A SURVEY ON CDN VULNERABILITY TO DOS ATTACKS

Maurizio D'Arienzo and Serena Gracco

Dipartimento di Scienze Politiche Universit`a della Campania "L.Vanvitelli" - Italy

ABSTRACT

Content Delivery Networks (CDN), or "content distribution networks" have been introduced to improve performance, scalability, and security of data distributed through the web. To reduce the response time of a web page when certain content is requested, the CDN redirects requests from users' browsers to geographically distributed surrogate nodes, thus having a positive impact on the response time and network load.

As a side effect, the surrogate servers manage possible attacks, especially denial of service attacks, by distributing the considerable amount of traffic generated by malicious activities among different data centers. Some CDNs provide additional services to normalize traffic and filter intrusion attacks, thus further mitigating the effects of possible unpleasant scenarios.

Despite the presence of these native protective mechanisms, a malicious user can undermine the stability of a CDN by generating a disproportionate amount of traffic within a CDN thanks to endless cycles of requests circulating between nodes of the same network or between several distinct networks. We refer in particular to Forwarding Loops Attacks, a collection of techniques that can alter the regular forwarding process inside CDNs. In this paper, we analyze the vulnerability of some commercial CDNs to this type of attacks and then propose some possible useful defensive strategies.

1. INTRODUCTION

The content delivery network is a quite young technology. Some of the first experiments with this approach were carried out in the mid-1990s by the NEXUS International Broad- casting Association, an IT business incubator based in Milan, using it for the streaming distribution of audio-video documents, as well as web services and ftp archives for NEXUS members. The first commercial version of the Content Delivery Network was launched in 1995 by NEXUS itself and its technological and financial potential was immediately un- derstood.

In the following years, the system was also exported to the United States and other Western countries for local usage, but it was in the global Internet that it found more inter- esting applications and greater margins for development. In 1998 a former Massachusetts Institute of Technology student founded Akamai, a leading global provider of CDN ser- vices. Akamai, which provides its performance to giants such as Adobe, Audi, Apple, IBM, Nintendo, IKEA, Reuters, and many others, make a mirror copy of the content of the client company's servers and saves several copies on different servers in its network. Although the URL remains the same, the user is automatically redirected to one of the Akamai servers, selected according to the type of file requested and the geographical location of the user. All this happens in a transparent manner, that is, without the user noticing it and, above all, without the quality of the services they want to use being affected.

The effective expansion of CDNs across the Internet, where a rising number of websites are being put behind CDNs, has been attributed to CDN providers' promotion of its capacity to defend against DoS attacks. It is generally accepted that CDN suppliers offer good DoS protection for the CDN-powered websites since the CDN absorbs distributed assault traffic with its enormous bandwidth capacity [1] [2]. However, CDNs are exposed to a particular type of attack where endless cycles of requests traveling between nodes of the same network or across multiple different networks can jeopardize the stability of a CDN

by creating an excessive quantity of traffic within it. Such attacks, namely Forwarding Loop Attacks [3], handle a single request repeatedly or endlessly, consuming unnecessary resources and increasing the risk of DoS attacks. In this paper we survey the effects of four type of attacks, the self loop attack, the intra-CDN loop attack, the inter-CDN loop attack, and the dam flooding, a highly damaging type of loop attack. Novel type of attacks, aiming either at starving the bandwidth or the connections of the origin server [4], as well as other type of DoS attacks are recently arising as a possible threat for CDNs [5] [6] [7], and are out of the scope of this survey.

2. ARCHITECTURE AND OPERATION OF A CDN

Content Delivery Network means a network of computers and servers, connected to each other via the Internet and used to distribute large files and data, such as movies, and live TV.

CDN is a network within the network: dozens of servers (sometimes even hundreds or thousands) spread over five continents, hosting the same content (usually movies and other large multimedia files), making it available to computers all over the world. The aim is to optimize the content delivery process to the various nodes of the Network that request it, ensuring that any peaks in requests do not have a negative effect on server performance[8][9].

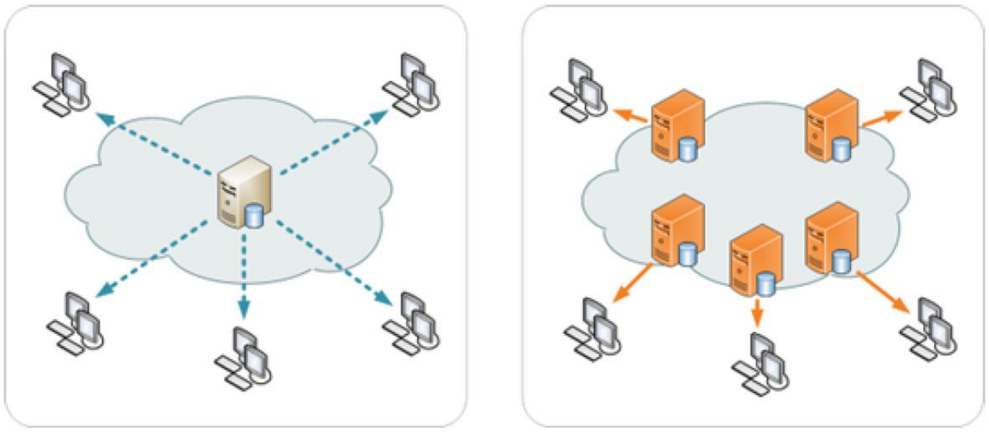


Fig. 1. Load distribution between Edge Servers in a CDN

Especially when dealing with media files, such as live streaming of TV events or movies, centralized systems based on a single distribution server can suffer significant performance degradation. This inevitably has negative effects on the audio and video quality of the distributed content: thanks to CDN it is possible to overcome this problem by diverting traffic to other servers, a perfect copy of the original, while maintaining the performance of the Network and the quality of the distributed content[10].

The four parts that generally make up a CDN are:

- Content Delivery: source web server plus CDN's set of Edge Servers;
- Request Routing: routes content requests to Edge Servers;
- Distribution: replicates content from Origin to Edge Server, and manages content con- sistency (expiration, on-demand/periodic update, update propagation);
- Accounting: produces logs, statistics, and analysis for the site administrator.

The heart of a CDN, and also the part a webmaster has to deal with, is the Request Routing. In particular, this part is of fundamental importance because it also establishes how the content is replicated, and therefore also sent to users. The main Request Routing devices (including Edge Server), are nothing more than authoritative Name Servers, with functions to control and monitor network traffic. By definition, a name server is absolute for a host if it always has a DNS record that translates the hostname of the host to the IP address of that host. When an absolute name server receives a request from a root name sever, it gives a DNS response that contains the requested correlation. The root server then sends the correlation to the local name server, which in turn runs it to the requesting host. Many name servers act as both local and absolute name servers [11]. So, the Request Routing handles what content to replicate, then how to send it to users[12].

3. VULNERABILITY OF A CDN

In this section, we present the key mechanisms that can be used to undermine a CDN by using four different types of attacks that exploit the flexibility of CDNs in terms of configuration of forwarding requests, and can compromise the stability of the entire system [13][14][15].

3.1. Collecting IP Addresses of Edge Servers

The identification of the edge servers of a CDN is based on the resolution of the hostnames from URLs served by the CDN using for example public platforms like PlanetLab or DipZoom, which allows to process a large amount of data in a short time [16]. To this purpose, if we consider the Coral CDN, a free peer-to-peer content distribution network running on PlanetLab, we can draw up a list of URLs served by it; then we randomly select a URL and resolve its hostname in its IP address from each of the DipZoom measurement points located all over the world. You can repeat this process for several hours to discover all 263 unique IP addresses of Coral's servers. A similar but much slower procedure exploits the nslookup command to find the IP address of an edge server of the CDN assigned to our device as shown in the following figure.

Once the IP address of the edge server is retrieved, we describe the next step.

3.2. Overwriting the Selection of Edge Servers

In order to recruit a large number of edge servers for the attack, it is necessary to send HTTP requests to them from a single device in order to override the selection of the server implemented by the CDN for that host. In other words, it is required to bypass the default DNS resolution by connecting to the desired edge server using its IP address, rather than the DNS hostname obtained from the URL [19].

A simple HTTP command line download tool can be used to send a request using the correct DNS hostname. For example, the following command will successfully download content from a selected CloudFlare Edge server with IP address 104.31.66.228 by providing the expected host header as the ”-H” command argument.

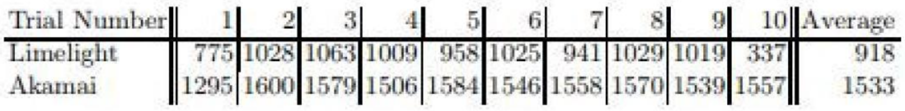
3.3. CDN Caching

The key component of this type of attack is to make sure that the attacker's HTTP request is answered by the origin server rather than the edge servers' cache memory. Normally, requiring an edge server to obtain a given object from the origin server can be done using the HTTP Cache-Control header [17]. Alternatively, in situations where some CDNs, i.e. Akamai, are protected against this approach, so the following observation can be made. On one hand, modern caches use the entire URL including search strings (the optional part of the URL after "?") as cache keys1. For example, a request for foo.jpg?randomstring will be forwarded to the origin server because it is unlikely that the cache already contains an object with this URL. On the other hand, origin servers ignore unexpected search strings contained in other valid URLs. The above request will then provide the "foo.jpg" image directly from the origin server rather than the cache.

To verify the effectiveness of this technique, we ensure that we can download a valid object via the CDN even if we add a random search string to its URL, e.g.,

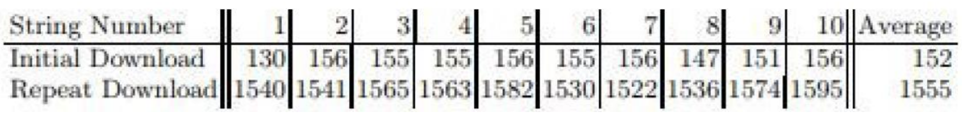
ak.buy.com/db/_assets/large/_images/093/207502093.jpg?random.

Both Akamai, Coral, and Limelight are vulnerable to such a strategy. Then you measure the throughput1 for downloading an object in the cache memory of an edge server. To this purpose, we first send a request to an edge server with a regular URL, without search strings, and measure its download throughput. We repeat the same procedure using the same URL to the same Edge server, this time measuring the download throughput of the object present in the cache of the edge server. Since the first request will cache the content on the edge server, the performance of subsequent downloads will represent the actual performance of the content delivery from the cache. Then, to verify that requests with search strings are processed by the origin server, we compare the performance of the first download of a URL with a given random search string, with repeated downloads from the same edge server using the same search string and downloading the same object from the cache. Presumably, repeated downloads of the same content will be performed by the edge server. Eventually, the modification of the search string results in significantly lower performance when downloading, while repeated downloads show throughput similar to that for cached objects. This will indicate that the initial request with the random search string will always be processed by the Origin server.



Trial Number	1	2	3	4	5	6	7	8	9	10	Average
Limelight	775	1028	1063	1009	958	1025	941	1029	1019	337	918
Akamai	1295	1600	1579	1506	1584	1546	1558	1570	1539	1557	1533

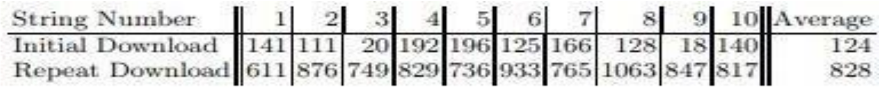
Fig. 2. Throughput for downloading objects in cache memory (Kb/s) without search strings



String Number	1	2	3	4	5	6	7	8	9	10	Average
Initial Download	130	156	155	155	156	155	156	147	151	156	152
Repeat Download	1540	1541	1565	1563	1582	1530	1522	1536	1574	1595	1555

Fig. 3. Throughput for first and subsequent downloads from Akamai (Kb/s) with search strings added to URLs

Fig. 2 shows the throughput of ten repeated downloads of a selected object, with respect to Akamai and Limelight. We need these results to provide an indication of the performance of content downloads already in cache memory. Fig. 3 and 4 show the throughput for initial and repeated downloads of the same object using ten different search strings. There



String Number	1	2	3	4	5	6	7	8	9	10	Average
Initial Download	141	111	20	192	196	125	166	128	18	140	124
Repeat Download	611	876	749	829	736	933	765	1063	847	817	828

Fig. 4. Throughput for first and subsequent downloads from Limelight (Kb/s) with search strings added to URLs.

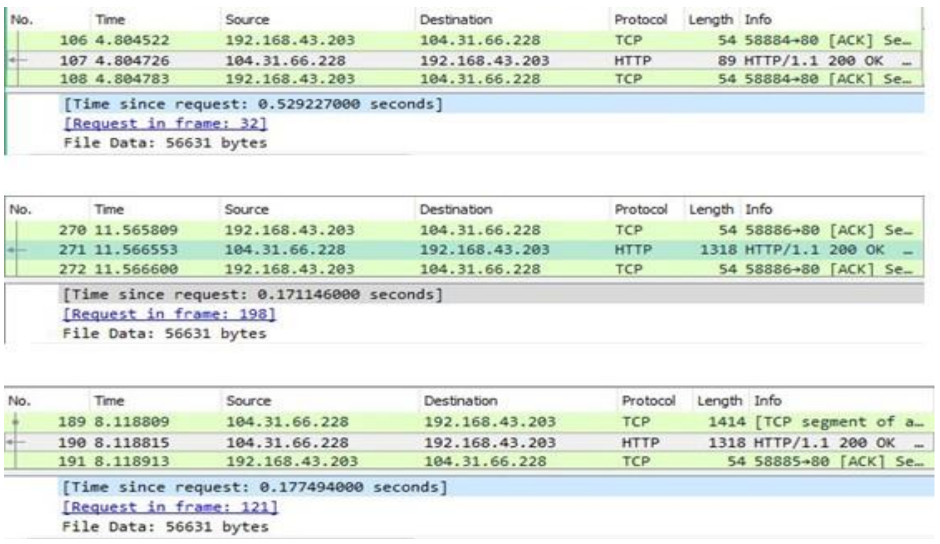
Is therefore a clear difference between the download speed between the first request and the subsequent ones related to the same content. The subsequent requests seem in fact to be 10 times faster for Akamai, and about 7 times faster for Limelight. Furthermore, in any case, any download with an always different search string achieves the performance of repeated downloads for the same object. At the same time, the performance for repeated downloads with random search strings seems to be similar to that obtained for cached content. All these statements lead us to the conclusion that repeated downloads with a random search string are served by the cache while appending a new search string to the URL overrides the caching system of the edge servers forcing them to retrieve the requested content directly from the origin server.

In the case of the CloudFlare CDN, we checked its handling request policy with random search strings on a website created for the test, We obtained the IP address of the edge server chosen by CloudFlare for our client by simply resolving the hostname cachingtest- website.altervista.org.To verify that CloudFlare puts the item in the cache memory, we requested the above image a few times without search strings and checked from the log that the item was requested only once. We then checked the difference in terms of through- put and download speed. It will provide us with information on the source of the content requested. Subsequently, different searches were executed with different search strings each time. This time, the website log will contain each access to content for each of the requests submitted. We conclude that placing an always different search string to the URL of a given content pushes the CloudFlare server to provide the object from the Origin server rather than from the cache, regardless of its content, in the same way of Akamai and Limelight.

4. AMPLIFYING THE ATTACK: DECOUPLED FILE TRANSFERS

We described how it is possible to manipulate an edge server so that it downloads a certain object from the origin server regardless of its cache content, as well as how to bypass the DoS defense mechanism of a CDN. Further methods are described in [2] [18]. We now present the methods to ”hire” an edge server to consume bandwidth resources from the source site without reducing the attacker’s bandwidth. We highlight that an edge server downloads files from the origin server and provides them to the client-server using two independent TCP connections, making the file transfer speeds along these two connections roughly unrelated. This comes from the natural implementation of an edge server and the desire to have a file available in the cache memory as soon as possible to quickly serve future requests. Unfortunately, this mechanism has serious implications for CDN security.

To verify the actual independence of the TCP connections, we arranged a simple testbed depicted in Fig. 7 Two clients act as a ”probe” and a ”monitor”. The former can model its bandwidth or close its connection right after sending the HTTP request. The latter works with the standard TCP/IP network stack.



```
No.   Time          Source           Destination      Protocol  Length Info
  106 4.804522      192.168.43.203   104.31.66.228    TCP           54 58884→80 [ACK] Se…
  107 4.804726      104.31.66.228    192.168.43.203   HTTP          89 HTTP/1.1 200 OK  …
  108 4.804783      192.168.43.203   104.31.66.228    TCP           54 58884→80 [ACK] Se…
        [Time since request: 0.529227000 seconds]
        [Request in frame: 32]
        File Data: 56631 bytes
```

```
No.   Time          Source           Destination      Protocol  Length Info
  270 11.565809     192.168.43.203   104.31.66.228    TCP           54 58886→80 [ACK] Se…
  271 11.566553     104.31.66.228    192.168.43.203   HTTP        1318 HTTP/1.1 200 OK  …
  272 11.566600     192.168.43.203   104.31.66.228    TCP           54 58886→80 [ACK] Se…
        [Time since request: 0.171146000 seconds]
        [Request in frame: 198]
        File Data: 56631 bytes
```

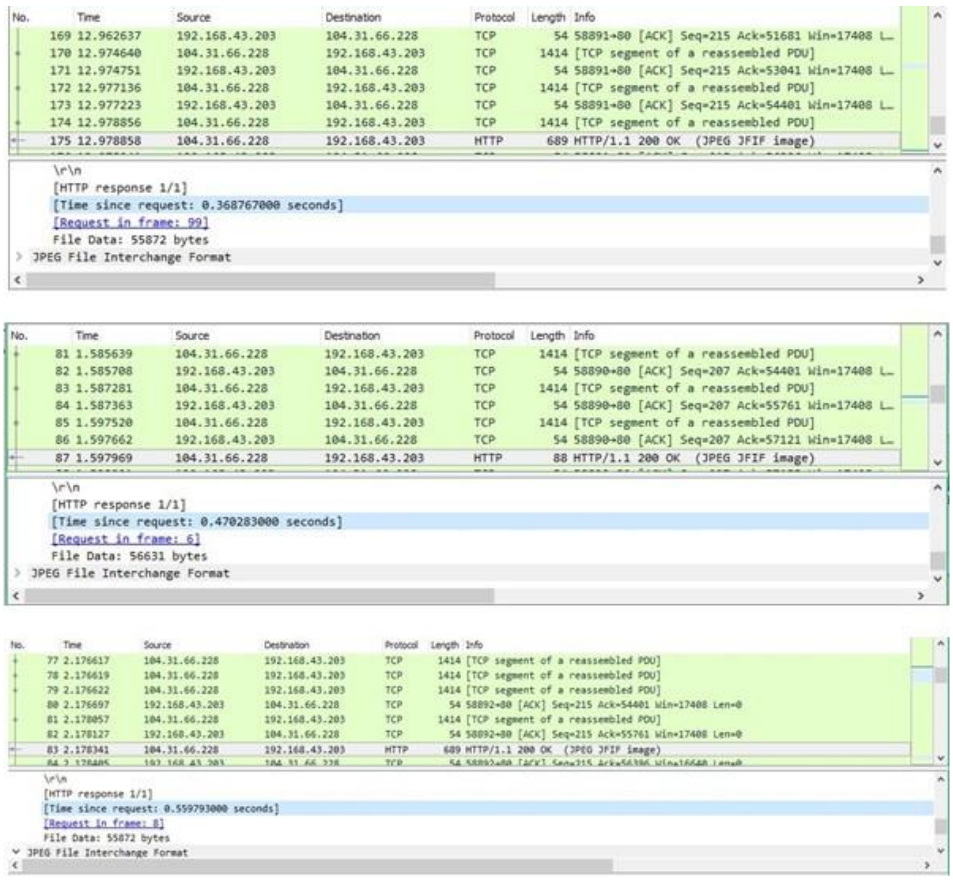
```
No.   Time          Source           Destination      Protocol  Length Info
  189 8.118809      104.31.66.228    192.168.43.203   TCP         1414 [TCP segment of a…
  190 8.118815      104.31.66.228    192.168.43.203   HTTP        1318 HTTP/1.1 200 OK  …
  191 8.118913      192.168.43.203   104.31.66.228    TCP           54 58885→80 [ACK] Se…
        [Time since request: 0.177494000 seconds]
        [Request in frame: 121]
        File Data: 56631 bytes
```

Fig. 5. First and subsequent HTTP requests for the object at node 104.31.66.228, served by the origin server.

The probe requests a CDN object from the Edge server ”E” using a random search string to make sure it is downloaded from the origin server rather than the cache. The probe then adapts its band so that it is very low or even cuts the connection completely after sending the HTTP request. While the client is therefore making slow (if any) progress in downloading the file, the monitor sends a request to the same URL with the same random search string used in the previous step to E and measures the download throughput. If this is consistent with the results obtained in the experiments described in the previous paragraph, this means that the edge server will have processed the monitor client request from its cache memory. Therefore, the edge server must have completed the file transfer from the origin server after the probe client accessed it, even if it has almost downloaded it. On the other hand, if the measured throughput is comparable to the initial download of content seen in section 3.3, this indicates that the edge server has not yet downloaded the requested file and is acquiring it from the origin server. Therefore, this second case will indicate that the edge server somehow approximates its download speed from the source server to its upload speed to the applicant.

As the edge servers may react differently to different client behaviors, we plan these experiments with the probe client: a) Reducing his connection; b) Silencing it, i.e. not send- ing any ACK following the HTTP request; c) Completely cut the connection by sending the TCP reset segment to the edge server as a response to its first data segment.

Since Akamai, Limelight, and Coral seem not to change their file download behavior in response to any of the three modes mentioned above, we will illustrate a more aggressive technique that sets the TCP input buffer to 256 bytes, so that the edge server will only send a small initial portion of data (this cuts the payload in the first data segment from 1470 bytes to 256 bytes), and then interrupts the TCP connection upon the transmission of the HTTP request (so that the edge server does not try to retransmit the first data



```
No.    Time          Source            Destination       Protocol  Length  Info
   169 12.962637     192.168.43.203    104.31.66.228     TCP         54 58891→80 [ACK] Seq=215 Ack=51681 Win=17408 L…
   170 12.974640     104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
   171 12.974751     192.168.43.203    104.31.66.228     TCP         54 58891→80 [ACK] Seq=215 Ack=53041 Win=17408 L…
   172 12.977136     104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
   173 12.977223     192.168.43.203    104.31.66.228     TCP         54 58891→80 [ACK] Seq=215 Ack=54401 Win=17408 L…
   174 12.978856     104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
   175 12.978858     104.31.66.228     192.168.43.203    HTTP       689 HTTP/1.1 200 OK  (JPEG JFIF image)

      \r\n
      [HTTP response 1/1]
      [Time since request: 0.368767000 seconds]
      [Request in frame: 99]
      File Data: 55872 bytes
  > JPEG File Interchange Format
```

```
No.    Time          Source            Destination       Protocol  Length  Info
    81 1.585639      104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
    82 1.585708      192.168.43.203    104.31.66.228     TCP         54 58890→80 [ACK] Seq=207 Ack=54401 Win=17408 L…
    83 1.587281      104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
    84 1.587363      192.168.43.203    104.31.66.228     TCP         54 58890→80 [ACK] Seq=207 Ack=55761 Win=17408 L…
    85 1.597520      104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
    86 1.597662      192.168.43.203    104.31.66.228     TCP         54 58890→80 [ACK] Seq=207 Ack=57121 Win=17408 L…
    87 1.597969      104.31.66.228     192.168.43.203    HTTP        88 HTTP/1.1 200 OK  (JPEG JFIF image)

      \r\n
      [HTTP response 1/1]
      [Time since request: 0.470283000 seconds]
      [Request in frame: 6]
      File Data: 56631 bytes
  > JPEG File Interchange Format
```

```
No.    Time          Source            Destination       Protocol  Length  Info
    77 2.176617      104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
    78 2.176619      104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
    79 2.176622      104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
    80 2.176697      192.168.43.203    104.31.66.228     TCP         54 58892→80 [ACK] Seq=215 Ack=54401 Win=17408 Len=0
    81 2.178057      104.31.66.228     192.168.43.203    TCP       1414 [TCP segment of a reassembled PDU]
    82 2.178127      192.168.43.203    104.31.66.228     TCP         54 58892→80 [ACK] Seq=215 Ack=55761 Win=17408 Len=0
    83 2.178341      104.31.66.228     192.168.43.203    HTTP       689 HTTP/1.1 200 OK  (JPEG JFIF image)

      \r\n
      [HTTP response 1/1]
      [Time since request: 0.559793000 seconds]
      [Request in frame: 8]
      File Data: 55872 bytes
  v JPEG File Interchange Format
```

Fig. 6. HTTP responses to requests with random search strings

segment after the timeout). The above tables show how Limelight and Akamai respectively transfer the required content with a throughput that is between 100 and 200 Kb/s. Since both files used in this example have a size of about 50Kb, we make sure that the request is sent from the monitor client 0.5s after the probe client, so that if our hypothesis is correct, each edge server will have already transferred in cache memory the entire file when the request from the monitor arrives [12].

The table in Fig. 8 shows that the throughput for the monitor is comparable with that for repeated downloads we have indicated in the figures on the previous pages. This means that the monitor got its contents from the edge server cache memory. Since the edge server could provide the file from its cache memory only upon a request from the probe client, which acquired a negligible part of it, we have shown that, with the help of the edge server, the probe can consume (object size)/0.5s, or approximately 100Kb/s, of the origin server's bandwidth while consuming a small portion of its own bandwidth.

5. FORWARDING-LOOP ATTACKS

Malicious CDN clients can deliberately manipulate the HTTP request submission process to create an infinite loop between CDN nodes [19]. As already widely seen, access to the network through a CDN requires two steps: first (Request Routing), a user's request is forwarded to a CDN server geographically close to the user; second, the CDN server obtains the requested content which is then sent to the user. As far as the first step, it is

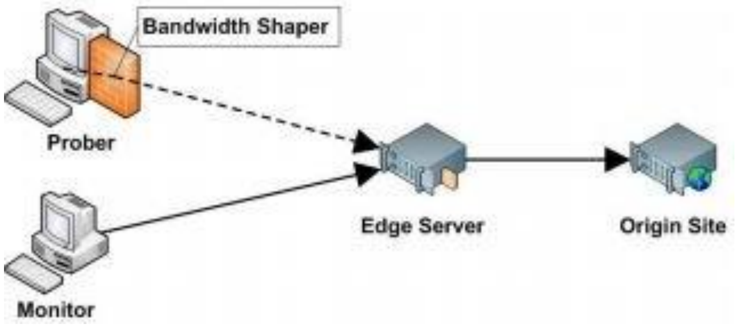


Fig. 7. Probe and Monitor clients

String Number	1	2	3	4	5	6	7	8	9	10	Average
Limelight	1058	1027	721	797	950	759	943	949	935	928	907
Akamai	1564	1543	1560	1531	1562	1589	1591	1600	1583	1544	1567

Fig. 8. Monitor client throughput in download (Kb/s)

possible to override the selection of a node CDN for the client by connecting to a network server through its IP address rather than its hostname. For this purpose, platforms like PlanetLab help retrieve such IP addresses by resolving subdomain names. Concerning the second step, i.e. obtaining the required content from the CDN servers, we remind that there are available both push and pull ways. The pull technique allows website owners to upload their content in advance to the CDN servers. The pull mode is based instead on the caching mechanism operated by CDN servers in advance, without sending further requests to the source server and therefore considerably shortening the response time.

The vulnerabilities examined in this discussion are concerned with the pull mode. In fact, adding the ”no-cache” header to the requests will ensure that the CDN will always re-evaluate the responses from the source server instead of running them out of its cache. A similar result is achievable by using POST requests that allow to write to the source server. In addition, many CDNs provide the ability to configure the network so that some URLs do not go into cache memory.

Forwarding-loops cause CDN nodes to repeatedly and indefinitely process a user’s request, thus generating a considerable amount of traffic. In general, before a node on the CDN forwards an HTTP request received from a client, it checks the Host field in the client for any forwarding destination previously specified by the user. In general, the target server provides a response which is then delivered by the CDN node to the client. However, if the forwarding destination is intentionally changed to point at another CDN node other than the default, the forwarding process may end up in a loop, causing detriment to device performance.

Fig. 9 illustrates this critical mechanism triggered between three distinct nodes that can be placed on the same CDN or distributed among several CDNs [19].

The possibility to generate this type of attack is mainly caused by CDN customers flexible control over the network forwarding configuration, and unfortunately CDNs lack an effective control mechanism to ensure that these configurations - especially if they

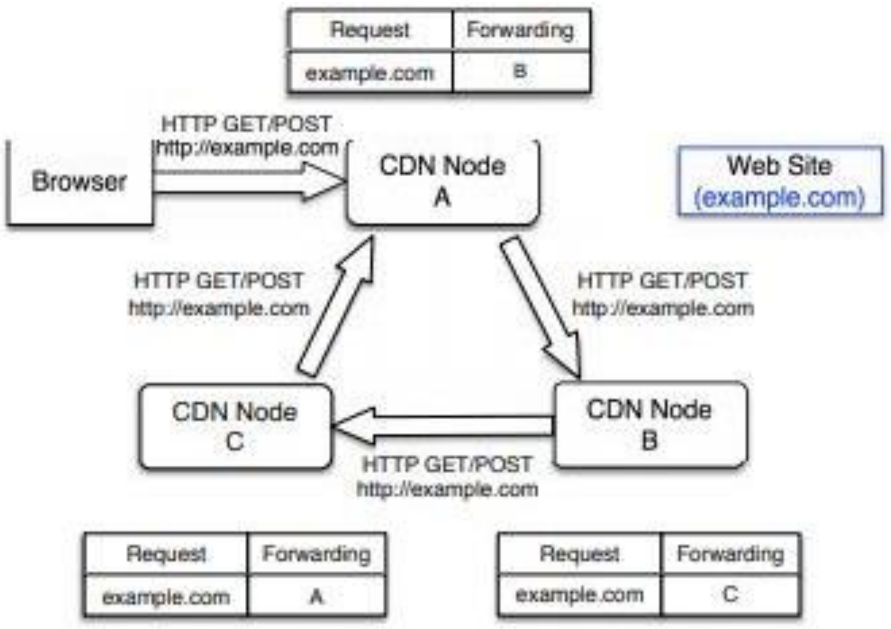


Fig. 9. Scheme of a Forwarding-Loop attack

affect several customers or several CDNs - do not allow to process the requests more than necessary.

There are four different types of forwarding-loops in a CDN:

– Self-loop: where a single node of a CDN is used;
– Intra-CDN loop: where more nodes than one CDN are used;
– Inter-CDN loop: which exploits nodes belonging to different CDNs;
– CDN Dam Flooding: which pairs infinite forwarding attacks with time control of HTTP responses to significantly increase the damage to the affected network.

We consider these types of attacks in the following subsections.

5.1. Self-Loop Attacks

The self-loop attack occurs when requests are routed in a single CDN node. The attack is replicated immediately: it simply needs the loopback address (127.0.0.1) or the IP address of a specific node on the network as the destination address for HTTP requests. Self- loop attacks can be particularly insidious because of the circulation of requests in the network is practically without latency, potentially consuming resources very quickly. Many of the most common CDNs are configurable to accept the loopback address or the IP address as one of the network nodes, except for Baidu and CloudFlare as a destination for requests; CloudFront also does not allow to directly enter an IP address or ”localhost” as a destination. It is important to stress that, to avoid the emergence of such a mechanism be harmful to the network, the implementation of a blacklist of prohibited destination addresses is not a sufficient defensive measure for CDNs that support the use of domain names as a forwarding destination. For example, CloudFlare allows to specify a CNAME domain (which allows to link one DNS name to another) as the forwarding address, thus providing the possibility of modifying the DNS resolution later by entering the loopback address or that of a CloudFlare network node.

It is also possible to test the vulnerability of three open-source reverse proxies that are often used by some commercial CDNs: Squid, Nginx and Varnish. The last two by default are completely vulnerable to this type of threat and do not have any mechanism to preserve their integrity; Squid instead prevents the establishment of forwarding loops by adding the Via header to the submitted requests and rejecting incoming requests that contain the same hostname in their header. This approach is similar to that adopted by CDN77, CDNlion, CDN.net, and CDNsun.

The syntax to prepare a Via header is as follows: Via: [¡ protocol-name¿ ”/”] ¡ protocol-version¿ ¡ host¿ [”:” ¡port¿] or Via: [¡ protocol-name¿ ”/”] ¡ protocol-version¿
¡ pseudonym¿ [14]

Testing the feasibility of a self-loop attack on commercial CDN networks requires special care to avoid service disruption to the network. To this purpose, a request with a slightly increased size than the maximum accepted size can be sent to a node of the CDN, then observe its response (i.e. 400 Bad Request, Request Header Or Cookie Too Large) and then send another self-loop request to the same node, but this time a little smaller (about 200 bytes less) than the maximum accepted size. Proceeding in this way, if the CDN is vulnerable to self-loop attacks, the submitted request will only bounce in the same node a few times before it reaches the maximum size allowed for the header. At this point, it is easy to conclude that if both requests sent are matched by the same response indicating that the maximum size has been exceeded, the CDN analyzed will be vulnerable to attack under analysis. Otherwise, the CDN will prohibit the forwarding of requests to the loopback address or your own address. Only the Azure network (China) was found vulnerable to this type of attack.

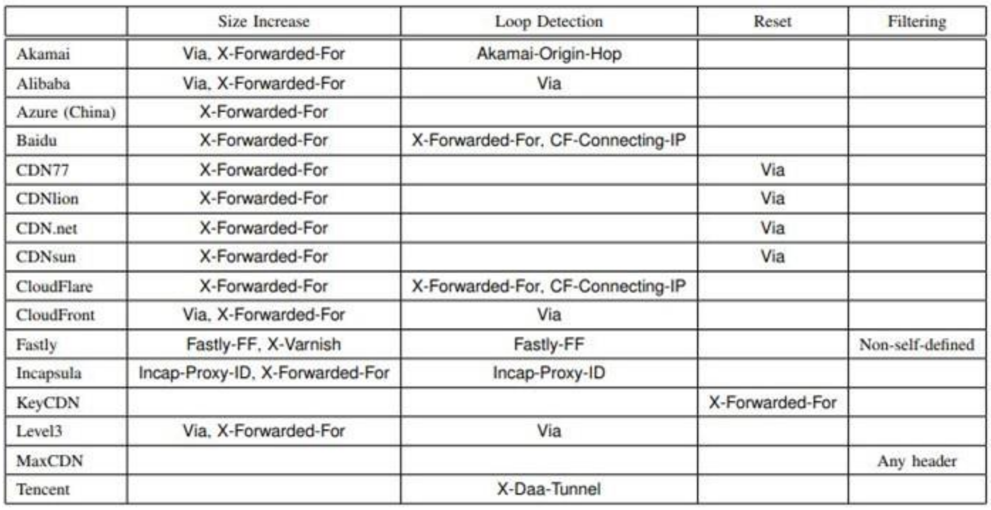
5.2. Intra-CDN Loop

Attacks to a CDN can also be conducted so that loops are created among several nodes on the same CDN. About 15 different CDN platforms allow to enter domain names as destination addresses. When a request is submitted to a domain, 10 out of 15 CDNs under analysis (except Azure China, Baidu, CloudFlare, Fastly, and Tencent) change the Host header to reflect the destination domain. For each of them, attack among several nodes can be generated using several different accounts and concatenating different target domains. For example, it can be set account A1 to forward requests from domain D1 to D2, account A2 to go from D3 to D4 to the An account, which closes the loop by forwarding the Dn domain to D1. This type of attack can be arranged by dynamically changing the forwarding destinations using the DNS service. None of the CDNs can enter domain names as forwarding destination share a global DNS cache. Different CDN nodes will independently resolve the forwarding domain.

A malicious user can then create a loop between two nodes A and B of the same CDN by checking the DNS resolution of their forwarding domains so that queries from A are provided with the IP address of B, and vice versa. Depending on how a CDN handles for DNS resolution the attacker may have to choose A and B from different datacenters or regions. This type of attack does not affect the 9 CDNs that use loop detection headers. A summary of these considerations is reported in Fig. 10

5.3. Inter-CDN loops

When an intra-CDN loop attack is extended to encompass multiple CDNs, it becomes possible to bypass the network protection mechanism based on the use of particular headers



	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. 10. Different approaches to change request header

in loop detection requests. This approach is based on the concatenation of loop-detection header CDNs with other CDNs that inhibit their functionality.

As shown in Fig. 11, Fastly and MaxCDN provide customer-defined header filtering. The headers filtering feature offered by Fastly does not make loop detection more difficult just because it adds a non filterable loop detection header. Inserting a MaxCDN node in a chain makes this type of attack possible because it allows an unlimited filtering of the headers and allows to affect their regular operations.

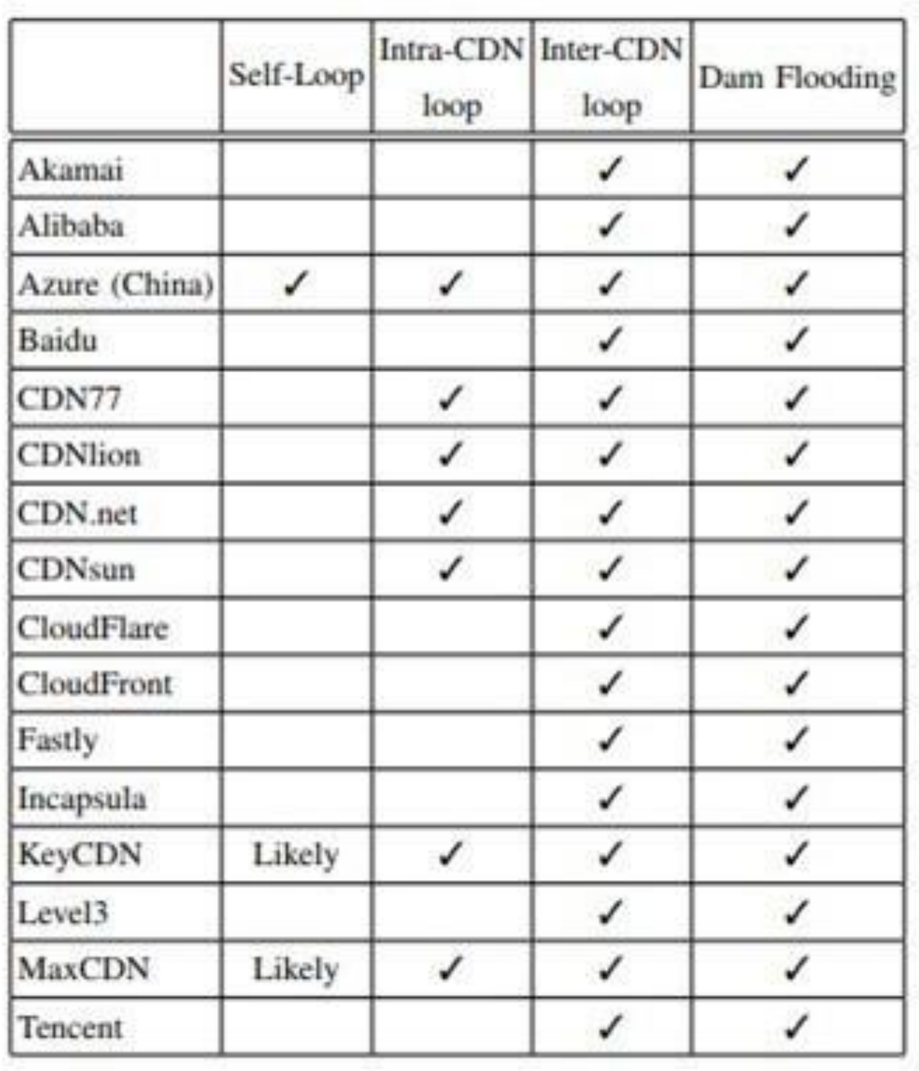
The simple introduction of a MaxCDN node in the configuration of a forwarding loop is sufficient to even cheat those CDNs having defense mechanisms against this type of attack.

Observing Fig. 11, we notice how some CDNs reset the header of the received requests. In particular, CDN77, CDNlion, CDN.net, and CDNsun reset the Via header, discussed in the previous paragraph, used by Alibaba, CloudFront, and Level3 to detect forwarding loops. Therefore, to carry out the attack, it will be sufficient to link a node belonging to one of the first 4 CDNs with one or more nodes of the last 3.

Another consequence of filtering or resetting the headers is the ability to block the header size increase so that the life cycle of a forwarding loop becomes independent of the normal header size limits. For example, we could form a continuous loop including a CloudFront node, one of CDN77, and one of KeyCDN. The CDN77 node, as we said, will reset the Via header used by CloudFront. The KeyCDN node will reset the X-Forwarded- For node, which seems to be the one whose size would constantly increase from CloudFront and CDN77. KeyCDN does not detect loops or increase the size of headers. In addition, since CDN77 does not adopt mechanisms to end forwarding-loops, the timeouts of the requests will not end the loop. This type of attack could last indefinitely.

5.4. CDN Dam Flooding Attack

This type of attack is based on the HTTP streaming additional feature of CDN. HTTP streaming was introduced in HTTPv1.1 protocol and is enabled by reporting the Transfer- Encoding chunked field instead of the Content-Length field inside the header, thus allowing to establish a persistent connection to transmit dynamically generated content on demand, without knowing its size in advance. Focusing on the case of CDNs, a node compatible



	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			✓	✓

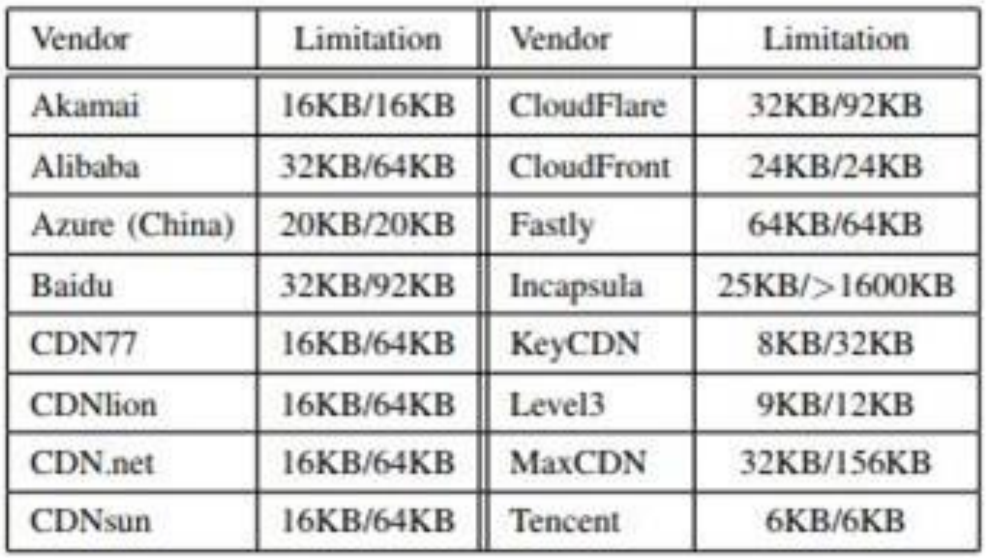
Fig. 11. Vulnerability of several CDNs with respect to forwarding-loop attacks

with this technology will start forwarding requests or responses to the next node on the network immediately after receiving the first ”chunk”, (chunk=piece), rather than waiting for the entire content. This mechanism allows the data to circulate faster and thus make the attack even more effective by completely saturating the connections.

As shown in Figure 2.20, Azure(China) is the only applicable target for streaming loops since it is the only CDN that support streaming requests and does not integrate a loop detection system Fig. 14. Since all the CDNs in our analysis support HTTP streaming for responses, we can extend the attack by using the responses rather than the requests to create a streaming loop.

This type of attack is so called because it is based on the two fundamental phases characterizing the process of filling and flooding a dam.

The attack evolves like the steps reported in diagram Fig. 15: 1) The Attacker Client sends a request to node A; 2) Node A interrogates attacker.com to forward the request, and is then directed to B; 3) The request circulates between nodes B and C and goes back to node A; 4) The attacker points attacker.com to one of its servers; 5) The next query from A for attacker.com is mapped to the attacker’s server; 6) A forwards the request to



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

Fig. 12. Headers size limitation

the attacker’s server; 7) This reacts with a streaming response; 8) The streaming response circulates through C and B, then back to A, repeating the cycle several times; 9) Finally A delivers the request to the Attacker Client.

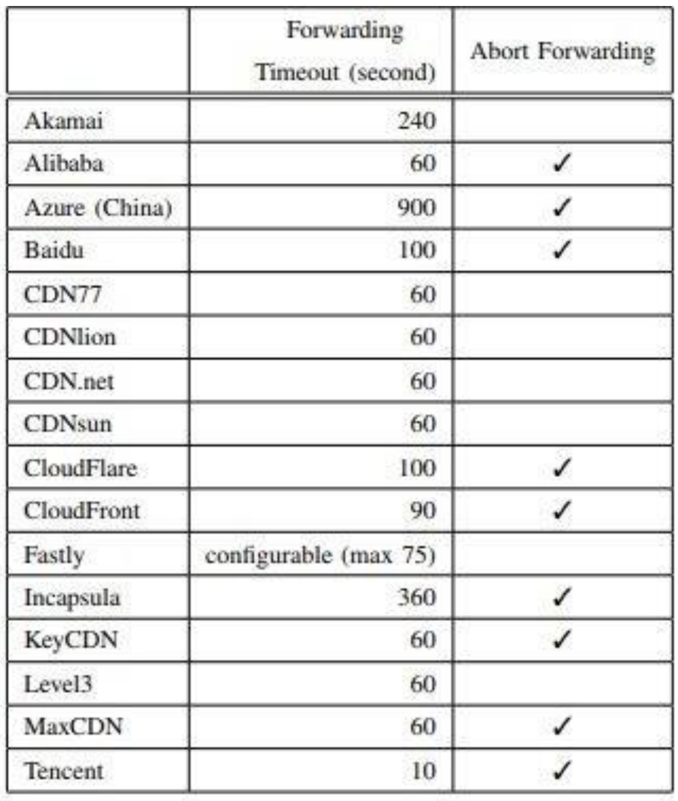
In the first ”filling phase”, several forwarding loops are launched that can be of one of the above types, using domain names as forwarding destinations. In the ”flooding phase”, the resolution of these names is changed to direct the forwarding to a server of the attacker that responds to incoming requests with a large file using HTTP streaming. As a result, for each forwarding-loop, a streaming response flows between the CDN nodes in reverse order, several times, until a connection is broken due to forwarding timeout or inside the client that originated the loop. Similar to a dam, during the first phase a huge amount of data traffic is accumulated so that in the second phase it exploded upon HTTP large chunks of continuous streaming. The effects are easily visible in Fig. 16.

Note that being able to dynamically change forwarding destinations using DNS reso- lution is not a necessary condition for creating streaming loops. In fact, if we follow the example reported in subsection 5.3, rather than chaining the node An with A1, we can alter the cycle so that An points to the server owned by the attacker so that after n jumps among the nodes of the CDN, a request is forwarded to that server that, with a streaming response then sent back into the network. That said, using DNS ensures more triggering control on the execution of the flood phase, thus causing as much damage as possible.

6. A SURVEY ON ADOPTED COUNTERMEASURES

Some of the providers analyzed in our discussion have become aware of the problem that could afflict the CDNs and have introduced countermeasures to prevent attacks. We present the approach adopted by some of them:

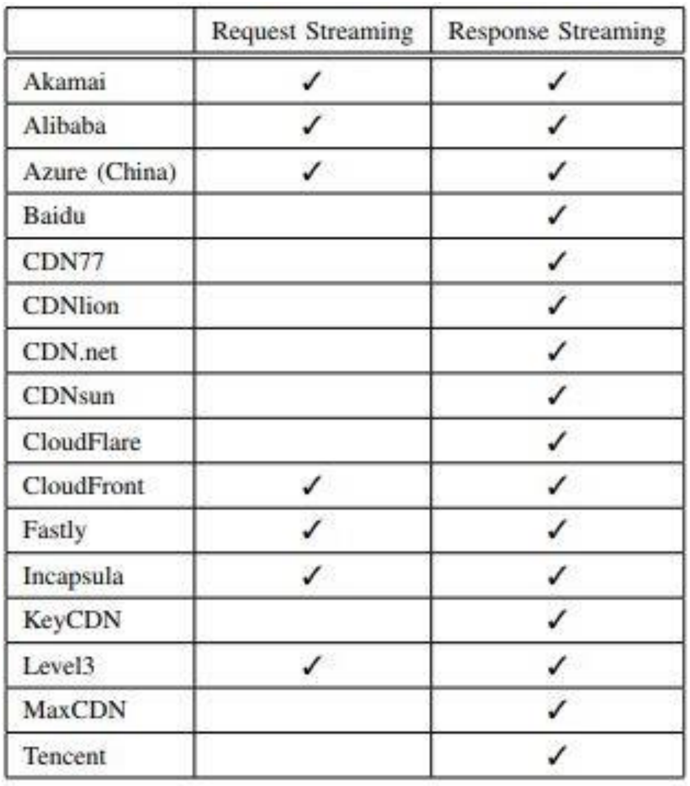
– CloudFlare: it recognized the problem, in particular the serious threat posed by the use of gzip packages and potential consequences, they recommended a report to CERT/CC1 for further discussion;



	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	✓

Fig. 13. Forwarding timeouts and adoption of forwarding abort

– Baidu: after discussing the problem, it admitted to having seen only a very small number of cases similar to those analyzed here and therefore adopted as a preventive measure the use of a special header for the detection of forwarding loops;
– Alibaba: Following a careful analysis of the issue, this CDN provider opted for a closer network control and a limit on connections to stem the damage of any attacks;
– Tencent : admitted the high risk of this vulnerability, seeing it as a serious problem for the CDN industry itself, and declared that he was working on his own to find a solution;
– Fastly : acknowledged and discussed the issue at length. In particular, it suggests the usage of a uniform standard header among all CDNs as an ideal approach to combat the issue of inter-CDN assaults and is committed to making a significant contribution in this regard as well as to enhance performance in comparison to other forms of attacks.



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

Fig. 14. List of CDNs supporting HTTP

7. AVAILABLE STRATEGIES TO MITIGATE THE EFFECTS OF ATTACKS

This section presents the available strategies to detect, or even avoid, the occurrence of malicious behavior triggered within the network.

7.1. No-Strings-Attached

One of the vulnerabilities analyzed above is based on the use of random search strings that allow the attacker to get past the defensive shield offered by the edge servers, and thus reach the origin server directly. Content providers may consider changing the settings of their CDN service as described below to defend themselves against attack. This method, however, does not apply to all CDN services; in those instances, the content provider cannot protect themselves unilaterally and must give up these services or rely on the mitigating options afterward suggested.

To protect itself from vulnerability to using random search strings to cross the cache, a content provider may want to set its CDN service so that only URLs without search strings are accepted by the CDN. As a further preventative measure, it would also be useful to

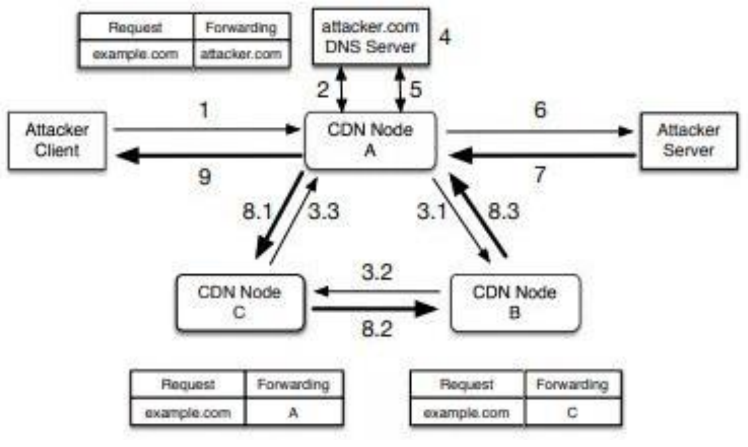


Fig. 15. Principles of a Dam Flooding attack

set the network origin server to report an error message for each request submitted by edge servers containing argument strings. We remind that this procedure is managed and sent from the central memory and consumes few network and server resources, so it seems to be a reasonable solution. In the case of Akamai, for example, customers are allowed to specify a URL pattern to ignore or reject. Therefore, content providers may consider adopting such a mechanism to ensure that edge servers do not resolve any requests with additional strings, thus eliminating the threat presented in the previous chapter. The only exception could be for search strings with a narrow set of legitimate values that can be set on edge servers. This type of resolving approach could be referred to as ”no-strings- attached”. Details on how the no-string-attached technique can be implemented depend on individual websites. For example, to illustrate the general idea, let’s consider a foo.com website that has some dynamic URLs that require apparently random parameters. A possible implementation could concentrate content whose delivery is external to the CDN in one sub-domain, e.g. outsourced.foo.com, and content requiring topic strings in another, e.g. self.foo.com.

Referring to Fig. 17, the DNS for foo.com would return a CNAME record pointing to the CDN only for queries from the first hostname and would respond instead by directly providing the source IP address for the queries for the second hostname. Note, however, that the no-strings-attached approach establishes an ”origin first” setup and eliminates the most popular option of a ”CDN-first” setup. As a result, this strategy sadly significantly reduces the flexibility of the CDN settings, but it does enable the implementation of a firm defense against this kind of attack.

7.2. Unification and Standardization of Loop-Detection Header

As previously discussed, one potential solution is to add a loop-detection header to re- quests. However, even if all CDNs implement this kind of preventive measure, it may still result in endless loops of requests between different CDNs if some of them unintentionally give customers a way to disable headers inserted by other CDNs. Therefore, using the same header across all CDNs and forbidding editing activities on the latter would un-

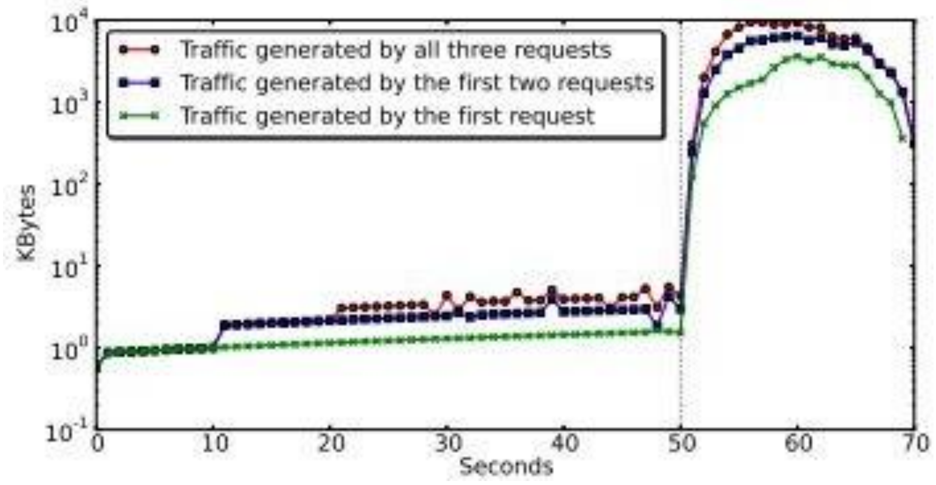


Fig. 16. Traffic generated by three forwarding loops in a ”dam-flooding” attack. The flooding event occurs after 50 seconds

doubtedly be more productive. The Via header is already employed by some CDNs with nodes compliant with the current HTTP standard protocol. The Via header is appended when submitting or returning HTTP requests, and can not be modified by proxies. This solution is better formalized in RFC 8586 [20].

7.3. Blinding Self-Defined Loop Detection Headers

A simple and lightweight solution might be to implement a self-defined header so that it resists tampering by attackers who might be setting a particular route or forwarding rules. To create such a header, for instance, one method would be to encrypt some keywords with other names such that they can be decoded by looking for them.

7.4. Monitoring and Limiting Data Traffic

Another form of mitigation that CDNs could implement is some kind of monitoring and possible reduction of data traffic. For example, a CDN might decide to monitor traffic volume or simultaneous connections per IP address or customer, thus rejecting or down- grading subsequent requests from the same source/customer once their activity exceeds a certain threshold. In particular, an effective strategy to differentiate between legitimate and malicious requests would be to send a 302 error message to the sender where nec- essary, inviting him to try again later. While a regular customer would be automatically redirected, any loops created within the network would cease because all the CDNs we have previously tested, upon receipt of such an error, would forward the response rather than follow the redirection process.

CloudFlare is keen to point out that it has already implemented a limit on simultaneous connections per source IP address and will degrade the performance of the connection with a strategy similar to the one proposed by sending the 302 error once a preset threshold has been exceeded. However, it should be pointed out that any limiting threshold can be circumvented with sufficient planning on the part of the attackers. In an extreme case, a forwarding loop could be formed so that the traffic comes from different IP addresses and is therefore attributed to different (fictitious) accounts. Moreover, the returning-with-302

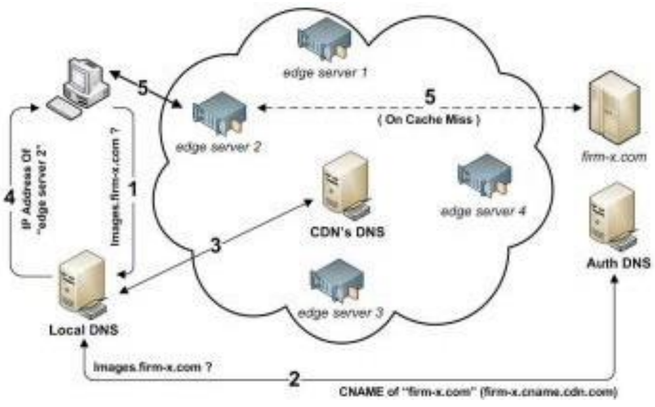


Fig. 17. Content Delivery Network

strategy would not work in the case of the dam-flooding-attack. Monitoring and limiting traffic, however, might make malicious forwarding loops more difficult and hence less common.

7.5. Imposition of Pre-Established Forwarding Addresses

Another possible mitigation would be to implement a blacklist-like policy for forwarding destinations. For example, a CDN may reject requests if their forwarding destination belongs to another CDN. However, more subtle and focused requirements could also be used to establish such restrictions. For instance, CloudFlare discloses its rule that forbids nodes from accepting requests if they originate from a specific CDN but have a different destination. Additionally, CDN777 organization has expressed interest in putting into practice a blacklist-based fix.

The drawback of this strategy is that it takes a lot of resources to keep a comprehen- sive list of CDN IP addresses. The usage of numerous CDNs in a chain would also be discouraged, despite the fact that this strategy has its benefits.

8. CONCLUSIONS

In this work, we presented a survey on a particular type of attack against CDNs named Forwarding loop attack. This attack leverages internal CDN flaws to originate infinite routing cycles. These dynamics, whether deliberate or accidental, should be carefully con- sidered by CDN vendors. Even though there are various mitigation strategies, mainly the introduction of a CDN identifier to append to requests exchanged among edge servers, it still requires perfect coordination among CDNs to be effective against attacks. The other solutions proposed in the final paragraphs of this paper are far from definitive and com- pletely conclusive, so this discussion focuses attention on the security issues of a service that is growing rapidly and is becoming more vulnerable due to novel types of attacks that require further in-depth analysis in a future work.

REFERENCES

[1] Mirkovic, J., Reiher, P.: A taxonomy of DDOS attack and DDOS defense mechanisms. ACM SIGCOMM Comput. Commun. Rev. 34(2), 39–53 (2004)

[2] Guo Run, Li Weizhong, Liu Baojun, Hao Shuang, Zhang Jia, Duan Haixin, Shen Kaiwen, Chen Jianjun, Liu Ying. (2020). CDN Judo: Breaking the CDN DoS Protection with Itself. 10.14722/ndss.2020.24411.

[3] Chen, J., Jiang, J., Zheng, X., Duan, H., Liang, J., Li, K., Paxson, V. (2016). Forwarding-Loop Attacks in Content Delivery Networks. Proceedings 2016 Network and Distributed System Security Symposium. https://doi.org/10.14722/NDSS.2016.23442

[4] Song Hengxian, Liu Jing, Yang Jianing, Lei Xinyu, Xue Gang. (2022). Two Types of Novel DoS Attacks Against CDNs Based on HTTP/2 Flow Control Mechanism. https : //10.1007/978−3−031−17140−623.

[5] Ghaznavi Milad, Jalalpour Elaheh, Salahuddin Mohammad, Boutaba R., Migault Daniel, Preda Stere. (2021). Content Delivery Network Security: A Survey. IEEE Communications Surveys and Tutorials. PP. 1-1. 10.1109/COMST.2021.3093492.

[6] Mai Dinh, Bao Pham, Truong Can, Tung Nguyen. (2023). DDoS Attacks Detection using Dynamic En- tropy in Software-Defined Network Practical Environment. International journal of Computer Networks and Communications. 15. 113-128. 10.5121/ijcnc.2023.15307.

[7] Oo Nan, Risdianto Aris, Ling Teck Chaw, Maw Aung Htein. (2020). Flooding Attack Detection and Mitigation in SDN with Modified Adaptive Threshold Algorithm. International Journal of Computer Networks and Communications. 12. 75-94. 10.5121/ijcnc.2020.12305.

[8] Zolfaghari Behrouz, Srivastava Gautam, Roy Swapnoneel, Nemati Hamid, Afghah Fatemeh, Koshiba Takeshi, Razi Abolfazl, Bibak Khodakhast, Mitra Pinaki, Rai Brijesh. (2020). Content Delivery Networks: State of the Art, Trends, and Future Roadmap. ACM Computing Surveys. 53. 1-34. 10.1145/3380613.

[9] Desai Ankit, Parmar Jekishan, Chaudhary Sanjay. (2015). Content Delivery Networks Technology Sur- vey and Research Challenges. 10.13140/RG.2.1.4805.7689.

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

[11] Ramdas, Anju, Muthukrishnan, Ramakrishnan. (2019). A Survey on DNS Security Issues and Mitiga- tion Techniques. 781-784. 10.1109/ICCS45141.2019.9065354.

[12] Triukose S., Al-Qudah Z., Rabinovich M. (2009). Content Delivery Networks: Protection or Threat?. In: Backes, M., Ning, P. (eds) Computer Security – ESORICS 2009. ESORICS 2009. Lecture Notes in Computer Science, vol 5789. Springer, Berlin, Heidelberg. 10.1007/978 − 3 − 642 − 04444 − 123

[13] Ghaznavi Milad, Elaheh Jalalpour, Mohammad Ali Salahuddin, Raouf Boutaba, Daniel Migault and Stere Preda. “Content Delivery Network Security: A Survey.” IEEE Communications Surveys & Tuto- rials 23 (2021): 2166-2190.

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

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

[16] Micah Adler, Ramesh K. Sitaraman, Harish Venkataramani, Algorithms for optimizing the bandwidth cost of content delivery, Computer Networks, Volume 55, Issue 18, 2011, Pages 4007-4020, ISSN 1389- 1286, https://doi.org/10.1016/j.comnet.2011.07.015.

[17] Triukose Sipat, Wen Zhihua,Rabinovich Michael. (2011). Measuring a commercial content delivery network. Proceedings of the 20th International Conference on World Wide Web, WWW 2011. 467-476. 10.1145/1963405.1963472.

[18] U. Rahamathullah and E. Karthikeyan, Distributed Denial of Service Attacks Prevention, Detec-tion and Mitigation – A Review (May 25, 2021). Proceedings of the International Conference on Smart Data Intelligence (ICSMDI 2021), Available at SSRN: https://ssrn.com/abstract=3852902 or http://dx.doi.org/10.2139/ssrn.3852902

[19] Chen, Jianjun, Jiang, Jian, Zheng, Xiaofeng, Duan, Haixin, Liang, Jinjin Wan, Tao Paxson, Vern. (2016). Forwarding-Loop Attacks in Content Delivery Networks. 10.14722/ndss.2016.23442.

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