Implementation Vulnerability Associated with OAuth 2.0 A Case Study on Dropbox

Bruce Wu, Tung Nguyen and Mohammad Husain
Department of Computer Science
California State Polytechnic University, Pomona, CA, USA

{brucewu,tungtnguyen, mihusain}@cpp.edu

Abstract— **Dropbox is a cloud based file storage service used by more than 200 million users. Its ability to seamlessly provide cloud storage with minimal user complexity is the key for its wide spread popularity. Despite of its high usability, Dropbox has been recently criticized for loose ends in security. Security and usability is not always mutually exclusive, and we believe there is still a lot of room to improve Dropbox's security without affecting the unique user experience. In this paper, we present a RAM analysis based method to extract the key security token for account access. In addition, we describe a new technique to bypass authentication and gain unauthorized access to Dropbox accounts by using the new *tray_login* feature on the most current Dropbox client (v2.4.x). Through these exploits, we demonstrate that most of these security issues are at the level of implementation, rather than design. Finally, we describe potential resolutions that can improve Dropbox's security without affecting its high usability.**

Keywords—Dropbox; RAM Analysis; Security token; Unauthorized Access; OAuth 2.0

I. INTRODUCTION

Dropbox is a cloud based storage service used by more than 200 million users. Dropbox also serves four million businesses and 97% of the Fortune 500 companies. As one of the most popular cloud storage services, Dropbox's security features have been examined extensively [1]. In this paper, we attempted to extend those concepts and identify similar vulnerabilities in the most current version of Dropbox client (v2.4.x).

Most attacks described in this paper take advantage of the *Bearer Token* described in *OAuth* standard [2]. *OAuth* provides a method for clients to access server resources on behalf of a resource owner, without sharing resource owner's credential. *OAuth* is needed in situations such as granting access right to third party applications, where handing off user credential is undesirable. A *Bearer Token* is then used in place of user credential for authentication. To prevent misuse, these tokens need to be protected from disclosure in storage and in transport.

Dropbox currently uses a *Bearer Token* known as the *host_id,* to replace all the necessary credential for account access. The token is generated when user registers a device with their Dropbox account. Current attacks that target the *host_id* involve decryption of the encrypted configuration database, or Reflective-DLL injections. In Section 2, we cover these existing exploits in details. We then propose a novel approach to obtain the tokens through RAM Analysis in Section 3. Although Dropbox released a patch on Sept. 27, 2013 to fix a known security issue of using desktop client's *tray_login* function to bypass authentication, we demonstrate a new attack exploiting the implementation of this most recent patch in Section 4.

We hope our work inspires the community to refine the implementation strategy of current *OAuth* standard, and conduct research in securing high usability systems and services. A demonstration of our exploits is available at [6]. Also, we have already notified Dropbox security team about these exploits.

II. RELATED WORK

In this section, we cover existing work related to security analysis of Dropbox and previously published methods to obtain the key security tokens (*Bearer Token*).

Kholia and Wegrzyn [1] analyzed Dropbox versions from 1.1.x to 2.4.x. By reverse engineering Dropbox's byte code, the authors were able to reveal the internal API and write a portable open-source Dropbox client. The internal API revealed that a key security token, *host_id* is used to replace all the necessary credential to access Dropbox's website. The token is generated when user registers a device with Dropbox account. In this registration process, the end-user device is associated with the token which is used for all future authentication operations. In other words, Dropbox client doesn't store or use user credentials once it has been linked to the user account. *Host_id* is not affected by password changes and it is stored locally on the end-user device.

In older versions of Dropbox (< 1.2.48), this *host_id* was stored locally in clear-text in a SQLite database named *config.db*. By simply copying this SQLite database file to another machine, it was possible to gain access to the target user's data. This attack vector is described in detail in [3].

From version 1.2.48 onward, Dropbox encrypted the *config.db* database. The secrets used in deriving the database encryption key are stored on the end-user device (local storage of such secrets can't be avoided since Dropbox client depends on them to work). For a Windows client, DPAPI encryption [5] is used to protect the secrets, while a Linux client used a custom obfuscator designed by Dropbox. Kholia and Wegrzyn mentioned tools such as "*dbx-keygen-linux*" [4] that are capable of recovering the database encryption key, but since our goal was to create a novel method of obtaining the *host_id*, we did not verify the capability of these tools during our experiment.

Another method described in the same paper [1] was to extract *host_id* from the Dropbox client by using Reflective DLL injection or LD_PRELOAD. The code snippet in Figure 1 shows how this can be achieved.

```
# 1. Inject code into Dropbox.
# 2. Locate PyRun_SimpleString using dlsym
#    from within the Dropbox process
# 3. Feed the following code to the located
#    PyRun_SimpleString

import gc

objs = gc.get_objects()
for obj in objs:
    if hasattr(obj, "host_id"):
        print obj.host_id
    if hasattr(obj, "host_int"):
        print obj.host_int
```

Figure 1: Extracting *host_id* from Dropbox as described in [1].

The code in Figure 1 examines the garbage collection objects within Dropbox client's running process in order to obtain the decrypted *host_id* and *host_int*. In our experiment described in Section 3, we further discovered that examining garbage collection objects is not necessary, because *host_id* is unprotected and stored in plain text within runtime memory.

Dropbox client has a feature for users to login to Dropbox's website without providing any credentials. By selecting *Launch Dropbox Website* from the Dropbox tray icon, a custom URL is generated by hashing current time, *host_id*, and a fixed secret together as shown in Figure 2. Accessing the URL output takes one to the Dropbox account of the target user. This access URL also bypasses Dropbox's two-factor authentication. This implies that only the *host_id* is needed to gain access to a target's data stored in Dropbox.

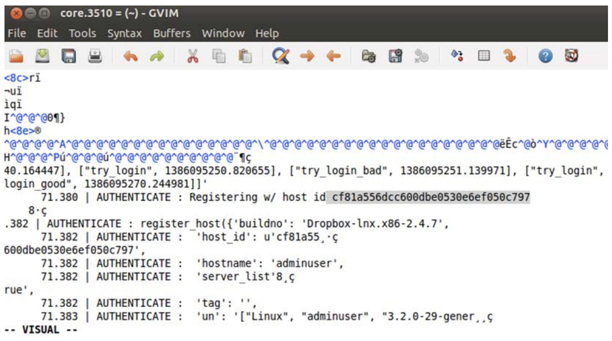
Since version 2.4.0 and onward, the above mentioned access URL based attack no longer works. It is due to the fact that the new mechanism Dropbox implemented rely on heavier obfuscation and random nonce (received from the server) to generate those auto-login URLs. However, in our experiment, we examined this new mechanism, and have successfully devised a method to exploit it and gain access to target's Dropbox account. A detail of this novel exploit is described in Section 4.

```
import hashlib
import time

host_id = <UNKNOWN>
host_int = <ASK SERVER>

now = int(time.time())

fixed_secret = 'sKeevie4jeeVie9bEen5baRFin9'

h = hashlib.sha1('%s%s%d'% (fixed_secret,
    host_id, now)).hexdigest()

url = ("https://www.dropbox.com/tray_login?"
       "i=%d&t=%d&v=%s&url=home&cl=en" %
       (host_int, now, h))
```

Figure 2: Custom URL, *host_id* and a fixed secret

III. RAM ANALYSIS

Inspired by the Reflective DLL injection attack described by [1], we decided to perform a RAM analysis in order to gain insight of the working status of a Dropbox client. We used the GNU Debugger (GDB) to generate a core dump of a running Dropbox process and we were able to find the key security token ("*host_id*") in plain text as shown in Figure 3.



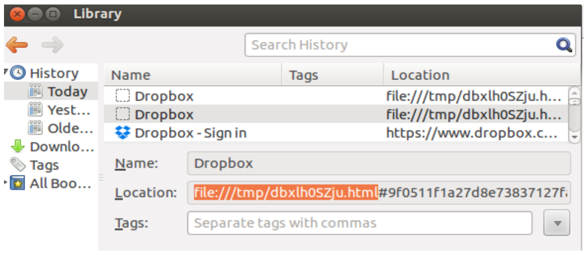
```
<8c>rï
¬uï
ìqï
I^@^@^@0¶}
h<8e>®
^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^\^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ëÊc^@ò^Y^@^@^@^@^
H^@^@^@^Pú^@^@^@ú^@^@^@^@^@^@^@^@^@^@^@¨¶ç
40.164447], ["try_login", 1386095250.820655], ["try_login_bad", 1386095251.139971], ["try_login",
login_good", 1386095270.244981]]'
      71.380 | AUTHENTICATE : Registering w/ host id cf81a556dcc600dbe0530e6ef050c797
    8·ç
.382 | AUTHENTICATE : register_host({'buildno': 'Dropbox-lnx.x86-2.4.7',
      71.382 | AUTHENTICATE :  'host_id': u'cf81a55¸·ç
600dbe0530e6ef050c797',
      71.382 | AUTHENTICATE :  'hostname': 'adminuser',
      71.382 | AUTHENTICATE :  'server_list'8¸ç
rue',
      71.382 | AUTHENTICATE :  'tag': '',
      71.383 | AUTHENTICATE :  'un': '["Linux", "adminuser", "3.2.0-29-gener¸¸ç
-- VISUAL --
```

Figure 3: A core dump of running Dropbox process shows the "*host_id*" in plain text

IV. TRAY LOGIN EXPLOIT

Since version 2.4.0 and onward, Dropbox clients no longer use a simple hashed URL to access Dropbox's website. Instead, a HTML with the name *dbx****.html is generated. The location of this file can be found by examining browser history as shown in Figure 4. In Linux, the file is stored under */tmp/* folder. This html file included three key variables to access Dropbox's website as shown in Figure 5: "c", "b", and "value" (which is *host_int* in the older version).

Variable "c" and "b" are two different SHA-256 hashed values generated by the Dropbox client. The exact mechanism of how these two variables are generated is unknown. Variable "c" is in plain text inside the HTML file, whereas variable "b" is in the URL link as highlighted in Figure 6.



Library

Search History

History | Today | Yest… | Olde… | Downlo… | Tags | All Boo…

Name	Tags	Location
Dropbox		file:///tmp/dbxlh0SZju.h…
Dropbox		file:///tmp/dbxlh0SZju.h…
Dropbox - Sign in		https://www.dropbox.c…

Name: Dropbox

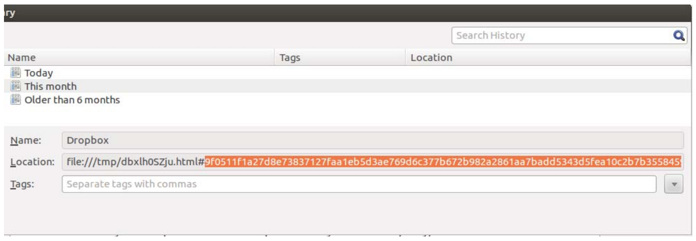
Location: file:///tmp/dbxlh0SZju.html#9f0511f1a27d8e73837127f

Tags: Separate tags with commas

Figure 4: File location on Linux

```
---------------------------------------------------------------------------------------
                                                      .
       .
       .
           <script type="text/javascript">
               (function () {
                   "use strict";
                   function go() {
                       var c =
'c56995950efbee29c84b37866bf65b467cf1820dd86870f931c4ba42236a6b1dd0439567da7fc334e3
e963';
                       var b = window.location.hash.substr(1);
                       window.location.hash = '';
                       var a = '';
                       if (c.length !== b.length || (c.length % 2) !== 0) {
                           a = '**' + c; // XXX - See note2 above
       .
       .
       .
       .
       <body>
       <form name="desktop_login" action="https://www.dropbox.com/desktop_login"
method="post">
           <input type="hidden" name="i" value="871964756">
           <input id='n' type="hidden" name="n" value="">
           <input type="hidden" name="u" value="home">
           <input type="hidden" name="c" value="">
           <noscript>
               <div class="center">
                   <meta id="meta-refresh" http-equiv="refresh"
content="2;URL='https://www.dropbox.com/login?cont=/home'">
                   <p>Dropbox cannot log you in automatically because your browser
has scripts disabled.</p>
                   <a href="https://www.dropbox.com/login?cont=/home">Sign in
manually</a>
               </div>
           </noscript>
       </form>
       </body>
       </html>
 ----------------------------------------------------------------------------
```

Figure 5: "c", "b" and "value" in the HTML file



Search History

Name	Tags	Location
Today		
This month		
Older than 6 months		

Name: Dropbox

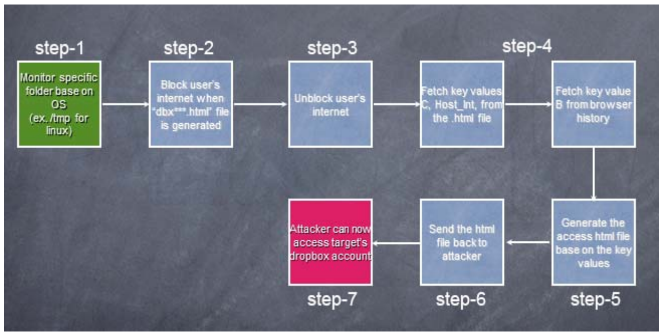
Location: file:///tmp/dbxlh0SZju.html#9f0511f1a27d8e73837127faa1eb5d3ae769d6c377b672b982a2861aa7badd5343d5fea10c2b7b355845

Tags: Separate tags with commas

Figure 6: A part of the three security tokens

In our experiment, we noticed that Dropbox clients remove this HTML from local device one minute after it is generated. The short life is designed to reduce exposure vulnerability. However, since all three value are stored unprotected, it is still very easy for automated script to transmit these variables to attacker before the file is deleted.

Variable "b" and "c" are used to generate the access token "a". This token can be used only one time. Therefore, it is crucial for attacker to prevent the token from transmitting to Dropbox's server. An example attack flow is described below in Figure 7:



step-1
Monitor specific folder base on OS (ex. /tmp for linux)
step-2
Block user's internet when "dbx***.html" file is generated
step-3
Unblock user's internet
step-4
Fetch key values C, Host_Int, from the .html file
Fetch key value B from browser history
step-5
Generate the access html file base on the key values
step-6
Send the html file back to attacker
step-7
Attacker can now access target's dropbox account

Figure 7: Flow of the *"tray login"* exploit

Step 1: Monitor the */tmp/* folder to find out about the HTML file generation,

Step 2: Block user's Internet when target HTML file is created to stop transmitting the secret values to Dropbox server,

Step 3: Unblock user's Internet,

Step 4: Fetch key values,

Step 5: Generate the access HTML file base on the key values,

Step 6: Send the access HTML file back to attacker,

Step 7: Attacker can now access target's Dropbox account.

V. SOLUTION SKETCH

To avoid future exploitations, we believe that these access tokens should become device specific. One implementation would be to include certain device specific information into the generation of hashed variable "b" or "c", then verify client's identity base on these information when the token is used to access the account. Also, we suggest to add the time of token creation into the hashed variable. This way the server would be able to reject access to tokens that are exposed and possibly compromised.

We believe the use of a single easily accessible crucial information, either the *host_id* used in previous Dropbox version, or the auto-generated *dblxxxxxx*.html file used currently, is not a good choice and pose a significant threat to the account security. To avoid future exploitations, an improvement in the implementation can be made as described below:

Step 1 : Dropbox client app sends a secure HTTP request to the Dropbox server, including currently used user identification keys (b, c, and value) plus a randomly generated *key* string,

Step 2 : Dropbox server processes the request, if user request is validated, then a URL link to user's page is encrypted with the randomly generated *key* included in the client request, as a response to client, Step 3 : Dropbox client app decrypts the response with the *key* string, then open the URL in browser.

This approach eliminates the generation of the *dblxxxxx*.html file, which made the timing of blocking user's internet connection almost impossible. It also prevents attacker from obtaining sensitive information such as a user's identification values or possible insight to the verification mechanism.

VI. Conclusion and future work

Tray login is a convenient feature that we enjoy to use as a Dropbox user, but as we have shown in this paper, it can be a security threat throughout Dropbox's client versions. Although, we believe that the use of a one-time only, short lived token is better then it's previous approach that relies solely on the obfuscation of the *host_id*, the new implementation further lacks any effort in protecting the key variables. Therefore, it is now easier for an attacker to hijack a Dropbox account compared to the previous versions. Unlike the *host_id*, which required attackers to have root privilege in order to launch the attack, the new implementation only requires an attacker to access the browser history. We hope our work inspires the community to refine the implementation strategy of the current *OAuth* standard, and conduct research in securing high usability systems and services.

References

[1] D. Kholia and P. Wegrzyn, "Looking inside the (Drop) box," in 7th USENIX Workshop on Offensive Technologies (WOOT), Washington, DC, USA, Aug. 2013, pp. 1–7.

[2] Jones, M., and D. Hardt. "RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage." Internet Engineering Task Force (IETF), Oct. 2012. Web. 13 Dec. 2013

[3] Newton, D. Dropboxauthentication:insecurebydesign.[Online] Available at: http://dereknewton.com/2011/04/dropbox-authentication-static-host-ids/, 2011.

[4] Ruff, N. and Ledoux, F. Encryption key extractor for dropbox dbx files. [Online] Available at: https://github.com/newsoft/dbx- keygen-linux.git, 2008.

[5] Windows Data Protection API. [Online] Available at: http://msdn.microsoft.com/en-us/library/ms995355.aspx

[6] Dropbox Exploit. [Online] Available at: http://youtu.be/ZcQv8oRHAow