Blueinfy
AppSecLabs

Discovery and Fuzzing for SQL injections with Web 2.0 Applications

Abstract

Web 2.0 application assessment is becoming increasingly challenging due to their behavior and implementation of the components. It is imperative to identify hidden Web 2.0 resources and fuzz them to detect SQL injection possibilities. This paper describes some techniques and approaches to perform effective assessment on Web 2.0 applications on the basis of our recent experience and cases which were analyzed on the field.

Detecting Web 2.0 calls

First step is to identify few hidden calls and resources from Web 2.0 applications to perform fuzzing and response analysis on it. Web 2.0 applications are running on Ajax or Flex/Flash platform in RIA space. Here are few techniques by which one can identify these resources and reconstruct the possible HTTP(S) requests.

a.) JavaScript Analysis

Ajax calls are hidden in Java Script and one needs to analyze Java Script thoroughly to identify these entry points into the system. To discover these points we did our analysis by parsing Java Scripts using JS parser or regex, it is one of the ways to identify key calls and dissect possible URLs along with invoking mechanism.

Examples,

Tracking XHR call and fetching all possible open calls

```
http.open('POST', '/json/jservice.ashx', true);
```

If application is using some toolkit then one needs to identify respective calls for it like over here we have it for prototype.

```
var myAjax = new Ajax.Updater(target, ‘json/jservice.ashx’, {method: 'get'});
```

By this mechanism either by parsing JavaScript thoroughly or using regex we can get all possible calls and that helps in reducing analysis surface.

b.) HTTP traffic analysis

Once analysis surface is restricted to few resources one can open those pages into browser and analyze HTTP traffic. It is also possible to spider/crawl these pages in automated fashion by driving browsers like IE/Firefox using their respective drivers like

Watir (for IE) or Chickenfoot (for FireFox). At the same time it is interesting to use plugins like Firebug to segregate XHR traffic from regular HTTP traffic. This can help in mapping actual JavaScript to HTTP traffic as shown below.

```
POST http://192.168.1.56/json/jservice.ashx HTTP/1.1
Host: 192.168.1.56
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.9.0.1)
Gecko/2008070208 Firefox/3.0.1 Paros/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Content-Type: text/plain; charset=utf-8
X-JSON-RPC: getProduct
Referer: http://192.168.1.56/
Content-Length: 51
Pragma: no-cache
Cache-Control: no-cache

{"id":4,"method":"getProduct","params":{ "id" : 4}}
```

In above case we are able to identify JSON request hitting to JSON-RPC running on backend server. We can use regex patterns to identify various different structures of HTTP request which are not falling into traditional name-value pair category. These structures clearly define Web 2.0 calls in the form of XML-RPC, SOAP, JSON-RPC, JS-Array, JS-Object etc.

c.) RIA call detections

We used JavaScript parsing to reduce analysis surface for Web 2.0 applications in first section, we can use same technique to identify RIA calls running over Flex based Adobe applications.

Example,

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swfla
sh.cab#version=6,0,0,0" WIDTH="645" HEIGHT="660" id="myApp"
ALIGN="Center">
<PARAM NAME=movie VALUE="products.swf"> <PARAM NAME=quality VALUE=high>
<PARAM NAME=bgcolor VALUE="blue"> <EMBED src="products.swf"
quality=high bgcolor=#333399 WIDTH="645" HEIGHT="660" NAME="myApp"
ALIGN="Center" TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
```

In above page we can identify hidden source of SWF file and we can restrict our analysis on it. Once again we can either load page into browser window and invoke SWF driven calls or decompile swf component and dissect information. Depending on the version of flash, we can perform decompiling tactic. It is also possible to capture remoting traffic

using proxy like Charles that provides AMF streams or using tracers for Flash applications. Hence, by adopting any of the above approaches one can discover hidden Web 2.0 calls running over Java Script or RIA. These calls can be sending JSON or XML traffic back to application.

Fuzzing Web 2.0 structures

Next, we need to fuzz Web 2.0 structures for various different values and try to figure out the possible vulnerabilities. In work we tried to fuzz for SQL injections and analyze responses. One can use any HTTP based fuzzer to send different values but one need to make sure it keeps the JSON and XML structures as it is else will not get processed at server end. You can also use wsScanner (http://blueinfy.com/tools.html) to do Web 2.0 fuzzing. It has capabilities to fuzz these structures and streams.

Example,

Here is a request which broke the JSON services on other side.

```
POST http://192.168.1.56/json/jservice.ashx HTTP/1.1
Host: 192.168.1.56
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.9.0.1)
Gecko/2008070208 Firefox/3.0.1 Paros/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Content-Type: text/plain; charset=utf-8
X-JSON-RPC: getProduct
Referer: http://192.168.1.56/
Content-length: 54
Pragma: no-cache
Cache-Control: no-cache

{"id":2,"method":"getProduct","params":{ "id" : "--"}}
```

In this case we injected hyphens (--) in the parameter value for id in JSON. While doing automated fuzzing one needs to identify these points and values should be injected in JSON or any other stream. Another approach is to identify it by seeing its structure or relevant JavaScript code.

We got following response

```
HTTP/1.1 200 OK
Date: Tue, 02 Sep 2008 10:05:30 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
```

```
Pragma: no-cache
Expires: -1
Content-Type: text/plain; charset=utf-8
Content-Length: 153

{"id":2,"error":{"name":"JSONRPCError","message":"Incorrect syntax near
'='.","errors":[{"name":"SqlException","message":"Incorrect syntax near '='."}]}}
```

Definite SQL injection point or vulnerability is identified over here. This is one of the places where an attacker can try different pay loads to gain unauthorized access. It is interesting to see that it is not 500 but 200 in which JSON response will come back and no point in looking for 500 signature blocks for vulnerability detection in many Web 2.0 centric cases.

Following is a list of fuzzing points for Web 2.0 application to discover SQL injections:

- Public functions of SOAP based Web Services
- XML-RPC services' entry points
- AMF remoting calls
- JSON parameters along with RPC
- Other type of JS Objects like Array or customized object
- Customized APIs and library structures

Conclusion

Discovery phase for Web 2.0 applications need better approach compared to traditional. One needs to dig through several components and scripts to identify backend resources. During our recent work against Web 2.0 applications we were able to identify few SQL injections over Web Services and JSON streams. As new applications are moving towards Web 2.0 space these sort techniques are required to be deployed and some innovative methodologies are needed as well along with tools.

About Blueinfy

Blueinfy specializes in application security. We provide services to evaluate and improve the overall security posture of web applications and websites deployed world wide by products, consulting and training services. We continually strive to ensure complete customer satisfaction with respect to the security of their application assets, and to achieve this through state-of-the-art know-how built by enhancing methodologies, evolving tools and researching technologies.

Web: http://www.blueinfy.com Email: contact@blueinfy.com

******This paper is authored by Blueinfy's research and consulting team.******