Meta-data anonymizing toolkit for file publication

jvoisin

Hello,
I am Julien Voisin, undergraduate computer science student from France.

I am interested to work on the “Meta-data anonymizing toolkit for file publication” project.
I know there is already a student interested in by this project, but I really want to do it :
I needed it for my own and have already thought about a potential design some time ago.

I would like to work for the EFF, because I am very concerned about privacy issues on the Internet.
I think privacy is an essential right, and not just an option.
Especially, I would really enjoy to work for the Tor project (or Tails, since it’s heavily based on him).
I am using it for quite some time and would like to get more involved and contribute back!

I use F/OSS on a daily basis (Ubuntu, Debian, Archlinux and Gentoo).
So far my major contributions were the writing of documentations for archLinux, openmw, xda-forum and Ubuntu.
Recently I have released a little matrix manipulation library written in C,
originally for an academic project (http://dustri.org/lib/).
I am interested to do the debian package, but heard that it can be quite tricky, so a little
help would be much appreciated

I do not have any major plan for this summer (but my holidays only begins the june 4th), so I can fully focus on the project and reasonably think that I could commit 5-6 hours per day on it.

Requirement/Deliverables:

- A command line and a GUI tool having both the following capabilities (in order of importance):
    - Listing the metadatas embedded in a given file
    - A batch mode to handle a whole directory (or set of directories)
    - The ability to scan files packed in the most common archive formats
    - A nice binding for srm (Secure ReMoval) or shred (GNU utils) to properly remove the original file containing the evil metas
    - Let the user delete/modify a specific meta

- Should run on the most common OS/architectures (And especially on Debian Squeeze, since Tails is based on it.)
- The whole thing should be easily extensible (especially it should be easy to add support for new file formats)
- The proper functioning of the software should be easily testable

I’d like to do this project in Python, because I already have done some personal projects whith it (for which I also used subversion) : an IRC bot tailored for logging (dustri.org/tor/depluie.tar.bz2 still under heavy WIP), a battery monitor, a simple search engine indexing FTP servers, ...

Why is Python a good choice for implementing this project ?

1. I am experienced with the language
2. There are plenty of libraries to read/write metadatas, among them is Hachoir (https://bitbucket.org/haypo/hachoir/)that looks very promising since it supports quite a few file formats
3. It is easy to wrap other libraries for our needs (even if they are not written in Python !)
4. Runs on almost every OS/architecture, what is a great benefit for portability
5. It is easy to make unit tests (thanks to the built-in Unittest module)

Proposed design:

The proposed design has three main components : one lib, a command line tool and a GUI tool.

The aim of the library (described with more details in the next part) is to make the development of tools easy. A special attention will be made on the API that it exposes. The ultimate goal being to be able to add the support of new file format in the library without changing the whole code of the tools.

Meta reading/writing library :

A library to read and write metas for various file formats. The main goal is to provide an abstraction interface (for the file format and for the underlying libraries used).
At first it would only wrap Hachoir.
Why hachoir :

- Autofix: Hachoir is able to open invalid / truncated files
- Lazy: Open a file is very fast since no information is read from file, data are read and/or computed when the user ask for it

- Types: Hachoir has many predefined field types (integer, bit, string, etc.) and supports string

with charset (ISO-8859-1, UTF-8, UTF-16, ...)
- Addresses and sizes are stored in bit, so flags are stored as classic fields
- Editor: Using Hachoir representation of data, you can edit, insert, remove data and then save in a new file.
- Meta : Support a very large scale of file format

But we could also wrap other libraries to support a particular file format. Or write ourself the support for a format, although this should be avoided if possible (it looks simple at first, but supporting different versions of the format and maintaining the thing over time is extremely time consuming)
The must would be to make the children libraries optional dependencies.

One typical use case of the lib is to ask for metadatas for a file, if the format is supported a list (or maybe a tree) of metas is returned.

Both the GUI and the cmdline tool will use this lib.

The cmdline/GUI tool features:

- List all the meta
- Removing all the meta
- Anonymising all the meta
- Let the user chose wich meta he wants to modify
- Support archives anonymisation
- Secure removal
- Cleaning wholes folder recursively

GUI:
Essentially the GUI tool would do the same features as for the cmd line too.
I do not have a significant GUI development experience, but I'm planing to fix that point during community bonding period.

Proposed development methods:
One way to develop this would be to do it layer by layer : first implementing the meta reading/writing lib for ale the formats in one shot, then making the command line application, …

However for this project, developing feature by feature seems more appropriate :
Starting by a skeleton implementing a thin slice of functionality that traverses most of the layers.
For example, I could start by focussing only on EXIF metas : make sure that the meta reading/writing library supports EXIF, then make the command line tool using the previous library.
And only then, when the skeleton is actually working, add supports for other features/formats.

This allows a more incremental development flow and after only a few weeks I would be able to deliver a working system. The list of features supported in the first iterations would be ridiculously short but at least that would enable me to get feedbacks and notice quickly if I am on the wrong tracks.
Also, since I would add one feature at a time, the structure of the system will tend to easily accommodate that. Thanks to this, adding the support of a new file format will be made easy, even after the end of the GSOC.

Testing

For such a tool, since the smallest crack could compromise the user, testing is critically important. I plan to implement two main kind of testings :

Unit tests

To test the proper working of the meta accessing library. Maintaining a collection of files with metas for every format we should support. Using Python's unittest I can setup expectations to make sure that the library will finds the good fields.

End-to-End testing:

A script (or set of script) to test the proper working of the command line tool. One way is to run the tool in batch mode on the input test files set. If in the output we are still able to find metas, the system is not doing his job right.
It is possible to write this script in Python, possibly using again the unittest lib (even if here the goal is to execute an external executable and see how it's behaving) to get output of the results that is consistent with the unit tests.

The goal is to be able to automatically test the whole system in a few commands (one for the unit tests one for the end-to-end tests). It will be easy to add a new document in the test set, so if someone from the community provides a file that was not cleaned properly, we can easily reproduce the problem and then decide on the proper action to take. The ability to tests everything easily might help a user to make sure that everything works fine on his OS/architecture. Also, if for one reason after some months it is decided to change from Hachoir to another underlying library (yes a pretty radical decision, just given as example) we still have the tests, so valuable to check than everything is still working with the new library.

Timeline:

- Community Bonding Period (in order of importance)
    - Playing around with pygobject
    - Playing with Hachoir
    - Learning git

- First two weeks :
  - create the structure in the repository (directories, README, ..)
  - Create a skeleton
  - Objectives : to have a deployable working system as soon as possible(even if the list of features is ridiculous). So that I can show you my work in an incremental way thereafter and get feedbacks early.
  - The lib will handle reading/writing EXIF fields (using Hachoir)
  - A set of tests files (and automated unit tests) to demonstrate that the lib does the job
  - The beginning of the command line tool, at this point must list and delete EXIF meta
  - An automated end-to-end test to show that the command line tool does properly remove the EXIF

After this first step (making the skeleton) I should be able to deliver a working system right after adding each of the following features. I Hope to get feedbacks so can fix problems quickly

- 3 weeks
  - adding support for (in order of importance) pdf, zip/tar/bzip (just the meta, not the content yet), jpeg/png/bmp, ogg/mpeg1-2-3, exe...
  - For every type of meta, that involves :
    - Creating some input test files with meta data
    - Implementing the feature in the library
    - Asserting that the lib does the job with unit tests
    - Modifying the cmd line tool to support the feature (if necessary)
    - Checking that the cmd line tool can properly delete this type of meta with automated end-to-end test

- about one day
  - Enable the command line tool to set a specific meta to a chosen value

- about 1 day
  - Implementation of the “batch mode” in the cmdline tool, to clean a whole folder
  - Implementation of secure removal

- about 2 days :
  - Add support for deep archive cleanup
    - Clean the content of the archives
    - Make a list of non supported format, for which we warn the user that only the

container can be cleaned from meta, not the content (at first that will include rar, 7zip, ..)
        - The supported formats will be those supported natively by Python (bzip2, gzip, tar)
        - Create some test archives for each supported format containing various files with metas
        - Implement the deep cleanup for the format
        - Assert that the command line passes the end-to-end tests (that is, it can correctly clean the content of the test archives)

- <u>about 2 days</u>
    - Add support for complete deletion of the original files
    - Make a binding nice for shred (should not be to hard using Python)
    - Implement the feature in the command line tool

- <u>3 weeks</u>
    - Implementation of the GUI tool
    - At this stage, I can use the experience from implementing the cmdline tool to implement the GUI tool, having the same features.

- <u>1 week</u>
    - Add support for more format (might be based on requests from the community)

- <u>Remaining weeks</u>
    - I want to keep those remaining week in case of problems, and for
        - Remaining/polishing cleanup
        - Bugfixing
        - Integration work
        - Missing features
        - Packaging
        - Final documentation

- <u>Every Week-end :</u>
    - Documentation time : both end-user, and design. I do not like to document my code while I'm coding it : it slows a lot the development process, but it's not a good thing to delay it too much : week-ends seems fine for this.
    - A blog-post, and a mail on the mailing list about what I have done in the week.

About the anonymisation process :

Since I'll relay on Hachoir in a first time, I don't know how much it's effective on every case.
I am planing to do some tests during the Community Bonding Period.

The plan is to first rely on the capabilities of Hachoir. I don't know yet how effective it is for each case. I am planing to do some tests during the Community Bonding Period. Following the test strategy described before, it will be easy to add a new document in the test set. If I could make a test file with metas not supported by Hachoir (or someone from the community provide such a file), we could then decide on the proper action : propose a patch to Hachoir or use another library for this specific format.

Doing R&D about supporting exotics fields, or improve existing support will depend of my progress. Speaking of the fingerprints, the subject of the project is “metadata anonymisation”, and not “fingerprint detection” : they are too many softwares, too many subtle ways to alter/mark a file (and I don’t even speak of steganography).

So, nop, I’m not planing to implement fingerprinting detection. Not that it is not interesting, it’s just that it’s not realistic to support it correctly (actually it’s not realistic to support it at all, given that I must first support the metas, in such a short time frame). Would you agree that it is better to first focus on making a working tool that does the job for the known metadatas ?

That done, if the design is good we could easily add support for more exotic fields (or some kind of fingerprinting). I think we should never loose track of the sad truth : no matter how big the effort we spend on this, making a comprehensive tool able of detecting every kind of meta and every pattern of fingerprinting is just not feasible.

About archives anonymisation

Task order for anonymisation of an archive :

a. Extract the archive
b. Anonymise his content
c. Recompress the archive
d. Securely deleting the extracted content
e. Anonymise the new created archive

And I’m planing to support the extraction only of archive formats supported by python (without this limitation, the tool will not be portable). If the user try to anonymize a .rar for example, the tool will popup “WARNING - only the container will be anonymize, and not the content - WARNING”

As for what I expect from my mentor, I think he should try to be available (not immediately but in the 48 hours) when I need him specifically (e.g. technical questions no one else on IRC can answer) but he doesn’t need to check on me all the time. I’m fine with mails to (since intrigeri is not always “online”, it will be I think, the best solution to communicate with him). I’d like to do a weekly reunion, on irc or jabber, to discuss things more smoothly.

I'm using irc quite a lot, and I’m hanging on #tor, and #tor-dev (nick : jvoisin).
I'm planing to do a blogpost every week-end, about the advancement of the project.
You can mail me at : julien.voisin@dustri.org for more information.