Aletheia Documentation

Daniel Quinn

Feb 08, 2019

Contents

This is the Python port of Aletheia. Details as to why this project exists at all can be found in the Aletheia whitepaper. This documentation relates strictly to how Aletheia is implemented in Python.

So! You've got some files that you want to attach your name & reputation to, or maybe you've got some files that claim to be from someone and you just want to confirm that. Either way, Aletheia can help you out.

CHAPTER 1

Setup

Aletheia is a reasonably simple program that stands on top of a few well-known software packages out there. In order to use it, you'll need to install the requirements first, and then install Aletheia with the Python package manager.

1.1 Requirements

1.1.1 System Dependencies

Aletheia requires `libmagic`, which comes standard with most Linux & BSD systems, but may need to be installed on your system if you're running OSX or Windows. For OSX, you can install this with Homebrew with `brew install libmagic`, but I'm not sure what happens with Windows.

In addition to libmagic, Aletheia needs to have the ability to talk to two external programs:

- FFmpeg: used to read & write audio & video files.
- exiftool: used to read & write image files.

Installing both of these is easy on any platform though, even Windows ;-)

The download & installation instructions for your operating system of choice can be found on their respective project pages. Once that's finished, and you can successfully execute `ffmpeg` and `exiftool` on the command line, you're ready to install Aletheia.

1.1.2 Python

Aletheia was written for modern versions of Python, so you'll need Python 3.5 or higher to get things running. If you're stuck using a system without a modern version available, the pyenv project provides a handy means of getting modern python on any system that can run Bash.

1.2 Installation

As Aletheia is just a Python package, installing it is easy with pip:

```
$ pip install aletheia
```

This will download the package and install it for you. Along with the Python library (so you can `import aletheia`), you also get the command line program, which you can call like this:

```
$ aletheia generate
$ aletheia sign /path/to/file.jpg example.com
$ aletheia verify /path/to/file.jpg
```

See *The Command Line API* for more information about how to use Aletheia on the command line, and the *The Python API* for how to use it in your own scripts.

CHAPTER 2

QuickStart

Getting up & running with Aletheia doesn't take long at all the process is simple:

1. Check out *Setup* to install Aletheia on your system.
2. Use the `aletheia` command to generate your public and private keys. Have a look at *The Command Line API* for details on that.
3. Put your public key on the web (details below)
4. Sign your files using the `aletheia` command.

If you only want to use Aletheia to verify stuff you find online, you don't event need to worry about steps 2-4.

2.1 A Little More Information

Let's go over steps 2-4 a little closer as step 1 is pretty well covered in *Setup*.

2.1.1 Generating Your Keys

Aletheia allows you to attach your authorship to a file through a process called *public key cryptography*. The process is pretty simple:

1. **You use Aletheia to create two files: a *private key* and a *public key*.**
    - You keep your private key safe and don't share it with anyone.
    - You put your public key on your webserver where Aletheia knows to look for it.
2. You use Aletheia to *"sign"* your files. This tags them in such a way that other people can then use Aletheia to verify the file came from you.

Key generation is a one-step process. Just open a shell and type this:

```
$ aletheia generate
```

That'll take a few moments. When you're done, you have to decide where you want to store your public key. You have two options:

1. In your DNS configuration as a `TXT` record.
2. On your webserver at `/aletheia.pub`.

You only need to do *one* of these, but it doesn't hurt to do both.

Storing Your Key in DNS

As DNS TXT records don't much line line breaks, you should store your key in OpenSSH format. So, the first step is to get your public key in said format:

```
$ aletheia public-key --format=openssh
```

Copy & paste the output of this command into a TXT record for your domain, prefixing it with *aletheia-public-key*=. The result should look something like this:

example.com. 3599 IN TXT "aletheia-public-key=ssh-rsa AAAAB3NzaC1yc2E...

Note that there's an RFC that requires that TXT records not exceed a length of 255 characters, but the work-around is to store the single key as a series of strings on the same record. If you're curious about what this looks like, make sure you've got `dig` installed and have a look at `danielquinn.org`:

```
$ dig danielquinn.org txt
```

Storing Your Key on Your Webserver

As an alternative to DNS, you can also just host your public key on your webserver so long as:

1. The file is accessible at `/aletheia.pub`
2. Your site supports SSL

Just get a copy of your public key:

```
$ aletheia public-key
```

And put the output of that command into a file called `aletheia.pub`. Finally, upload that file to your website. You'll know you've got it right if you can go to `https://yourwebsite.com/aletheia.pub` and the result is your public key.

2.1.2 Signing Your File(s)

Finally, you've got your public key where other people running Aletheia can find it, so now it's time to sign your files. Have a look at *The Command Line API* again for more info, but here's the quick version:

```
$ aletheia sign /path/to/my/file.jpg my-website.com
```

CHAPTER 3

The Command Line API

The command line interface supports 5 different subcommands: `generate`, `sign`, and `verify`, which generate keys, sign files, and verify files respectively, and `public-key` and `private-key` which simply print out the relevant key for you to make use of it.

3.1 Generating Keys

If you're just planning on using Aletheia to verify the origin of a file, then you don't need to generate keys. This command is only for cases when you want to sign a file.

Generation is easy though, as Aletheia takes the complication out of the process. All you have to do is:

```
$ aletheia generate
```

That's it. After a few moments, Aletheia will generate new public and private keys and store them in `${HOME}/.config/aletheia/` by default. The private and public keys will be named `aletheia.pem` and `aletheia.pub` respectively.

3.1.1 Changing the Default Home Directory

By default, Aletheia assumes that you want all of your Aletheia-related files stuffed into `${HOME}/.config/aletheia/`, but you can change that if you like by setting `ALETHEIA_HOME` in your environment:

```
$ ALETHEIA_HOME="/path/to/somewhere/else" aletheia generate
```

3.2 Signing

Once you have some keys generated, you can use them to sign files with the `sign` subcommand. Importantly signing requires two things:

- The path to the file you want to sign.
- The domain to which you're attributing the origin of the file.

```
$ aletheia sign /path/to/file.jpg example.com
```

What we're doing here is writing some instructions to your file that (a) claim authoriship of the file, and (b) tell people where they can find the public key that proves that authorship. This means that **your public key must always be available at that domain** (either via DNS or at `https://example.com/aletheia.pub`) for verification to work.

3.2.1 Domain in the Environment

In cases where you might want to sign a lot of files and don't want to have to specify the domain name in every case (it's likely to be the same every time after all), you can specify the domain in your environment:

```
$ export ALETHEIA_DOMAIN=example.com
$ aletheia sign /path/to/file.jpg
$ aletheia sign /path/to/file.mkv
$ aletheia sign /path/to/file.html
```

3.3 Verification

Verification is easy, but note that it might require an internet connection as Aletheia will attempt to fetch the public key (based on the domain in the signed file) if it hasn't cached it already. By now, you can probably guess what the command looks like:

```
$ aletheia verify /path/to/file.jpg
$ aletheia verify /path/to/file.mkv
$ aletheia verify /path/to/file.html
```

That's all there is to it.

3.4 Getting Your Public & Private Keys

Aletheia provides a handy interface for reading your public & private keys so you can copy/paste the text somewhere useful.

3.4.1 Your Private Key

You can get your private key with the `private-key` subcommand:

```
$ aletheia private key
-----BEGIN RSA PRIVATE KEY-----
MIISKQIBAAKCBAEA0qKTDRq/sPsLLZ+C+kr2eONfKYUZFYYNJ+if2oMKqj8pXr4s
J6qG8Z3FBMlcvx9gmKslByUv68DbGVrH/zBdEU+/XOI3cCqn1+Pblz0r2UDgl97z
7xThq3y6CA1NvI36kcipuzA1HOTMXVdb4voG095CbRo96K+eLXtLpYSvAkzZTCCa
O2UZTcAdb0Nc+BUB3c9GWioLSXADgJKjaqZGMGEGuOKEsHovXc3t+9yNm4Q4YlBl
...
```

3.4.2 Your Public Key

Your public key can be recognised in either `PKCS1` format or `OpenSSH` format. Handily, you can use the `public-key` subcommand to get your key in either format.

To see the default `PKCS1` format, you can call `public-key` without any options, or with `--format pkcs1`:

```
$ aletheia public-key
-----BEGIN RSA PUBLIC KEY-----
MIIECgKCBAEA0qKTDRq/sPsLLZ+C+kr2eONfKYUZFYYNJ+if2oMKqj8pXr4sJ6qG
8Z3FBMlcvx9gmKslByUv68DbGVrH/zBdEU+/XOI3cCqn1+Pblz0r2UDgl97z7xTh
q3y6CA1NvI36kcipuzA1HOTMXVdb4voG095CbRo96K+eLXtLpYSvAkzZTCCaO2UZ
```

For OpenSSH format, use `--format openssh`:

```
$ aletheia public-key --format openssh
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAEAQDSopMNGr+w+wstn4L6SvZ4418phRkVhg...
```

CHAPTER 4

The Python API

The Python API is really just a lower-level of access to what's going on in the command-line client. The process however is the same: generate, sign, and verify.

Handily, Aletheia has each of these common processes bundled into simple functions that you can import from `aletheia.utils`.

4.1 Generating Keys

To generate keys, simply import can call `generate()`:

```
from aletheia.utils import generate

generate()
```

Not much to see here. The behaviour is identical to that of the `generate` command on the command line. Similarly, the location of the generated keys can be altered by setting `ALETHEIA_HOME` in your environment. See *The Command Line API* for more details.

4.2 Signing

Like generating, signing is done with a single function. The only difference is that you must pass in two arguments:

- The path to the file you want to sign
- The domain to which you're ascribing authorship

Have a look at *Signing* for an explanation as to the importance of that second argument:

```
from aletheia.utils import sign

sign("/path/to/file.jpg", "my-website.com")
```

4.2.1 Bulk Signing

The process of setting up and tearing down Aletheia every time can be avoided by using `sign_bulk()`:

```
from aletheia.utils import sign_bulk

sign_bulk(
    ("/path/to/file.jpg", "/path/to/file.mkv", "/path/to/file.html"),
    "my-website.com"
)
```

4.3 Verification

By now this should be looking familiar. Verification is also handled by way of a function:

```
from aletheia.utils import verify

verify("/path/to/file.jpg")
```

This will attempt to fetch the public key from the domain stored in the file and then use that public key to verify the file. If you don't have an internet connection available, and the public key hasn't already been cached, this process will fail.

4.3.1 Bulk Verification

Just like signing, verification has a bulk helper function:

```
from aletheia.utils import verify

verify_bulk(
    ("/path/to/file.jpg", "/path/to/file.mkv", "/path/to/file.html"),
)
```

4.4 Optional Keyword Arguments

Each of the above commands will accept a series of keyword arguments that will get passed up to the `Aletheia` class:

- `private_key_path`: The path to the private key you want to generate or use to sign a file.
- `public_key_path`: The path to where your public key should be generated
- `cache_dir`: The path to the directory where you want Aletheia to store all the public keys it caches while verifying files.

4.4.1 Examples

```
from aletheia.utils import generate, sign, verify

generate(
```

(continues on next page)

(continued from previous page)

```
    private_key_path="/path/to/private-key.pem",
    public_key_path="/path/to/public-key.pub",
)
sign(
    "/path/to/file.jpg",
    "my-website.com",
    private_key_path="/path/to/private-key.pem"
)
verify(
    "/path/to/file.jpg",
    cache_dir="/path/to/cache"
)
```

CHAPTER 5

Extending Aletheia

Aletheia is the kind of project that works best if lots of people use it, so any contribution you want to make toward making Aletheia more awesome is always appreciated.

These are the top 3 categories that could use some attention, but even if you just want to fix a typo in the documentation, send a pull request!

5.1 Packaging

In a perfect world, Aletheia should be able to be installed without `pip`, but rather with your operating system's packaging manager. One day, I'd like to see an `apt install aletheia`. If you think that's something you could help with, give it a shot and post to the issue queue if you have questions.

The priority package formats would be:

- Debian (apt)
- Redhat (yum)
- Gentoo (ebuild)
- Homebrew

5.2 Supporting Additional Formats

The library of supported formats is growing all the time, but if you have a particular format in mind that you'd like to see working, Aletheia has been designed to be very extendable. Basically you've got three steps:

1. Create a file in the appropriate folder (either `audio`, `documents`, `images`, or `video`) and in it create a class that subclasses `aletheia.file_types.base.File`.
2. At the very least, your class must define three methods:
    - `.get_raw_data()`: This should return the "data" portion of the file (as opposed to the metadata).

- `.sign()`: While the actual signature generation and formatting of the metadata is done by the `File` class, the `.sign()` method needs to define *how* that metadata is written to your file type.
- `.verify()`: Like `.sign()`, the hard part is done for you. However you still need to define how we can retrieve the metadata from your file type.

3. Write some tests to cover your new file type.

If you're looking for a good place to start for an example, have a look at the `HtmlFile`, `Mp3File`, and `JpegFile` classes. If you need help, just post a question to the issue queue.

5.3 Porting to another language

As not everyone uses Python, it's probably a good idea to try to port Aletheia to other languages. For the most part, this is all just an elaborate wrapper around FFmpeg, so porting shouldn't be all that difficult. Priority languages would include:

- Javascript
- Ruby
- Go
- Rust
- C#

If you like this project and would like to use it in another language, why not give this a shot?

CHAPTER 6

Troubleshooting

6.1 Libmagic

Aletheia depends on libmagic, and uses the `file-magic` module to access it. Unfortunately, libmagic operates differently from environment to environment, and sometimes it can have trouble. In cases when this presents a problem, it'll be documented here.

6.1.1 Alpine Linux

Alpine systems currently require a patch to play nice with Python's `file-magic` module, so you'll have to add the following to your `Dockerfile`:

```
FROM python:3-alpine

...

RUN wget https://patch-diff.githubusercontent.com/raw/python/cpython/pull/
↪10461.patch -O - \
| patch /usr/local/lib/python3.7/ctypes/util.py

ENV LD_LIBRARY_PATH /usr/lib/
```

6.2 FFmpeg

A lot of Aletheia's functions require a working installation of FFmpeg. Thankfully, this program is Free software and is available for all major platforms out there. Installing it should be relatively easy, but if Aletheia is complaining about how you don't have it installed even after you're *sure* you installed it, note that FFmpeg must be installed and available in your system `PATH`.

This means that if you type `ffmpeg -version` on the command line, regardless of what directory you're in, you should see something like this:

```
built with gcc 8.1.1 (GCC) 20180531
configuration: --prefix=/usr --disable-debug --disable-static --disable-stripping --
↪enable-avresample --enable-fontconfig --enable-gmp --enable-gnutls --enable-gpl --
↪enable-ladspa --enable-libaom --enable-libass --enable-libbluray --enable-libdrm --
↪enable-libfreetype --enable-libfribidi --enable-libgsm --enable-libiec61883 --
↪enable-libmodplug --enable-libmp3lame --enable-libopencore_amrnb --enable-
↪libopencore_amrwb --enable-libopenjpeg --enable-libopus --enable-libpulse --enable-
↪libsoxr --enable-libspeex --enable-libssh --enable-libtheora --enable-libv4l2 --
↪enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwebp --enable-
↪libx264 --enable-libx265 --enable-libxcb --enable-libxml2 --enable-libxvid --enable-
↪nvenc --enable-omx --enable-shared --enable-version3
libavutil      56. 14.100 / 56. 14.100
libavcodec     58. 18.100 / 58. 18.100
libavformat    58. 12.100 / 58. 12.100
libavdevice    58.  3.100 / 58.  3.100
libavfilter     7. 16.100 /  7. 16.100
libavresample   4.  0.  0 /  4.  0.  0
libswscale      5.  1.100 /  5.  1.100
libswresample   3.  1.100 /  3.  1.100
libpostproc    55.  1.100 / 55.  1.100
```

If you see `command not found` or `Bad command or file name`, then either FFmpeg isn't installed, or it's not in your `PATH`. You'll have to talk to someone who knows more about your operating system to figure out how to get that to work.

CHAPTER 7

Changelog

7.1 3.0.0

- **Breaking change**: This release contains a considerable rewrite around how we handle FFmpeg-based signing. It turns out that there's a difference between how FFmpeg hashes audio & video files between versions, and this has a direct effect on how we calculate signatures. This change fixes that by ashing the streams directly, and has been tested to work with old and present versions of FFmpeg.
- Handle FFmpeg's inability to write Webm metadata in older versions. Unfortunately, FFmpeg versions older than 3.4 could read metadata from Webm files, but would silently fail to write to them. This release introduces a check for the local FFmpeg version and barks at you if you try to use an old version to write to a Webm file.
- Moved to Gitlab, which allows us to use their fantastic test runner so we can verify operation in multiple operating systems and Python versions in a few lines of YAML. Currently we've got Debian (python:3.5,3.6,3.7), Arch, and Fedora (26,27,28,29).
- Added a cross-platform test script that lets you run all the stuff that Gitlab does, but locally with a single Bash script.
- Move pytest.ini into setup.cfg Additionally, this rolls in marker checks and `-n auto` as defaults.
- Fixed a bug that would allow you to sign a file with a `null` domain, making verification impossible.
- Added some code to better handle cases where Aletheia is installed on operating systems that don't play well with `file-magic`.
- Added more test coverage.

7.2 2.0.2

- Use cleaner exception handling.
- Allow for an update to dnspython without breaking Aletheia.

7.3 2.0.0

Considerable changes have been made to the way we connect a file to a domain. While v1.x assigned ownership of a file to a domain based on the existence of a public key *anywhere* on that domain, v2.x now introduces further restrictions on where that key must be hosted. This is to prevent hosting providers from being implicated by people who have the ability to host files on their platform.

The new rules are a lot simpler though. You can host your public key in only one of two places:

1. On your web server at `https://your-domain.com/aletheia.pub`
2. In a DNS TXT record for your domain. In this case, your public key should be stored in OpenSSH format so it all fits on one line. Have a look at the DNS record for danielquinn.org if you want an example of what this looks like.

A few other features were added as well:

- All keys are now using SHA512 instead of the previous SHA256.
- You can now call `aletheia public-key` to display your public key. Similarly, you can call `aletheia public-key --format=openssh` for the aforementioned OpenSSH formatting required for DNS storage.
- You can also call `aletheia private-key` to display your private key.
- We now check the schema version in the verification step and error out if the version is in the future.

7.4 1.1.0

- Added support for Markdown files.

7.5 1.0.1

- Added a `--version` flag to the command line interface.
- Added some performance tweaks to how we're calling exiftool.
- Updated various exceptions to include a little more information about what went wrong.

7.6 1.0.0

- Use of Pillow and piexif have been dropped in favour of exiftool. This was due largely to the fact that Pillow's `.tobytes()` method performs differently from environment to environment, making Aletheia's job quite impossible. Standardising on exiftool means reproducible results regardless of what operating system you're using. Unfortunately, this also means that files signed with past versions of Aletheia will fail a verification check in this new version.
- GIF and PNG files are now supported, thanks to the inclusion of exiftool.
- The tests were restructured to handle a multi-threaded test environment better. You can now run the tests with `pytest -n auto` on multi-cored machines for a significant speed improvement.

7.7 0.6.4

- Bugfix release: Attempting to run Aletheia on a video file will now no longer explode with a traceback if FFMpeg isn't installed.

7.8 0.6.3

- Changed the dependency on file-magic to require a minimum of v0.3.0 rather than v0.4.0. This is to make packaging for Arch Linux easier.
- Added a new function to `setup.py` to automatically generate a PKGBUILD file.
- Removed `scripts/*` from the `MANIFEST` file as that directory is no longer used.

7.9 0.6.2

- Added support for Python 3.5. Aletheia now supports CPython 3.5, 3.6, 3.7, and PyPy 3.5 v6.0.0

7.10 0.6.1

- Switched to using file-magic instead of python-magic. The effects & performance are the same, but file-magic appears to be more commonly used in different Linux distros and I'd like for packaging to be as easy as possible.
- Added tox tests for Python 3.7

7.11 0.6.0

- We now make use of FFmpeg's hashing features rather than trying to determine a "safe" way of drawing out the raw data from a file.
- Support for MKV files added.
- Support for Webm files added.
- The means of determining file type now includes support for guessing from file suffixes.

7.12 0.5.0

- Support for HTML files added.

7.13 0.4.0

- After some tinkering with a few alternatives, FFmpeg is now the standard way to generate the "raw data" component for audio & video.
- Support for MP4 files added.

7.14 0.3.4

- Fix a bug in environment variable referencing for public key URL.

7.15 0.3.3

- Error out gracefully if we attempt to verify a file that doesn't contain a signature.

7.16 0.3.2

- Prettied up the CLI output with a few emojis and colours.

7.17 0.3.1

- Add tox to help get us to a point where Python 2.7 is supported
- Fix a bug in the shebang in the CLI script and modify setup.py to use `entry_points=` instead of `scripts=` as the latter method had a tendency to overwrite the shebang line in the `aletheia` script.
- Lastly, we now have Even More Tests.

7.18 0.3.0

- Added colours to the output of the command-line script. This means a new dependency on the `termcolor` library.
- **Breaking**: `verify()` now raises various exceptions on failure rather than simply returning `False`. This was done to allow the command-line script to show useful error messages.
- The command-line script is a lot more helpful now in terms of error messages.

7.19 0.2.0

- Dropped support for signing JpegImageFile objects. The process was ugly and the overhead less-than-awesome. Signing image files can still be done the standard way though: by operating on the file rather than the PIL object.
- More tests!

7.20 0.1.0

- Support for MP3 files
- You can now sign & verify images by either specifying a file name or passing in a `Pillow.JpegImageFile` instance.
- Location of the signature data in JPEG images was moved to `ImageIFD.HostComputer`.

- Dropped pyexiv2 and added mutagen & piexif as dependencies.

7.21 0.0.3

- A working implementation of Aletheia for JPEG images.