caching POST," https://www.mnot.net/blog/2012/09/24/caching_POST, 2012, [Accessed Aug. 2015].

[17] OpenCDN, "The Idea of Traffic Amplification Attacks," http://drops.wooyun.org/papers/679, 2013, [Accessed Aug. 2015].

[18] L. Peterson, B. Davie, and R. van Brandenburg, "Framework for Content Distribution Network Interconnection (CDNI)," *IETF RFC 7336*, 2014.

[19] J. Roberts, "How does CloudFlare Handle HTTP Request Headers? ," https://support.cloudflare.com/hc/en-us/articles/200170986-How-does-CloudFlare-handle-HTTP-Request-headers, 2015, [Accessed Aug. 2015].

[20] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind akamai (travelocity-based detouring)," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 435–446, Aug. 2006. [Online]. Available: http://doi.acm.org/10.1145/1151659.1159962

[21] A.-J. Su and A. Kuzmanovic, "Thinning Akamai," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement (IMC)*. ACM, 2008, pp. 29–42.

[22] S. Triukose, Z. Al-Qudah, and M. Rabinovich, "Content Delivery Networks: Protection or Threat?" in *Computer Security–ESORICS 2009*. Springer, 2009, pp. 371–389.

[23] Y. Xi, "Method and Device for Defending CDN Flow Amplification Attacks," https://www.google.com/patents/CN103685253A?cl=en, 2013, [Accessed Aug. 2015].