「マルチメディア，分散，協調とモバイル
(DICOMO2016)シンポジウム」 平成28年7月

WebBrowserBotnet における HTML ベースの DDoS 攻撃

上久保遼　齊藤泰一

概要： 従来の DDoS 攻撃の手法はネットワーク層で行われることが主であったが、アプリケーション層で行われる DDoS 攻撃が増加してきている。2015 年 4 月には広告ネットワークを通して悪意のあるスクリプトを配信することによってボットネットを形成し、DDoS 攻撃が行われたという事例があった。今回は JavaScript を使用しない場合のにおける HTML ベースの DDoS 攻撃の方法を新たに提案し、それについて評価を行った。

HTML based DDoS attacking of WebBrowserBotnet

RYO KAMIKUBO　TAIICHI SAITO

1. はじめに

情報通信ネットワークが普及したころから DoS 攻撃と呼ばれる攻撃対象であるコンピュータのリソースを意図的に消費するなどしてサービスを妨害する攻撃があり、特にその攻撃が複数の IP アドレスから行われる場合を DDoS 攻撃と呼ぶ。これまでの DDoS 攻撃はマルウェアに感染して bot と化したコンピュータ群(以下ボットネットと言う)を制御するサーバを用意し、そのサーバからボットネットにコマンドを出して行うものが多かった。しかしブラウザベースの DDoS 攻撃が去年確認され新たな脅威となっている。ブラウザベースの DDoS 攻撃は確認される以前から理論上可能なのではないかと考えられていた。しかし攻撃が確認されなかったのは攻撃コードを拡散させることが課題となっていたからである。2015 年 4 月に確認された DDoS 攻撃ではスマートフォンなどに配信される広告の広告ネットワークを悪用し効果的に攻撃コードを拡散したのではないかと考えられている。

2015 年 4 月、アメリカの IT 企業 GitHub への史上最大の DDoS 攻撃があった。この攻撃では秒間 27 万以上のリクエストを発生させる大規模な攻撃であった。トロント大学 CitizenLab1が発表した報告書では GreatCannon と呼ばれる新しい攻撃システムを用いられたと発表した。このシステムを利用し中国にサーバがある広告ネットワークに攻撃コードを挿入することによって web サイトを閲覧したユーザが無意識の参加者としてリクエストを発生させ大規模な DDoS 攻撃につながったのではないかと予想されている。

2. 広告ネットワークを利用した DDoS 攻撃

2015 年 4 月に起きた大規模な DDoS 攻撃について、米セキュリティ企業の CloudFlare の調査によると、web サイトを閲覧しているユーザのブラウザ上で表示される広告ネットワークを経由して攻撃コードを含む広告が配信され、大量のリクエストを発生させていたという報告がある。この DDoS 攻撃ではリクエストの 99.8%は中国からであり、72%はスマートフォンなどのモバイル端末が発信源であった。

3. GreatCannon

中国ではネットの情報を検閲する GreatFirewall(金盾)というシステムが存在していることはよく知られているが、これとともに配置された別の攻撃システム「GreatCannon」の存在がトロント大学 CitizenLab らの調査によって判明している。このシステムでは特定の IP アドレスに対するトラフィックを乗っ取りその内容を自由に置き換える中間者攻撃を行う能力がある。また中国では暗号化による保護を採用していない。報告書によると GreatCannon は国内のアクセスはもちろん、中国にあるサーバと通信を行う国外からのアクセスも監視下におかれると明かした。

4. 先行研究

G.Pellegrino らの先行研究では、GreatCannon で使用された HTML ベースの攻撃は XMLHttpRequest、WebSocket、Server-Sent-Events を利用していると予想し、それぞれの攻撃性を比較している。

1 参考文献 1 参照

しかし XMLHttpRequest はブラウザ上で JavaScript を使用するため、JavaScript がオフになっている場合クライアントが対応していても攻撃不可である。そこで JavaScript が動作しない環境でも攻撃が可能な HTML ベースの DoS 攻撃を考えることにより、踏み台となるクライアントが増え攻撃能力が高くなるのではないかと考えた。

5. 研究内容

5-1.今回利用する HTML の機能

新たに提案する HTML ベースの攻撃で使用する機能は multipart/x-mixed-replace と metarefresh である。いずれも web ページを動的に表示するダイナミックドキュメントの機能である。

・Multipart/x-mixed-replace の機能

serverpush 手法の 1 つであり、サーバが任意のタイミングで複数の文章を返し、紙芝居的にレンダリングを切り替えさせるものである。基本的な使い方は下記のようになる。

```
Content-type:multipart/x-mixed-replace;boundary=End
  --End
  Content-type: image/gif
  Image #1
  --End
  Content-type: image/gif
  Image #2
  --End
```

boundary で決められた境界文字列を境界として 2 つのデータブロックに分ける。この境界文字列を続けることによっていくつもの送信したいデータブロックを作ることが出来る。

・Metarefresh の機能

ページをリダイレクトするタグであり、その基本的な使い方は下記のとおりである。

```
<meta name="refresh"
content="5;URL=http://example.com">
```

content でリダイレクトするまでの待機秒数を指定し、URL でリダイレクト先の URL を指定する。ここに記載がない場合同じページにリダイレクトする。

5-2. 攻撃方法

multipart/x-mixed-replace の場合実際の攻撃部分は以下のようなサーバプログラムで生成される。

```
for ($i = 1; $i < 20; $i++) {
echo 'Content-Type: text/html; charset=utf-8;
<p>test <b>'. $i .'</b></p>';
for ($j = 1; $j< 100; $j++) {
```

実際には img タグの他 iframe タグおよび存在するデータしないデータで検証した。

metarefresh の場合実際の攻撃部分は以下のようなサーバプログラムで生成される。

```
for ($i = 1; $i< 100; $i++) {
for ($j = 1; $j< 100; $j++) {
print '<meta http-equiv="refresh" content=0.1>';
}
```

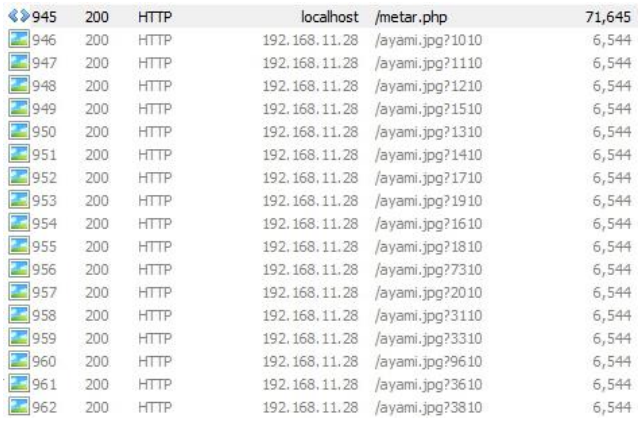
metarefresh も同様に存在するデータ及び存在しないデータで検証をした。

5-3.攻撃の効率化

攻撃先の URL をそのまま記述し実行するとブラウザによっては 2 回目以降のリクエストに If-Modified-Since ヘッダを付加して送信する場合がある。これが付加している場合リクエスト先のデータの更新がない限り対象サーバはデータの送信を行わない。そこで URL の後に文字クエリを付けことによって異なる URL と解釈させ If-Modified-Since にならないようにしている。下記の図では If-Moidified-Since の場合の通信の状態とそうでない場合の状態を Fiddler で観察した。図 1 では URL が一緒(/ayami.jpg?)であり、2 回目以降の HTTP ステータスコードが 304(If-Modified-Since)である。図 2 では URL の末尾がそれぞれ異なっており(/ayami.jpg?****)、HTTP ステータスコードはすべて 200 番であることがわかる。

3	200	HTTP	localhost	/metar.php	71,258
4	200	HTTP	192.168.11.28	/ayami.jpg?	6,544
5	200	HTTP	localhost	/favicon.ico	7,782
6	200	HTTP	localhost	/metar.php	71,258
7	304	HTTP	192.168.11.28	/ayami.jpg?	0
8	304	HTTP	pki.google.com	/GIAG2.crl	0
9	200	HTTP	localhost	/metar.php	71,258
10	304	HTTP	192.168.11.28	/ayami.jpg?	0
11	200	HTTP	localhost	/metar.php	71,258
12	304	HTTP	192.168.11.28	/ayami.jpg?	0
13	200	HTTP	localhost	/metar.php	71,258
14	304	HTTP	192.168.11.28	/ayami.jpg?	0
15	200	HTTP	localhost	/metar.php	71,258
16	304	HTTP	192.168.11.28	/ayami.jpg?	0
17	200	HTTP	localhost	/metar.php	71,258
18	304	HTTP	192.168.11.28	/ayami.jpg?	0
19	200	HTTP	localhost	/metar.php	71,258
20	304	HTTP	192.168.11.28	/ayami.jpg?	0

図 1　URL 末尾に文字クエリを付けない場合



945	200	HTTP	localhost	/metar.php	71,645
946	200	HTTP	192.168.11.28	/ayami.jpg?1010	6,544
947	200	HTTP	192.168.11.28	/ayami.jpg?1110	6,544
948	200	HTTP	192.168.11.28	/ayami.jpg?1210	6,544
949	200	HTTP	192.168.11.28	/ayami.jpg?1510	6,544
950	200	HTTP	192.168.11.28	/ayami.jpg?1310	6,544
951	200	HTTP	192.168.11.28	/ayami.jpg?1410	6,544
952	200	HTTP	192.168.11.28	/ayami.jpg?1710	6,544
953	200	HTTP	192.168.11.28	/ayami.jpg?1910	6,544
954	200	HTTP	192.168.11.28	/ayami.jpg?1610	6,544
955	200	HTTP	192.168.11.28	/ayami.jpg?1810	6,544
956	200	HTTP	192.168.11.28	/ayami.jpg?7310	6,544
957	200	HTTP	192.168.11.28	/ayami.jpg?2010	6,544
958	200	HTTP	192.168.11.28	/ayami.jpg?3110	6,544
959	200	HTTP	192.168.11.28	/ayami.jpg?3310	6,544
960	200	HTTP	192.168.11.28	/ayami.jpg?9610	6,544
961	200	HTTP	192.168.11.28	/ayami.jpg?3610	6,544
962	200	HTTP	192.168.11.28	/ayami.jpg?3810	6,544

図 2　URL 末尾に文字クエリを付けた場合

5-4.実験環境

今回の実験環境を以下の表に示す。

表 1　実験環境

	クライアント(攻撃側)	サーバ(標的)
OS	windows8.1	Ubuntu15.10
CPU	intel corei3-4160 3.6GHz*2	intel corei3-4130 3.4GHz*4
RAM	8GB	8GB

・サーバソフトとして apache2 を使用
・C&C サーバはクライアント上に仮想で設置

6. 実験結果

実際の攻撃結果を以下に示す。multipart/x-mixed-replace は m/x、metarefresh は meta、img タグは img、iframe タグは if rame、存在するファイルは存、存在しないファイルは無と表記する。また、単位は request/s である。

表 2　実験結果

	FireFox		Chrome	
req/s	平均	最大	平均	最大
m/x,img,無	143.0	174.6	141.2	146.3
m/x,img,存	132.6	173.0	-	-
m/x,iframe,無	55.98	168.2	41.67	70.50
m/x,iframe,存	62.76	168.2	42.58	73.50
meta,img,無	75.55	171.0	92.61	144.0
meta,img,存	72.18	86.60	98.60	111.0
XHR	202.4	211.0	45.76	75.00

7. 評価

ブラウザによって差異があるが先行研究の XMLHttpReq uest と比較しても multipart/x-mixed-replace では遜色のない程度にはリクエストが発生していることがわかる。また m ultipart/x-mixed-replace を使用した場合はブラウザにあまり依存せずリクエストを発生させるという結果を得た。しかし Chrome では multipart の動作が正常ではなく表示がリフレッシュされない為、長い時間動作させているとクライアント側のリソースを大幅に消費してしまうという問題点がある。

8. 今後の課題

攻撃効率がクライアント依存である為、使用する PC や端末に合わせてリクエスト数やリクエストを送る間隔を考える必要がある場合がないか試す。今後クライアントを増やし実際に攻撃を行った場合の挙動を見る必要がある。

今回実験を行っていない WebWorker や WebSocket を用いた場合の攻撃能力・手軽さなどを調査する必要がある。また単一の方法ではなく例えば XMLHttpRequest と meta-Refresh を使用した場合などの組み合わせではどうなるかなどを実験し最適な方法を考えていく。

サービス障害以外にも AWS のような従量課金制の web サーバにおいて不当に使用料を上げるようなことが可能であるか、また効率的であるかどうかなどを検証していきたい。

参考文献

[1]　G.Pellegrino, C.Possow, F.J.Ryba, T. C.Schmidt, M.Wahlisch Chasing out the Great Cannon?On Browser-Based DDoS Attacks and Economics

[2]　Yokochin のハイパーテキストマニュアル・第 4 章フォームと CGI・ダイナミックドキュメント機能,入手先 http://www.yokochin.com/manual/cgi/dynamicd.html (参照 2016-02-04).

[3]　citizenlab・China's Great Cannon、入手先 https://citizenlab.org/2015/04/chinas-great-cannon/ (参照 2016-02-02)

[4]　CGI Programming on the World Wide Web・6.6 Animation、入手先 http://www.oreilly.com/openbook/cgi/ch06_06.html (参照 2016-02-02)

[5]　MOZILLA DEVELOPER NETWORK・XMLHttpRequest の利用 入手先 https://developer.mozilla.org/ja/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest (参照 2016-01-30)

[6]　CLOUDFLARE・Mobile Ad Networks as DDoS Vectors: A Case Study, 入手先 https://blog.cloudflare.com/mobile-ad-networks-as-ddos-vectors/ (参照 2016-03-08)

[7]　Lavakumar Kuppan Attacking with HTML5 入手先 https://media.blackhat.com/bh-ad-10/Kuppan/Blackhat-AD-2010-Kuppan-Attacking-with-HTML5-wp.pdf (参照 2016-01-30)