Computer Communications 99 (2017) 84–92

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Exploiting Content Delivery Networks for covert channel communications

Yongzhi Wang[a,b], Yulong Shen[a,*], Xiaopeng Jiao[a], Tao Zhang[a], Xu Si[a], Ahmed Salem[a], Jia Liu[c]

[a] School of Computer Science and Technology, Xidian University, 2 South Taibai Road, Xi'an, Shaanxi, PR China
[b] Key Laboratory of Grain Information Processing and Control, (Henan University of Technology), Ministry of Education, PR China
[c] School of Systems Information Science, Future University Hakodate, Hakodate 041-8655, Japan

ARTICLE INFO

Article history:
Available online 25 July 2016

Keywords:
Content Delivery Networks
Covert channel
Information hiding

ABSTRACT

Content Delivery Networks (CDNs) became an important infrastructure in today's Internet architecture. More and more content providers use CDNs to improve their service quality and reliability. However, providing better quality of service (QoS) by using CDNs could also be abused by attackers to commit network crimes. In this paper, we show that CDNs can be used as a covert communication channel to circumvent network censorships. Specifically, we propose the CDN covert channel attack, where accessing contents through different CDN nodes can form a unique pattern, which can be used in encoding secret messages. We implemented a proof-of-concept covert channel based on our proposed attack on CloudFront, a commercial CDN service provided by Amazon Web Service. We showed that our constructed covert channel can transmit messages with various lengths with an average transmission efficiency as 2.29 bits per request (i.e., each *penetration request* transmits 2.29 bits of secret message on average). After presenting the CDN covert channel attack, we also discuss possible countermeasures.

1. Introduction

Content Delivery Networks (CDNs) have received a wide acceptance as a solution to provide on-demand capacity and faster content accessibility. Akamai [1], a leader CDN provider, has deployed 85,800 servers in about 1800 districts within a thousands of different ISPs in more than 79 countries. Today, most of the major content providers, such as CNN, Reuters, Yahoo, Youtube, utilize CDNs to achieve high speed content access.

CDNs enable the user to access the objects from the closest edge server, thus obtains a much higher access speed. On the other hand, due to CDNs' open characteristic, CDNs can be also abused by the attacker for illegal purposes. For instance, [2] described a denial of service (DoS) attack that makes a huge number of CDN edge servers serve as malicious content visitors, which can exhaust the original content server's resource. In our paper, we continue to play the devil's advocate to explore other possible attacks. By exploring vulnerabilities from a different perspective, we show that a *malicious content provider* (MCP) and a *malicious content visitor* (MCV) can use a CDN to construct a covert channel. Specifically, the MCV can access the MCP's contents through different CDN edge servers. Using different CDN edge servers generates different access patterns, which can be used to encode secret messages.

We call our proposed attack as the *CDN covert channel attack*. As far as we know, this is the first paper that describes such an attack. We performed a proof-of-concept CDN covert channel attack on a real commercial CDN, the *Amazon CloudFront*. In our experiments, we showed that secret messages with arbitrary lengths can be sent through this channel. Our experiments in Section 4 shows that the transmission efficiency can be 2.25 bits per *penetration request*. We also discussed the traditional HTTP-based covert channel attack and explored its possible attack scheme in an environment where the CDN is introduced. The ultimate goal of our research is to prevent proposed attacks. Therefore, after presenting the attack details, we discussed possible countermeasures regarding to the proposed attacks.

The paper is organized as follows. We declare the research motivation, the system model and the attacker model in Section 2. We describe the design of the CDN-based covert channel attack and the HTTP-based covert channel attack in the CDN environment in Section 3. We describe the experimental details and results in Section 4. We discuss possible countermeasures in Section 5. We

* Corresponding author. Tel: +862988201779; Fax: +862988202427.
E-mail addresses: yzwang@xidian.edu.cn (Y. Wang), ylshen@mail.xidian.edu.cn (Y. Shen), xpjiao@mail.xidian.edu.cn (X. Jiao), taozhang@xidian.edu.cn (T. Zhang), bryant123@foxmail.com (X. Si), engahmedsalem2@yahoo.com (A. Salem), jliu871219@gmail.com (J. Liu).

http://dx.doi.org/10.1016/j.comcom.2016.07.011
0140-3664/

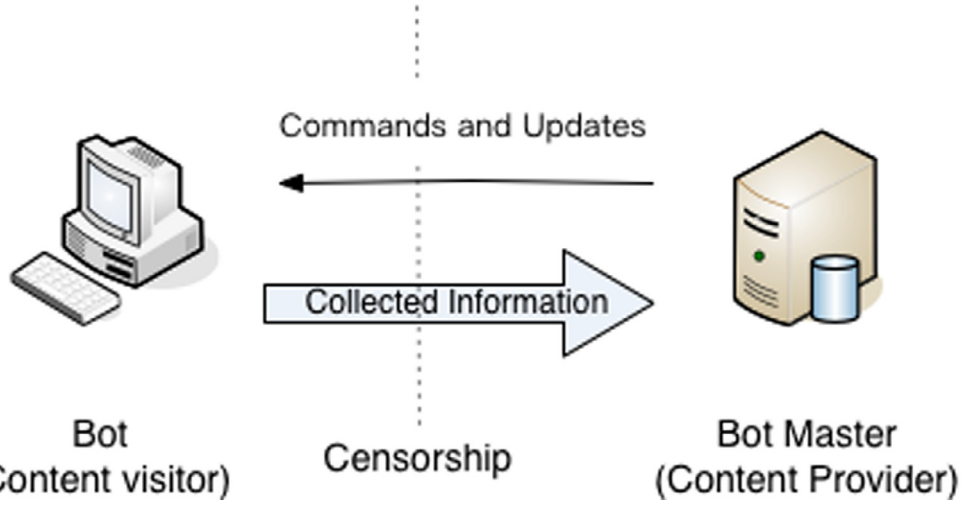

Fig. 1. The Scenario of CDN covert channel attack.

discuss the related work in Section 6. We conclude the paper and point out the future works in Section 7.

2. Research motivations, the system model and the attacker model

2.1. Research motivations

The covert channel communication is usually used for transmitting secret information while circumventing the Internet's censorship. One of the applications of the covert channel is the botnet command-and-control (C & C) as shown in Fig. 1. A botnet usually consists of a number of bots and a bot master. The bot master sends commands or updates to the bots. Each bot, according to the bot master's command, performs attacks or steals information from the infected host, and submits stolen data or attack results back to the bot master. The bot master can be deployed on the content provider's server. Bots can be deployed on the content visitor's machines. The botnet commands or updates can be sent to the bots when bots visit the content provider's server as a visitor. Given the high volume of the content to be transmitted from a content provider to the visitor, it is fairly easy to hide the botnet commands and updates in the transmitted content. However, the information sent from a content visitor to the content provider is usually limited. In many protocols such as HTTP and DNS, content visitors only send content requests to the content provider. The amount of information hidden in a request is usually quite limited. Besides, certain censorship mechanisms exist in the Internet to recognize such requests. On the other hand, in a botnet, the amount of information sent from the bot to the bot master are usually large. Therefore, finding an effective and safe covert channel that can transmit a large amount of information is still a challenging problem. Our research thus focuses on the secret information transmission in this direction, i.e., from the content visitor to the content provider. Attackers have constructed different covert channels on existing protocols for the message transmission, including Relay Chat Protocol (IRC), HTTP [3], Domain Name System (DNS) [4], email [5], etc. Our research propose a novel covert channel, which uses CDN services to transmit information from the bot to the bot master secretly.

2.2. The system model

A CDN can be modeled as in Fig. 2. A typical CDN consists of a large number of *edge servers* and *domain name system* (DNS) servers. Edge servers cache contents provided by content providers. DNS servers direct content visitors to the edge server that caches the requested content. As shown in Fig. 2, when a content visitor wants to access an object that is hosted on the content provider, the request is sent to the local DNS server (step 1). The server will return the IP address of the requested object if such information is cached in the local DNS server (step 4). Otherwise, it will forward the request to the CDN's recursive DNS server (step 2), which will return the IP address of the closest edge server to the content visitor (step 3). The IP address will be returned from the local DNS server to the content provider (step 4). With the edge server's address, the content visitor sends a request to the edge server and obtains the object if the object is cached in the edge server (step 5 and 6). If the object is not cached, the edge server will forward the request to the content provider to fetch the object and return the received object to the content visitor. Meanwhile, the edge server may cache the obtained object using different caching algorithms [1,6] to avoid future cache miss (step 7 and 8).

2.3. The attacker model

In our research, we assume that the bot master is deployed on a server, which provides contents through CDN services. Thus the content provider is a *Malicious Content Provider* (MCP). This server can belong to a benign content provider, which is compromised by being installed a bot master. It can also be a server belonging to a malicious content provider. In this case, the bot master is deployed intentionally by the malicious content provider. We assume that the bot is installed on the visitor's host. Thus the visitor is a *Malicious Content Visitor* (MCV). A benign visitor can be installed a bot when it visits a malicious content server. Our paper will show that when the visitor, i.e., the bot, accesses the content provided by the content provider, i.e., the bot master, it can send covert messages to the bot master through the CDN. In our paper, we mainly focus on the information transmission from the bot to the bot master. We assume that the information sent from the bot master to the bot can be hidden in the content provided by the MCP through traditional information hiding techniques [7].

3. Attack design

3.1. Architecture

The architecture of the CDN covert channel attack is shown in Fig. 3. A MCP (the bot master) hosts a server to provide contents

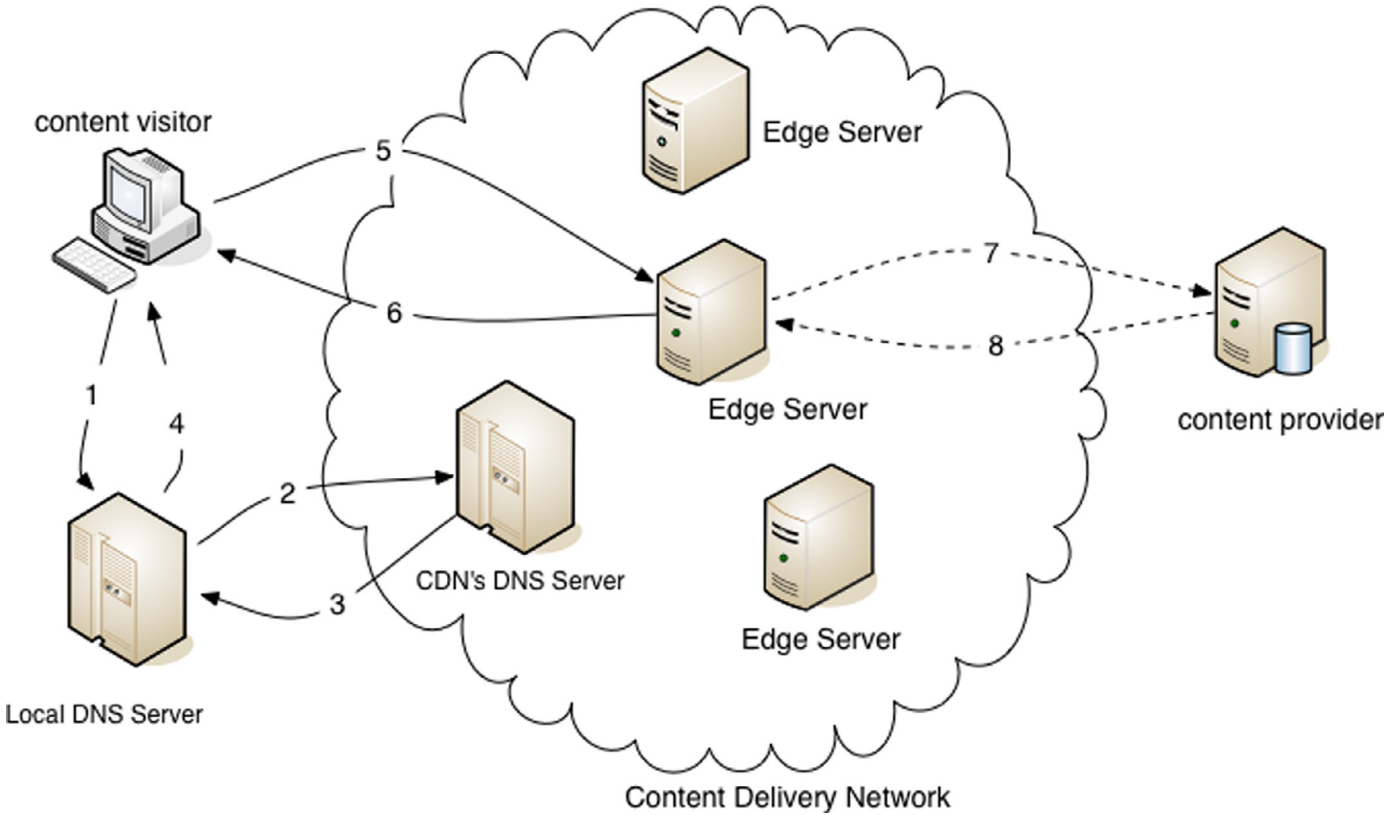

Fig. 2. The system model of a CDN.

to its visitors via a CDN. A MCV (the bot) sends requests to the MCP to access contents provided by the MCP. Since the MCP employs a CDN to distribute its content, the MCV's requests are sent to the edge servers of the CDN. Usually, the MCV's requests will not achieve the MCP, if the edge server caches the requested content. In this case, the edge server will directly return the cached objects to the MCV. However, in our proposed attack, we force the MCV to send a special request, called the *penetration request*, which will force the edge server to fetch the latest contents from the MCP. Additionally, we force the MCV to select an edge server to send the requests. The MCP thus can know which edge server was used to pass the MCV's request. Such information can be used to encode secret messages sent from the MCV to the MCP.

The CDN covert channel attack can be divided into six steps, as shown below.

a. Set up a MCP server that uses a CDN service.
b. Collect the IP address information of the CDN edge servers.
c. Generate the information encoding scheme.
d. Broadcast the collected IP information and the encoding scheme to all MCP and MCVs.
e. On the MCV, encode the secret message into a series of penetration requests and send these requests.
f. On the MCP, decode received requests to restore the original secret message.

Notice that step b and c can be performed by MCP, or a MCV, or any other host that is controlled by the attacker. We verified that the collected IP information in step b is stable (the details are described in Section 4). The collected IP information can be performed only once and be reused by all other MCVs and the MCP. Thus steps b and c only need to be performed once. In step d, the collected IP information and the encoding scheme need to be sent to the MCP and all the MCVs through an offline channel, which can be through an email or a file transferring. This step also only needs to be performed once and will be reused afterwards. Step e and f need to be performed in rounds repeatedly. In each round, the MCV sends a certain amount of information to the MCP.

The essential ingredients that achieves the above six steps are the MCP server setup, IP address information collecting and the encoding scheme design. We will elaborate these three techniques in the next section.

3.2. Channel construction details

In this section, we will describe the three important ingredients required to ensure a successful covert channel, i.e., setting up the MCP server, collecting edge server's IP address information, and encoding the secret message.

3.2.1. Setting up the MCP server

Setting up a MCP that uses a CDN service is similar to setting up a benign content provider. The only difference is that the MCP should be able to log the IP addresses of its direct visitor (i.e. the visitor that directly sends requests to the MCP, it can be different from the MCV, as described later). For example, in our experiments in Section 4, we set up a Tomcat web server as the content provider. The Tomcat server has the logging function that can log each HTTP request received by the MCP, along with the IP address of the request sender. The sample log file is shown in Fig. 4. Each line of the log records the information of a request. For example, the first line in Fig. 4 indicates that a visitor with IP address 216.137.58.45 has sent a *GET* request for the object */testApp/css/test.css* with a query string *batch_1*.

3.2.2. Collecting IP address information

We noticed that IP addresses in the log file belong to edge servers that directly access the content. They usually are not the

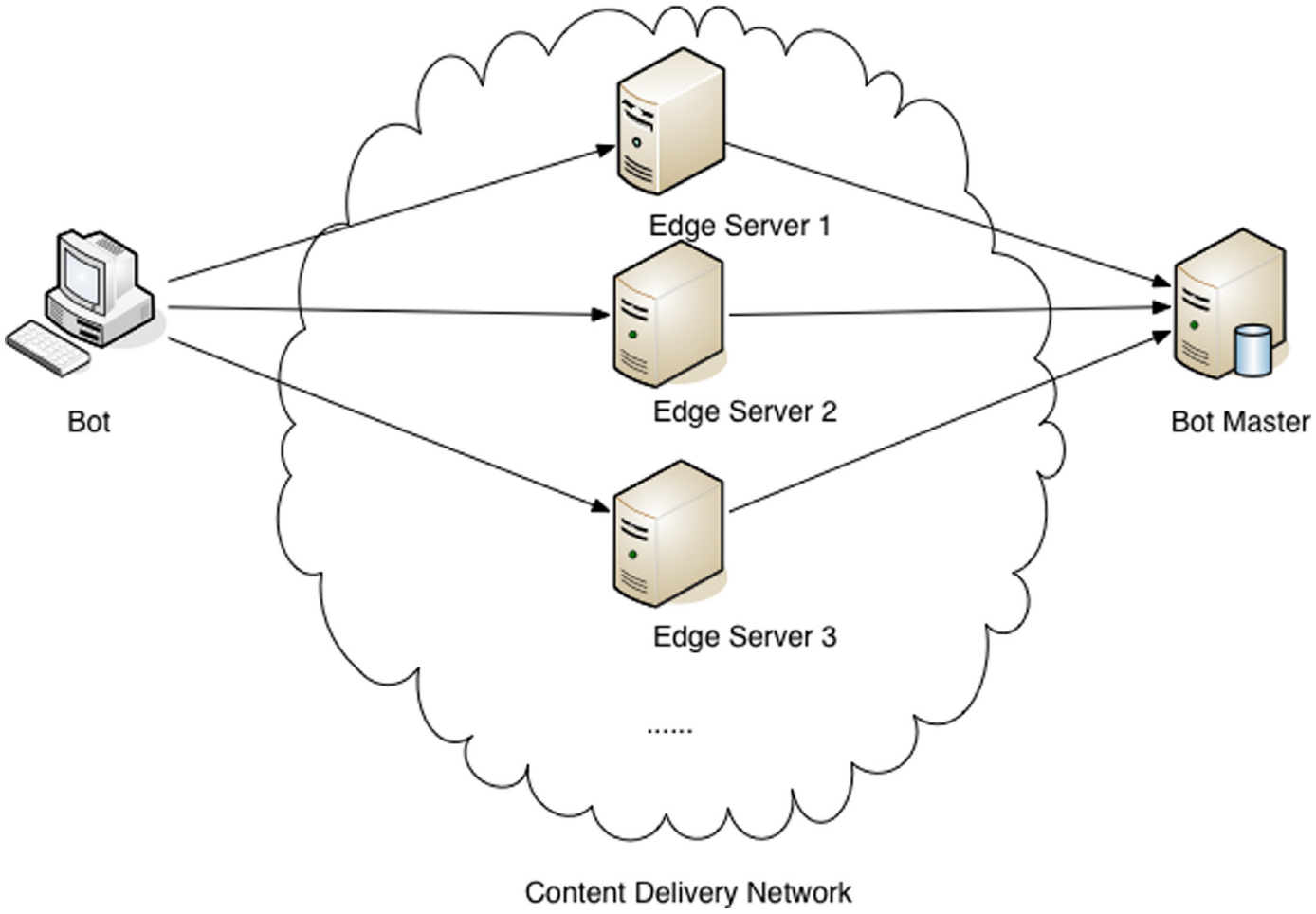

Fig. 3. The architecture of CDN covert channel attack.

```
216.137.58.45 - - [10/Dec/2015:06:41:26 +0000] ``GET /
    testApp/css/test.css?batch_1 HTTP/1.1" 200 38
54.240.159.109 - - [10/Dec/2015:06:45:50 +0000] ``GET /
    testApp/css/test.css?batch_1 HTTP/1.1" 200 38
205.251.221.106 - - [10/Dec/2015:06:47:38 +0000] ``GET /
    testApp/css/test.css?batch_1 HTTP/1.1" 200 38
```

Fig. 4. A sample of the MCP's log file.

edge servers that directly receive the MCV's requests. To make our future presentation clear, we will call the edge server that directly access the content from the MCP as the *last hop edge server* (or the *last hop server* for short) and we will call the edge server that directly receives the request from the MCV as the *first hop edge server* (or the *first hop server* for short). For instance, our experiments on Amazon CloudFront (in Section 4) showed that for each request, the first hop server and the last hop server usually are not the same edge server. Although the documentation for the CloudFront does not explain why the edge server receiving a request is different from the one fetching the object, we guess that Amazon CloudFront might use the following request forwarding mechanism. The first hop edge server that receives the request is far away from the MCP. To fetch the latest object, the first hop edge server might go through multiple edge servers (multiple hops) before reaching the MCP. In other words, when an edge server is selected as the next hop server, it will recursively select the hops for the next steps, until the MCP is reached. The requested object is thus cached in the first hop edge server for future access (it is possible that the requested object will be cached in each hop to speed up future requests).

Our goal is to make the MCP to be able to recognize from which edge server the request is sent. Such information can be used in encoding secret messages. Our following techniques enable us to choose the first hop edge server in which the request will be sent to. However, the IP address in the log file belongs to the last hop edge server. Fortunately, we verified that the correspondence between the first hop servers and the last hop servers are fairly

stable (the verification details are described in Section 4). Thus if we can collect the mapping information between first hop servers and last hop servers, it will help the MCP to derive the first hop edge server from the requesting last hop edge server and therefore distinguish the information sent from the MCV.

We break our IP information collecting step (step b) into two sub-steps as follows:

b.1. Collect the IP addresses of valid first hop edge servers.
b.2. Collect the mapping information between the first hop edge server and the last hop edge server.

In order to collect the mapping information, we first need to ensure that each request sent by the MCV will be received by the MCP. In other words, the request needs to bypass the CDN's DNS lookup and the content caching mechanism. We call such requests as *penetration requests*. To generate penetration requests, we reuse the technique described in [2]. Specifically, to access an object at the MCP, the MCV will use a command-line tool named *curl* to specify which edge server the object will be requested from and append a search string (the optional portion of a URL after "?") to ensure that the requests are sent to the MCP. We can verify whether a request has been sent to the MCP by reading the log file on the MCP. For instance, suppose that a MCP has an original domain name *mydomain.com*. When this MCP employs a CDN service, the distribution domain name that will be assigned to this MCP by the CDN is *mydomain.cloudfront.net*. If a MCV wants to request an object http://mydomain.com/testApp/css/test.css through a specific CDN server 54.16.44.39, it can issue the following command instead of simply accessing http://mydomain.cloudfront.net/testApp/css/test.css in a browser.

```
curl – H Host : mydomain.cloudfront.net
http : //54.16.44.39/testApp/css/test.css?batch_1
```

In this command, the MCV sends the HTTP *GET* request to the CDN edge server 54.16.44.39 to access the object */testApp/css/test.css* at the MCP. By specifying the expected host header through the "-H" argument, the CDN edge server knows that the object belongs to the distribution mydomain.cloudfront.net. Normally, the edge server will check if it has already cached the expected object locally. It will return the cached object if the check result is true. In this case, the request will not achieve at the MCP, which will fail the covert communication. In order to avoid this, in the curl request, the MCV appends a random query string, such as*? batch_1*, which forces the edge server to fetch the latest object from the MCP. It is because the CDN cache uses the entire URL strings, including the search string, as the cache key. When an url that requests the same object but with a different query string is received by CDN, it will be considered as a new key. Since such a key does not exist in the cached keys, the request will be forwarded to the MCP.

With the help of the penetration request, we are able to collect the IP information with the two sub-steps mentioned previously (i.e., the sub-step b.1 and b.2). In the sub-step b.1, we first collect as many valid edge server IP addresses as possible. Our starting point is to find the IP address ranges that could be CDN edge servers. Our experiments were performed on Amazon CloudFront. In our experiments, we were able to find the IP address ranges of Amazon's CDN edge servers in its document [8]. We found that not each address could be used to send the penetration request. Specifically, when using the *curl* command with different edge server IP address, some servers returned error code, such as "Empty reply from server", or "Operation timed out". In order to collect as many valid IP addresses as possible, we wrote a script to try different IP addresses that are within one address range reported in [8]. In total, we were able to find 47,925 valid IP addresses. These edge servers can be used as first hop edge servers.

The sub-step b.2 is to collect the mapping information between first hop edge servers and last hop edge servers. This information is collected at the same time of performing sub-step b.1. For each edge server address we tried in the above script, if the address is valid, we recorded the first hop and the last hop edge server address pairs. Specifically, our script on the MCV requests the object from different first hop edge servers. The MCP logs the requests received from the last hop edge server. In order to track the correspondence relationship between first hop edge servers and last hop edge servers, our script appends a unique query string for each request that will be sent through a different first hop server. By doing so, the query string and its corresponding last hop edge server address will be reflected in the same line of the MCP's log. Thus, the first hop edge server and the last hop edge server are connected by the query string. Therefore, we can collect the mapping information between the first hop server and the last hop server. In the mapping information that we collected, we found that usually multiple first hop addresses correspond to the same last hop address. For example, both the first hop servers 54.192.15.127 and 54.192.15.128 correspond to one last hop server 126.137.58.45. Therefore the two first hop servers are set into one *group* that corresponds to the last hop server 126.137.58.45. When sending a penetration request through this last hop server to the MCP, we only need to select one first hop server from the group that corresponds to it and send the request through the selected server. It is because that the MCP cannot distinguish the two first hop servers. In our experiments, although we found 47,925 valid first hop server addresses, we only collected 78 last hop servers, which means we broke the first hop servers into 78 groups.

3.2.3. Secret message encoding scheme

With the previous preparation, we are able to encode secret message into a set of penetration requests. Any secret message can be represented as a binary sequence. We can therefore send the binary sequence through the CDN covert channel. The detailed encoding scheme is as follows. Suppose we collected N first hop groups, we denote the collection of the groups as $\{g_1, \ldots, g_N\}$ and their corresponding last hop servers as $\{l_1, \ldots, l_N\}$. We can send the secret message in multiple batches, each batch will send at most N bits. We call the maximum number of bits can be sent in a batch as the *channel base*. Specifically, for a batch that consists of N bits, requesting an object through any first hop server in group g_i represents sending a 1 on the ith bit; not requesting an object through g_i represents sending a 0 on the ith bit. Different batches can be distinguished by requesting different objects (for instance, sending an object request *testApp/css/batch_j.css* through any first hop server in group g_i indicates that the ith bit in the jth batch is 1), or appending different query strings if requesting the same object (for instance, sending an object request *testApp/css/test.css? batch_j* through any first hop server in group g_i indicates that the ith bit in the jth batch is 1). In order to decode the secret message, the MCP collects the information for each batch through grouping the requests by the requested objects or by the query strings. For each group of requests, based on the availability of the requests received from last hop servers $\{l_1, \ldots, l_N\}$, the MCP can restore each batch into a binary sequence.

Our experiments chose to distinguish different batches by appending different query strings in the penetration requests. In this section, we describe such an encoding scheme briefly. The detailed scheme can be found in Section 4.3. The scheme that distinguishes different batches according to the different requested objects is similar to this scheme, we skip discussing it in this paper. Fig. 5 uses an example to show the idea of the encoding scheme. In

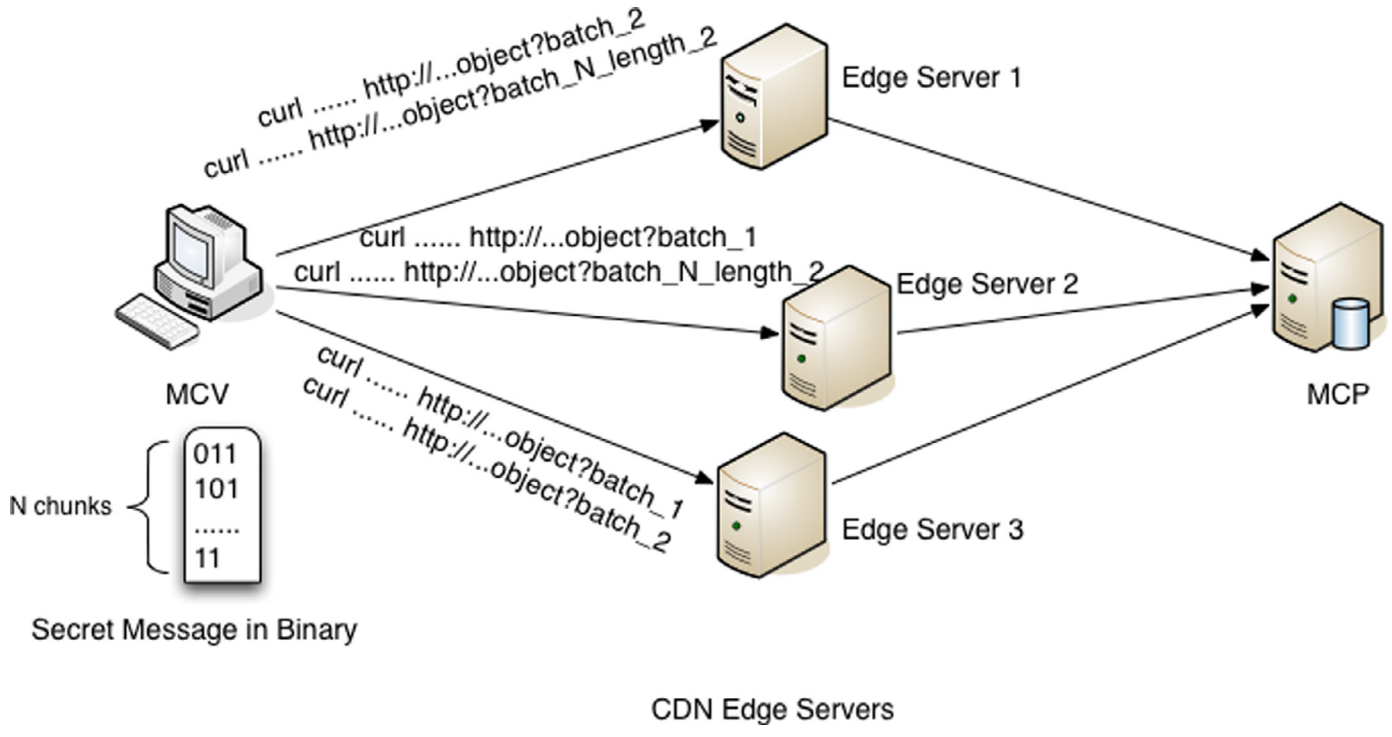

Fig. 5. An example of message encoding.

Fig. 5, we assume there are three edge servers. In other words, the channel base is 3. To encode a sequence of binary "011101...11", we first split the original sequence into 3-bit chunks. If the length of the binary is not an integral of 3, the length of the last chunk will be less than 3. For instance, in the figure, the last chunk only has 2 bits. For each chunk, we send a request through an edge server in the first hop server group g_i, if the value of the corresponding bit i in this chunk is 1. If the corresponding bit i is 0, we do not send any request. For requests corresponding to chunk j, the query string is *batch_j*. For instance, for the first chunk "011", the requests with the query string *batch_1* are sent through the edge servers 2 and 3, since only the second and the third bits in this chunk are 1. For the chunk that does not have the full length, we also specify the chunk length in the query string. For instance, to transmit the last chunk (the chunk N) that only has two bits, the query string will be *batch_N_length_2*. Such requests will be sent through the edge servers 1 and 2, indicating that the first 2 bits in this chunk are both 1.

The encoding scheme has created a basic channel. However, to make this channel practical, we still have to consider other factors. For instance, if some edge servers are unreliable, requests will not be delivered to the MCP. In this case, sending a bit 1 will be mis-received as 0. To evade such a problem, error correction code can be used. The internet censorship can also perform pattern matching on the request query string to detect the secret message. To evade such detection, the query strings need to be encrypted. Those problems are classic communication problems. Mature solutions already exists [9] and can be applied to our encoding schemes. In this paper, we only focus on the basic channel. Therefore, our solution does not consider the above problems. Integrating solutions of the above problems to the constructed covert channel will be our future work.

3.3. HTTP-based covert channel attacks through CDN

We also explore the possibility of applying HTTP-based covert channel attacks [10] under the CDN environment. The idea is to have the MCP and the MCV agree on a certain encoding scheme, where each file name corresponds to a secret message to be sent. The files are hosted on the MCP but initially NOT cached on the CDN. To send a secret message, the MCV sends requests to the MCP through the CDN, requesting different files. Since the files are not cached on the CDN, the CDN edge server will fetch the file from the MCP. The MCP, upon receiving this request, will send the requested file to the CDN edge server, which will be forwarded to the MCV. Meanwhile, the MCP decodes the file name into the secret message. When the same secret message is going to be sent again, the MCP and the MCV will update the name of the requested file corresponding to the secrete message based on a certain pre-agreed update protocol, (e.g., the file name can be a random number generated by a pseudo random generator). By doing so, we ensure that all secret messages will be delivered to the MCP.

We give a concrete covert channel design following the above principle: The MCP and the MCV share a key (or a random seed for a pseudorandom number generator), K, which is used to generate distinct file names. The MCP hosts N files, where N is a large number. The names of the files are unique random numbers generated based on K and thus known by the MCP and the MCVs. Each file has an index, ranging from 0 to $N-1$, known to both the MCP and the MCVs (e.g., the index is based on the sorted order of the file names). Notice that initially, the files are not cached by the CDN. To transmit a $logN$ bit long value v, the client requests the file at index v. Since the file is not cached by the CDN, the edge server will fetch the file from the MCP. Once the file is requested, it will be cached by the CDN, which means the next request of the same information will not reach the MCP. To solve such a problem, once a file is requested, the file name is replaced by a different file name generated based on K, on both the MCP and the MCV. The indexes of the file names will be updated (e.g., the indexes are reassigned based on the sorted order of the updated file names). Note that since both the MCP and the MCV know the requested file, if the file name replacement algorithm and the index update algorithm are deterministic, the updated indexes on the MCP and

the MCV will be synchronized. Therefore, we ensure that all secret messages will be delivered to the MCP.

The advantage of such a covert channel is that the size of the message (i.e., the number of bits) transmitted in each request can be defined based on the encoding scheme (e.g, the above concrete design can transmit *logN* bits of message in each request.). However, such a covert channel is more of a HTTP-based covert channel applied under the CDN environment. We therefore only propose this idea and do not thoroughly explore this direction.

4. Experiments

We constructed a proof-of-concept covert channel on Amazon CloudFront, a commercial CDN service, and performed a series of experiments.

4.1. Environment setup

The experiments was performed on *Amazon Web Services* (*AWS*). We hosted a Tomcat web server on *Amazon EC2* instance, acting as the MCP. To show the concept and to make experiments simple, this web server only hosts one object (i.e., testApp/css/test.css), which can be accessed through HTTP GET requests. We leveraged Tomcat server's log function to record all the object requests received by the MCP. The log format is shown in Fig. 4. In each entry of the log file, the IP address is the last hop IP address that directly requests the object, the "GET" command records the requested object and the query string, if available. We created a CloudFront *distribution* that caches the hosted objects on CloudFront's edge servers. We performed the CDN covert channel attack on this experimental environment.

4.2. IP information collection

We collected the IP addresses and the IP address mapping information with the methods described in Section 3.2.2. Specifically, we obtained the IP address ranges of the CloudFront's edge servers from AWS's documentation [8]. That document indicates 17 IP address ranges that are assigned to AWS's edge servers. We only chose one out of the 17 address ranges to perform our experiments. This range contains 65,535 IP addresses. We wrote a script to send penetration requests using each IP address in the selected range as the first hop server. By using the technique described in Section 3.2.2, each request tries to penetrate the CDN cache to arrive at the MCP through a last hop edge server. Each request will "GET" the same object (testApp/css/test.css). To penetrate the caching mechanism, we appended different query strings for each request. The query string contains the IP address of the first hop server that we used in that request. As a result, the first hop server IP address will also be reflected in the Tomcat server's log, which facilitates our log file processing. During our experiments, although most penetration requests were successfully sent to the MCP, several requests through several IP addresses timed out and did not receive any reply. As a result, we collected 62,538 out of 65,536 available IP addresses that can be used as first hop servers. By processing the Tomcat log, we found 78 first hop edge server mapping groups that map to 78 last hop edge servers.

We verified the stability of those mappings. Specifically, we sent penetration requests repeatedly through the same first hop IP address and verify whether the requests can be received from its corresponding last hop IP address. We found that the mappings between the first hop edge server and the last hop edge server preserves at the beginning. As time passes, some requests sent by the first hop edge server are received by another last hop edge server. Such a finding means that the mapping information can be directly used for a short time after it was collected. As time passes, to guarantee an accurate transmission, either error correction codes should be applied to the secret messages, or the mapping information should be re-collected. In this paper, we only focus on the case that the mappings are preserved. We will explore the solution to the case that the mappings are partially preserved in the future.

We also verified the stability of the mappings in the space dimension. Specifically, we set up multiple EC2 instances in different regions and sent penetration requests through the same first hop IP address from those instances. We found that those requests were all sent to the MCP through the last hop edge server that was previously mapped to the current group. The verification results indicate that the mapping information, once collected by one MCV, can be re-used by other MCVs from different locations.

4.3. Secret message transmission

We collected 78 last edge servers in the previous step. Thus we set the channel base as 78 in our experiment. The message encoding and decoding procedure follow Section 3.2.3. In each batch, we sent 78 bits. Since the MCP in our experiments only hosts one object, to distinguish different batches, requests that belong to the same batch will use the same query string; requests belonging to a different batch will use a different query string. The encoding scheme follows the example in Fig. 5. To indicate the batch order, we include the order information in the query string. To extend the scheme to the scenario where multiple MCVs send information to the MCP, we also include the MCV's information in each query string. For instance, suppose the channel base of a CDN covert channel is b, if a MCV x sends requests in batch i, the query strings will be *MCV_x_batch_i*. For the last batch, whose length l is less than the channel base b, we also specify the length in the query string, marked as *MCV_x_batch_i_length_l*. As we transmit the secret message, we found that in some batches, penetration requests through some first hop servers timed out and did not receive any respond, which failed the transmission of several bits. We also noticed that those first hop servers restored their functionalities in later batches. We guess that when an edge server is overloaded, it simply drops the request, which makes the request timed out. To avoid such a transmission failure, we dynamically change the selected first hop server to ensure the communication quality. Specifically, if a penetration request sent through a first hop server in group g failed, we will switch to another first hop server in this group to send the request. If this server still fails, we switch to another, until the request is sent successfully from a first hop server in group g. However, if all the first hop servers in group g fails to send the request, we fails to directly send 1 in bit g. In this case, we might lose one bit during the transmission. To remedy this failure, we send this bit in the query string of another request that was successfully sent. For example, suppose in batch i, the MCV x fails to send requests through all the first hop servers in group g. The query string of another successfully sent penetration requests will be *MCV_x_batch_i_drop_i_g*, indicating that the gth bit in batch i is 1, but failed to be transmitted. The MCP receiving this request knows that the gth bit in batch i should be 1, although it did not receive such a request.

Based on the above encoding rule, we wrote a python script to automatically translate secret messages into a binary strings and send a series of penetration requests based on the values of the binary strings. Our experiments showed that, if we send penetration requests too frequently, CloudFront will trade it as a denial-of-service attack and will disable the CDN service on our account. To avoid such a detection, the time interval between each two penetration requests is set to be a random large number. The requests were logged by the Tomcat server. We wrote another python script to restore the requests collected in the log file back into the original secret message. To measure the performance of our CDN covert

Table 1
Experimental results.

Message size (Bytes)	64	1k	2k	4k	8k	16k
Request number	218	3359	7219	14,606	29,377	58,819
Transmission efficiency (bits/request)	2.04	2.05	1.95	1.98	1.09	2.15

channel, we performed the message transmission with different sizes of secret messages. Our experiments showed that all the messages are transmitted successfully. Table 1 shows the message *transmission efficiency* for different lengths of secret messages, defined as the number of bits sent when sending one penetration request. Table 1 shows that the request number is proportional to the message size, which makes the values of transmission efficiencies fairly stable. On average, our experiments achieved a transmission efficiency of 2.29 bits/request. In this experiment, we did not measure the number of bits sent in a certain amount of time, defined as the *transmission speed*. The reason is that such a metrics is largely related to the random time intervals inserted between any two consecutive penetration requests, which has a very large variance.

5. Countermeasures

In this section, we first discuss countermeasures that can mitigate the CDN covert channel attack. After that, We discuss the countermeasures of the HTTP-based covert channel attacks in the CDN environment.

In order to construct a CDN covert channel, three requirements needs to be satisfied. Firstly, the MCV can choose an edge server to request an object. Secondly, the penetration request can bypass edge servers' caching mechanism. Lastly, the mapping between the first hop edge server and the last hop edge server are stable. If any of the three prerequisites is dissatisfied, the CDN covert channel can be blocked. We discuss the possibility of disabling the three requirements separately. In this paper, our focus is to analyze the causes of our proposed attack and discuss general directions for the defense. We will leave detailed defense solutions as our future work.

Our experiments showed that Amazon CloudFront allows users to specify an edge server by adjusting the IP address to send requests. According to [2], Akamai, the biggest commercial CDN provider, and CoralCDN, the biggest free public CDN, also support such a function. To disable such a support, the CDN provider can set up a special type of server, called *routing servers*, to forward users' requests to edge servers. With the routing server, the edge server only accepts requests from routing servers. One disadvantage of such solution is that the routing server can become the bottleneck of the CDN infrastructure. Even worse, performing DoS attack to routing servers will compromise the availability of the entire CDN service. To mitigate such a disadvantage, one improved solution using routing servers is as follows: Whenever a routing server receives a request from a user, it selects an edge server for the requesting user and generates a *certificate token*, which is based on the information of the requesting user and the selected edge server. The certificate token will be sent to the selected edge server and the requesting user. In addition, the IP address of the selected edge server will be returned to the requesting user so that the user can connect to the edge server. The requesting user then sends the request and the certificate token to the selected edge server through the received IP address. The edge server will first check the certificate token, if it matches the one it received from the routing server, it will process the request. By following this procedure, the routing server does not need to maintain the request connection, thus reduces its workload. Such a solution will make the CDN itself to be able to determine the edge server, which will prevent the MCV from bypassing the CDN's routing mechanism.

In order to prevent bypassing the edge server's cache, the edge server can disable the query string parameter. By doing so, the edge server will update the cached object only when the original server updates its own original object. Thus there will be no necessity for the user to fetch the object directly from the original server, which will break the covert channel. However, enabling query string parameter is a feature to better support dynamic content delivery. For instance, sometimes, different objects should be returned based on different query strings. Simply disabling the query string parameter will degrade the CDN's service quality. Therefore, under the current service requirement, it is infeasible to prevent bypassing the cache mechanism without downgrading the service quality.

Our experiments on Amazon CloudFront shows that the edge server that receives the request from the MCV (the first hop server) is different from the edge server that is requesting the object from the MCP (the last hop server). According to our experiments, although first hop servers and last hop servers are not the same set of edge servers, the mappings between the two types of servers are stable. Requesting an object from a first hop server always ends up with receiving the request from a fixed last hop server. The MCP can thus derive the first hop server mapping group from the last hop server, from which the request was received. To interfere the covert channel, the CDN can randomize the mapping between the first hop server and the last hop server. Specifically, by adjusting the routing algorithm to make the route different for each object requests can randomize the mapping between the first hop server and the last hop server. Since the exact routing algorithm used in Amazon CloudFront is not public, it is difficult to provide a concrete solution on our experimental environment. However, designing a generalized routing algorithm that introduce randomness to defend the CDN covert channel attack is an interesting topic.

The HTTP-based covert channel attacks can be detected by detecting the request pattern signature. When such attacks are performed in the CDN environment (as described in Section 3.3), frequent cache missing is a request pattern signature that can be used to detect such attacks. To hide such a signature, the MCV can inject requests to the original requests set. The injected requests are repetitions of the previously sent requests, which will not reach the MCP, but only to make the cache missing frequency similar to that of the normal access. However, such a strategy cannot reduce the total number of cache missing incidents. The size of the secret message to be sent is proportional to the number of cache missing incidents. When the secret message to be sent is large enough, the number of the cache missing incidents is large, which can be used as a request pattern signature in detecting such attacks.

6. Related works

Previous studies of CDN security mainly focus on the availability of the service. In the attacking side, Su et al. [11] focus on Akamai's streaming service, describes an attack to downgrade the streaming service quality. Triukose et al. [2] describe a method to perform a DDoS attack to the CDN user by abusing CDN' infrastructure. Recently, Chen et al. [12] describes a Denial-of-Service attack that leverages request forwarding mechanism in CDNs. Most previous attacks are focusing on the content availability. As far as we know, we are the first to describe a covert channel-based attack.

On the defense side, Wang et al. [13] describe the defense strategy applied in preventing the edge server breaks-in in order to provide a reliable service. Lee et al. [14] propose a hash-based

routing algorithm that evenly distributes requests to multiple CDN edge servers to mitigate the DDoS attack. Unfortunately, the edge server selection in this paper is still deterministic. Thus, it does not help to defend our covert channel attack. In [15], Jung et al.propose the dynamic caching hierarchies to handle a high volume of request events. In this strategy, the edge server selection is dynamic. Still, the defense is focused on the availability, the edge server to handle the request changes only when the server is overloaded or underloaded.

Although covert channel attacks in CDNs are rarely studied, covert channel attacks on other network protocol have been discussed for decades. HTTP is one of the most frequently used protocols for Internet browsing and can be exploited in different ways for covert communication. Covert data can be encoded in URL parameters [10] of HTTP requests. Advanced techniques encode information into HTTP header field values, order of header fields, use of lower or uppercase, presence or non-presence of optional headers, use of multiple white spaces and nonstandard header fields [16–18]. Domain Name Service (DNS) maps domain names to IP addresses. Because DNS is one of the basic protocols on the Internet, most firewalls allow DNS traffic, which opens the door of abusing DNS for covert channels. PSUDP [19] is a tool that creates a network-wide messaging system by piggy-backing on legitimate network DNS traffic. In [20], Born presents a covert DNS channel using JavaScript with modern browsers. Heyoka [21] is a prototype exfiltration tool which uses spoofed DNS requests to create a bidirectional tunnel. Xu et al. [22] explore the feasibility of only using DNS queries to stealthily and effectively maintain large botnets. They hide tunneling DNS query activities by piggybacking tunneling queries with legitimate DNS queries and temporally distributing DNS queries.

7. Conclusion and future work

Content Delivery Networks have received widely adoption by most content providers. While offering convenience and higher service quality, content delivery networks can also be abused for illegal purpose. In this paper, we proposed a CDN-based covert channel, through which the malicious content visitor can send secret message to the malicious content provider. We constructed such channel on a commercial CDN service, Amazon CloudFront, and measured its transmission efficiency. Based on the proposed attacks, we discussed possible countermeasures.

This paper only explored the basic encoding scheme of the CDN-based covert channel. In the future, we will explore advanced encoding schemes that can address practical problems such as the request loss, the IP address mapping instability, etc. On the defense side, this paper only discussed the general directions of countermeasures. Designing and implementing a practical CDN system that defends the covert channel attack will be another direction in our future work.

Acknowledgment

We acknowledge the suggestions and comments from reviewers. This paper is supported in part by the Open Fund of the Chinese Key Laboratory of the Grain Information Processing and Control (under the grant No. KFJJ-2015-202), the Fundamental Research Funds for the Central Universities (under the grant No. XJS16042, JB160312 and BDY131419), as well as the National Natural Science Foundation of China (under the grant No. U1536202, 61571352, and 61373173).

References

[1] B.M. Maggs, R.K. Sitaraman, Algorithmic nuggets in content delivery, SIGCOMM Comput. Commun. Rev. 45 (3) (2015) 52–66, doi:10.1145/2805789.2805800.
[2] S. Triukose, Z. Al-Qudah, M. Rabinovich, Content delivery networks: Protection or threat?, in: M. Backes, P. Ning (Eds.) Computer Security ESORICS 2009, Lecture Notes in Computer Science, vol. 5789, Springer Berlin Heidelberg, 2009, pp. 371–389, doi:10.1007/978-3-642-04444-1_23.
[3] D. Fifield, G. Nakibly, D. Boneh, Oss: using online scanning services for censorship circumvention, in: E. De Cristofaro, M. Wright (Eds.), Privacy Enhancing Technologies, Lecture Notes in Computer Science, vol. 7981, Springer Berlin Heidelberg, 2013, pp. 185–204, doi:10.1007/978-3-642-39077-7_10.
[4] K. Xu, P. Butler, S. Saha, D. Yao, Dns for massive-scale command and control, Dependable Secure Comput. IEEE Trans. 10 (3) (2013) 143–153, doi:10.1109/TDSC.2013.10.
[5] K. Singh, A. Srivastava, J. Giffin, W. Lee, Evaluating email's feasibility for botnet command and control, in: Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on, 2008, pp. 376–385, doi:10.1109/DSN.2008.4630106.
[6] B. Carbunar, R. Potharaju, M. Pearce, V. Vasudevan, M. Needham, A framework for network aware caching for video on demand systems, ACM Trans. Multimedia Comput. Commun. Appl. 9 (4) (2013) 30:1–30:22, doi:10.1145/2501643.2501652.
[7] F.A.P petitcolas, R.J Anderson, M.G Kuhn, Information hiding-a survey, Proc. IEEE 87 (7) (1999) 1062–1078.
[8] Locations and ip address ranges of CloudFront edge servers, (http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/LocationsOfEdgeServers.html). Accessed: 2015-12-11.
[9] R. Roth, Introduction to Coding Theory, Cambridge University Press, New York, NY, USA, 2006.
[10] L. Bowyer, Firewall bypass via protocol steganography, Netw. Penetration (2002).
[11] A.-J. Su, A. Kuzmanovic, Thinning akamai, in: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement, ACM, 2008, pp. 29–42.
[12] J. Chen, J. Jiang, X. Zheng, H. Duan, J. Liang, K. Li, T. Wan, V. Paxson, Forwarding-loop attacks in content delivery networks, NDSS, 2016.
[13] L. Wang, K. Park, R. Pang, V.S. Pai, L.L. Peterson, Reliability and security in the codeen content distribution network., in: USENIX Annual Technical Conference, General Track, 2004, pp. 171–184.
[14] K.-W. Lee, S. Chari, A. Shaikh, S. Sahu, P.-C. Cheng, Improving the resilience of content distribution networks to large scale distributed denial of service attacks, Comput. Netw. 51 (10) (2007) 2753–2770. http://dx.doi.org/10.1016/j.comnet.2006.11.025.
[15] J. Jung, B. Krishnamurthy, M. Rabinovich, Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites, in: Proceedings of the 11th international conference on World Wide Web, ACM, 2002, pp. 293–304.
[16] A. Dyatlov, S. Castro, Exploitation of data streams authorized by a network access control system for arbitrary data transfers: tunneling and covert channels over the http protocol. Grayworld, USA, 2003 http://grayworld.net/projects/papers/html/covert_paper.html.
[17] Z. Kwecka, Application layer covert channel analysis and detection. Ph.D. thesis, Edinburgh Napier University, 2006.
[18] M. Van Horenbeeck, Deception on the network: thinking differently about covert channels, in: Proceedings of the 7th Australian Information Warfare and Security Conference, 2006, pp. 174–184.
[19] K. Born, PSUDP: a passive approach to network-wide covert communication, Black Hat USA (2010). https://www.blackhat.com/html/bh-us-10/bh-us-10-archives.html#Born.
[20] K. Born, Browser-based covert data exfiltration, 2010. arXiv preprint arXiv:1004.4357
[21] A. Revelli, N. Leidecker, Playing with heyoka: spoofed tunnels, undetectable data exfiltration and more fun with dns packets, in: ShakaCon Security Conference, 2009.
[22] K. Xu, P. Butler, S. Saha, D. Yao, Dns for massive-scale command and control, Dependable Secure Comput. IEEE Trans. 10 (3) (2013) 143–153.