Dropbox: Looking inside the protocol behind file synchronization

Dropbox[1] is one of the most popular file hosting services on the web that offers cloud storage and file synchronization across multiple devices. It basically allows users to have a special folder on their computers or mobile devices where they can manage their files and the Dropbox service automatically syncs those files to the cloud and other devices that the user has.

Dropbox Client Security

The Dropbox client, available on Windows, Linux and MacOS, is mainly written in Python. Because of this and the fact that Python bytecode is easily reverse engineered, the developers of Dropbox used a modified encrypted Python bytecode. The bytecode itself is decrypted and run when the Dropbox client is executed.

From a networking perspective, Dropbox syncs its files to the cloud service via standard HTTPS and uses both client and server certificate verification upon TLS session establishment. Because of this, intercepting it via a proxy is impossible to accomplish.

Also, for authentication purposes, Dropbox uses a two-factor method where the device on which the Dropbox client is running is first registered with the cloud service using the user's credentials and then the device is subsequently re-authenticated at periodic intervals.

How we intercepted the Dropbox protocol

Since decompiling the application would have involved too much of an effort [2], we mainly focused our efforts on the library it was using to establish the TLS session, namely OpenSSL.

By using the dynamic instrumentation tool platform DynamoRIO[3] we managed to patch the OpenSSL SSL_Write and SSL_Read functions so that we can dump the unencrypted form of the HTTP requests and responses to a file.

The functions we used are presented below and they may be used on any other application that uses the OpenSSL library. They have been adapted starting from the wrap.c example present in DynamoRIO.

```
// Usage:
//
// bin64/drrun -v -c samples/bin/libwrap.so -- ~/.dropbox-dist/dropbox

#include "dr_api.h"
#include "drwrap.h"
#include <stdio.h>
```

```
static void *lock;

static void wrap_pre_write(void *wrapcxt, OUT void **user_data);
static void wrap_pre_read(void *wrapcxt, OUT void **user_data);
static void wrap_post(void *wrapcxt, void *user_data);

unsigned char *read_bufer = NULL;
int read_bufer_len =0;
void *ssl;

static
void module_load_event(void *drcontext, const module_data_t * mod, bool loaded)
{
	app_pc towrap = (app_pc)
	    dr_get_proc_address(mod->handle, "SSL_write");
	if (towrap != NULL) {
		bool ok = drwrap_wrap(towrap, wrap_pre_write, NULL);
		if (!ok) {
			printf("Couldn’t wrap\n");
			drwrap_exit();
		}
	}

	app_pc towrap2 = (app_pc)
	    dr_get_proc_address(mod->handle, "SSL_read");
	if (towrap2 != NULL) {
		bool ok = drwrap_wrap(towrap2, wrap_pre_read, wrap_post);
		if (!ok) {
			printf("Couldn’t wrap\n");
			drwrap_exit();
		}
	}
}

DR_EXPORT void dr_init(client_id_t id)
{
	/* make it easy to tell, by looking at log file, which client executed */
	dr_log(NULL, LOG_ALL, 1, "Client 'wrap' initializing\n");
#ifdef SHOW_RESULTS
	if (dr_is_notify_on()) {
#ifdef WINDOWS
		dr_enable_console_printing();
#endif
```

```
                dr_fprintf(STDERR, "Client wrap is running\n");
        }
#endif
        drwrap_init();
        dr_register_module_load_event(module_load_event);
        lock = dr_mutex_create();
}

static void wrap_pre_write(void *wrapcxt, OUT void **user_data)
{
        // http://www.openssl.org/docs/ssl/SSL_write.html
        // int SSL_write(SSL *ssl, const void *buf, int num);
        void *ssl = (void*) drwrap_get_arg(wrapcxt, 0);
        unsigned char *buf = (unsigned char *) drwrap_get_arg(wrapcxt, 1);
        size_t sz = (size_t) drwrap_get_arg(wrapcxt, 2);
        char filename[50] = "";
        sprintf(filename, "/tmp/trace%p.write", ssl);
        FILE *f = fopen(filename, "ab+");
        fwrite(buf, 1, sz, f);
        fclose(f);
}

static void wrap_pre_read(void *wrapcxt, OUT void **user_data)
{
        dr_mutex_lock(lock);
        ssl = (void*)drwrap_get_arg(wrapcxt, 0);
        read_bufer = (unsigned char *) drwrap_get_arg(wrapcxt, 1);
        read_bufer_len = (size_t) drwrap_get_arg(wrapcxt, 2);
}

static void wrap_post(void *wrapcxt, void *user_data)
{
        dr_mutex_unlock(lock);
        if (read_bufer_len == 0) {
      return;
   }
        char filename[50] = "";
        sprintf(filename, "/tmp/trace%p.read", ssl);
        FILE *f = fopen(filename, "ab+");

        fwrite(read_bufer, 1, read_bufer_len, f);
        fclose(f);
}
```

By using these, each TLS connection will be dumped to /tmp/trace<connection handle address>.[read|write].

Dropbox Protocol Structure

Now that we've seen how we can intercept and reverse engineer the protocol, let's take a look at some of the important requests and what they contain. For brevity we omit some of the less important request and response payloads.

Authentication

When the client is first started on a new device it proceeds to check if the current user is signed in as shown in Fig. 1. Also present in Fig. 1 are the custom HTTP headers that Dropbox uses. It also uses chunked encoding for the majority of requests and responses.

```
POST /check_sso_user HTTP/1.1
Host: client37.dropbox.com
Accept-Encoding: identity
Content-type: application/x-www-form-urlencoded
Connection: keep-alive
X-Dropbox-Locale: en_US
Content-Length: 70
User-Agent: DropboxDesktopClient/2.8.3 (Linux; 3.8.0-29-generic; i32; en_US)

host_id=58a29b6911f683d39f77d906d9cec063&email=test%40ixiacom.com
```

```
HTTP/1.1 200 OK
Server: nginx
Date: Sat, 09 Aug 2014 16:45:54 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Connection: keep-alive
x-db-timeout: 60
x-content-type-options: nosniff
x-server-response-time: 28
x-dropbox-messages: {}
x-dropbox-request-id: fe234f28caade228ad8614642ec00591
pragma: no-cache
cache-control: no-cache
x-dropbox-http-protocol: None
x-frame-options: SAMEORIGIN

3d
{"user_sso_state": "none", "chillout": 8.031287501158066e-08}
0
```

Fig. 1. Dropbox Single Sign-On user check

After the SSO (Single Sign-On) check, the host is linked with the user account via the request/response pair presented in Fig. 2. The user's email/password is provided in the request and the gvc cookie is returned via the response.

```
POST /link_host_with_ret HTTP/1.1
...

is_preauth_link=False&nonce=None&display_name=viserion&post_2fa_token=None&instal
l_type=None&oem_model_info=None&tag=None&installer_tags=None&host_id=58a29b69
11f683d39f77d906d9cec063&password=ixiaati&is_sso_link=False&email=test%40ixiaco
m.com
```

```
HTTP/1.1 200 OK
...
set-cookie:
gvc=MjUxMDgwMzUwMjczNDg1Mzk5MDAxMzc1MDM1MDQzMjIzMDQxOTk%3D;
expires=Thu, 08 Aug 2019 16:45:55 GMT; Path=/; httponly
...

32
{"upgrade_prompt_post_install": true, "ret": "ok"}
0
```

Fig. 2. Dropbox Link Host request/response

The authentication is afterwards maintained via the Desktop Login Sync request/response pair presented in Fig. 3. The response is composed of a series of nonces which are probably used to derive the us parameter present in the subsequent request.

```
POST /desktop_login_sync HTTP/1.1
...

ia=1&host_id=6925bf34850a552594c6c23c40bc1a80&us=eJyrrgUAAXUA-Q%7E%7E
```

```
HTTP/1.1 200 OK
...

11c
{"nonces": {"-DJ7dSCOoChgzYbsu9uEjeX9MEY5joXr6uBP8zxToC4": 1408578695,
"ORRpj81_zuXGXCGJPAdkAh5eU12z1yes74tszOFK8TU": 1408578695,
"pDPYQXA36NAYQFE2kOhxsZq_9td18HUxB4AWIlaheE4": 1408578695,
"W8qBqkKk_fBFQCvHY0ClNhfr_HCxnXqbA4MejNTDv1w": 1408578695}, "chillout":
5.7366339293986195e-08}
0
```

Fig. 3. Dropbox Desktop Login Sync request/response

As can be noted, the majority of the responses are in JSON format while the request are done by using form encoding (application/x-www-form-urlencoded).

Retrieving file contents and directory structure

Another interesting aspect that we should touch upon is the request/responses used for syncing the actual file contents. These can broadly be broken down into retrieving file contents/changes from the cloud service and committing/uploading files/changes to the service.

Fig. 4 and 5 show the requests/responses used for retrieving the directory structure of the current user’s Dropbox folder and the list of files in those directories.

```
POST /list_dirs HTTP/1.1
…

ns_map=678000691_-1&host_id=58a29b6911f683d39f77d906d9cec063
```

```
HTTP/1.1 200 OK
...

af7
{"host_int": 2576532816, "blockexcserver": "dl-debug3.dropbox.com,...", "server_time":
1407602800, "uid": 325938066, "in_use": 249184, "poll_interval": 30, "more_results": false,
"quota": 2147483648, "ns_cursors_before": {"678000691":
"AAC8x-Wu3ci60GhVnsM23MejDO8Zt9Qjbwx8wt0m-FAjD6V7VhlBXrzODoMGrncIh27B2N
ao-Y2aVzvNgmqkqJg4j3cRmpJT7DL368BIJ0xRPA"}, "namespace_permissions": {},
"webserver": "www.dropbox.com", "sandboxes": [], "excserver": "dl-debug6.dropbox.com",
"tombstones_map_before": {"678000691": -1}, "tombstones_map_after": {"678000691":
47922640947}, "cu_data": null, "ns_map_before": {"678000691": -1},
"upgrade_retry_interval": 1801, "send_trace": null, "limits":
{"upload_hash_batch_max_size": 4194304, "download_hash_batch_max_size": 4194304,
"upload_hash_batch_parallel_connections": 1, "download_hash_batch_max": 100,
"download_hash_batch_parallel_connections": 1, "upload_hash_batch_max": 100},
"ns_cursors_after": {"678000691":
"AAB1ipwK6pxgqK7VcXvBHY384-JrSUUyxvGibIW8hc6OQRirwTbUoFXrjOEs47e9VHgOT
kjpASIcCCI_xCoTLtzHd0zYSXaC652XYFkM1ERG1A"}, "max_data_size": 131072, "list":
[{"blocklist": "", "sjid": 9267935283, "attrs": {}, "mtime": 1407304273, "path":
"678000691:/test", "dir": 1, "size": 0}], "pubserver": "dl.dropboxusercontent.com", "ret": "ok",
"cu_rotation_type": "V1", "notifyserver": "notify7.dropbox.com:80", "ns_map_after":
{"678000691": 47922640947}, "blockserver": "dl-client1.dropbox.com", "metaserver":
"client-lb.dropbox.com", "chillout": 5.7366339293986195e-08, "metaexcserver":
"d.dropbox.com"}
0
```

Fig. 4. Dropbox List Dirs request/response

```
POST /list HTTP/1.1
…
```

ns_map=678000691_-1&buildno=Dropbox-lnx.x86-2.8.3&ns_p2p_key_map=%7B%22678000691%22%3A+null%7D&dict_return=1&server_list=True&host_id=58a29b6911f683d39f77d906d9cec063&initial_list=1&need_sandboxes=1&xattrs=True

HTTP/1.1 200 OK
...

2466
{"host_int": 2576532816, "blockexcserver": "dl-debug31.dropbox.com,...", "ns_p2p_key_map": {"**678000691**": {"pk": "-----BEGIN RSA PRIVATE KEY-----\n...\n-----END RSA PRIVATE KEY-----\n", "cert": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----\n", **"md5": "c21d0a102b9a7bdda178371b46708b81"**}}, "server_time": 1407602801, "uid": 325938066, "in_use": 249184, "poll_interval": 30, "more_results": false, "quota": 2147483648, "ns_cursors_before": {"678000691": "AAALLGzhPziMk14zL2buWJVdMG7qfRp7_om0P95lbQdON6s4uEPwQeqUP-mA3LdpFM6IXjtVZMfPFyQQm5p-AJ7vd6YxAKedvOk0ezyuElv5fQ"}, "send_trace": null, "webserver": "www.dropbox.com", "sandboxes": [], "excserver": "dl-debug27.dropbox.com", "tombstones_map_before": {"678000691": -1}, "tombstones_map_after": {"678000691": 47922640947}, "cu_data": null, "ns_map_before": {"678000691": -1}, "upgrade_retry_interval": 1801, "dict_return": 1, "limits": {"upload_hash_batch_max_size": 16777216, "download_hash_batch_max_size": 16777216, "upload_hash_batch_parallel_connections": 4, "download_hash_batch_max": 400, "download_hash_batch_parallel_connections": 4, "upload_hash_batch_max": 400}, "ns_cursors_after": {"678000691": "AADTBBWbjdJGfkKjNLEELaoUx6mKzIwdgHVk7rVxuCqlGo35AoXvmo4o_J-UIsOVdUpop9pLgloQRiSPj5uw2_2VMP9nDWgCNe_I70bsDvMGzw"}, "namespace_permissions": {}, "debug_tombstones_present": true, "max_data_size": 131072, "list": [{"event_type": 1, "attrs": {}, "mtime": 1407599976, **"path": "/Getting Started.pdf"**, "is_dir": false, "size": 249159, "user_id": 216, **"blocklist": "pBTCVnmA6QJnyEaeJHS_Gd4-FqssClN_XbMVhFkkRl4"**, "ts": 1407599975, "sjid": 1, "host_id": 1, "ns_id": 678000691, **"client_sjid": 4972967987**}, {"event_type": 1, "attrs": {}, "mtime": 1407304273, **"path": "/test", "is_dir"**: true, "size": 0, "user_id": 325938066, "blocklist": "", "ts": 1407600191, "sjid": 2, "host_id": 2576385874, "ns_id": 678000691, "client_sjid": 9267935283}, {"event_type": 3, "attrs": {}, "mtime": -1, "path": "/test/.a.txt.swp", "is_dir": false, "size": -1, "user_id": 325938066, "blocklist": "", "ts": 1407600563, "sjid": 4, "host_id": 2576406241, "ns_id": 678000691, "client_sjid": 17857869875}, {"event_type": 1, "attrs": {}, "mtime": 1407304645, **"path": "/test/a.txt", "is_dir": false**, "size": 7, "user_id": 325938066, "blocklist": "-PQmVCxoVGsWhguOWL2lh-nPTpIp7e7TksZGmxhTnJ4", "ts": 1407600563, "sjid": 5, "host_id": 2576406241, "ns_id": 678000691, "client_sjid": 22152837171}, {"event_type": 3, "attrs": {}, "mtime": -1, "path": "/test/.b.txt.swp", "is_dir": false, "size": -1, "user_id": 325938066, "blocklist": "", "ts": 1407602124, "sjid": 10, "host_id": 2576494466, "ns_id": 678000691, "client_sjid": 43627673651}, {"event_type": 2, "attrs": {}, "mtime": 1407306207, "path": "/test/b.txt", "is_dir": false, "size": 18, "user_id": 325938066, "blocklist": "PkGP-VCdxXxUo7-BCP4qkYMwvZQftsPcPXEEbg0yBCQ", "ts": 1407602125,

```
"sjid": 11, "host_id": 2576494466, "ns_id": 678000691, "client_sjid": 47922640947}],
"pubserver": "dl.dropboxusercontent.com", "ret": "ok", "max_inline_attr_size": 4095, ...
0
```

Fig. 5. Dropbox List request/response

As in any file system, each file has an associated list of blocks (of maximum 4 MB), with each block having an associated hash which uniquely identifies it. In Fig. 5 we can see that several file names are returned in the JSON response, each file having an associated block list.

The actual contents of the files are obtained via the Retrieve Batch request which will retrieve a specified list of file blocks. An example of such a request and its response is presented in Fig. 6.

```
POST /retrieve_batch HTTP/1.1
...

host_id=58a29b6911f683d39f77d906d9cec063&hashes=%5B%5B%22-PQmVCxoVGsWh
guOWL2Ih-nPTpIp7e7TksZGmxhTnJ4%22%2C+null%2C+null%5D%2C+%5B%22PkGP
-VCdxXxUo7-BCP4qkYMwvZQftsPcPXEEbg0yBCQ%22%2C+null%2C+null%5D%2C+%5
B%22pBTCVnmA6QJnyEaeJHS_Gd4-FqssClN_XbMVhFkkRl4%22%2C+null%2C+null%5
D%5D
```

```
HTTP/1.1 200 OK
...

3ea6
{"hash": "PkGP-VCdxXxUo7-BCP4qkYMwvZQftsPcPXEEbg0yBCQ", "len": 21}
<gzip encoded contents of length 21>
{"hash": "-PQmVCxoVGsWhguOWL2Ih-nPTpIp7e7TksZGmxhTnJ4", "len": 13}
<gzip encoded contents of length 13>
```

Fig. 6. Dropbox Retrieve Batch request/response

Pushing file contents and changes to the cloud service

The file contents of newly added files are pushed in a similar way to the previous Retrieve Batch request. The Store Batch request/response is presented in Fig. 7.

```
POST /store_batch HTTP/1.1
Host: dl-client594.dropbox.com
Accept-Encoding: identity
Content-Type: multipart/form-data; boundary=----------------------------1407631214627
Content-Length: 36033
X-Dropbox-Locale: en_US
User-Agent: DropboxDesktopClient/2.10.27 (Linux; 3.8.0-29-generic; i32; en_US)

------------------------------1407631214627
```

```
Content-Disposition: form-data; name="host_id"

6925bf34850a552594c6c23c40bc1a80
-------------------------------1407631214627
Content-Disposition: form-data; name="batch_info_version"

2
-------------------------------1407631214627
Content-Disposition: form-data; name="batch_info"

[{"advisory_nses": [678000691], "hash":
"HCDOaVCVgUHqzOaAysnhcogmHjsbV9jH3Q8YvROPQrM", "parent": null}]
-------------------------------1407631214627
Content-Disposition: form-data;
name="HCDOaVCVgUHqzOaAysnhcogmHjsbV9jH3Q8YvROPQrM";
filename="HCDOaVCVgUHqzOaAysnhcogmHjsbV9jH3Q8YvROPQrM"
Content-Type: application/octet-stream

<gzipped block contents>
```

```
HTTP/1.1 200 OK
...

40
[["HCDOaVCVgUHqzOaAysnhcogmHjsbV9jH3Q8YvROPQrM", {"ret": "ok"}]]
0
```

Fig. 7. Dropbox Store Batch request/response

After the file blocks are uploaded through the Store Batch request, a Commit Batch request followed by a Close Changeset request are used in order to store the changes atomically. The requests/response for these are presented in Fig. 8 and Fig. 9.

```
POST /commit_batch HTTP/1.1
...

autoclose=&changeset_map=&host_id=6925bf34850a552594c6c23c40bc1a80&commit_i
nfo=eJxtjkFrwkAQhf-K5FxsEksSexPqwUJb8FQoZdkks-nobDbsTmyq-N-7sdoE9DLwv
vdm3nwcgkZaqFnkZlotoePgcVK3RHeTiyOZrRvoOBi8tbCl4m1b0vNTDs16pXbZbiNWz
WzT4U-4X6y_X6t8-Y7qJfC7mIGD34sewjSZRXEWn2r4q791z-D4NKbccR9HJ0q03IOSH
HjA0lbAoh69c_nuEJTWNLnpRGGIoGA0taiwHJLQoRrUf1pqsFK0DRlZXtsKCQTeMHT
LMFAti0EoGqvzwvUFltUZHj2t3V9NkmZhGCbzyDOH-74jnR8_fwHg3lx8&allow_guid_sji
d_hack=0&paired_host_id=None&extended_ret=True
```

```
HTTP/1.1 200 OK
```

```
...

bc
{"chillout": 4.097595663856157e-08, "changeset_id": {}, "results": ["nb"], "need_blocks":
["OueA12kudlJDbepRIfv8vj_Ip3jxiy0zARwNgbEXifM"], "delta_info": {"ns": {}, "user": {},
"team": {}}}
0
```

Fig. 8. Dropbox Commit Batch request/response

```
POST /close_changeset HTTP/1.1
...

host_id=6925bf34850a552594c6c23c40bc1a80&changeset_id=406454814&ns_id=678000
691
```

```
HTTP/1.1 200 OK
...

30
{"chillout": 8.031287501158066e-08, "ret": "ok"}
0
```

Fig. 9. Close Changeset request/response

The commit info parameter present in the Commit Batch request contains a Base64 encoded gzipped JSON string. Its structure is presented in Fig. 10.

```
[{"parent_blocklist": null, "parent_attrs": null, "blocklist":
"HCDOaVCVgUHqzOaAysnhcogmHjsbV9jH3Q8YvROPQrM", "mtime": 1407631213,
"path": "/test/14577.txt.utf-8", "is_dir": false, "target_ns": null, "attrs":
{"dropbox_collection_gid": null, "exif": null, "dropbox_camera_upload": null,
"dropbox_file_id": null, "dropbox_mute": null, "mac": null, "flac": null, "dropbox": null,
"dropbox_tag": null}, "ns_id": 678000691, "size": 101517}][{"parent_blocklist": null,
"parent_attrs": null, "blocklist":
"HCDOaVCVgUHqzOaAysnhcogmHjsbV9jH3Q8YvROPQrM", "mtime": 1407631213,
"path": "/test/14577.txt.utf-8", "is_dir": false, "target_ns": null, "attrs":
{"dropbox_collection_gid": null, "exif": null, "dropbox_camera_upload": null,
"dropbox_file_id": null, "dropbox_mute": null, "mac": null, "flac": null, "dropbox": null,
"dropbox_tag": null}, "ns_id": 678000691, "size": 101517}][{"parent_blocklist": null,
"parent_attrs": null, "blocklist": "OueA12kudlJDbepRIfv8vj_Ip3jxiy0zARwNgbEXifM",
"mtime": 1407631282, "path": "/test/test.txt", "is_dir": false, "target_ns": null, "attrs":
{"dropbox_collection_gid": null, "exif": null, "dropbox_camera_upload": null,
"dropbox_file_id": null, "dropbox_mute": null, "mac": null, "flac": null, "dropbox": null,
"dropbox_tag": null}, "ns_id": 678000691, "size": 79}][{"parent_blocklist": null, "parent_attrs":
null, "blocklist": "OueA12kudlJDbepRIfv8vj_Ip3jxiy0zARwNgbEXifM", "mtime": 1407631282,
"path": "/test/test.txt", "is_dir": false, "target_ns": null, "attrs": {"dropbox_collection_gid": null,
```

"exif": null, "dropbox_camera_upload": null, "dropbox_file_id": null, "dropbox_mute": null, "mac": null, "flac": null, "dropbox": null, "dropbox_tag": null}, "ns_id": 678000691, "size": 79}]

Fig. 10. Commit Info JSON structure

Emulating Dropbox Traffic with Ixia ATI

With our lastest ATI strikepack we now support emulating the Dropbox file synchronization protocol. The protocol allows configuration of actual file contents that get uploaded/downloaded as well as other useful parameters found throughout the request/responses.

All of the presented requests/responses are supported in our protocol implementation as well as a few extra ones so update your ATI strikepack and start testing your network with Dropbox traffic.

References

[1] https://www.dropbox.com/

[2] Kholia, Dhiru, and Przemyslaw Wegrzyn. "Looking Inside the (Drop) Box."*WOOT*. 2013. http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/12058-woot13-kholia.pdf

[3] http://dynamorio.org/