MegaFS, a FUSE filesystem wrapper for Mega. Part 1: Listing files.

In this series of articles, we will implement a FUSE filesystem wrapper for Mega, that will allow us to mount our Mega space on Linux. FUSE (Filesystem in userspace) is a kernel module allowing to create our own filesystems directly in userspace, without editing kernel code. Implementing a filesystem wrapper for FUSE in Python is really simple thanks to *fuse-python* (*aptitude install python-fuse*). We only have to subclass *fuse.Fuse* to implement our filesystem operations and add a few lines of boilerplate code:

```
import fuse

fuse.fuse_python_api = (0, 2)

class DummyFS(fuse.Fuse):
  def __init__(self, *args, **kw):
   fuse.Fuse.__init__(self, *args, **kw)

  def getattr(self, path):
    return 0

  def readdir(self, path):
    return 0

  def open(self, path, flags):
    return 0

  def read(self, path, length, offset):
    return 0

  # ...
fs = DummyFS()
```

```
fs.parse(errex=1)
fs.main()
```

For more information about FUSE, feel free to read the Wikipedia article, the FUSE Python tutorial, and the FUSE simple filesystem howto.

In this first article, we will focus on showing the list of our files, and having a filesystem we can *cd* and *ls* around in. To achieve this, we'll have to implement two functions:

- *getattr()*, to get the attributes of a file (type, size, creation/access/modification time, owner, permissions…)
- *readdir()*, to list the contents of a directory.

But first, we need to think about the data structure that will hold our filesystem structure. Let's keep things simple and just use a dictionnary mapping paths to file objects. The file objects are the ones returned by the API *f* method (see my previous article for a reminder), with the attributes decrypted and the key decomposed into its three parts (*k*, *iv* and *meta_mac*). The directories will contain a additional entry *children* listing their contents.

```
{'/Cloud Drive': {
  'a': {'n': 'Cloud Drive'},
  'children': ['file1.mp3', 'file2.png', 'lol'],
  'h': 'hash1',
  'k': '',
  'p': '',
  't': 2,
  'ts': 1234567890,
  'u': 'user1'
},
'/Cloud Drive/file1.mp3': {
  'a': {'n': 'file1.mp3'},
  'h': 'hash2',
  'k': (12345, 67890, 54321, 09876),
  'iv': (12345, 67890, 0, 0),
  'meta_mac': (12345, 67890),
  'p': 'hash1',
  's': 12345,
```

```
  't': 0,
  'ts': 1234567890,
  'u': 'user1'
},
'/Cloud Drive/file2.png': {
  'a': {'n': 'file2.png'},
  'h': 'hash3',
  'k': (12345, 67890, 54321, 09876),
  'iv': (12345, 67890, 0, 0),
  'meta_mac': (12345, 67890),
  'p': 'hash1',
  's': 12345,
  't': 0,
  'ts': 1234567890,
  'u': 'user1'
},
'/Cloud Drive/lol': {
  'a': {'n': 'lol'},
  'children': ['lol1.png'],
  'h': 'hash4',
  'k': (12345, 67890, 54321, 09876),
  'p': 'hash1',
  't': 1,
  'ts': 1234567890,
  'u': 'user1'
},
'/Cloud Drive/lol/lol1.png': {
  'a': {'n': 'lol1.png'},
  'h': 'hash5',
  'iv': (12345, 67890, 0, 0),
  'k': (12345, 67890, 54321, 09876),
  'meta_mac': (12345, 67890),
  'p': 'hash4',
  's': 12345,
  't': 0,
```

```
    'ts': 1234567890,
    'u': 'user1'
  }}
```

To build this dict, we can simply iterate over the list of files returned by the API. But in order to add a file to its parent's list of children, we need to create the parent entry in the dict before the child entries. We could have sorted the dict to ensure that (sort of topological sort), but instead we will simply test if the parent entry exists when adding a child, and create it if necessary.

```
for file_h, file in self.client.getfiles().items():
  path = self.getpath(files, file_h)
  dirname, basename = os.path.split(path)
  if not dirname in self.files:
    self.files[dirname] = {'children': []}
  self.files[dirname]['children'].append(basename)
  if path in self.files:
    self.files[path].update(file)
  else:
    self.files[path] = file
    if file['t'] &gt; 0:
      self.files[path]['children'] = []
```

We now need to get the path associated with a file, that its, its name concatened with the name of all its parents (with a "/" delimiter between them). The only trick is that Mega allows files in the same directory to have the same name (and thus, the same path). But we need the paths to be unique, so we will remember all the computed paths, and add a suffix to files that collide with other files. Thus, we will end up with *file.ext*, *file (1).ext*, *file (2).ext*, etc.

```
def getpath(self, files, hash):
  if not hash:
    return ""
  elif not hash in self.hash2path:
    path = self.getpath(files, files[hash]['p']) + "/" + files[hash]['a']['n']

    i = 1
    filename, fileext = os.path.splitext(path)
    while path in self.hash2path.values():
      path = filename + ' (%d)' % i + fileext
      i += 1
```

```
    self.hash2path[hash] = path.encode()
  return self.hash2path[hash]
```

A little reminder about the client.getfiles() function, slightly modified from the previous article to add the decrypted attributes and key to the file objects, and wrap it into a class (see megaclient.py on GitHub):

```
def getfiles(self):
  files = self.api_req({'a': 'f', 'c': 1})
  files_dict = {}
  for file in files['f']:
    if file['t'] == 0 or file['t'] == 1:
      key = file['k'][file['k'].index(':') + 1:]
      key = decrypt_key(base64_to_a32(key), self.master_key)
      if file['t'] == 0:
        file['k'] = (key[0] ^ key[4], key[1] ^ key[5], key[2] ^ key[6], key[3] ^ key[7])
        file['iv'] = key[4:6] + (0, 0)
        file['meta_mac'] = key[6:8]
      else:
        file['k'] = key
      attributes = base64urldecode(file['a'])
      attributes = dec_attr(attributes, file['k'])
      file['a'] = attributes
    elif file['t'] == 2:
      self.root_id = file['h']
      file['a'] = {'n': 'Cloud Drive'}
    elif file['t'] == 3:
      self.inbox_id = file['h']
      file['a'] = {'n': 'Inbox'}
    elif file['t'] == 4:
      self.trashbin_id = file['h']
      file['a'] = {'n': 'Rubbish Bin'}
    files_dict[file['h']] = file
  return files_dict
```

Now we can implement our FUSE callbacks! Let's start with *getattr()*:

```
def getattr(self, path):
  if path not in self.files:
    return -errno.ENOENT

  file = self.files[path]
```

```
  st = fuse.Stat()
  st.st_atime = file['ts']
  st.st_mtime = st.st_atime
  st.st_ctime = st.st_atime
  if file['t'] == 0:
    st.st_mode = stat.S_IFREG | 0666
    st.st_nlink = 1
    st.st_size = file['s']
  else:
    st.st_mode = stat.S_IFDIR | 0755
    st.st_nlink = 2 + len([child for child in file['children'] if self.files[os.path.join(path, child)]['t'] &gt; 0])
    st.st_size = 4096
  return st
```

Our *files* dict makes it very concise and easy to write. We set the filesystem attributes of the files according to the informations given by the Mega API:

- We only know the last modification time of the file, so we set the creation, access and modification time to the same value.
- We set the other attributes according to the file type:
  - In case of a file, we can fill in the file size, the number of hard links pointing to this file is only 1, and we set some convenient permissions (0666, rw-rw-rw-).
  - In case of a directory, the size is 4096, the number of hard links pointing to it is 2 + its number of sub-directories (the directory itself, the '.' special file inside of it, and all the '..' special files inside its subdirectories). We set 0755 (rwxr-xr-x) permissions.

And now *readdir()*:

```
def readdir(self, path, offset):
  dirents = ['.', '..'] + self.files[path]['children']
  for r in dirents:
    yield fuse.Direntry(r)
```

Very concise too, it simply returns the list of the given directory's children, without forgetting the two special files '.' and '..'.

That's it! We can now test our fresh new filesystem:

```
julienm@rchand:~$ python MegaFS/megafs.py ~/megafs
julienm@rchand:~$ cd megafs
julienm@rchand:~/megafs$ ls -la
total 36
drwxr-xr-x   5 root    root     4096 janv. 29 16:44 .
drwxr-xr-x 101 julienm julienm 20480 janv. 29 16:44 ..
drwxr-xr-x   4 root    root     4096 janv. 19 18:45 Cloud Drive
drwxr-xr-x   2 root    root     4096 janv. 19 18:45 Inbox
drwxr-xr-x   2 root    root     4096 janv. 19 18:45 Rubbish Bin
julienm@rchand:~/megafs$ cd Cloud\ Drive/
julienm@rchand:~/megafs/Cloud Drive$ ls -la
total 612656
drwxr-xr-x 4 root root      4096 janv. 19 18:45 .
drwxr-xr-x 5 root root      4096 janv. 29 16:44 ..
-rw-rw-rw- 1 root root   5951970 janv. 21 12:48 Call Me Maybe vs She Wolf (YaYa Mashup).mp3
-rw-rw-rw- 1 root root     18599 janv. 29 14:09 epicwin (1).png
-rw-rw-rw- 1 root root     18599 janv. 29 14:08 epicwin (2).png
-rw-rw-rw- 1 root root     18599 janv. 28 02:49 epicwin.png
drwxr-xr-x 3 root root      4096 janv. 20 21:25 lol
drwxr-xr-x 2 root root      4096 janv. 28 15:45 lol (1)
-rw-rw-rw- 1 root root     39315 janv. 24 21:58 lulz.png
-rw-rw-rw- 1 root root    270695 janv. 25 15:27 WHOOOHOOOOOO.PNG
julienm@rchand:~$ sudo umount ~/megafs
julienm@rchand:~$
```

Seems to work  All the files belong to *root:root*, because we did not provide a *uid* and *gid* in our *getattr()* method, but we will handle that later. In the next article, we will see how to open and read files, so that we can cp them or open them directly from our Mega mountpoint! Meanwhile, you can find the complete source code of this example and follow the project on GitHub: https://github.com/CyberjujuM/MegaFS.

Google+

This entry was posted in Uncategorized on January 29, 2013 [/web/20140527233312/http://julien-marchand.fr/blog/megafs-a-fuse-filesystem-wrapper-for-mega-part-1-listing-files/] .

---

6 thoughts on “MegaFS, a FUSE filesystem wrapper for Mega. Part 1: Listing files.”

ProlyX
February 1, 2013 at 4:58 pm

Nice Work!
Can’t wait for Part 2 

eternauta2001
February 1, 2013 at 11:47 pm

Nice! Please: cp to and from Mega.

Pingback: MegaFS, a FUSE filesystem wrapper for Mega. Part 2: Reading and writing files. | Julien Marchand • Dev blog

Julien Marchand Post author
February 2, 2013 at 10:46 pm

Part 2 is there, with cp now working (in both directions!): http://julien-marchand.fr/blog/megafs-a-fuse-filesystem-wrapper-for-mega-part-2-reading-and-writing-files/

scavenger
March 23, 2013 at 3:14 pm

can I install OSXFUSE on Fedora ?

Daan
April 12, 2013 at 6:03 pm

Great work! Actually had something of sorts in mind already, given that I happen to have a server with a 100mbit line, but limited storage. This is great for some extra storage!

Very interesting and educational articles overall. Thanks!
