

## Man in the Cloud (MITC) Attacks



## 1. Executive Summary

In this report, we demonstrate a new type of attack we call “Man in the Cloud” (MITC). These MITC attacks rely on common file synchronization services (such as GoogleDrive and Dropbox) as their infrastructure for command and control (C&C), data exfiltration, and remote access. Without using any exploits, we show how simple re-configuration of these services can turn them into a devastating attack tool that is not easily detected by common security measures.

MITC does not require any particular malicious code or exploit to be used in the initial “infection” stage, thus making it very difficult to avoid. Furthermore, the use of well-known synchronization protocols make it extremely difficult (if not impossible) to distinguish malicious traffic from normal traffic. Even if a compromise is suspected, the discovery and analysis of evidence will not be easy, as little indication of the compromise is left behind on the endpoint. In the MITC attacks, the attacker gets access to the victim’s account without compromising the victim’s user name or password. As we show in this report, this type of compromise is very hard to detect (contrary to attacks that involve compromising passwords). More importantly, recovery of the account from this type of compromise is not always feasible.

Since most organizations either allow their users to use file synchronization services, or even rely on these services as part of their business toolbox, we think that MITC attacks will become prevalent in the wild. As a result, we encourage enterprises to shift the focus of their security effort from preventing infections and endpoint protection to securing their business data and applications at the source.

## 2. Key Findings

- File synchronization services such as OneDrive, Dropbox, Google Drive, and Box can be easily turned into an infrastructure for endpoint compromise, providing a channel for C&C, data exfiltration, and remote access
- File synchronization services can be compromised without using plaintext credentials
- Recovery of the compromised file synchronization accounts is not always possible (i.e., the account has to be closed)
- Attacks based on the above architecture have been witnessed in the wild (for example, in [“The Inception Framework”](#) – analysis by Blue Coat)
- Endpoint and perimeter security measures are insufficient at detecting and mitigating this threat as no malicious code persists on the endpoint, and no abnormal outbound traffic channels are observed on the wire
- Consequently, we believe organizations must invest more effort in monitoring and protecting their enterprise data resources, either on premise or in the cloud. By detecting abusive access patterns to such resources, enterprises can protect against this next generation of breaches.

## 3. Background

In the past couple of years, we have witnessed massive hacks and data breaches. Big-name billion dollar enterprises were embarrassed by hackers, some losing millions of dollars in market capitalization and firing executives from key positions. In addition, credit cards, Personal Identifiable Information (PII), and intellectual property are stolen and traded on the underground network using the anonymous currency, Bitcoins.

As attacks are getting bigger, the “attack surface” available to attackers is also growing. Both individuals in their personal lives and entire business institutions are transitioning from local applications to cloud services. File synchronization services make data accessible to multiple users and devices – sometimes across different countries and even organizations.

Additionally, with increased usage of mobile devices, tablets, VPNs, Remote Desktop Access, and SaaS applications, more data is stored in the cloud, where it is potentially spreading outside defined boundaries. As the boundaries of data storage become less defined, so does perimeter security. When there is no clear perimeter to secure, how can an organization protect against data leaking out and malware coming in?

File synchronization services are a good example of the shift that both individuals and businesses experience. These services are probably familiar to many, and include services such as GoogleDrive, Dropbox, OneDrive, and Box. They allow an individual (potentially an employee of an enterprise) to keep synchronized copies of file repositories across multiple devices. This is achieved by connecting the devices of an individual to a central hub in the cloud using the same account. Any change to the file repository in one device is quickly synchronized with the hub, and then proliferates to the other devices. Many enterprises currently adopt one of these services as a tool for business data synchronization on employee devices. File synchronization services are usually configured to synchronize with a specific folder on enterprise machines (i.e., Windows and Linux based desktops and laptops). Files added locally to this folder or updated are uploaded to the cloud and files loaded to the cloud are downloaded to the specific local folder. We call this the *sync folder*.

Synchronization is performed using dedicated software installed on the end station. This is the synchronization software. The synchronization software is provided by the service provider and takes care of monitoring the local *sync folder* as well as polling the cloud hub for changes. Once a change is detected locally, it is conveyed to the cloud hub through a dedicated communication channel. If a change notification is received from the cloud hub, it is then mirrored locally.

Communicating with the cloud hub on behalf of the user requires authentication. Most of the services we analyzed avoid the use of explicit credentials (account name and password), and authenticate to the cloud using a *synchronization token*. The alleged advantage of a token over storing the password is that a compromise of a token does not compromise the entire account – and as a result, the rightful owner of the account can use the account name and password to revoke the rights related to the compromised token (and thus “save” the account). For the most part, the synchronization services we analyzed use the standard OAuth (actually OAuth 2.0) token. OAuth is designed for authentication and authorization between applications. Consequently, each instance of the drive application installed on an end station is viewed as an “application” rather than an end station. This observation explains some of the security weaknesses we point out later.

## 4. Motivation

Organizations are putting a lot of emphasis on endpoint security. Older solutions were often based on pattern matching against files that may contain malicious code. Newer solutions try to execute incoming files in a controlled environment to find out if they exhibit malicious behavior, or monitor endpoint activity for such malicious behavior. This includes looking for unusual code injection practices, use of well-known exploits, and other anomalies. Additionally, some solutions are attempting to identify C&C communication either through reputation (i.e., blacklist IP addresses of known C&C servers) or anomaly detection (i.e., detect an unusual new address with which an endpoint is communicating regularly).

While attackers seem to do well against these solutions with variations of their current practices (e.g., fast polymorphism, 0-day exploits, large C&C infrastructure), we attempted to foresee the future of this, asking ourselves the question, “At what point would all these endpoint solutions completely fail and how far are we from that point?” In particular, we looked for methods that use existing code on the endpoint and existing communication channels to maintain a persistent attack.

We quickly zoomed in on file synchronization services as they are becoming very common on enterprise machines (e.g., OneDrive comes bundled with Microsoft Office), and inherently maintain an open communications channel to the Internet. Additionally, file synchronization services are designed to constantly deliver files from local machines to the Internet (which hackers can use for exfiltration), and pull files from the Internet onto the local machine (which hackers can exploit for remote access). This was our first “Aha” moment.

That was followed by our second “Aha” moment. Instead of trying to break into a victim’s file synchronization account, we found that it is possible to synchronize the victim’s endpoint with an attacker-controlled account.

## 5. Controlling the Synchronization Destination

In the course of normal operations, when a user of a synchronization application wants to make an initial connection to an account or to switch between accounts, that user is required to interactively provide account credentials (account name and password) through a noticeable user interface.

As we mentioned earlier though, the synchronization application relies on a synchronization token, obtained after the above authentication process, for further operations. This token is stored on the endpoint either in the registry or in a file. In other words, after initial authentication, no explicit credentials are needed (or stored) by an application in order to access the user's account. We further determined that the synchronization token is machine independent - the same token can be used from different machines. For each of the synchronization applications that we evaluated, by simply copying a token for an account into the right place in the end station, we were able to make the synchronization application switch to the account represented by the token. Therefore, an attacker can gain access to the victim's account by stealing a token and does not need to compromise the victim's password. As we show later in this report, not having to use explicit credentials (typically, an account name and password) allows the attack to go mostly undetected by the owner of the account. The following table shows a summary of tokens and token locations for the application we evaluated.

SYNCHRONIZATION APPLICATION	ONEDRIVE	BOX	GOOGLE DRIVE	DROPBOX
Token Type	OAuth Refresh Token	OAuth Refresh Token	OAuth Refresh Token	Proprietary
Location	Windows Credential Manager	Windows Credential Manager	Encrypted in Registry	Encrypted SQLite file

**Table 1 - Drive Application to Token Type and Location Map**

Based on our findings indicated in this table, we built a tool called "Switcher" for each of the synchronization applications. The tool takes as input a synchronization token and stores it into the appropriate place on the victim's end station to synchronize with the account represented by the token. The token provided to the Switcher is extracted from the attacker's machine and represents an account created (and controlled) by the attacker.

Before making the actual switch, the Switcher stores the original token from the victim's machine to a file. This file is copied by the Switcher to the *sync folder* immediately after the switch so that the original token is synchronized to the attacker controlled account.

The Switcher's code is very simple. It does not interact with the Internet and does not rely on use of exploits. It only modifies some specific files or registry keys. In this way, it becomes extremely difficult to identify this code as malicious without affecting legitimate software. Once the Switcher's code is done running, it can be removed from the endpoint, leaving no remaining traces of evidence of the compromise.

## 6. Man in the Cloud Attacks

### 6.1 Quick Double Switch

This rather simple attack enables the attacker to share the victim's file synchronization account. The attacker is then able to access files which are synchronized by the victim and infect these files with malicious code. The attack consists of running the Switcher tool mentioned above on the victim's machine. This can be achieved through a drive-by-download exploit or through a simpler Phishing attack. The flow of the attack is described in Figure 1.

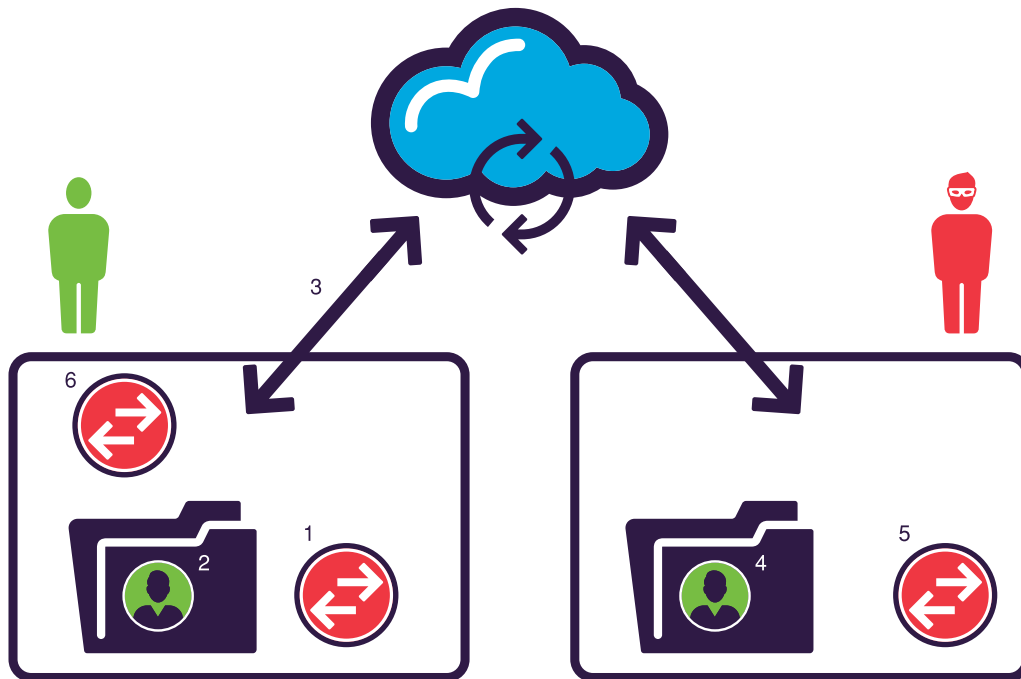


Figure 1 - Quick Double Switch Attack Flow

1. The attacker tricks the victim (using social engineering, for example) or uses an exploit in order to execute the Switcher. The Switcher then plants the attacker's *synchronization token* into the Drive Application.
2. When this first switch is complete, the Switcher copies the original *synchronization token* into the synced folder.
3. The Drive Application syncs with the attacker's account.
4. The attacker then has possession of the victim's *synchronization token*.
5. The attacker then uses the stolen *synchronization token* to connect with the victim's file synchronization account (using, for example, the Switcher tool on the attacker's machine).
6. The Switcher tool runs for the second time on the victim's machine (hence, "double switch") restoring the original *synchronization token* of the victim, essentially restoring the Drive Application to its original state.

This is the "cleanest" form of attack. After this attack is complete, the victim's Drive Application's state is the same as before the attack. The Switcher code deletes itself, which does not typically leave behind malicious code. Once the attack is complete, data placed by the victim in the *sync folder* is synchronized to the attacker's machine as well.

The double switch attack can be effective over time because the various file synchronization services may not restrict access of multiple devices from multiple locations to the synchronization account. For example: while Google responds to unusual access to the Google account itself (e.g., connecting from a new device or from an unusual geographic location invokes email messages to the account

owner), it does not provide a similar alert to suspect synchronization activity. Hence, it is possible for an attacker to maintain the synchronization activity with the victim's account from anywhere, anytime without notification to the owner of the account.

In addition to grabbing sensitive information, the attacker can manipulate files in the *sync folder* on the attacker's machine and have the changes propagate to the victim's machine. For example, the attacker can insert Macro code into MS Word documents or Scripts into PDF documents and have those execute when the victim tries to access these files. The results of execution can be written back to the *sync folder* and then collected by the attacker. After collecting the results, the attacker can remove them from the *sync folder* and even restore the original manipulated document to remove any traces of the attack. It is possible to devise other potential attack schemes based on the quick double switch. One possibility is a form of Ransomware in which the attacker encrypts the files on the attacker-controlled machine and have them synchronized to all other devices, which synchronizes with the victim's account.

## 6.2 Persistent Double Switch

This attack is very similar to the previous one, with the exception that the attacker wishes to maintain remote access to the victim. This access allows the attacker to interact with the victim's machine from time to time, execute arbitrary code, and collect that code's output. The flow of the attack is described in two phases in Figures 2 and 3.

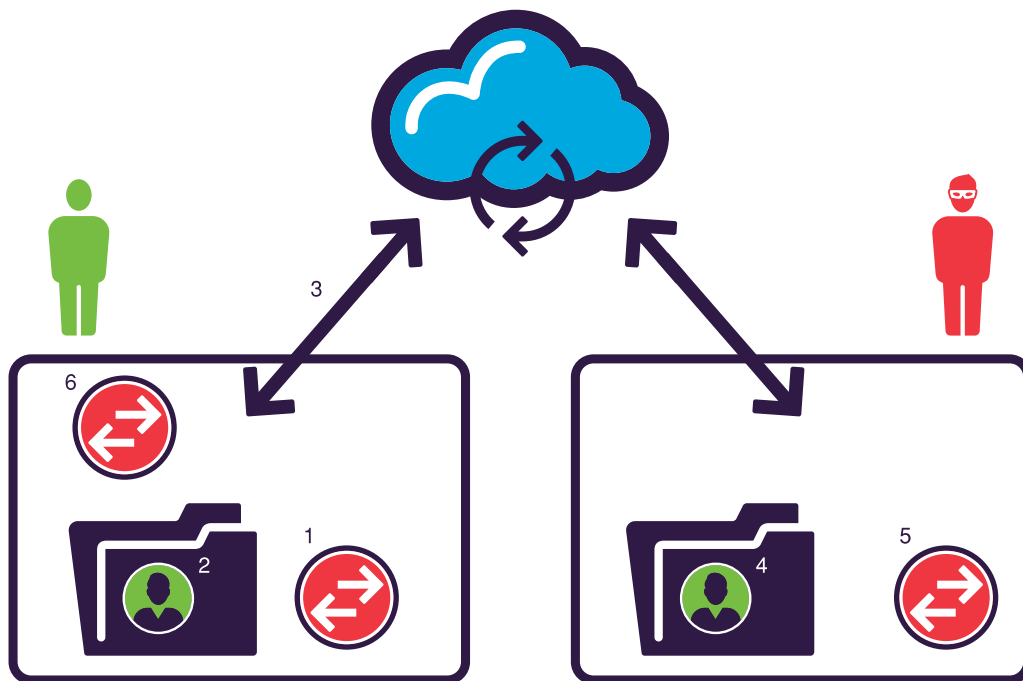


Figure 2 - Persistent Double Switch Attack Flow, Phase 1



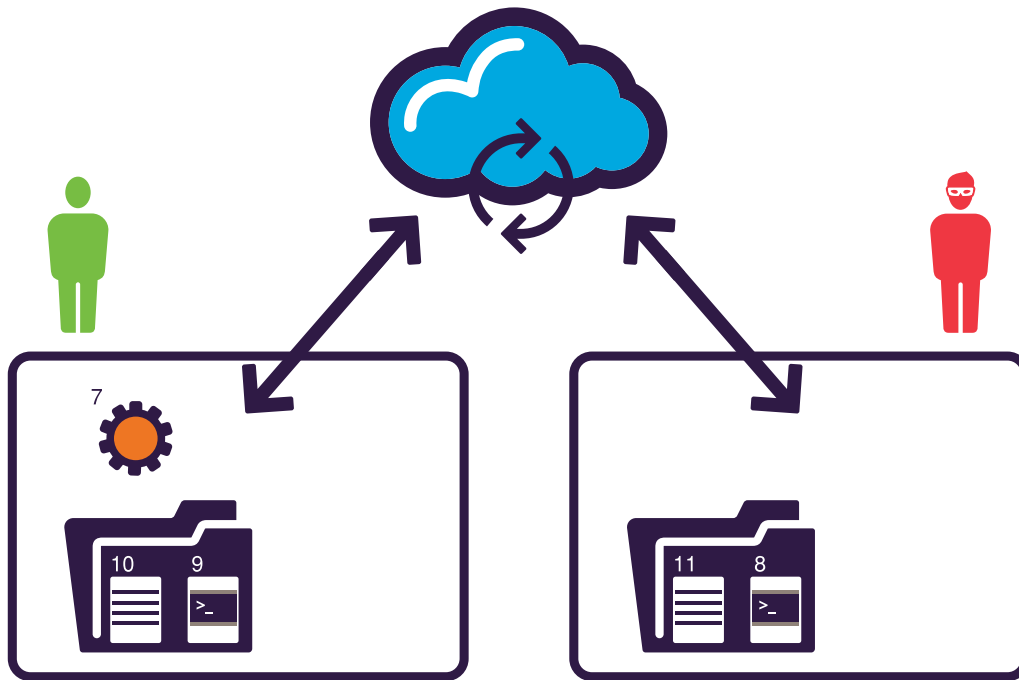


Figure 3 - Persistent Double Switch Attack Flow, Phase 2

1. The attacker tricks the victim (using social engineering, for example) or uses an exploit in order to execute the Switcher. The Switcher then plants the attacker's *synchronization token* into the Drive Application.
2. When this first switch is complete, the Switcher copies the original *synchronization token* into the synced folder.
3. The Drive Application syncs with the attacker's account.
4. The attacker then has possession of the victim's *synchronization token*.
5. The attacker then uses the stolen *synchronization token* to connect with the victim's file synchronization account (using, for example, the Switcher tool on the attacker's machine).
6. The Switcher tool runs for the second time on the victim's machine (hence, "double switch") restoring the original *synchronization token* of the victim, essentially restoring the Drive Application to its original state.
7. After the second switch the attacker sets up remote access to the victim's computer. (In the attack simulation we conducted, since we want the attack to leave little to no trace on the victim's computer, we did not set this as a running process.) The remote access is set up by waiting for a file to show up in a specific location in the *sync folder*, and then executing that file (assuming that the file deletes itself once execution is done). There are multiple methods for setting such a backdoor. Examples include scheduled tasks, setting conditional events through WMI, and setting registry entries for login triggered execution.

Once the attacker enabled remote access, the attacker can execute arbitrary code on the victim's machine by using the following process:

8. Place the code in a specific location in the *sync folder* on the attacker's machine.
9. The code gets synchronized to the victim's machine. The backdoor mechanism identifies the new file and executes it (see above).
10. The output of the code is written to the *sync folder* on the victim's machine and is synchronized to the attacker's machine.
11. The attacker collects the output, then removes the output and the code.

In this scenario, the attacker uses the victim's cloud storage as a C&C and remote access infrastructure.

### 6.3 Single Switch (Quick or Persistent)

In a Single Switch attack, the victim's data is synchronized with an attacker's controlled account. The flow of the attack is described in Figure 4.

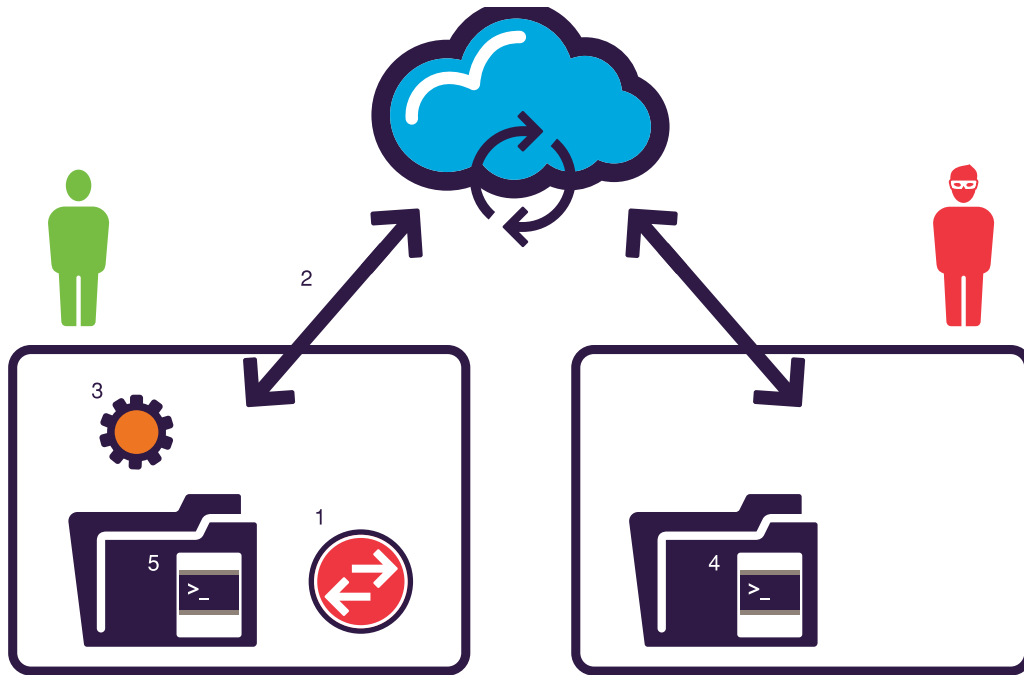


Figure 4 - Single Switch Attack Flow

1. The attacker tricks the victim (using social engineering, for example) or uses an exploit in order to execute the Switcher. The Switcher then pushes the attacker's *synchronization token* into the Drive Application.
2. Victim's data syncs with the attacker's account.
3. The quick attack is over, and the attacker now has access to the victim's data. In a persistent attack, the attacker also sets up remote access to the victim's machine.
4. In order to execute malicious code on the victim's machine, the attacker puts the code in the synced folder of the attacker controlled machine.
5. The malicious code is copied to the victim's synced folder and executed.

The advantage of this technique is that it overcomes synchronization services that detect change to the device or the location from which synchronization is done. Notification of such anomalies are sent to the attacker rather than the victim. The disadvantage of this attack is that the victim's local Drive Application will not sync correctly with other devices or the web interface. To minimize suspicion by the victim, the attacker may periodically switch back the victim's *synchronization token* - allowing the Drive Application to synchronize with the original account.



## 7. Persistence of MITC Attacks

MITC attacks are not easily detected. Even if a victim becomes aware that an account was compromised, it can be extremely difficult to deny further access of the attacker to the account. In most synchronization applications we evaluated, this translates into revoking the attacker's synchronization token, which is not always possible.

Dropbox is an example of how tricky it can be to revoke an attacker's synchronization token. The Dropbox *synchronization token* (host\_id value) is only changed when the user disconnects from her account, or "unlinks" her device (password changes do not affect the token's value). The user can only unlink devices that connected to her account, meaning that if an attacker retrieved several tokens from the victim, she won't be able to revoke them until he actually uses them.

While Box constantly refreshes tokens, we found that once an attacker gets hold of the victim's token, the attacker is able to generate a new "thread" of refresh tokens which is not revoked by default when the account owner changes the account password. The account owner must use an explicit "revoke all tokens" option during the password change in order to recover from the attack.

GoogleDrive is perhaps the best-case scenario if you found out that your *synchronization token* was stolen. We learned from our evaluation that changing the account's password forces all refresh tokens to be revoked and requires each device to re-authenticate using explicit account credentials. Revocation is immediate and any device that uses an existing token will not be granted further access to the data.

OneDrive behaves almost the same as Google Drive with respect to synchronization tokens and changing the account password. Changing the account password revokes all refresh tokens and require explicit credentials for authentication. Unlike GoogleDrive, devices that have already presented a valid token continue to receive access to data (through a "session identifier"). These devices need to be manually removed from the account.

## 8. Platform Specific Details

This section provides details about each of the applications we analyzed. Below, we provide information about *synchronization token* format and storage, as well as the effects of stealing a *synchronization token* over time, based on our evaluation.

One common attribute of all the applications we analyzed is that each treats *synchronization tokens* as sensitive (it grants access to the user's data). Hence, they all store the token using encryption with a key derived from the user's end-station credentials. Thus, the clear-text *synchronization token* is accessible only through an application running from the user's account.

The following table shows the applications' versions we analyzed.

SYNCHRONIZATION APPLICATION	ONEDRIVE	BOX	GOOGLE DRIVE	DROPBOX
Version	17.3.5860.0512	4.0.6477	1.18.7821.2489	3.6.8

### 8.1 Google Drive

Google uses OAuth 2.0 for its application's authentication. When a user first logs on from a Google Drive application, Google Drive prompts the user for credentials. In our experience using GoogleDrive, once the application verifies the credentials, the application receives a "refresh token". The application uses this refresh token to further request and receive access tokens. Whenever an access token expires, the application can request a new one using the "refresh token".

Google Drive keeps important data inside the registry: *HKEY\_CURRENT\_USER\Software\Google\Drive*. One of the keys in that path goes by the telling name: "OAuthToken\_o3hPm\*\*\*\*\*Bni0=". This is where Google stores its token in an encrypted form.

To determine how Google Drive encrypts the token, we hooked several cryptographic functions found in "Crypt32.dll" - a commonly used library in Windows OS. This DLL is called by many applications because it provides a method to encrypt and decrypt data

using the currently logged on user's credentials. We found out that Google Drive uses this library to encrypt its token by calling the *CryptProtectData* function. By intercepting this call, we were able to determine which data is encrypted.

Google Drive encrypts a base64-encoded string made up from different fields that the application needs in order to authenticate. These are:

- Refresh Token – essentially what we want to steal (the *synchronization token*)
- Client ID – an identifier of the Google Drive application
- Client Secret – a “secret” that the application uses to authenticate
- A List of APIs – these are APIs that the google application requests access to

The above data is concatenated to form a single character string in the following manner:

```
'2G' + Base64Encode(Refresh Token) + '|' + Base64Encode(Client ID) + '|' + Base64Encode(Client Secret) + '|' +  
Base64Encode(Base64Encode(feeds API) + '|' + Base64Encode(drive API) + '|' + Base64Encode(google talk API) + '|' +  
Base64Encode(docs API))
```

Google Drive stores some additional data that is important for proper functionality of the application. Some of this data is stored inside the “*sync\_config.db*” database. This database holds information such as application version, path of synced folder, the user's email address, etc. This file is important because an attacker will need to manipulate this file also in order to switch the identity of the application.

With the above knowledge, an attacker may retrieve the clear text token using a simple code that performs the following tasks:

- Read the value name inside the registry path: *HKEY\_CURRENT\_USER\Software\Google\Drive*
- Decrypt this data using *Crypt32.dll*'s function *CryptUnprotectData*
- Extract the refresh token according to the structure we mentioned in the previous section

Making the switch requires some extra effort and manipulation of the *sync\_config.db* file on the victim's machine.

First, the attacker must obtain a refresh token for the attacker's controlled account. This token should be generated with the same version of Drive Application as the victim's. The attacker obtains this token by creating a Google Drive account and performing the steps mentioned above. The attacker embeds this token in the Switcher code that runs on the victim's machine. In addition to the token itself, the attacker needs to provide the name of the registry entry for the token. The code required for making the switch performs the following actions:

- Terminate the Google Drive application (if it is running)
- Create a string that contains the attacker's token and encrypt it using *CryptProtectData* – which will use the victim's credentials to encrypt this token
- Delete the old entry from the registry of *OAuthToken\_\*\*\*\*\**
- Create a new entry in the registry, with the name copied from the attacker's machine (*OAuthToken\_o3hPm\*\*\*\*\*Bni0=*)
- Encrypt the previously created string using *CryptProtectData* and plant it inside the new registry entry
- Replace each row inside the data table in *sync\_config.db* with the attacker's data, except for the “*local\_sync\_root\_path*” entry (which holds the path to the victim's synced folder)
- Restart the Google Drive application

After the malicious code finishes running, the application will sync with the attacker's Google Drive account. In order to switch back to the victim's token, the attacker only needs to restore the previous registry keys and values after terminating the Google Drive application, and then restart it again.

There are several security gaps related to the behavior of Google Drive that make these types of attacks harder to detect and mitigate.

Google keeps track of logon activity for each account. It registers which devices logged on to an account and from where. This way, users are able to view their [security activity](#) and check whether suspicious activity is taking place. This, however, is not the case for applications. It makes sense that third party applications, that a user authorized to access his account are not tracked for “anomalies”

regarding location. Google is unable to determine the location from which a third party application connects to its servers. When it comes to its own Google Drive application, Google considers each new synchronizing device to be the same as a third party application, and does not track the log-in location.

Google does list connected apps in the [permission section](#), enabling the user to disconnect individual devices. For example, to save a compromised account, a victim can disconnect all devices from his or her Google account (because it is difficult to ascertain the device from which the token was stolen). To be on the safe side, the user can also change his or her password – which will revoke all tokens associated with the account.

## 8.2 Dropbox

Unlike Google Drive, Dropbox does not use the OAuth protocol for authentication. As mentioned in a [2011 blog post by Derek Newton](#), Dropbox uses what it calls a “*host\_id*”. This identity serves as its *synchronization token*, and is stored inside an SQLite file called “*config.dbx*” (or *config.db* in older versions). This *host\_id* is the only piece of data required for accessing not only the files within the Dropbox account, but the account itself (i.e., for account management). Perhaps even more troubling is the fact that this *host\_id* does not change even if the user changes the password for the account. In order to improve the security of this *host\_id*, the Dropbox application encrypts the *config.dbx* file. The encryption key, however, can be easily extracted by simple available tools. These tools follow the footsteps of the reverse engineering job done by Dhru Kholia (“[Looking Inside the \(Drop\\_Box\)](#)”).

In order to retrieve the victim’s original *synchronization token*, an attacker needs to decrypt the *config.dbx* file. The attacker can obtain the encryption key from the Windows registry by using a Python script written as part of Dhru Kholia’s work called [dbx-keygen-windows](#). The code retrieves an encrypted encryption key from the registry, and then uses standard Windows APIs to decrypt it and obtain the actual key. These APIs, as mentioned before, use the credentials of the currently logged-in user for decryption.

```
C:\Users\user\Desktop>dbx-keygen-windows.py
KEYSTORE: got user key (zd, zs) 0 '\xca$\xc6}\x997\xe7\xc9\xe0\x89\xd2\x04\xce\x
lf\x0f\xa3'
User key: ca24c67d[REDACTED]f0fa3
Database key: 98d[REDACTED]fd0237
```

Figure 5 - Getting Database Key in Dropbox

The attacker would next use [sqlite3-dbx](#), which was written by the same author, and uses the extracted encryption key to decrypt the *config.dbx* file. This file contains a single database called “*config*”. The attacker can find the “*host\_id*” key and its value inside this database.

The full process of making the switch from one account to another involves the following steps:

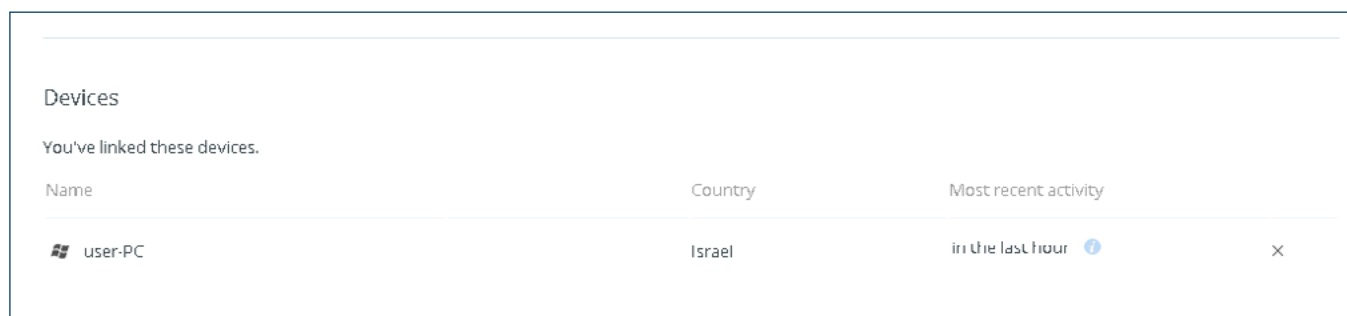
- Terminate running instance of Dropbox (if any)
- Get the database key by running *dbx-keygen-windows*
- Open the *config.dbx* file using *sqlite3-dbx* and the previously obtained database key
- Replace the existing *host\_id* entry in the *config.dbx* file with a new *host\_id* value (obtained from an attacker controlled account)
- Run Dropbox

The attacker would then use the stolen *host\_id* from an attacker-controlled machine. By retrieving the “*device\_id*” from the *config.dbx* file and using that same “*device\_id*” in the attacker-controlled machine, an attack can go unnoticed.

There are several security gaps related to the behavior of Dropbox that make these types of attacks harder to detect and mitigate. Perhaps the most glaring security issue of Dropbox is the use of *host\_id*. An attacker can use this token to access data within Dropbox, as well as control the account through the Dropbox web interface. The *host\_id* value used to be fixed for the entire lifespan of the

Dropbox account. Now, DropBox grants a new `host_id` for each new interactive login, and associates this `host_id` with the connected device. The victim can disconnect connected `host_id`'s from her account; thus revoking stolen `host_ids`.

It is easy for an attacker to avoid being recognized as a new device connecting to the victim's account (which would have invoked a warning email message to the victim). This requires retrieval of the "device\_id" from the victim's machine and using it in the attacker-controlled machine. As long as the attacker does not invoke too many changes in the data, it will become very difficult to notice the takeover. While Dropbox keeps track of the location from which each device is connecting (visible through the account management interface), it keeps only a single entry per device. Thus, if two separate machines are connected to the account using the same "device\_id", it becomes very hard for the victim to notice the compromise – as most account activity is performed by the victim, and hence the victim's location will be displayed.






Devices			
You've linked these devices.			
Name	Country	Most recent activity	
 user-PC	Israel	in the last hour 	

Figure 6 - DropBox Connected Devices

Even access by the attacker to the account management interface can go unnoticed. This can be achieved even though Dropbox presumably provides an audit trail for that exact purpose. An attacker, once connected to the management interface, can remove all entries written to the log, thus covering the tracks.

### 8.3 OneDrive

OneDrive uses OAuth 2.0 for its application's authentication. After an initial authentication process that requires the actual credentials, the drive application receives a "refresh token" (this is a private case of our *synchronization token* for OneDrive). The application uses this refresh token to request and receive access tokens. Whenever an access token expires, the application can request a new one using the refresh token. The synchronization service also gives each refresh token a "user-id" identifier, which is the same for all refresh tokens generated for the same account.

OneDrive uses Windows' Credentials Store for refresh token storage. The Credentials Store encrypts data using the credentials of the logged on user. Retrieving the refresh token is trivial for a code that runs on the victim's machine when the victim is logged in. Simple well-known Windows API calls are used for accessing the Credentials Store and retrieving the entry named "OneDrive Cached Credential". The refresh token is stored in the "password" field of the entry, while the "user-id" is stored in the "user name" field.

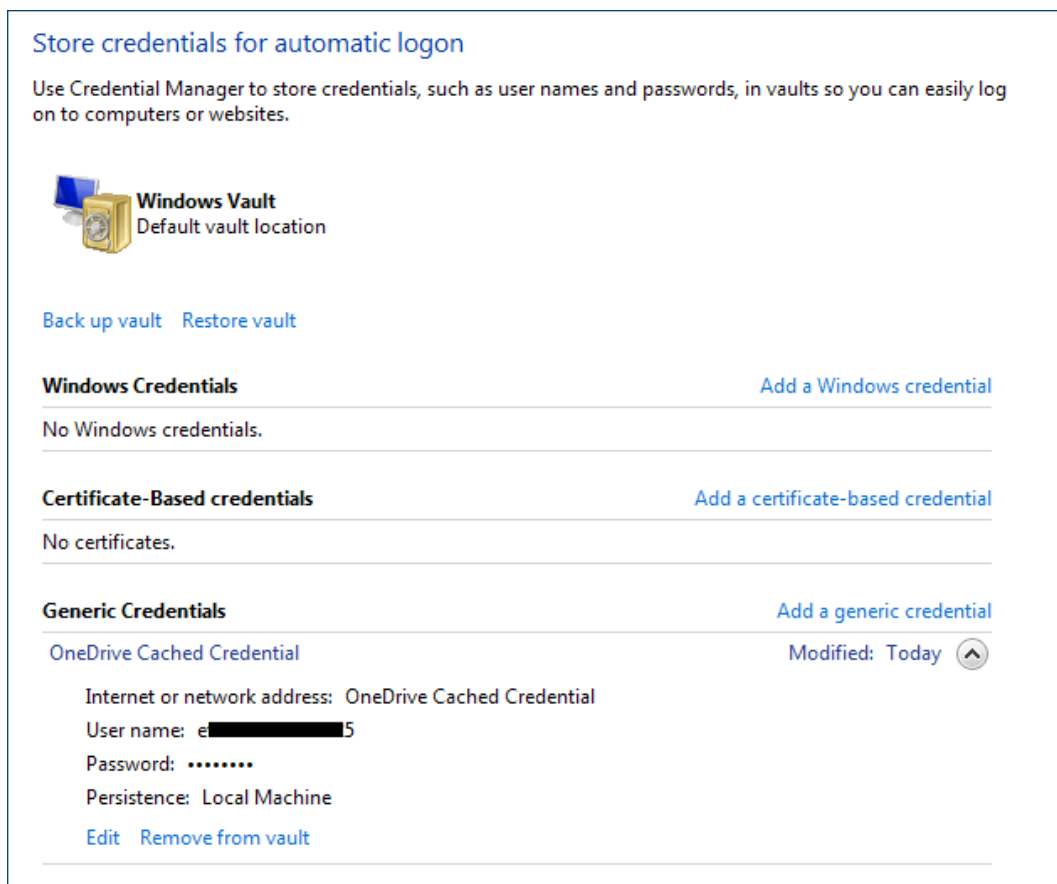


Figure 7 - OneDrive Credentials in Windows' Credential Manager

Performing the actual switch of accounts requires the following steps:

- Terminate running OneDrive application (if any)
- Replace the "OneDrive Cached Credentials" entry with the user name (user-id) and password (refresh token) values of the attacker controlled account (this is done via standard Windows APIs)
- Delete existing <user-id>.txt file inside the victim's settings folder (i.e., %USERNAME%\AppData\Local\Microsoft\OneDrive\settings\Personal)
- Copy the content of the <user-id>.txt from the attacker controlled machine into the victim's settings folder
- Run OneDrive

Microsoft's OneDrive application provides some security when it comes to access tokens. OneDrive shows the user which devices are connected to the OneDrive account and where they were connected from. After performing a switch, the victim will be able to see in OneDrive "Recent Activity" that there was a connection to the account from a different location. However, OneDrive does not proactively alert the user if the same "user-id" value was used from two distinct locations within a short period of time.

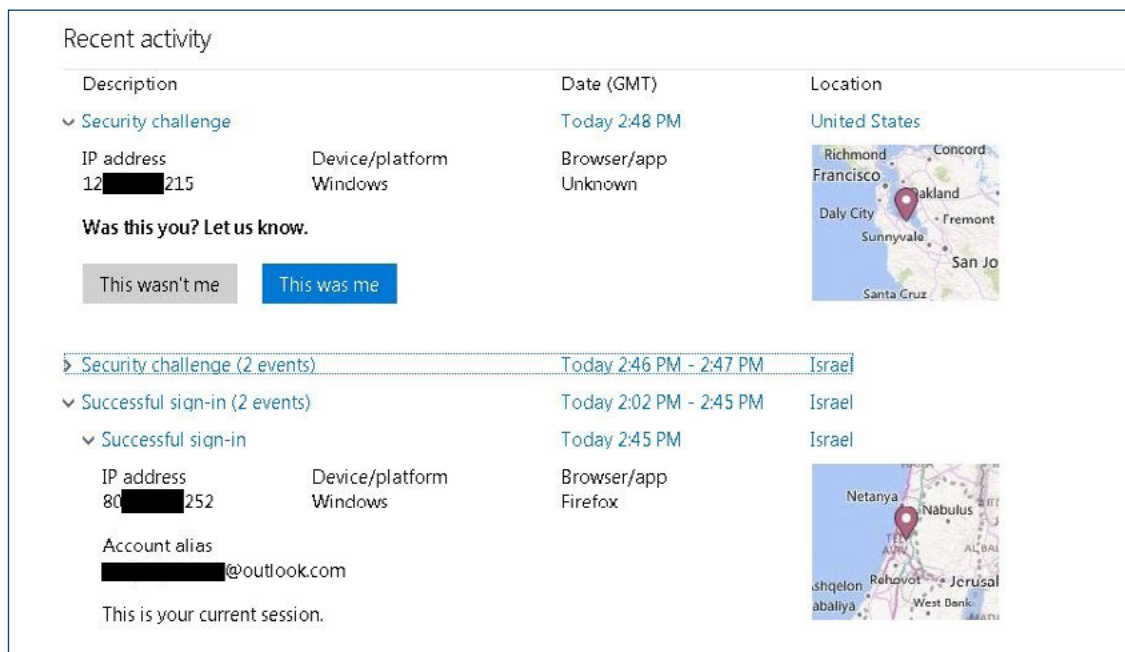


Figure 8 - OneDrive Recent Activity after Switch

The OneDrive application tries to uniquely identify each device connected to the account. This is coupled with the ability of a user to disconnect specific devices, and, therefore, requires explicit credentials for the device to reconnect (i.e., user name and password). However, device identification seems disconnected from refresh token validation. We were able to use the refresh token of an alleged "disconnected device" from a different device. Thus, an attacker can maintain access to the victim's data even if the victim detects a suspicious device and disconnects it. However, we were not able to manipulate the data in this scenario. The possession of a refresh token also bypasses the two-factor authentication mechanism that protects the account, as no message is sent when the token is used from a different device.

We observed that once a refresh token is presented by the drive application to the OneDrive synchronization service, a temporary session identifier is generated. This session identifier can be used for further interaction with the service even if the refresh token itself is revoked. In order to revoke the session identifier, the victim must first change the account password and disconnect all devices. Another troubling aspect of stolen refresh tokens is that the token remains valid even when victim changes the account password.

## 8.4 Box

Very similar to OneDrive, Box uses OAuth for authentication after a user logs on to the account, and stores a refresh token inside Windows Credentials Manager under "Box Sync". The user name is the email address of the account, and the stored value is the encrypted refresh token. If for some reason the Credentials Manager is not available, the token is stored in a text file: "C:\Users\%USERNAME%\AppData\Local\Box Sync\wincrypto\_pass.cfg". For encryption, Box uses built in Windows API, the same as OneDrive.

Unlike OneDrive or GoogleDrive, Box obtains a new refresh token whenever it loads up. This occurs whenever a user logs on to the machine or chooses to close or open the Box application. We will further discuss the implications of this feature later on.

In order to perform the switch on a victim's account, an attacker's malicious code needs to complete the following steps:

- Terminate running Box application (if any)
- Replace the password value (the refresh token) original Box entry in Windows Credentials
- Delete Box's *sync folder* (otherwise there will be issues on startup)
- The start of the Box application should hide a pop up window and select "Sync Folder"

We believe the best built-in security feature of Box is the user's ability to receive notifications for operations performed on the account – such as file upload, download, deletion, etc. Additionally, Box shows location information regarding connected devices.

When we "switched" a refresh token on a device, we noticed a new entry in the Box security tab. The user, however, does not receive any alert (an email, for example) that a logon was performed from a new location, as it does in the event of a plaintext logon via the Box web interface. An attacker can avoid detection by using a proxy that resides in the same country/city as the victim.

#	Applications	Added	Last Accessed	Last Access Location	Forget App
1	Box Sync for Windows	Jul 7, 2015	Jul 7, 2015	Redwood City, CA, USA	
2	Windows Firefox	Jul 7, 2015	Jul 7, 2015	Israel	Current session
3	Box Sync for Windows	Jul 7, 2015	Jul 7, 2015	Israel	
4	Box Sync for Windows	Jul 6, 2015	Jul 6, 2015	Tel Aviv, Israel	
5	Box Sync for Windows	Jul 6, 2015	Jul 6, 2015	Tel Aviv, Israel	
6	Box Sync for Windows	Jul 6, 2015	Jul 6, 2015	Israel	
7	Box Sync for Windows	Jul 5, 2015	Jul 5, 2015	Israel	
8	Windows Firefox	Jul 5, 2015	Jul 5, 2015	Tel Aviv, Israel	

Figure 9 - Location Information on Connected Devices to Box

When the Box application starts up, it requests a new refresh token using the currently stored token. After this operation is complete, the Box application will revoke the old token and deny use of the old token. However, an attacker that grabs a valid token can immediately use it for authentication by simply connecting to the service from an attacker-controlled endpoint before the Box application has the opportunity to revoke that old token. Thus, the attacker is granted a new token over which the victim has less control. The victim is required to provide explicit credentials the next time the Box application is started, but the attacker retains access to the account even after the password for the account is changed. Only when the victim changes the password and explicitly requests to disconnect all devices and 3rd party applications, will she be able to revoke the stolen token. The extra security measure of 2-factor authentication will only be activated when logging on using plaintext credentials. Using refresh-tokens bypasses this security mechanism.

In summary, once the attacker has stolen the victim's token and refreshed it, the victim should change password and explicitly request to disconnect all devices in order to revoke the attacker's access.



## 9. Conclusions and Mitigations

Based on our research, we were able to take control of a victim's machine for data extrusion as well as gaining remote access by using common file synchronization services. From an attacker's point of view, there are advantages in utilizing this technique - malicious code is typically not left running on the machine and the data flows out through a standard, encrypted channel. In the MITC attack, the attacker does not compromise explicit credentials (e.g., account name and password) of the victim. As we have described, in the case of file synchronization services, explicit credentials are actually less valuable than a synchronization token. Additionally, we learned that it is unlikely for an unsuspecting victim who is not carefully monitoring device-sync activity to detect a compromise. Furthermore, it is extremely difficult (or painful) to recover from an attack once it is detected, and may require the victim to cancel the existing account and open a new one. While testing our concepts in the lab, we found some evidence that these types of attacks are already occurring in the wild (for example, as mentioned in "[The Inception Framework](#)" - an analysis by Blue Coat). The feasibility of this type of attack indicates that a mitigation strategy that relies mostly on preventing and detecting infection through malicious code detection or C&C communications detection will likely fail.

We propose a solution that has two aspects to it - identify the compromise of file synchronization account, and even more importantly, identify the abuse of the internal data resource. We believe that the attackers are eventually after the enterprise data rather than the information stored at endpoints. Hence, an attack is bound to express itself by the attacker trying to access business data in a way that is not typical for normal enterprise users.

For the first part of the mitigation strategy, we urge organizations to use a Cloud Access Security Broker (CASB) solution that monitors access and usage of enterprise cloud services by the enterprise users. We believe CASB solutions can effectively detect, in a timely manner, anomalies in the way an account for a file synchronization service is used and access. The more effective CASB solutions (such as those deployed virtually online) can also make mitigation easier by blocking access of unrecognized devices to the data.

The second part requires that organizations deploy controls such as DAM and FAM around their business data resources, and identify abnormal and abusive access to the data. Again, the better solutions are also capable of quickly mitigating the threat and containing the compromised device or account by restricting further access to all enterprise data or to the sensitive part of it.

## Hacker Intelligence Initiative Overview

The Imperva Hacker Intelligence Initiative goes inside the cyber-underground and provides analysis of the trending hacking techniques and interesting attack campaigns from the past month. A part of the Imperva [Application Defense Center](#) research arm, the [Hacker Intelligence Initiative](#) (HII), is focused on tracking the latest trends in attacks, web application security and cyber-crime business models with the goal of improving security controls and risk management processes.

Additional support provided by Students of Techion University Information Security Project.