Viper Documentation

Release 1.2

Claudio Guarnieri

Jun 14, 2018

Contents

CHAPTER 1

What is Viper?

Viper is a binary analysis and management framework. Its fundamental objective is to provide a solution to easily organize your collection of malware and exploit samples as well as your collection of scripts you created or found over the time to facilitate your daily research. Think of it as a *Metasploit* for malware researchers: it provides a terminal interface that you can use to store, search and analyze arbitraty files with and a framework to easily create plugins of any sort.

Viper is released under BSD 3-Clause license and is copyrighted by Claudio Guarnieri. The source code is available on GitHub, where also all development efforts and contributions are coordinated. For questions and inquiries, you can find the author's contact details here.

CHAPTER 2

Table of Content

2.1 Installation

Viper is written in Python and **Python >= 3.4 is recommended** (python 2.7 *should* still work) to function properly. In this documentation we will use Debian GNU/Linux based distributions, such as Ubuntu, as a reference platform. The following installation instructions should apply similarly to other distributions and possibly to Mac OS X as well, although it has not been properly tested.

Before proceeding, you should make sure you have the basic tools installed to be able to compile additional Python extensions:

```
$ sudo apt-get install gcc python3-dev python3-pip
```

In order to have support for certain modules, you will need to install the following dependencies too before proceeding:

```
$ sudo apt-get install libssl-dev swig libffi-dev ssdeep libfuzzy-dev
```

To install Viper:

```
$ git clone https://github.com/viper-framework/viper
$ cd viper
$ git submodule init
$ git submodule update
$ sudo make install
```

2.1.1 Core dependencies

Viper makes use of a number of Python library for its core functioning, which can be installed with the command:

```
$ sudo pip3 install -r requirements.txt
```

Although it is optional, we recommend that you install YARA pattern matching project and the YARA-Python library by following the setup instructions on their official websites.

Viper can retrieve files remotely through Tor, if you're interested in such feature you should install SocksiPy:

```
$ sudo apt-get install python-socksipy
```

You will also clearly need a running Tor daemon, refer to the official website for setup instructions.

2.1.2 Extra dependencies

Please be aware that all the modules that are available in Viper might have their own dependencies that are unrelated to Viper's core. We will try to make such dependencies available in our `requirements.txt` file that you can provide to `pip`:

```
$ sudo pip3 install -r requirements.txt
```

In case a dependency is missing or it is not available on PyPi, you should be instructed by the module itself on how to retrieve and install it.

2.1.3 First launch

If everything worked out fine, you should be able to launch Viper's shell without raising any exceptions, like following:

```
nex@nex:~/$ ./viper-cli
          _
         (_)
   _   _ _ _____  ______  _____
  | | | | |  _ \| ____ |/ ____)
   \ V /| | |_| | _____| |
    \_/ |_|  __/|______)_| v1.3
          |_|

You have 0 files in your default repository
shell >
```

2.1.4 Uninstall

To uninstall Viper:

```
$ cd viper
$ sudo make uninstall
```

2.2 Usage

2.2.1 Concepts

Before proceeding in learning the functioning of each available command and module, you need to understand some fundamental design concept that represent the foundation of Viper itself.

Projects

Viper allows you to create and operate on a collection of files. One collection represent one **project**.

You can create as many projects as you want and you can easily switch from one to another. Each project will have its own local repositories of binary files, a SQLite database containing metadata and an history file which contains all the commands you provided through Viper's shell exclusively in the context of the opened project.

In this way you can for example create different workbenches for each malware campaign, malware family or threat actor you're investigating. You can also easily pack up and share the whole project folder with your friends and colleagues.

As you can see from Viper's help message, you can specify a project name at startup:

```
nex@nex:~/$ viper-cli -h
usage: viper-cli [-h] [-p PROJECT]

optional arguments:
  -h, --help            show this help message and exit
  -p PROJECT, --project PROJECT
                        Specify a new or existing project name
```

When doing so, Viper will try to open an existing project with the given name and if it doesn't exist it will initialize it under the `projects/` folder.

If you opened a project, it will appear both in a startup message as well as in Viper's terminal:

```
nex@nex:~/$ viper-cli -p test
         _
        (_)
   _   _ _ ____  _____  ____
  | | | | |  _ \| ___ |/ ___)
   \ V /| | |_| | ____| |
    \_/ |_|  __/|_____)_| v1.3
          |_|

You have 0 files in your test repository
test shell >
```

From within the terminal, you can see which projects exist and eventually you can switch from one to another:

```
test1 shell > projects --list
[*] Projects Available:
+--------------+--------------------------+---------+
| Project Name | Creation Time            | Current |
+--------------+--------------------------+---------+
| test2        | Fri Jul 11 02:05:55 2014 |         |
| test1        | Fri Jul 11 02:05:51 2014 | Yes     |
+--------------+--------------------------+---------+
test1 shell > projects --switch test2
[*] Switched to project test2
test2 shell >
```

More details on the `projects` command are available in the *Commands* chapter.

Sessions

Most of commands and especially modules provided by Viper, are designed to operate on a single file, being a Windows executable or a PDF or whatever else.

In order to do so, you'll have to open the file of your choice and every time you do so a new **session** will be created. You'll be able to see the name of the file you opened in the terminal:

```
shell > open 9f2520a3056543d49bb0f822d85ce5dd
[*] Session opened on ~/viper/binaries/2/d/7/9/
↪2d79fcc6b02a2e183a0cb30e0e25d103f42badda9fbf86bbee06f93aa3855aff
shell darkcomet.exe >
```

From then on, every command and module you launch will execute against the file you just opened (if the module requires to do so obviously).

Similarly to the projects, you can just as easily see which sessions you have currently opened:

```
shell darkcomet.exe > sessions --list
[*] Opened Sessions:
+---+----------------+----------------------------------+---------------------+------
↪---+
| # | Name           | MD5                              | Created At          |␣
↪Current |
+---+----------------+----------------------------------+---------------------+------
↪---+
| 1 | blackshades.exe | 0d1bd081974a4dcdeee55f025423a72b | 2014-07-11 02:28:45 |     ␣
↪   |
| 2 | poisonivy.exe   | 22f77c113cc6d43d8c12ed3c9fb39825 | 2014-07-11 02:28:49 |     ␣
↪   |
| 3 | darkcomet.exe   | 9f2520a3056543d49bb0f822d85ce5dd | 2014-07-11 02:29:29 | Yes ␣
↪   |
+---+----------------+----------------------------------+---------------------+------
↪---+
```

You can eventually decide to switch to a different one:

```
shell darkcomet.exe > sessions --switch 1
[*] Switched to session #1 on ~/viper/binaries/1/5/c/3/
↪15c34d2b0e834727949dbacea897db33c785a32ac606c0935e3758c8dc975535
shell blackshades.exe >
```

You can also abandon the current session with the `close` command (the session will remain available if you wish to re-open it later):

```
shell blackshades.exe > close
shell >
```

A session will also keep track of the results of the last `find` command so that you'll be able to easily open new sessions without having to perform repeated searches on your repository. You can find more details about this in the *Commands* chapter.

Please note that if you switch to a whole different project, you'll lose the opened sessions.

Commands & Modules

The operations you can execute within Viper are fundamentally distinguished between **commands** and **modules**. Commands are functions that are provided by Viper's core and enable you to interact with the file repository (by

adding, searching, tagging and removing files), with projects and with sessions. They are static and they should not be modified.

Modules are plugins that are dynamically loaded by Viper at startup and are contained under the `modules/` folder. Modules implement additional analytical functions that can be executed on an opened file or on the whole repository, for example: analyzing PE32 executables, parsing PDF documents, analyzing Office documents, clustering files by fuzzy hashing or imphash, etc.

Modules are the most actively developed portion of Viper and they represent the most important avenue for contributions from the community: if you have an idea or you want to re-implement a script that you have lying around, make sure you submit it to Viper.

Database

The database that stores all meta inforation is per default in an sqlite database stored at:

```
$HOME/.viper/viper.db
```

Binaries

The files are stored in a folder structure within:

```
$HOME/.viper/binaries
```

2.2.2 Commands

Viper provides a set of core commands used to interact repositories of files you want to collect. In order to see which commands are available, type `help`:

```
shell > help
Commands:
+------------+-----------------------------------------------+
| Command    | Description                                   |
+------------+-----------------------------------------------+
| about      | Show information about this Viper instance    |
| analysis   | View the stored analysis                      |
| clear      | Clear the console                             |
| close      | Close the current session                     |
| copy       | Copy opened file(s) into another project      |
| delete     | Delete the opened file                        |
| exit, quit | Exit Viper                                    |
| export     | Export the current session to file or zip     |
| find       | Find a file                                   |
| help       | Show this help message                        |
| info       | Show information on the opened file           |
| new        | Create new file                               |
| notes      | View, add and edit notes on the opened file   |
| open       | Open a file                                   |
| parent     | Add or remove a parent file                   |
| projects   | List or switch existing projects              |
| rename     | Rename the file in the database               |
| sessions   | List or switch sessions                       |
| stats      | Viper Collection Statistics                   |
| store      | Store the opened file to the local repository |
```

(continues on next page)

(continued from previous page)

```
| tags       | Modify tags of the opened file                |
+----------+-----------------------------------------------+
```

Following are details for all the currently available commands.

about

The **about** command can be used to display some useful information regarding the Viper instance you are currently running. This includes the versions of both Viper itself and of your Python installation. Additionally the path of the active configuration file is shown:

```
viper > about
+---------------+-------------------------------------------------+
| About         |                                                 |
+---------------+-------------------------------------------------+
| Viper Version | 1.3-dev                                         |
| Python Version | 3.4.3                                          |
| Homepage      | https://viper.li                                |
| Issue Tracker | https://github.com/viper-framework/viper/issues |
+---------------+-------------------------------------------------+
+-------------------------+-------------------------------------------------+
| Configuration           |                                                 |
+-------------------------+-------------------------------------------------+
| Configuration File      | /home/user/.viper/viper.conf                    |
| Storage Path            | /home/user/.viper                               |
| Current Project Database | Engine(sqlite:////home/user/.viper/viper.db)   |
+-------------------------+-------------------------------------------------+
```

projects

As anticipated in the *Concepts* section, Viper provides a way to create multiple **projects** which represent isolated collections of files. You can create a project by simply specifying a value to the `--project` argument at launch of `viper-cli`.

From within the Viper shell, you can list the existing projects and switch from one to another by simply using the `projects` command. Following is the help message:

```
usage: projects [-h] [-l] [-s=project]

Options:
    --help (-h) Show this help message
    --list (-l) List all existing projects
    --switch (-s)   Switch to the specified project
```

Each project will have its own local file repository, its own `viper.db` SQLite database and its own `.viperhistory` file, which is used to record the history of commands you entered in the terminal.

For example, this is how to launch Viper with a specific project:

```
nex@nex:$ viper-cli --project test1
         _
        (_)
   _   _ _ ____  _____  ____
  | | | | |  _ \| ___ |/ ___)
```

(continues on next page)

(continued from previous page)

```
   \ V /| | |_| | _____| |
    \_/ |_|  __/|______)_| v1.1
          |_|

You have 0 files in your test1 repository
test1 shell >
```

From within the terminal, you can see which projects exist:

```
test1 shell > projects -l
[*] Projects Available:
+--------------+--------------------------+---------+
| Project Name | Creation Time            | Current |
+--------------+--------------------------+---------+
| test1        | Sat Jul 12 00:53:06 2014 | Yes     |
+--------------+--------------------------+---------+
```

You can eventually switch to a different one:

```
test1 shell > projects --switch test2
[*] Switched to project test2
test2 shell >
```

Note that if you specify a name of a project that doesn't exist to the `--switch` parameter, Viper will create that project and open it nevertheless.

open

As explained in the *Concepts* chapter, Viper supports the concept of **session**, which is an execution context created when a specific file is opened and closed only when requested by the user. In order to create a session, you need to issue an `open` command. Following is the help message:

```
usage: open [-h] [-f] [-u] [-l] [-t] <target|md5|sha256>

Options:
    --help (-h) Show this help message
    --file (-f) The target is a file
    --url (-u)  The target is a URL
    --last (-l) Open file from the results of the last find command
    --tor (-t)  Download the file through Tor

You can also specify a MD5 or SHA256 hash to a previously stored
file in order to open a session on it.
```

You can fundamentally open:

- A file available in the local repository
- Any file available on the local filesystem
- Any URL

If you don't specify any option, Viper will interpret the value you provided as an hash it has to look up in the local database, for example:

```
shell > open 22f77c113cc6d43d8c12ed3c9fb39825
[*] Session opened on ~/viper/binaries/5/0/8/5/
↪50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
shell poisonivy.exe >
```

If you want to open a file elsewhere on the filesystem, you need to specify the `--file` (or `-f`) flag:

```
shell > open -f /tmp/poisonivy.exe
[*] Session opened on /tmp/poisonivy.exe
```

If you want to open an URL you can use the `--url` flag:

```
shell > open --url http://malicious.tld/path/to/file.exe
[*] Session opened on /tmp/tmpcuIOIj
shell tmpcuIOIj >
```

If you have Tor running, you can fetch the file through it by additionally specifying `--tor`.

Through the `open` command you can also directly open one of the entries from the results of the last executed `find` command, for example:

```
shell > find all
+---+---------------+-----------------------+----------------------------------+
| # | Name          | Mime                  | MD5                              |
+---+---------------+-----------------------+----------------------------------+
| 1 | poisonivy.exe | application/x-dosexec | 22f77c113cc6d43d8c12ed3c9fb39825 |
+---+---------------+-----------------------+----------------------------------+
shell > open --last 1
[*] Session opened on ~/viper/binaries/5/0/8/5/
↪50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
shell poisonivy.exe >
```

sessions

You can see which sessions are currently active and eventually switch from one to another through the `sessions` command. Following is the help message:

```
usage: sessions [-h] [-l] [-s=session]

Options:
    --help (-h) Show this help message
    --list (-l) List all existing sessions
    --switch (-s)   Switch to the specified session
```

An example of execution is the following:

```
shell poisonivy.exe > sessions --list
[*] Opened Sessions:
+---+---------------+----------------------------------+---------------------+--------
↪-+
| # | Name          | MD5                              | Created At          |␣
↪Current |
+---+---------------+----------------------------------+---------------------+--------
↪-+
| 1 | poisonivy.exe | 22f77c113cc6d43d8c12ed3c9fb39825 | 2014-07-12 01:36:14 | Yes    ␣
↪ |
```

(continues on next page)

(continued from previous page)

```
| 2 | zeus.exe      | 9b2de8b062a5538d2a126ba93835d1e9 | 2014-07-12 01:36:19 |
↪ |
| 3 | darkcomet.exe | 9f2520a3056543d49bb0f822d85ce5dd | 2014-07-12 01:36:23 |
↪ |
+---+---------------+----------------------------------+---------------------+--------
↪-+
shell poisonivy.exe > sessions --switch 2
[*] Switched to session #2 on ~/viper/binaries/6/7/6/a/
↪676a818365c573e236245e8182db87ba1bc021c5d8ee7443b9f673f26e7fd7d1
shell zeus.exe >
```

export

The `export` command is used to export the currently opened file to the target path or archive name. You can zip up the file in a new archive too:

```
usage: export [-h] [-z] <path or archive name>

Options:
    --help (-h) Show this help message
    --zip (-z)  Export session in a zip archive
```

close

This command simply abandon a session that was previously opened. Note that the session will actually remain available in case you want to re-open it later.

store

The `store` command is used to store the currently opened file to the local repository. There are many options and filters you can apply, as shown in the following help message:

```
usage: store [-h] [-d] [-f <path>] [-s <size>] [-y <type>] [-n <name>] [-t]

Options:
    --help (-h) Show this help message
    --delete (-d)   Delete the original file
    --folder (-f)   Specify a folder to import
    --file-size (-s)    Specify a maximum file size
    --file-type (-y)    Specify a file type pattern
    --file-name (-n)    Specify a file name pattern
    --tags (-t) Specify a list of comma-separated tags
```

If you specify `--delete` it will instruct Viper to delete the original copy of the file you want to store in the local repository, for example:

```
shell > open -f /tmp/poisonivy.exe
[*] Session opened on /tmp/poisonivy.exe
shell poisonivy.exe > store --delete
[+] Stored file "poisonivy.exe" to ~/viper/binaries/5/0/8/5/
↪50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
```

(continues on next page)

(continued from previous page)

```
[*] Session opened on ~/viper/binaries/5/0/8/5/
↪50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
shell poisonivy.exe >
```

If you want, you can store the content of an entire folder by specifying its path to the `--folder` parameter. In case the folder contains a large variety of files, you can filter which ones you're particularly interested in: with `--file-size` you can specify a maximum size in bytes, with `--file-type` you can specify a pattern of magic file type (e.g. *PE32*) and with `--file-name` you can specify a wildcard-enabled pattern to be matched with the file names (e.g. *apt_*).

If you want, you can already specify a list of comma separated tags to apply to all files stored through the given command.

Following is an example:

```
shell > store --folder /tmp/malware --file-type PE32 --file-size 10000000 --file-name␣
↪apt_* --tags apt,trojan
```

find

In order to quickly recover files you previously stored in the local repository, you can use the `find` command. Following is its help message:

```
usage: find [-h] [-t] <all|latest|name|md5|sha256|tag|note> <value>

Options:
    --help (-h) Show this help message
    --tags (-t) List tags
```

This command expects a key and eventually a value. As shown by the help message, these are the available keys:

- **all**: this will simply return all available files.
- **latest** *(optional limit value)*: this will return the latest 5 (or whichever limit you specified) files added to the local repository.
- **name** *(required value)*: this will find files matching the given name pattern (you can use wildcards).
- **md5** *(required value)*: search by md5 hash.
- **sha256** *(required value)*: search by sha256 hash.
- **tag** *(required value)*: search by tag name.
- **note** *(required value)*: find files that possess notes matching the given pattern.

For example:

```
shell > find tag rat
+---+---------------+-----------------------+----------------------------------+
| # | Name          | Mime                  | MD5                              |
+---+---------------+-----------------------+----------------------------------+
| 1 | poisonivy.exe | application/x-dosexec | 22f77c113cc6d43d8c12ed3c9fb39825 |
+---+---------------+-----------------------+----------------------------------+
```

info

The `info` command will return you some basic information on the file you currently have opened, for example:

```
shell poisonivy.exe > info
+--------+-------------------------------------------------------------------------------
↪-------------------------------------------------+
| Key    | Value
↪                                                 |
+--------+-------------------------------------------------------------------------------
↪-------------------------------------------------+
| Name   | poisonivy.exe
↪                                                 |
| Tags   | rat, poisonivy
↪                                                 |
| Path   | ~/viper/binaries/5/0/8/5/
↪50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
↪                    |
| Size   | 133007
↪                                                 |
| Type   | PE32 executable (GUI) Intel 80386, for MS Windows
↪                                                 |
| Mime   | application/x-dosexec
↪                                                 |
| MD5    | 22f77c113cc6d43d8c12ed3c9fb39825
↪                                                 |
| SHA1   | dd639a7f682e985406256468d6df8a717e77b7f3
↪                                                 |
| SHA256 | 50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
↪                                                 |
| SHA512 |
↪6743b06e8b243d513457949ad407d80992254c99b9835eb1ed03fbc0e88a062f0bb09bfd4dd9c0d43093b2a5419ecdb689
↪|
| SSdeep | 3072:I4lRkAehGfzmuqTPryFm8le+ZNX2TpF3Vb:I4lRkAehaKuqT+FDl7NXs7B
↪                                                 |
| CRC32  | 4090D32C
↪                                                 |
+--------+-------------------------------------------------------------------------------
↪-------------------------------------------------+
```

notes

During an analysis you might want to keep track of your discoveries and results. Instead of having unorganized text files lying around, Viper allows you to create notes directly linked to the relevant files and even search across them. When you have a file opened, you can add any number of text notes associated to it through the `notes` command. This is the help message:

```
usage: notes [-h] [-l] [-a] [-e <note id>] [-d <note id>]

Options:
    --help (-h) Show this help message
    --list (-h) List all notes available for the current file
    --add (-a)  Add a new note to the current file
    --view (-v) View the specified note
    --edit (-e) Edit an existing note
    --delete (-d)   Delete an existing note
```

As shown in the help message, you can list add a note:

```
shell poisonivy.exe > notes --add
Enter a title for the new note:
```

Now you should enter a title, when you proceed Viper will open your default editor to edit the body of the note. Once done and the editor is closed, the new note will be stored:

```
[*] New note with title "Domains" added to the current file
```

Now you can see the new note in the list and view its content:

```
shell poisonivy.exe > notes --list
+----+---------+
| ID | Title   |
+----+---------+
| 1  | Domains |
+----+---------+
shell poisonivy.exe > notes --view 1
[*] Title: Domains
[*] Body:
- poisonivy.malicious.tld
- poisonivy2.malicious.tld
```

tags

In order to easily group and identify files, Viper allows you to create one or more tags to be associated with them. This is the help message:

```
usage: tags [-h] [-a=tags] [-d=tag]

Options:
    --help (-h) Show this help message
    --add (-a)  Add tags to the opened file (comma separated)
    --delete (-d)   Delete a tag from the opened file
```

Once you have a file opened, you can add one ore more tags separated by a comma:

```
shell poisonivy.exe > tags --add rat,poisonivy
[*] Tags added to the currently opened file
[*] Refreshing session to update attributes...
[*] Session opened on ~/viper/binaries/5/0/8/5/
↪50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
```

Once added, the session will be refreshed so that the new attributes will be visible as you can see from the output of an `info` command:

```
shell poisonivy.exe > info
+--------+-------------------------------------------------------------------------------
↪-----------------------------------------------------+
| Key    | Value
↪                                                    |
+--------+-------------------------------------------------------------------------------
↪-----------------------------------------------------+
| Name   | poisonivy.exe
↪                                                    |
| Tags   | rat, poisonivy
↪                                                    |
```

(continues on next page)

(continued from previous page)

```
| Path   | ~/viper/binaries/5/0/8/5/
↪50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
↪                    |
| Size   | 133007
↪                                                          |
| Type   | PE32 executable (GUI) Intel 80386, for MS Windows
↪                                                          |
| Mime   | application/x-dosexec
↪                                                          |
| MD5    | 22f77c113cc6d43d8c12ed3c9fb39825
↪                                                          |
| SHA1   | dd639a7f682e985406256468d6df8a717e77b7f3
↪                                                          |
| SHA256 | 50855f9321de846f6a02b264e25e4c59983badb912c3c51d8c71fcd517205f26
↪                                                          |
| SHA512 |
↪6743b06e8b243d513457949ad407d80992254c99b9835eb1ed03fbc0e88a062f0bb09bfd4dd9c0d43093b2a5419ecdb689
↪|
| SSdeep | 3072:I4lRkAehGfzmuqTPryFm8le+ZNX2TpF3Vb:I4lRkAehaKuqT+FDl7NXs7B
↪                                                          |
| CRC32  | 4090D32C
↪                                                          |
+--------+-----------------------------------------------------------------------------
↪-------------------------------------------------------+
```

copy

The `copy` command let's you copy the opened file into another project. By default the stored analysis results, notes and tags will also be copied. If the file has children related to it then these will not be copied by default. Also copying all children (recursively) can be enabled by passing the `--children` or `-c` flag.

If the `--delete` or `-d` is passed then the files will be copied to the specified project and then deleted from the local project:

```
viper foo.txt > copy -h
usage: copy [-h] [-d] [--no-analysis] [--no-notes] [--no-tags] [-c] project

Copy opened file into another project

positional arguments:
  project         Project to copy file(s) to

optional arguments:
  -h, --help      show this help message and exit
  -d, --delete    delete original file(s) after copy ('move')
  --no-analysis   do not copy analysis details
  --no-notes      do not copy notes
  --no-tags       do not copy tags
  -c, --children  also copy all children - if --delete was selected also the
                  children will be deleted from current project after copy


viper foo.txt > copy -d foobar
[+] Copied: e2c94230decedbf4174ac3e35c6160a4c9324862c37cf45124920e63627624c1 (foo.txt)
[*] Deleted: e2c94230decedbf4174ac3e35c6160a4c9324862c37cf45124920e63627624c1
[+] Successfully copied sample(s)
```

delete

The `delete` command you simply remove the currently opened file from the local repository:

```
shell poisonivy.exe > delete
Are you sure you want to delete this binary? Can't be reverted! [y/n] y
[+] File deleted
shell >
```

2.2.3 HTTP Interfaces

Viper has two HTTP Components that can optionally be enabled alongside the console access.

- REST based API interface
- Web interface

The first one can be used to easily integrate Viper with other tools, while the second one provides an user-friendly alternative to the traditional command-line interface.

Security Considerations

Beware that Viper does not provide any security mechanism to protect neither of the HTTP interfaces. Enabling either interface access to the Internet would severely expose the security of your system, as users are able not only to access and operate on your Viper repositories, but also execute commands on the hosting system.

Make sure you take all necessary precautions to implement authentication and encryption whenever necessary through the tools provided by your firewall and web server.

API

Viper provides a REST-based API interface through which you can operate on the repositories as well as execute almost all functions that you would normally get through the command-line interface.

To start the API:

```
user@system:~/$ viper-api
Bottle server starting up (using WSGIRefServer())...
Listening on http://127.0.0.1:8080/
Hit Ctrl-C to quit.
```

You can bind it on a different IP and port by providing additional options:

```
usage: viper-api [-h] [-H HOST] [-p PORT]

optional arguments:
  -h, --help            show this help message and exit
  -H HOST, --host HOST  Host to bind the API server on
  -p PORT, --port PORT  Port to bind the API server on
```

The API commands will provide JSON results. Following ou can find details on the commands available.

/test

GET /test

Test the API server.

Example request:

```
curl http://127.0.0.1:8080/test
```

Example response:

```
{
    "message": "test"
}
```

Parameters:

- None

Status codes:

- 200 - no errors

/file/add

POST /file/add

Submit a file to Viper.

Example request:

```
curl -F file=@FILE -F tags='foo,bar' -X POST http://127.0.0.1:8080/
↪file/add
```

Example response:

```
{
    "message": "added"
}
```

Parameters:

- tags: comma separated list of tags
- file: path to the file to submit

Status codes:

- 200 - no errors
- 500 - something failed when adding the file

/file/add_url

POST /file/add_url

Submit a url to Viper.

Example request:

```
curl -F url='http://google.com' -F tags='foo,bar' -F tor='true' -X␣
↪POST http://127.0.0.1:8080/file/add_url
```

Example response:

```
{
    "message": "added"
    "sha256":
↪"a28a9ca63e8460b03dff84b5645c6c2a30f48149c0e5b273525cf4b80fe8a8ca"
}
```

Parameters:

- `url`: url of the file to download
- `tags`: comma separated list of tags
- `tor`: if a value is set, tor will be used to download the file

Status codes:

- `200` - no errors
- `500` - something failed when adding the file

/file/get

GET /file/get/ *(str: MD5 or SHA256 hash)*

Retrieve a file from Viper.

Example request:

```
curl http://127.0.0.1:8080/file/get/
↪9ce49435b67d531bbd966186920c90ecf0752e88b79af246886b077c8ec9b649
```

Parameters:

- `hash`: MD5 or SHA256 hash of the file to retrieve

Status codes:

- `200` - no error
- `400` - you did not provide a valid hash (MD5 or SHA256)
- `404` - file not found

/file/delete

GET /file/delete/ *(str: MD5 or SHA256 hash)*

Delete file from Viper.

Example request:

curl http://127.0.0.1:8080/file/delete/9ce49435b67d531bbd966186920c90ecf0752e88b79af246886b077c8ec9b649

Example response:

```
{
    "message": "deleted"
}
```

Status codes:

- `200` - no error
- `400` - invalid hash format
- `404` - file not found
- `500` - unable to delete file

/file/find

POST /file/find

Find a file in Viper default repository or project

Example request:

```
curl -F tag=rat http://127.0.0.1:8080/file/find
```

Example response:

```
{
    "default": [
        {
            "sha1": "13da502ab0d75daca5e5075c60e81bfe3b7a637f",
            "name": "darkcomet.exe",
            "tags": [
                "rat",
                "darkcomet"
            ],
            "sha512":
↪"7e81e0c4f49f1884ebebdf6e53531e7836721c2ae41729cf5bc0340f3369e7d37fe4168a7434b2b0420b299f5
↪",
            "created_at": "2015-03-30 23:13:20.595238",
            "crc32": "2238B48E",
            "ssdeep":
↪"12288:D9HFJ9rJxRX1uVVjoaWSoynxdO1FVBaOiRZTERfIhNkNCCLo9Ek5C/
↪hlg:NZ1xuVVjfFoynPaVBUR8f+kN10EB/g",
            "sha256":
↪"2d79fcc6b02a2e183a0cb30e0e25d103f42badda9fbf86bbee06f93aa3855aff",
            "type": "PE32 executable (GUI) Intel 80386, for MS␣
↪Windows",
            "id": 10,
            "md5": "9f2520a3056543d49bb0f822d85ce5dd",
            "size": 774144
        },
        {
            "sha1": "dbcea714f43aa06a7f1c3d11cbfd67e4f8e0c23e",
            "name": "poisonivy3.exe",
            "tags": [
                "rat",
                "poisonivy"
            ],
```

(continues on next page)

(continued from previous page)

```
            "sha512":
↪"4b2d61211b059400d5f8701908c6f4cb25a70a44882c67f887301dfc3e02d29b562032fc11333cca29f8bb9a3
↪",
            "created_at": "2015-03-30 23:13:20.595238",
            "crc32": "BCD8287D",
            "ssdeep":
↪"3072:lR+yF1aa8hCqTevS0IjhAPCoGm3vkazsW2mq:lR+KrWCqavSFhmCoGm3h0mq
↪",
            "sha256":
↪"15846af22582f06fde215a0e506fdf5f88d3262b3d62d1eabd6bdf00f91e0df7",
            "type": "PE32 executable (GUI) Intel 80386 (stripped to␣
↪external PDB), for MS Windows",
            "id": 28,
            "md5": "23c3b61ecdff3d67479d70b5d4d91dea",
            "size": 143560
        },
        ...
    ]
}
```

Parameters:

- `md5`: search by MD5
- `sha256`: search by SHA256
- `ssdeep`: search by ssdeep
- `tag`: search by tag
- `name`: search by name
- `all`: retrieve all files
- `latest`: retrieve only the most recently added files (specify 1 as the value)
- `project`: a project name to search the file in (default is none, you can also specify "all" to search across all projects)

Status codes:

- `200` - no error
- `400` - invalid search term

/file/tags/add

POST /file/tags/add

Add one or more tags to one or more files

Example request:

```
curl -F tags=foo,bar -F md5=23c3b61ecdff3d67479d70b5d4d91dea http://
↪127.0.0.1/file/tags/add
```

Example response:

```
{
    "message": "added"
}
```

Parameters:

- `tags`: comma-separated list of tags
- `md5`: select by MD5
- `sha256`: select by SHA256
- `ssdeep`: select by ssdeep
- `tag`: select by tag
- `name`: select by name
- `all`: retrieve all files
- `latest`: retrieve only the most recently added files

Status codes:

- `200` - no error
- `404` - file not found

/tags/list

GET /tags/list

Retrieve a list of all tags

Example request:

```
curl http://127.0.0.1:8080/tags/list
```

Example response:

```
[
    "rat",
    "darkcomet",
    "poisonivy",
    "njrat",
    "embedded_win_api",
    "nettraveler",
    "xtreme"
]
```

Status codes:

- `200` - no error#

/file/notes/add

POST /file/notes/add

Add a note to a sample

Parameters:

- `sha256`: select by SHA256
- `title`: title of the note
- `body`: body of the note

Example request:

```
curl http://127.0.0.1 -F sha256=
↪"2e766eabed666510a385544b79a5d344b48a2de2040c62fee9addb19c554ed4c"␣
↪-F title="asd"  -F body="bodddy" http://127.0.0.1:8080/file/notes/
↪add
```

Example response:

```
{
    "message": "Note added"
}
```

Status codes:

- `200` - no error

/file/notes/view

POST /file/notes/view

Retrieve a list of all notes

Example request:

```
curl -F sha256=
↪"2e766eabed666510a385544b79a5d344b48a2de2040c62fee9addb19c554ed4c"␣
↪http://127.0.0.1:8080/file/notes/view
```

Example response:

```
{
    "message": {
        "1": {
            "body": "bodddy",
            "title": "asd"
        }
    }
}
```

Status codes:

- `200` - no error

/file/notes/update

POST /file/notes/update

Updates a note from a sample

Parameters:

- `sha256`: select by SHA256

- `title`: title of the note
- `body`: body of the note
- `id`: id of the note

Example request:

```
curl http://127.0.0.1 -F sha256=
↪"2e766eabed666510a385544b79a5d344b48a2de2040c62fee9addb19c554ed4c"␣
↪-F title="asd" -F id="1" -F body="bodddy" http://127.0.0.1:8080/
↪file/notes/update
```

Example response:

```
{
    "message": "Note updated"
}
```

Status codes:

- `200` - no error

/file/notes/delete

POST /file/notes/delete

Delete a note from a sample

Parameters:

- `sha256`: select by SHA256
- `id`: id of the note

Example request:

```
curl http://127.0.0.1 -F sha256=
↪"2e766eabed666510a385544b79a5d344b48a2de2040c62fee9addb19c554ed4c"␣
↪-F id="1" http://127.0.0.1:8080/file/notes/delete
```

Example response:

```
{
    "message": "Note deleted"
}
```

Status codes:

- `200` - no error

/projects/list

GET /projects/list

Retrieve a list of all projects

Example request:

```
curl http://127.0.0.1:8080/projects/list
```

Example response:

```
[
    "project1",
    "project2",
    "project3"
]
```

Status codes:

- `200` - no error
- `404` - no projects found

/modules/run

POST /modules/run

Execute a command

Example request:

```
curl -F␣
↪sha256=d5042d68b813d5c45c03fe6883f5b83ff079cb9c394ddcc9c84f58e0369c6cdf␣
↪-F cmdline="pe compiletime" http://127.0.0.1:8080/modules/run
```

Example response:

```
[{'data': 'Compile Time: \\x1b[1m1992-06-20 00:22:17\\x1b[0m', 'type
↪': 'info'}]
```

Parameters:

- `project`: project name
- `sha256`: SHA256 hash of the file to execute the command on
- `cmdline`: the full command line as you would normally pass to the CLI

Status codes:

- `200` - no error
- `404` - invalid command line

Web Interface

Viper comes with a basic single threaded HTML Browser interface that can run alongside the command-line interface and API. Its main features are:

- Project Switching / Creation
- Multiple File Upload
- File Download
- Unpack Compressed uploads
- Full Search (including tag, name, mime, note, type)

- Hex Viewer
- Run Modules
- Enter Notes
- Add / Delete / Modify Yara rules
- Add / Delete / Modify Tags

Launch the web interface

To launch the web application move into the viper directory and run the `web.py` script. By default it launches a single threaded bottle web server on localhost:9090:

```
user@localhost:~/$ viper-web
Bottle v0.12.8 server starting up (using WSGIRefServer())...
Listening on http://localhost:9090/
Hit Ctrl-C to quit.
```

You can set the listening IP address and port with options -H and -p

```
user@localhost:~/$ viper-web -H 0.0.0.0 -p 8080
Bottle v0.12.8 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.
```

Use viper in a (web) production environment

In production use, its often not recommended to use the default Bottle WSGIRefServer as it can be quite slow and requires manual start.

A dedicated webserver that serves pages on standard ports and with a standardized configuration is possible with viper, but requires some additional setup To make this work, we are using uwsgi and nginx as stack.

To use nginx as the webserver serving web.py,:

```
$ sudo apt-get remove apache2 #avoid conflicts
$ sudo apt-get install nginx-full uwsgi uwsgi-plugin-python
```

Move (or copy) the source to the web directory and make sure the permissions match

```
$ sudo mv viper-* /srv/www/viper
$ chown -R www-data:www-data /srv/www/viper
```

Change the VIPER_ROOT variable in web.py to reflect your installation.:

```
$ sudo vi /srv/www/viper/web.py
```

Modify the user section, and uncomment and change the following line to reflect your environment:

```
VIPER_ROOT ='/srv/www/viper'
```

Create a file /etc/uwsgi/apps-available/bottle.ini:

```
[uwsgi]
socket = /run/uwsgi/app/bottle/socket
chdir = /srv/www/viper/
master = true
plugins = python
file = web.py
uid = www-data
gid = www-data
```

Link the /etc/uwsgi/apps-available/bottle.ini file to apps-enabled:

```
$ sudo  ln -s /etc/uwsgi/apps-available/bottle.ini /etc/uwsgi/apps-enabled/bottle.ini
```

Restart nginx and uwsgi:

```
$ sudo service nginx restart
$ sudo service uwsgi restart
```

Tail the uwsgi logfile to make sure everything works ok:

```
$ tail -f /var/log/uwsgi/app/bottle.log
```

And browse to the default port 80 interface of the webserver. You should be greeted with the viper webpage.

Setting up SSL and web-based authentication is out of scope for this document, but infromation can be found in the nginx manual and in one of the many tutorials on the web.

2.3 Create new modules

Viper in itself is simply a framework, modules are what give it analytical capabilities. We receive and include new modules all the time from contributors, but there are always new features to add. If you have an idea, you should implement a module for it and contribute it back to the community.

The following paragraphs introduce you to the first steps to create a new module.

2.3.1 First steps

First thing first, you need to create your *.py* script under the `modules/` directory: all modules are dynamically loaded by Viper from that folder exclusively. You can create subfolders and place your modules anywhere, Viper will be able to find them.

Any module needs to have some basic attributes that will make it recognizable. It needs to be a Python class inheriting `Module`, it needs to have a `cmd` and `description` attribute and it needs to have a `run()` function. For example the following would be a valid, although not very useful, Viper module:

```
1 from viper.common.abstracts import Module
2
3 class MyModule(Module):
4     cmd = 'mycmd'
5     description = 'This module does this and that'
6
7     def run(self):
8         print("Do something.")
```

2.3.2 Arguments

When a module is invoked from the Viper shell it can be provided with a number of arguments and options. These should be parsed with the python `argparse` module as show in the example below.

```
from viper.common.abstracts import Module

class MyModule(ModuleName):
    cmd = 'mycmd'
    description = 'This module does this and that'
    authors = ['YourName']

    def __init__(self):
        super(ModuleName, self).__init__()
        self.parser.add_argument('-t', '--this', action='store_true', help=
↪'Do This Thing')
        self.parser.add_argument('-b', '--that', action='store_true', help=
↪'Do That')

    def run(self):
        if self.args.this:
            print("This is FOO")
        elif self.args.that:
            print("That is FOO")
```

2.3.3 Using the Config File

Viper provides a config file that will allow you to store user editable sections in a single file rather than inside the modules.

/usr/share/viper/viper.conf.sample

You can easily access the config file:

```
from viper.core.config import __config__

cfg = __config__
```

From here you can access any element in the config file by name:

```
from viper.core.config import Config

cfg = Config()

config_item = cfg.modulename.config_item

# Example Getting VirusTotal Key

vt_key = cfg.virustotal.virustotal_key
```

2.3.4 Using common config settings for outbound http connections

A common use case for modules is to implement the API of an external web service (e.g. https://koodous.com/). The (great!) requests library (https://github.com/requests/requests/) provides an easy interface for making outbound http connections. Viper provides a global configuration section `[http_client]` where certain requests options can be

set for Proxies, TLS Verfication, CA_BUNDLE and TLS Client Certificates. Please check the current `viper.conf.sample` for more details.

When implementing a custom module settings from the global `[http_client]]` can be overridden by specifying them again in the configuration section of the custom module and then calling the `Config.parse_http_client` method for the custom module configuration section. Example:

```
# viper.conf

[http_client]
https_proxy = http://prx1.example.internal:3128
tls_verify = True

[mymodule]
base_url = https://myapi.example.internal
https_proxy = False
tls_verify = False
```

```
import requests
from viper.common.abstracts import Module
from viper.core.config import __config__

cfg = __config__
cfg.parse_http_client(cfg.mymodule)

class MyModule(Module):
    cmd = 'mycmd'
    description = 'This module does this and that'

    def run(self):
        url = cfg.mymodule.base_url
        r = requests.get(url=url, headers=headers, proxies=cfg.mymodule.
↪proxies, verify=cfg.mymodule.verify, cert=cfg.mymodule.cert)
```

2.3.5 Accessing the session

In most cases, you will probably want to execute some analysis function on the currently opened file and in order to do so you'll need to access the session. Sessions are internally made available through a global object called `__sessions__`, which has the following attributes:

- `__sessions__.current`: a `Session` object for the currently opened file.
- `__sessions__.sessions`: the list of all `Session` objects opened during the current Viper execution.
- `__sessions__.find`: a list contains all the results from the last executed `find` command.

A `Session` object has the following attributes:

- `Session.id`: an incremental ID for the session.
- `Session.created_at`: the date and time when the session was opened.
- `Session.file`: a `File` object containing common attributes of the currently opened file (generally speaking, the same information returned by the `info` command).

Following are the information available on the opened file:

- `__sessions__.current.file.path`
- `__sessions__.current.file.name`

- `__sessions__.current.file.size`
- `__sessions__.current.file.type`
- `__sessions__.current.file.mime`
- `__sessions__.current.file.md5`
- `__sessions__.current.file.sha1`
- `__sessions__.current.file.sha256`
- `__sessions__.current.file.sha512`
- `__sessions__.current.file.crc32`
- `__sessions__.current.file.ssdeep`
- `__sessions__.current.file.tags`

Here is an example:

```
from viper.common.abstracts import Module
from viper.core.session import __sessions__

class MyModule(Module):
    cmd = 'mycmd'
    description = 'This module does this and that'

    def run(self):
        # Check if there is an open session.
        if not __sessions__.is_set():
            # No open session.
            return

        # Print attributes of the opened file.
        print("MD5: " + __sessions__.current.file.md5)

        # Do something to the file.
        do_something(__sessions__.current.file.path)
```

2.3.6 Accessing the database

In case you're interested in automatically retreiving all files stored in the local repository or just a subset, you'll need to access the local database. Viper provides an interface called `Database()` to be imported from `viper.core.database`.

You can then use the `find()` function, specify a key and an optional value and you will obtain a list of objects you can loop through. For example:

```
from viper.common.abstracts import Module
from viper.core.database import Database

class MyModule(Module):
    cmd = 'mycmd'
    description = 'This module does this and that'

    def run(self):
        db = Database()
        # Obtain the list of all stored samples.
```

(continues on next page)

(continued from previous page)

```
        samples = db.find(key='all')

        # Obtain the list of all samples matching a tag.
        samples = db.find(key='tag', value='apt')

        # Obtain the list of all samples with notes matching a pattern.
        samples = db.find(key='note', value='maliciousdomain.tld')

        # Loop through results.
        for sample in samples:
            print("Sample " + sample.md5)
```

2.3.7 Printing results

Viper provides several function to facilitate and standardize the output of your modules. Viper uses a logging function to return the output to the console or web application. The format is `self.log('type', "Your Text")` and the following types are made available in Viper.

- `info`: prints the message with a `[*]` prefix.
- `warning`: prints the message with a yellow `[!]` prefix.
- `error`: prints the message with a red `[!]` prefix.
- `success`: prints the message with a green `[+]` prefix.
- `item`: prints an item from a list.
- `table`: prints a table with headers and rows.

You can also easily print tables, such as in the following example:

```
from viper.common.abstracts import Module

class MyModule(Module):
    cmd = 'mycmd'
    description = 'This module does this and that'

    def run(self):
        self.log('info', "This is Something")
        self.log('warning', "This is the warning Text")

        # This is the header of the table.
        header = ['Column 1', 'Column 2']
        # These are the rows.
        rows = [
            ['Row 1', 'Row 1'],
            ['Row 2', 'Row 2']
        ]

        self.log('table', dict(header=header, rows=rows))
```

2.4 Known issues

2.4.1 Various errors when using unicode characters

unicode and python is a not easy and using unicode in notes, tags or filenames (or other modules where userinput is allowed) might result in unhandled exceptions.

2.4.2 Error storing file names containing unicode characters in database

If you try to store a file with a filename containing Unicode chars it will not be stored to the database.

2.4.3 Problem importing certain modules

If you experience an issue like:

```
[!] Something wrong happened while importing the module modules.office: No module␣
↪named oletools.olevba
```

You are likely missing dependencies.

To install required python modules run:

```
pip install -r requirements.txt
```

2.4.4 The API interface isn't fully aware of projects

Most of the API commands are not able yet to interact with different projects, so most of the commands will be executed against the default repository.

2.4.5 update.py from 1.1 to 1.2 IOError 'data/web/'

If you are running a Viper version 1.1 und using update.py to update to 1.2 you might run into some error like:

```
python update.py
[!] WARNING: If you proceed you will lose any changes you might have made to Viper.
Are you sure you want to proceed? [y/N] y
Traceback (most recent call last):
File "update.py", line 79, in <module>
  main()
File "update.py", line 66, in main
  new_local = open(local_file_path, 'w')
  IOError: [Errno 2] No such file or directory: 'data/web/'
```

That issue is known and already adressed in the new version of update.py (you might wanna pull that file manually

2.4.6 PreprocessError: data/yara/index.yara:0:Invalid file extension '.yara'.Can only include .yar

If you running yara or RAT module and receiving that issue:

```
...
PreprocessError: data/yara/index.yara:0:Invalid file extension '.yara'.Can only␣
↪include .yar
...
```

It is most likely the versions of yara are not correct, try to run:

```
viper@viper:/home/viper# yara -version
yara 2.1
```

And check for the yara-python bindings:

```
viper@viper:/home/viper# pip freeze | grep yara
yara-python==2.1
```

If you have installed yara-python using pip it is likely you are running an older version of yara (see yara documentation for compiling howto)

2.5 Final Remarks

2.5.1 Developers

Viper is an open and collaborative development effort. It is built by volunteers from all over the world. Following are the ones who contributed up to the latest stable release:

```
$ git shortlog -s -n
   447    Nex
    95    kevthehermit
    51    Raphaël Vinot
    26    Alexander J
    22    emdel
     8    Luke Snyder
     7    Neriberto C.Prado
     6    Sascha Rommelfangen
     5    Beercow
     5    SnakeByte Lab
     4    Seth Hardy
     3    =
     3    Csaba Fitzl
     3    Dionysis Grigoropoulos
     3    Jerome Marty
     3    nidsche
     2    Sam Brown
     2    haellowyyn
     1    Alex Harvey
     1    Ali Ikinci
     1    Boris Ryutin
     1    Nasicus
     1    S0urceC0der
     1    Tobias Jarmuzek
     1    binjo
     1    dewiestr
     1    dukebarman
     1    dukebarman@gmail.com
     1    jekil
```

2.5.2 Join Us

The best way to start contributing to the project is by start digging through the open tickets on our GitHub. Before submitting code make sure you read our Contribution Guidelines and that you thoroughly tested it.

You can also join our conversions by getting on IRC on FreeNode on channel `###viper`.