IRMA Documentation

Release unpackaged

Quarkslab

Sep 20, 2019

Contents

CHAPTER 1

Introduction

This publication is intended for advanced technical users of IRMA Enterprise. It assumes the reader has working knowledge of Systems Administration, the GNU/Linux operating system and basic Python.

IRMA (Incident Response and Malware Analysis) is a flexible content analysis orchestration platform. This guide will explain how to install, configure, use and customize it. This is an introductory chapter to IRMA. It describes IRMA's overall purpose, architecture and process.

1.1 Purpose

IRMA provides its users with the ability to objectively assess whether content is malicious or not. Content may be delivered by various means as described in this document, and subsequently distributed to various configurable analysis engines ("probes"). After analysis, this information is then conveyed to the user.

In addition to this core functionality, IRMA provides an overview of the overall analysis process and incident.

Finally, IRMA is designed to be installed and maintained in self-contained on-premises environment. This enables discreet and self-contained analysis for organizations which do not wish to disclose potentially confidential files to third parties.

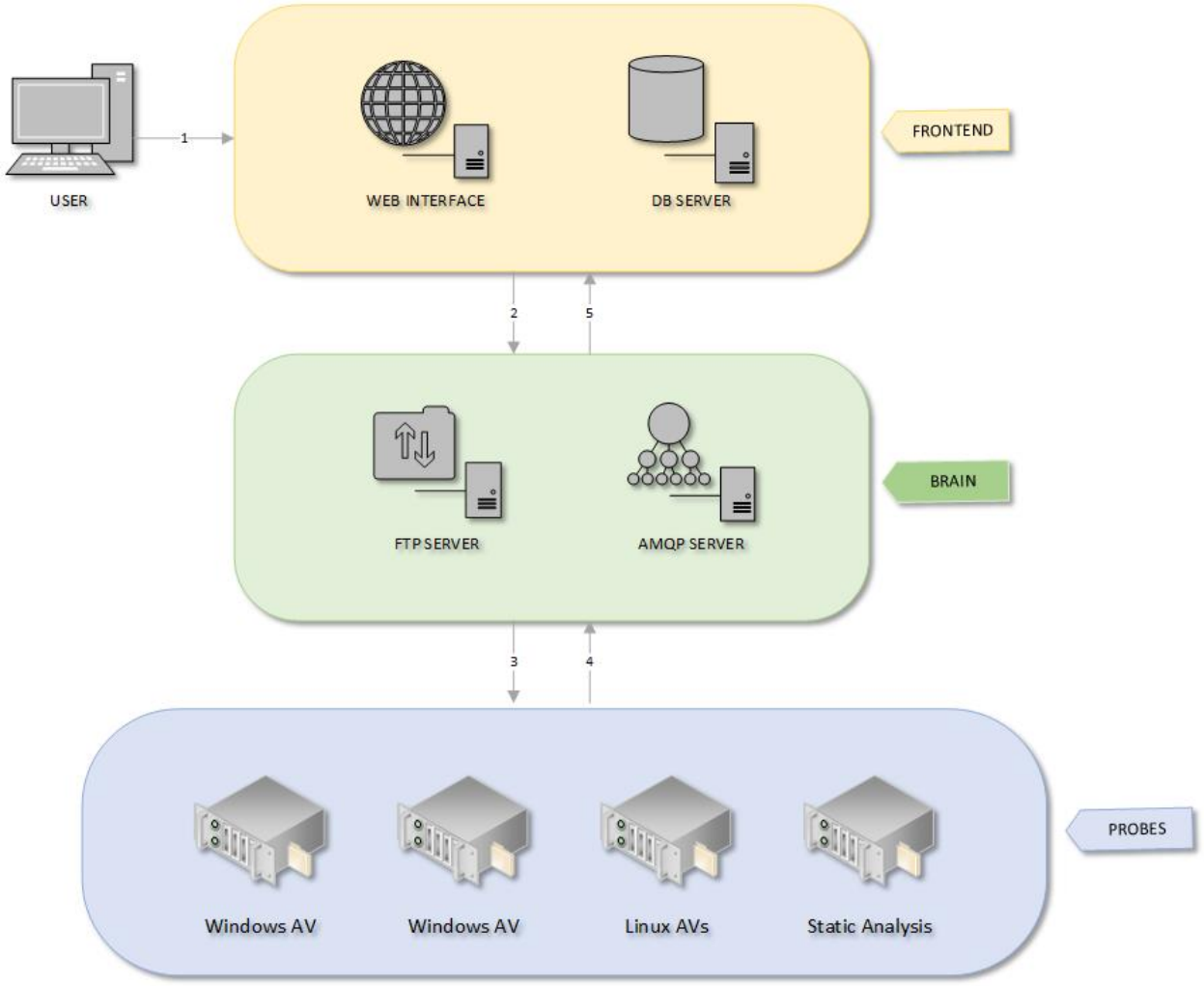
The ultimate purpose of IRMA is to orchestrate the entire analysis process and provide organizations with a flexible platform with which to manage and asses the content flowing through their organizations.

1.2 File Analysis Process

IRMA consists or three basic components. the **Frontend**, **Brain** and **Probes**. The basic functionality of **frontend** is to store results and host the API. **Brain** split analysis jobs on every **probes** involved, and **Probes** analyze files and returns results.

1. An analysis begins when a user uploads files to the **Frontend**.
2. **Frontend** checks for existing files and results in SQL. If needed, it stores the new files and calls asynchronously scan jobs on **Brain**.

3. **Brain** worker sends as much subtasks to **Probe(s)** as needed.
4. **Probe** workers process their jobs and send back results to **Brain**.
5. **Brain** sends results to **Frontend**.



1.3 Supported Analyzers

Here is the list of analyzers that are bundled with IRMA.

1.3.1 Antiviruses

Probe Name	Anti-Virus Name	Platform
ASquaredCmdWin	Emsisoft Command Line	Microsoft Windows CLI
AvastCoreSecurity	Avast Core Security	GNU/Linux CLI
AVGAntiVirusFree	AVG	GNU/Linux CLI
AviraWin	Avira	Microsoft Windows CLI
BitdefenderForUnices	Bitdefender	GNU/Linux CLI
ClamAV	ClamAV	GNU/Linux CLI
ComodoCAVL	Comodo Antivirus for Linux	GNU/Linux CLI
DrWeb	Dr.Web	GNU/Linux CLI
EScan	eScan	GNU/Linux CLI
EsetFileSecurity	Eset File Security	GNU/Linux CLI
FProt	F-Prot	GNU/Linux CLI
FSecure	F-Secure	GNU/Linux CLI
GDataWin	G Data Antivirus	Microsoft Windows CLI
Kaspersky	Kaspersky File Server	GNU/Linux CLI
KasperskyWin	Kaspersky Internet Security	Microsoft Windows CLI
McAfeeVSCL	McAfee VirusScan Command Line	GNU/Linux CLI
McAfeeVSCLWin	McAfee VirusScan Command Line	Microsoft Windows CLI
Sophos	Sophos	GNU/Linux CLI
SophosWin	Sophos Endpoint Protection	Microsoft Windows CLI
SymantecWin	Symantec Endpoint Protection	Microsoft Windows CLI
VirusBlokAda	VirusBlokAda	GNU/Linux CLI
Zoner	Zoner Antivirus	GNU/Linux CLI

1.3.2 External analysis platforms

Probe Name	Analysis Platform	Description
ICAP	ICAP	Query an ICAP server
VirusTotal	VirusTotal	Report is searched using the sha256 of the file which is not sent

1.3.3 File database

Probe Name	Database	Description
NSRL	National Software Reference Library	collection of digital signatures of known, traceable software applications

1.3.4 Metadata

Probe Name	Description
LIEF	PE/ELF File analyzer
PEiD	PE File packer analyzer
TrID	File type identification
StaticAnalyzer	PE File analyzer adapted from Cuckoo Sandbox
Yara	Checks if a file match yara rules

CHAPTER 2

Installation

This chapter describes the methods available to install IRMA using Ansible scripts.

2.1 Software requirements

- Ansible; You can see the requirement version of ansible in `ansible/requirements.txt`

```
ansible==2.4.2.0
```

2.2 Hardware requirements

The IRMA platform is divided in three major components: the **Frontend**, the **Brain** and one or multiple **Probes**.

These three components can be installed on a unique host or on multiple hosts, according to the kind of probes that are being used.

The **Frontend** and the **Brain** must be installed on a GNU/Linux system[1]. Quarkslab recommends using a Debian Stable distribution which is supported and known to work.

According to the kind of probes and their dependencies, each analyzers can be installed on a separate hosts or share the same host as far as they do not interfere with each other[2]. Currently, only Debian Stable and Microsoft Windows 8 and 10 hosts have been tested.

Quarkslab does not provide any estimates regarding performance. However, the following configuration is known to provide reasonable performance for small deployments:

[1] Theorically, it should be possible, with some efforts, to make IRMA work on Microsoft Windows systems as most of the components used for the platform are known to work or to have equivalents on these systems.

[2] For instance, we managed to host several GNU/Linux anti-viruses on an unique probe by preventing it to launch daemons at startup. This is difficult for Microsoft systems on which it is not recommended to install multiple anti-viruses on a single host.

whole IRMA platform on a single machine by hosting it with multiple systems inside virtual machines: this setup gives fairly high throughput as long as it has reasonable IO (ideally, SSDs), and a good amount of memory (test setup was an i7 cpu with 16 GB ram on regular drives (at least 200 GB required),

For larger deployments, the following configuration is known to work: a single high-memory machine, with 16+ cores, and SSDs, could run IRMA platform and bear the workload load with reasonable response time.

2.3 Automated Installation

The IRMA platform is easily installed thanks to a set of ansible roles and playbooks. It permits a user to build, install or maintain different setups.

There are 2 different types of IRMA environment, and multiple setups for each environment:

- **Development environment (sources rsync'd between host and vms)**
  - `allinone_dev`: everything installed in the same vm
  - `dev`: every component on its own vm
- **Production environment (sources installed through generated archives, install on vms/physical servers)**
  - `allinone_prod`: everything installed in the same vm/physical server (default environment)
  - `prod`: every component on its own vm/physical server

For specific instructions on these 2 environments see the related section.

Note: Vagrant step is optional in production mode.

2.3.1 Environment file

IRMA installation uses ansible and optionally Vagrant, and supports a common configuration format that allows launching of Vagrant and/or ansible. `VagrantFile` automatically parses the configuration file to allow vagrant to launch required virtual machines, and `irma-ansible.py` parses this same file to create an inventory and an extra variable (vars) file before launching ansible.

Format

For examples look at the files `*.yml` in the `ansible/environments` directory. Whole IRMA infrastructure is described here:

```
servers:
  - name: <hostname>
    ip: <ip address>
    ansible_groups: [list of ansible groups]
    box: [vagrant box name]
    cpus: [vagrant cpus (optional)]
    memory: [vagrant memory (optional)]
    shares: [vagrant share (optional)]
    [...]

libvirt_config:
  driver: kvm
```

(continues on next page)

(continued from previous page)

```
  # connect_via_ssh: true
  # host:
  # username:
  # storage_pool_name:
  # id_ssh_key_file:

ansible_vars:
  key: value
  [...]
```

- `servers` section both described ansible usage of the server and its vagrant configuration if needed.
- `libvirt_config` section is a vagrant-only section for using libvirt hypervisor.
- `ansible_vars` section is an ansible-only section for defining extra ansible variables.

Example of a development environment with vagrant:

```
servers:
  - name: brain.irma
    ip: 172.16.1.30
    ansible_groups: [frontend, sql-server, brain, comodo, trid]
    box: quarkslab/debian-9.0.0-amd64
    cpus: 2
    memory: 2048
    shares:
      - share_from: ../common
        share_to: /opt/irma/irma-common/releases/sync
        share_exclude:
          - .git/
          - venv/
      - share_from: ../frontend
        share_to: /opt/irma/irma-frontend/releases/sync
        share_exclude:
          - .git/
          - venv/
          - web/dist
          - web/node_modules
      - share_from: ../brain
        share_to: /opt/irma/irma-brain/releases/sync
        share_exclude:
          - .git/
          - venv/
          - db/
      - share_from: ../probe
        share_to: /opt/irma/irma-probe/releases/sync
        share_exclude:
          - .git/
          - venv/

libvirt_config:
  driver: kvm

ansible_vars:
  irma_environment: development
  vagrant: true
```

And an example of an environment without vagrant:

```
servers:
  - name: frontend.irma
    ip: 172.16.1.30
    ansible_groups: [frontend, sql-server]
  - name: brain.irma
    ip: 172.16.1.31
    ansible_groups: [brain]
  - name: avs-linux.irma
    ip: 172.16.1.32
    ansible_groups: [avast, avg, bitdefender, clamav, comodo, escan]
  - name: mcafee-win.irma
    ip: 172.16.1.33
    ansible_groups: [mcafee-win]
    windows: true

ansible_vars:
  irma_environment: production
  vagrant: true
  irma_release: HEAD
```

Extra vars

It is possible to customize IRMA variables in section `ansible_vars` (see `irma_vars.yml.sample` for a full list of available vars).

2.3.2 Vagrant setup

Requirements

- Vagrant 1.9 or higher has to be installed
- **a supported hypervisor:**
  - kvm/qemu (libvirt required, vagrant-libvirt plugin required)
  - Virtualbox

Vagrant setup

```
(venv)$ export VM_ENV=dev
(venv)$ export VM_ENV=allinone_dev
(venv)$ export VM_ENV=prod
(venv)$ export VM_ENV=allinone_prod  # (default)
```

Simply run in the *Vagrantfile* directory:

```
(venv)$ vagrant up (--provider=libvirt)
```

Vagrant will launch one/many VM(s).

Note: The basebox used in this project is provided by Quarkslab. The code source to build it is here.

Useful commands

Some useful commands with vagrant:

```
$ vagrant ssh <server_name>        # login through ssh
$ vagrant halt <server_name>       # shutdown the machine
$ vagrant reload <server_name>     # restart the machine
$ vagrant up <server_name>         # start the machine
$ vagrant destroy <server_name>    # delete the machine
```

2.3.3 Ansible setup

Common requirements

- Ansible 2.0+ (see requirements.txt for version required)

```
(venv)$ pip install -r requirements.txt
```

Warning: Due to ansible breaking releases, the ansible version supported is now fixed

Ansible playbooks

IRMA Installation is split in playbooks (in ansible/playbooks directory):

- playbooks/provisioning.yml for dependencies setup
- playbooks/updating.yml for av update only
- playbooks/deployment.yml for irma code setup
- playbooks/playbook.yml (provisioning + updating + deployment)

Launch Ansible

Note: If your environment requires some virtual machines handled by vagrant, you must do this first.

To launch one of these playbook, the full command is:

```
# Dependencies setup
(venv)$ python irma-ansible.py environments/allinone_prod.yml playbooks/provisioning.
↪yml

# AV update only
(venv)$ python irma-ansible.py environments/allinone_prod.yml playbooks/updating.yml

# IRMA code install
(venv)$ python irma-ansible.py environments/allinone_prod.yml playbooks/deployment.yml

# Full install (provisioning + updating + deployment)
(venv)$ python irma-ansible.py environments/allinone_prod.yml playbooks/playbook.yml
```

Last one will do the full install of IRMA. It can take a while (from 15 to 30 min) depending on the amount of RAM available on the machine and the hard disk drive I/O speed.

The default IRMA interface is available at http://172.16.1.30. According to your frontend server configuration.

References

Some roles from Ansible Galaxy used here:

- NodeJS role from JasonGiedymin/nodejs
- Nginx role from jdauphant/ansible-role-nginx
- OpenSSH role from Ansibles/openssh
- UFW role from weareinteractive/ansible-ufw
- Sudo role from weareinteractive/ansible-sudo
- Users role from mivok/ansible-users

2.3.4 Windows provisioning

Generate Windows base box

```
$ git clone https://github.com/boxcutter/windows
$ cd windows
$ make virtualbox/eval-win10x64-enterprise
```

Adding to Vagrant boxes

```
$ vagrant box add --name eval-win10x64-enterprise box/virtualbox/eval-win10x64-
↪enterprise*.box
```

Creating an instance of the base box

```
$ VM_ENV=<your_env> vagrant up
```

Provisioning with ansible

In the config file don't forget to add `windows: true` in the server. Example:

```
servers:
  - name: mcafee-win.irma
    ip: 172.16.1.33
    box: eval-win10x64-enterprise
    ansible_groups: [mcafee-win]
    windows: true
```

Provisioning a windows host is done the same way as other hosts:

```
(venv)$ python irma-ansible.py environments/allinone_prod.yml playbooks/playbook.yml
```

2.3.5 Production environment

IRMA will be installed on physical servers.

Requirements

- One or multiple 64-bit Debian 9 servers.

1. Prep servers

Create an account for ansible provisioning, or use one which has already been created. To speed up provisioning, you can:

- Authorize your SSH key for password-less authentication (optional):

```
*On your local machine*
$ ssh-copy-id user@hostname # -i if you want to select your identity file
```

- If you don't want to have to type your password for `sudo` command execution, add your user to sudoers, using `visudo` command (optional):

```
user ALL=(ALL) NOPASSWD: ALL
```

2. Configure the installation

Modify ansible `extra_vars` especially the `provisioning_ssh_key` section, you'll need to add private keys from user for password-less connection to the default IRMA server user.

Warning: Be careful, you'll need to change all passwords from this configuration files (`password` variables for most of them).

You'll need to create a configuration file and adapt it to your infrastructure.

2.3.6 Extras

Installation behind a corporate proxy

Thanks to the vagrant-proxyconf plugin, IRMA can be installed behind corporate proxy.

First, `vagrant-proxyconf` has to be installed:

```
$ vagrant plugin install vagrant-proxyconf
```

Then, the `vagrant-proxyconf` configuration has to be added to *ansible/Vagrantfile*. Here is an example:

```
if Vagrant.has_plugin?("vagrant-proxyconf")
  config.proxy.http     = "http://corporate.proxy:3128"
  config.proxy.https    = "http://corporate.proxy:3128"
  config.proxy.no_proxy = "localhost,127.0.0.1"
end
```

Finally, `vagrant up` can be launched, as usual.

It has to be noted that using such mechanism has two limitations:

- it is not working with Windows based boxes
- it is not working with tools that are not able do deal with environment based proxy definition (`http_proxy` and `https_proxy` environment variables). For instance, AVG updater does not take into account such definition.

CHAPTER 3

Use IRMA

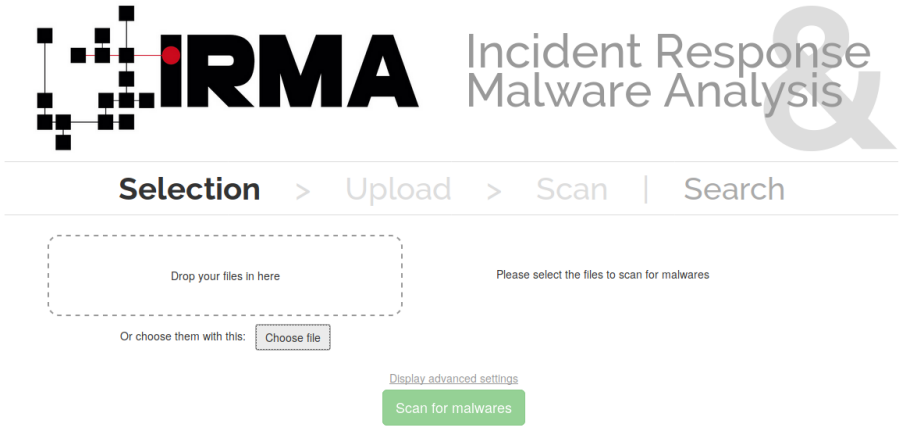
There are 2 ways to use IRMA :

3.1 Web Interface

3.1.1 How to do a scan

First choose one or multiple files to scan by:

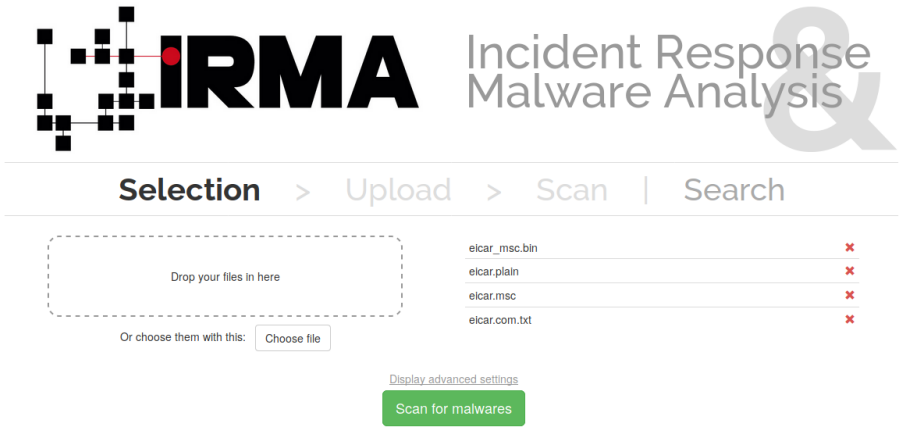
- Drop it in the select area
- Click on “Choose file” button



Now, you can see the selected files on the right.

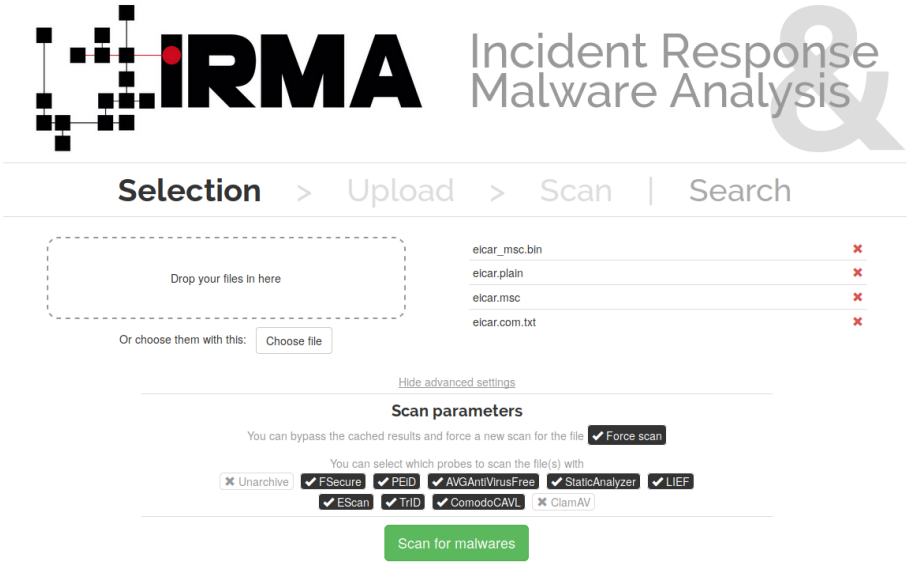
To cancel a file selection, click on the red cross next to the filename.

By clicking on the "Display advanced settings", you can see and determine scan parameters. Note that the defaults parameters are not reset by default after a scan.

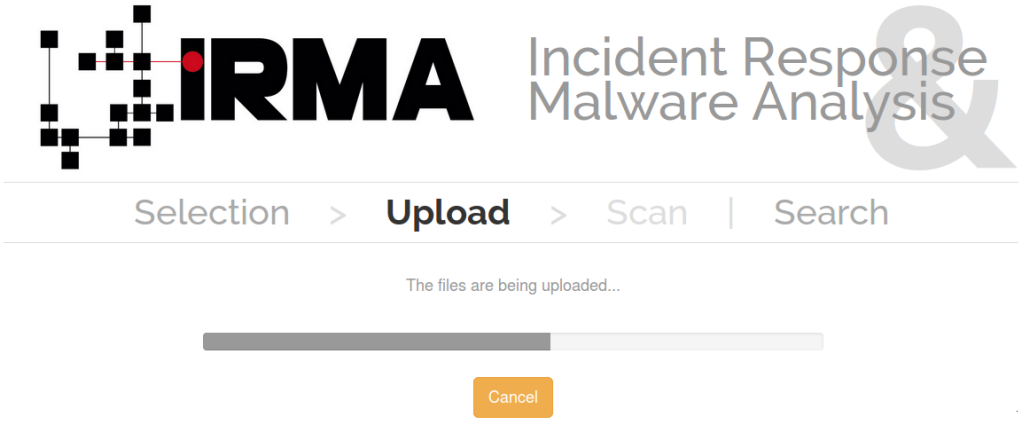


In the scan parameters you can choose if the scans will be forced, meaning that the files are unconditionally scaned, even if there is a cached result. You can choose too which probes will be lauched.

When you are ready, launch the scan by clicking on "Scan for malwares" button.



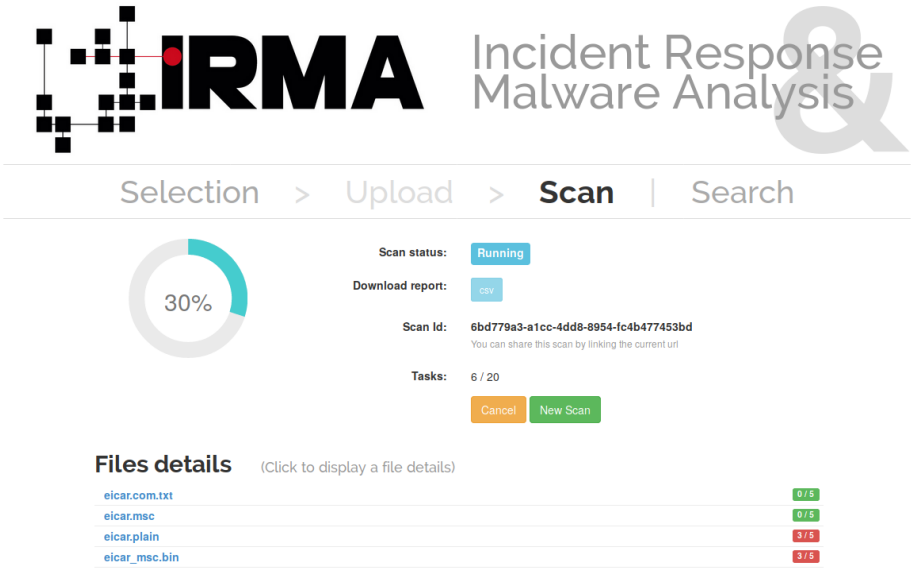
Wait during the upload of your files.



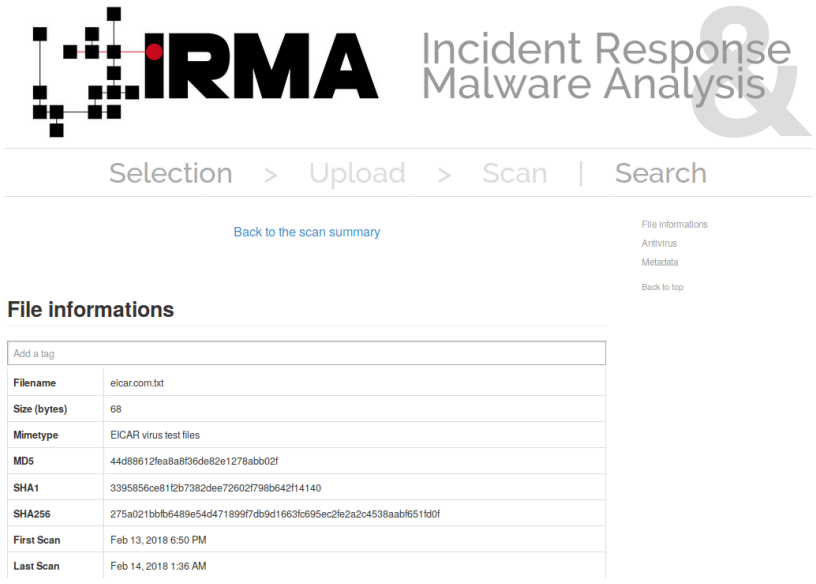
By now, you are on the results page. At the top, the scan status is displayed :

- The progression rate
- The scan status (if the scan is running or finish)
- The link to download the scan report in csv format
- The scan Id, a unique id to identify this scan that you can share
- The number of probe tasks done on the total number of probe tasks for the scan.

Next, the page displays the list of scan's files and their status. Click on the file's name to display the detailed scan result of a file scan.



In the first part of detailed scan result page, you can obtain information about the scanned file: filename, size, mime-type, different hashes, date of the first scan and the last scan of this file.



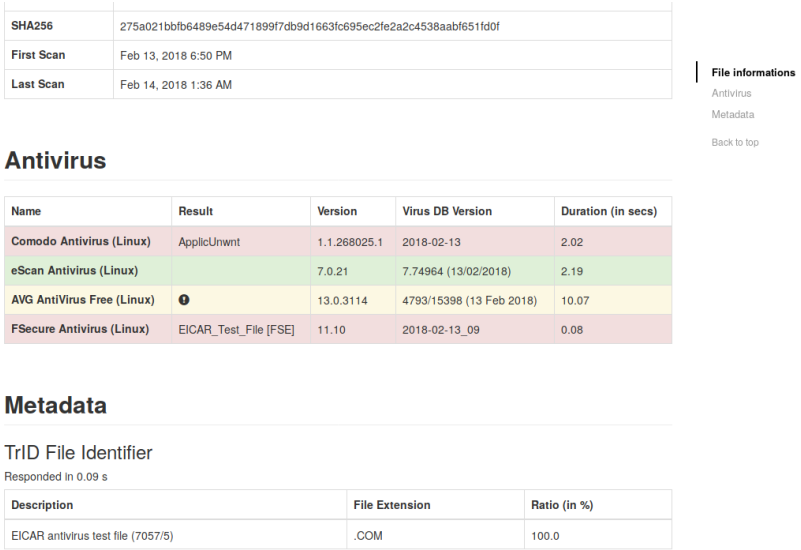
In the second part, you can see the details of the differents probe tasks ranked by probe type.

Firstly, the antivirus. For each antivirus, the following information are given:

- The name and the platform used
- The name of threat if it exist
- The version of the antivirus
- The version of the virus database
- The duration of the task

Note there is a color code to quickly see the status of the probe : green if everything is ok, red if a threat was be founded or orange if there was a problem with the probe.

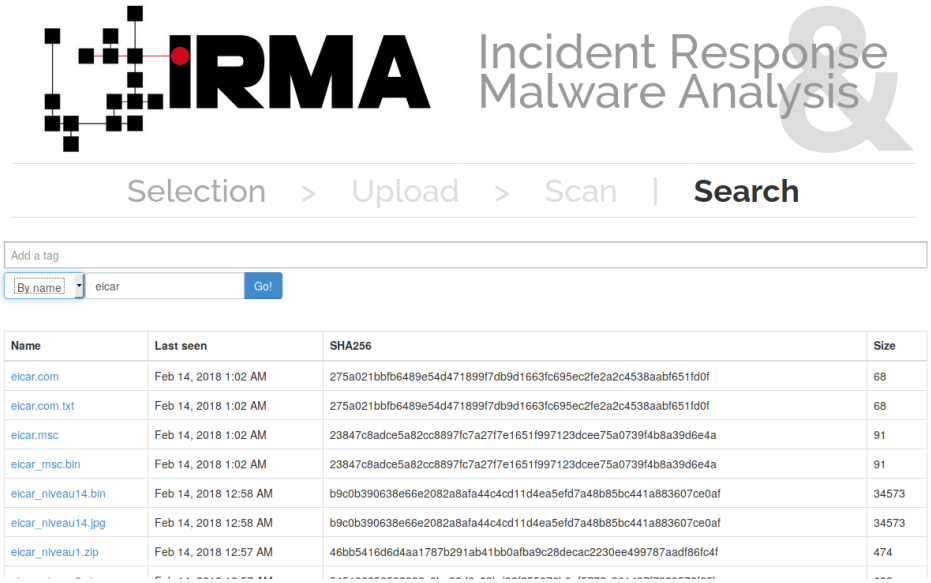
Then, it's the metadata and external parts : each probes of those classes have different ways to display their results.



3.1.2 How to do a research

It's possible to recover scan results in the "Search" section.

There are two ways to search scan : a research by name or a research by hash with a sha256. To this end, select in the scroll bar "By name" or "By hash" and effect your research : then a list of files' results ranked by date is displayed.



Note: To add a filter tag, see the section "Playing with tags"

3.1.3 Playing with tags

Note: Tags are available in IRMA from version 1.3.0

Creating a tag

You could create tags by using the command line tools

```
>>> from irmacl.helpers import *
>>> tag_list()
[]

>>> tag_new("archive")
{u'text': u'archive', u'id': 1}

>>> tag_list()
[Tag archive [1]]:
```

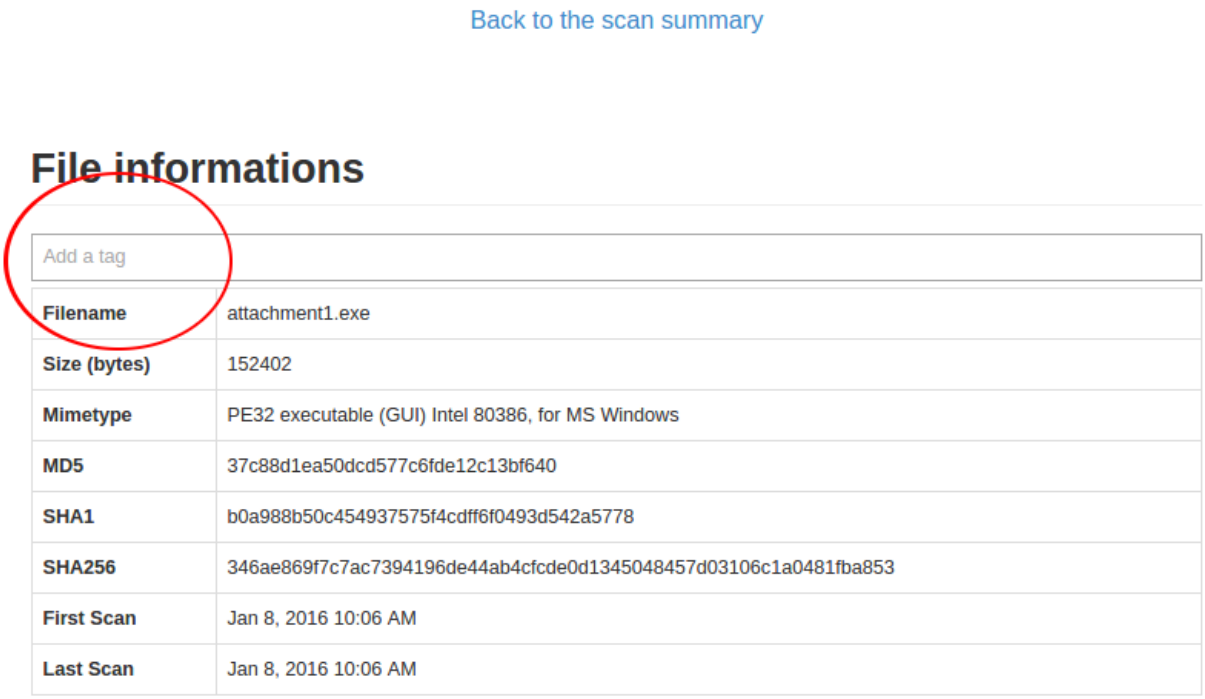
or directly from your terminal by using curl and posting a json with 'text' key:

```
$ curl -H "Content-Type: application/json; charset=UTF-8"  -X POST -d '{"text":"<your␣
↪tag>"}' http://172.16.1.30/api/v1.1/tags
```

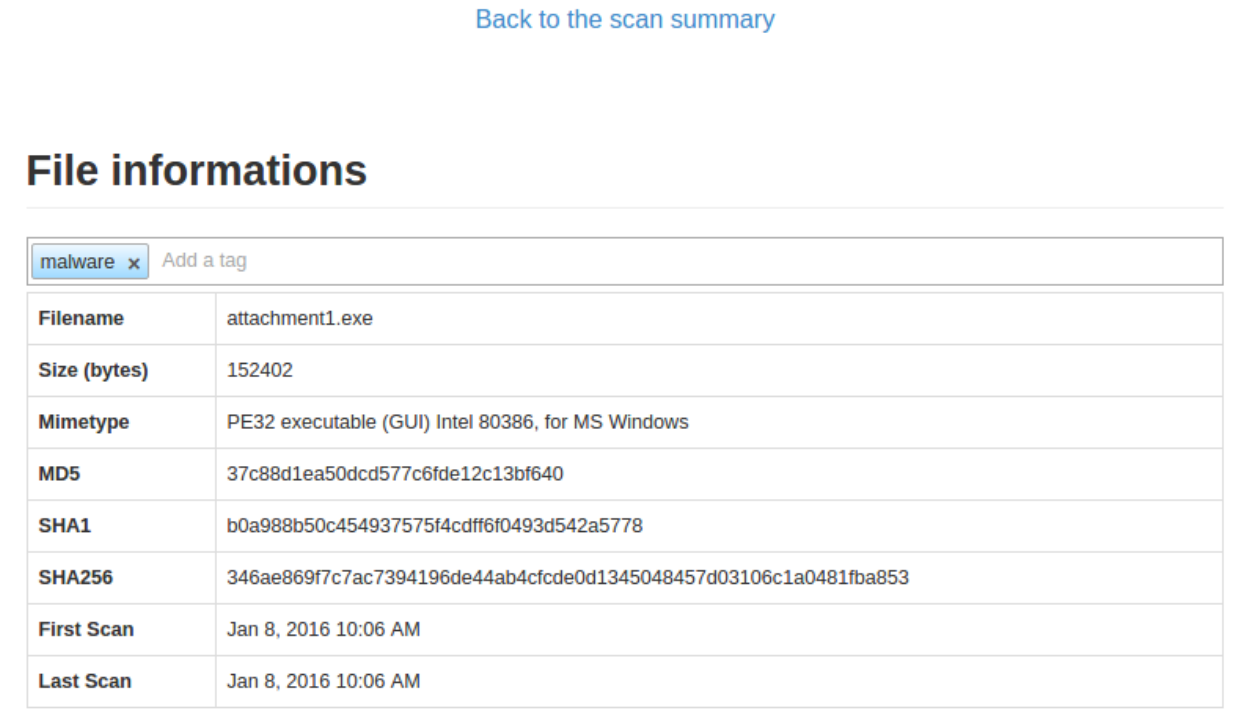
Note: There is currently no way to create a tag directly from the web IHM.

Tagging a File

Directly in web IHM, once you are on a file details page:



Just click the tag bar and you will see all available tags. You could add multiple tags.



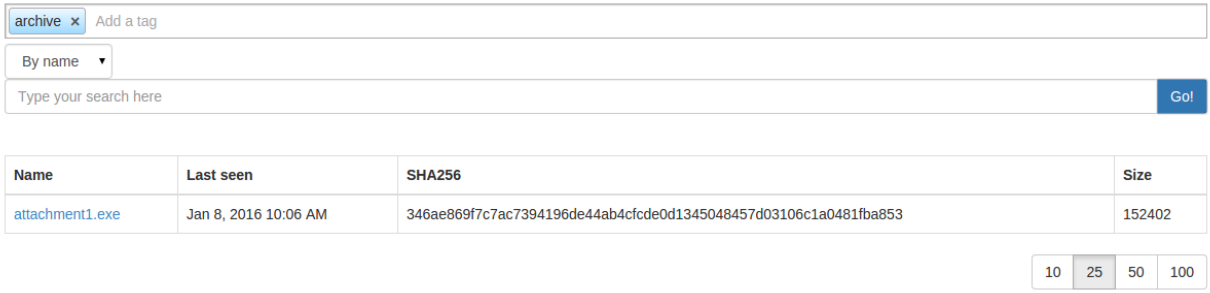
It is also possible to add a tag through command line tools:

```
>>> from irmacl.helpers import *
>>> help(file_tag_add)
Signature: file_tag_add(sha256, tagid, verbose=False)
Docstring:
Add a tag to a File

:param sha256: file sha256 hash
:type sha256: str of (64 chars)
:param tagid: tag id
:type tagid: int
:return: No return

>>> file_tag_add("346ae869f7c7ac7394196de44ab4cfcde0d1345048457d03106c1a0481fba853",1)
```

Searching by tag

You could specify one or more tags while searching for files too:

choose your tag list then hit the search button:



or by command line:

```
>>> from irmacl.helpers import *
>>> file_search(tags=[1])
(1, [<irma.apiclient.IrmaResults at 0x7f079ca23890>])
```

3.2 Command Line Interface

For a use of IRMA by command line, use the command line tools

This api client is only made for IRMA API version 1.1.

3.2.1 Installation

```
$ python setup.py install
```

Configuration file contains the API endpoint (full url) and some optional paramters (max number and delay in second between retries)

```
[Server]
api_endpoint=http://172.16.1.30/api/v1.1
max_tries=3
pause=1
```

and is searched in these locations in following order:

- current directory
- environment variable (“IRMA_CONF”)
- user home directory
- global directory (“/etc/irma”)

Once you set up a working irma.conf settings file, you could run tests on your running IRMA server:

```
$ python setup.py test
```

Pip Install

Install it directly with pip:

```
$ pip install irmacl
```

Usage

```
>>> from irmacl.helpers import *
>>> probe_list()
[u'StaticAnalyzer', u'Unarchive', u'VirusBlokAda', u'VirusTotal']

>>>  tag_list()
[Tag malware [1], Tag clean [2], Tag suspicious [3]]

>>>  scan_files(["./irma/tests/samples/eicar.com"], force=True, blocking=True)
Scanid: ca2e8af4-0f5b-4a55-a1b8-2b8dc9ead068
Status: finished
Options: Force [True] Mimetype [True] Resubmit [True]
Probes finished: 2
Probes Total: 2
Date: 2015-11-24 15:43:03
Results: [<irma.apiclient.IrmaResults object at 0x7f3f250df890>]

>>> scan = _
>>> print scan.results[0]
Status: 1
Probes finished: 2
Probes Total: 2
Scanid: ca2e8af4-0f5b-4a55-a1b8-2b8dc9ead068
Scan Date: 2015-12-22 14:36:21
Filename: eicar.com
Filepath: ./irmacl/tests/samples
ParentFile SHA256: None
Resultid: 572f9418-ca3c-4fdf-bb35-50c11629a7e7
FileInfo:
None
Results: None

>>> print scan_proberesults("572f9418-ca3c-4fdf-bb35-50c11629a7e7")
Status: 1
Probes finished: 2
Probes Total: 2
Scanid: ca2e8af4-0f5b-4a55-a1b8-2b8dc9ead068
Scan Date: 2015-12-22 14:36:21
Filename: eicar.com
Filepath: ./irmacl/tests/samples
ParentFile SHA256: None
Resultid: 572f9418-ca3c-4fdf-bb35-50c11629a7e7
FileInfo:
Size: 68
Sha1: 3395856ce81f2b7382dee72602f798b642f14140
Sha256: 275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f
Md5: 44d88612fea8a8f36de82e1278abb02fs
First Scan: 2015-11-24 14:54:12
Last Scan: 2015-12-22 14:36:21
Id: 3
Mimetype: EICAR virus test files
Tags: []
```

(continues on next page)

(continued from previous page)

```
Results: [<irmacl.apiclient.IrmaProbeResult object at 0x7f3f250b9dd0>, <irmacl.
↪apiclient.IrmaProbeResult object at 0x7f3f250b9850>]

>>> fr = _
>>> print fr.probe_results[0]
Status: 1
Name: VirusBlokAda (Console Scanner)
Category: antivirus
Version: 3.12.26.4
Duration: 1.91s
Results: EICAR-Test-File
```

Searching for scans

```
>>> scan_list()
(89, [Scanid: ef0b9466-3132-40b7-990a-415f08377f09
  Status: finished
  Options: Force [True] Mimetype [True] Resubmit [True]
  Probes finished: 1
  Probes Total: 1
  Date: 2015-11-24 15:04:27
[...]
```

Searching for files

```
>>> file_search(name="ei")
(1, [<irmacl.apiclient.IrmaResults at 0x7f3f250491d0>])

>>> (total, res) = _
>>> print res[0]
Status: 1
Probes finished: 1
Probes Total: 1
Scanid: 7ae6b759-b357-4680-8358-b134b564b1ca
Filename: eicar.com
[...]

>>> file_search(hash="3395856ce81f2b7382dee72602f798b642f14140")
(7,
 [<irmacl.apiclient.IrmaResults at 0x7f3f250b96d0>,
  <irmacl.apiclient.IrmaResults at 0x7f3f24fdc1d0>,
  <irmacl.apiclient.IrmaResults at 0x7f3f24fdca90>,
  <irmacl.apiclient.IrmaResults at 0x7f3f24fdcdd0>,
  <irmacl.apiclient.IrmaResults at 0x7f3f24fdc690>,
  <irmacl.apiclient.IrmaResults at 0x7f3f2504f390>,
  <irmacl.apiclient.IrmaResults at 0x7f3f24fea350>])

>>> file_search(hash="3395856ce81f2b7382dee72602f798b642f14140", tags=[1,2])
(0, [])

# looking for an unexisting tagid raise IrmaError
>>> file_search(hash="3395856ce81f2b7382dee72602f798b642f14140", tags=[100])
IrmaError: Error 402
```

Objects (apiclient.py)

class irmacl.apiclient.IrmaFileInfo(id, size, timestamp_first_scan, timestamp_last_scan, sha1, sha256, md5, mimetype, tags)

Bases: "object"

IrmaFileInfo Description for class

Variables:

- **id** – id
- **timestamp_first_scan** – timestamp when file was first scanned in IRMA
- **timestamp_last_scan** – timestamp when file was last scanned in IRMA
- **size** – size in bytes
- **md5** – md5 hexdigest
- **sha1** – sha1 hexdigest
- **sha256** – sha256 hexdigest
- **mimetype** – mimetype (based on python magic)
- **tags** – list of tags

pdate_first_scan – property, humanized date of first scan

pdate_last_scan – property, humanized date of last scan

raw()

class irmacl.apiclient.IrmaProbeResult(kwargs)**

Bases: "object"

IrmaProbeResult Description for class

Variables:

- **status** – int probe specific (usually -1 is error, 0 nothing found 1 something found)
- **name** – probe name
- **type** – one of IrmaProbeType ('antivirus', 'external', 'database', 'metadata'. . .)
- **version** – probe version
- **duration** – analysis duration in seconds
- **results** – probe results (could be str, list, dict)
- **error** – error string (only relevant in error case when status == -1)
- **external_url** – remote url if available (only relevant when type == 'external')
- **database** – antivirus database digest (need unformatted results) (only relevant when type == 'antivirus')
- **platform** – 'linux' or 'windows' (need unformatted results)

to_json()

class irmacl.apiclient.IrmaResults(file_infos=None, probe_results=None, **kwargs)

Bases: "object"

IrmaResults Description for class

Variables:

- **status** – int (0 means clean 1 at least one AV report this file as a virus)
- **probes_finished** – number of finished probes analysis for current file
- **probes_total** – number of total probes analysis for current file
- **scan_id** – id of the scan
- **scan_date** – date of the scan
- **name** – file name
- **path** – file path (as sent during upload or resubmit)
- **result_id** – id of specific results for this file and this scan used to fetch probe_results through file_results helper function
- **file_infos** – IrmaFileInfo object
- **probe_results** – list of IrmaProbeResults objects

to_json()

pscan_date – property, humanized date of scan date

class irmacl.apiclient.IrmaScan(id, status, probes_finished, probes_total, date, force, resubmit_files, mimetype_filtering, results=[])

Bases: "object"

IrmaScan Description for class

Variables:

- **id** – id of the scan
- **status** – int (one of IrmaScanStatus)
- **probes_finished** – number of finished probes analysis for current scan
- **probes_total** – number of total probes analysis for current scan
- **date** – scan creation date
- **force** – force a new analysis or not
- **resubmit_files** – files generated by the probes should be analyzed or not
- **mimetype_filtering** – probes list should be decided based on files mimetype or not
- **results** – list of IrmaResults objects

is_finished()

is_launched()

pdate – property, printable date

pstatus – property, printable status

class irmacl.apiclient.IrmaTag(id, text)

Bases: "object"

IrmaTag Description for class

Variables:

- **id** – id of the tag
- **text** – tag label

Helpers (helpers.py)

irmacl.helpers.file_download(sha256, dest_filepath, verbose=False)

Download file identified by sha256 to dest_filepath

Parameters:

- **sha256** (*str of 64 chars*) – file sha256 hash value
- **dest_filepath** (*str*) – destination path
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return tuple of total files and list of results for the given file

Return type: tuple(int, list of IrmaResults)

irmacl.helpers.file_results(sha256, limit=None, offset=None, verbose=False)

List all results for a given file identified by sha256

Parameters:

- **sha256** (*str of 64 chars*) – file sha256 hash value
- **limit** (*int*) – max number of files to receive (optional default:25)
- **offset** (*int*) – index of first result (optional default:0)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: tuple(int, list of IrmaResults)

irmacl.helpers.file_search(name=None, hash=None, tags=None, limit=None, offset=None, verbose=False)

Search a file by name or hash value

Parameters:

- **name** (*str*) – name of the file ('*name*' will be searched)
- **hash** (*str of (64, 40 or 32 chars)*) – one of sha1, md5 or sha256 full hash value
- **tags** (*list of int*) – list of tagid
- **limit** (*int*) – max number of files to receive (optional default:25)
- **offset** (*int*) – index of first result (optional default:0)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return tuple of total files and list of matching files already scanned

Return type: tuple(int, list of IrmaResults)

irmacl.helpers.file_tag_add(sha256, tagid, verbose=False)

Add a tag to a File

Parameters:

- **sha256** (*str of (64 chars)*) – file sha256 hash

- **tagid** (*int*) – tag id

Returns: No return

irmacl.helpers.file_tag_remove(sha256, tagid, verbose=False)

Remove a tag to a File

Parameters:

- **sha256** (*str of (64 chars)*) – file sha256 hash
- **tagid** (*int*) – tag id

Returns: No return

irmacl.helpers.probe_list(verbose=False)

List availables probes

Parameters: **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return probe list

Return type: list

irmacl.helpers.scan_add_data(scan_id, data, filename, post_max_size_M=100, verbose=False)

Add files to an existing scan

Parameters:

- **scan_id** (*str*) – the scan id
- **data** (*str*) – data to scan
- **filename** (*str*) – filename associated to data
- **post_max_size_M** (*int*) – POST data max size in Mb (multiple calls to the api will be done if total size is more than this limit, note that if one or more file is bigger than this limit it will raise an error)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return the updated scan object

Return type: IrmaScan

irmacl.helpers.scan_add_files(scan_id, filelist, post_max_size_M=100, verbose=False)

Add files to an existing scan

Parameters:

- **scan_id** (*str*) – the scan id
- **filelist** (*list*) – list of full path qualified files
- **post_max_size_M** (*int*) – POST data max size in Mb (multiple calls to the api will be done if total size is more than this limit, note that if one or more file is bigger than this limit it will raise an error)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return the updated scan object

Return type: IrmaScan

irmacl.helpers.scan_cancel(scan_id, verbose=False)

Cancel a scan

Parameters:

- **scan_id** (*str*) – the scan id
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return the scan object

Return type: IrmaScan

irmacl.helpers.scan_data(data, filename, force, post_max_size_M=100, probe=None, mimetype_filtering=None, resubmit_files=None, blocking=False,blocking_timeout=60, verbose=False)

Wrapper around scan_new / scan_add / scan_launch

Parameters:

- **data** (*str*) – data to scan
- **filename** (*str*) – filename associated to data
- **force** (*bool*) – if True force a new analysis of files if False use existing results
- **post_max_size_M** (*int*) – POST data max size in Mb (multiple calls to the api will be done if total size is more than this limit, note that if one or more file is bigger than this limit it will raise an error)
- **probe** (*list*) – probe list to use (optional default: None means all)
- **mimetype_filtering** (*bool*) – enable probe selection based on mimetype (optional default:True)
- **resubmit_files** (*bool*) – reanalyze files produced by probes (optional default:True)
- **blocking** (*bool*) – wether or not the function call should block until scan ended
- **blocking_timeout** (*int*) – maximum amount of time before timeout per file (only enabled while blocking is ON)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return the scan object

Return type: IrmaScan

irmacl.helpers.scan_files(filelist, force, post_max_size_M=100, probe=None, mimetype_filtering=None, resubmit_files=None, blocking=False,blocking_timeout=60, verbose=False)

Wrapper around scan_new / scan_add / scan_launch

Parameters:

- **filelist** (*list*) – list of full path qualified files
- **force** (*bool*) – if True force a new analysis of files if False use existing results
- **post_max_size_M** (*int*) – POST data max size in Mb (multiple calls to the api will be done if total size is more than this limit, note that if one or more file is bigger than this limit it will raise an error)
- **probe** (*list*) – probe list to use (optional default: None means all)
- **mimetype_filtering** (*bool*) – enable probe selection based on mimetype (optional default:True)
- **resubmit_files** (*bool*) – reanalyze files produced by probes (optional default:True)
- **blocking** (*bool*) – wether or not the function call should block until scan ended

- **blocking_timeout** (*int*) – maximum amount of time before timeout per file (only enabled while blocking is ON)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return the scan object

Return type: IrmaScan

irmacl.helpers.scan_get(scan_id, verbose=False)

Fetch a scan (useful to track scan progress with scan.pstatus)

Parameters:

- **scan_id** (*str*) – the scan id
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return the scan object

Return type: IrmaScan

irmacl.helpers.scan_launch(scan_id, force, probe=None, mimetype_filtering=None, resubmit_files=None, verbose=False)

Launch an existing scan

Parameters:

- **scan_id** (*str*) – the scan id
- **force** (*bool*) – if True force a new analysis of files if False use existing results
- **probe** (*list*) – probe list to use (optional default None means all)
- **mimetype_filtering** (*bool*) – enable probe selection based on mimetype (optional default:True)
- **resubmit_files** (*bool*) – reanalyze files produced by probes (optional default:True)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return the updated scan object

Return type: IrmaScan

irmacl.helpers.scan_list(limit=None, offset=None, verbose=False)

List all scans

Parameters:

- **limit** (*int*) – max number of files to receive (optional default:25)
- **offset** (*int*) – index of first result (optional default:0)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return tuple of total scans and list of scans

Return type: tuple(int, list of IrmaScan)

irmacl.helpers.scan_new(verbose=False)

Create a new scan

Parameters: **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return the new generated scan object

Return type: IrmaScan

irmacl.helpers.scan_proberesults(result_idx, formatted=True, verbose=False)

Fetch file probe results (for a given scan one scan <-> one result_idx

Parameters:

- **result_idx** (*str*) – the result id
- **formatted** (*bool*) – apply frontend formatters on results (optional default:True)
- **verbose** (*bool*) – enable verbose requests (optional default:False)

Returns: return a IrmaResult object

Return type: IrmaResults

irmacl.helpers.tag_list(verbose=False)

List all available tags

Returns: list of existing tags

Return type: list of IrmaTag

irmacl.helpers.tag_new(text, verbose=False)

Create a new tag

Parameters: **text** (*str*) – tag label (utf8 encoded)

Returns: None

CHAPTER 4

Administration

4.1 Environment configuration

Conf VMs with choice of probes

4.2 Components configuration

4.2.1 Frontend configuration

Configuration

The configuration file is located at `config/frontend.ini` in the installation directory.

Note: Detailed meaning of each field in `config/frontend.ini`:

Section	Key	Type	Default	Description
log	syslog	`integer`	0	enable rsyslog (experimental)
	prefix	`string`	irma-frontend:	prefix to append to rsyslog entries
	debug	`boolean`	False	enable Debug log
	sql_debug	`boolean`	False	enable SQL debug log
sqldb	username	`string`		database username
	password	`string`		database password
	host	`string`		database host
	port	`integer`		database port
	dbname	`string`		database name

Continued on next page

Table 1 – continued from previous page

Section	Key	Type	Default	Description
	tables_prefix	`string`		database tables prefix
samples_storage	path	`string`		Samples storage path
celery_brain	timeout	`integer`	60 (sec)	time before considering that the brain has timed-out
celery_frontend	timeout	`integer`	30 (sec)	time before considering that the frontend has timed-out
celery_options	concurrency	*integer*	0	number of concurrent workers (0 means nb of cores)
	soft_time_limit	`integer`	300 (sec)	time limit before task soft interrupt
	time_limit	`integer`	1500 (sec)	time limit before task is killed
	beat_schedule	`string`	/var/irma/frontend_beat_schedule	celery beat schedule file
broker_brain	host	`string`		hostname for the RabbitMQ server
	port	`integer`	5672	port for the RabbitMQ server
	vhost	`string`		virtual host configured for brain
	username	`string`		username used for brain on the RabbitMQ server
	password	`string`		password used for brain on the RabbitMQ server
	queue	`string`		queue to poll new tasks on the RabbitMQ server
broker_frontend	host	`string`		hostname for the RabbitMQ server
	port	`integer`	5672	port for the RabbitMQ server
	vhost	`string`		virtual host configured for this frontend
	username	`string`		username used for this frontend on the RabbitMQ server
	password	`string`		password used for this frontend on the RabbitMQ server
	queue	`string`		queue to poll new tasks on the RabbitMQ server

Continued on next page

Table 1 – continued from previous page

Section	Key	Type	Default	Description
ftp	protocol	`string`	"sftp"	choose File Transfer Protocol ("sftp" or "ftps")
ftp_brain	host	`string`		hostname for the FTP server
	port	`integer`	22	port for the FTP server
	auth	`string`	"password"	SFTP authentication method ("password" or "key")
	key_path	`string`		sftp private key absolute path
	username	`string`		username used by this frontend on the FTP server
	password	`string`		password used by this frontend on the FTP server
cron_clean_file _age	**clean_fs_max** _age	`string`	"0"	remove file when not scanned for givent time 0 means disabled ("1 hour", "5 days", "3w", "1year")
	clean_fs_age _cron_hour	`string`	0	cron hour settings
	clean_fs_age _cron_minute	`string`	0	cron minute settings
	clean_fs_age _cron_day_of_week	`string`	*	cron day of week settings
cron_clean_file _size	**clean_fs_max** _size	`string`	"0"	space's maximum size dedicated to the file system ("100 Mb", "512 Mb", "1.5Gb")
	clean_fs_size _cron_hour	`string`	*	cron hour settings
	clean_fs_size _cron_minute	`string`	0	cron minute settings
	clean_fs_size _cron_day_of_week	`string`	*	cron day of week settings
interprocess _lock	path	`string`	/var/run/lock/ir ma-frontend.lock	Concurrency file lock
ssl_config	activate_ssl	`boolean`	False	Enable RabbitMQ ssl

Continued on next page

Table 1 – continued from previous page

Section	Key	Type	Default	Description
	ca_certs	`string`		RabbitMQ SSL certs
	keyfile	`string`		RabbitMQ SSL keyfile
	certfile	`string`		RabbitMQ SSL certfile

Note: The default path for samples is /var/irma/samples/ make sure it exists with correct rights for irma user before launching your first scan.

4.2.2 Brain configuration

Configuration

The configuration file is located at `config/brain.ini` in the installation directory. Update it with your specific info.

Note: Detailed meaning of each field in `config/brain.ini`:

Section	Key	Type	Default	Description
log	syslog	`integer`	0	enable rsyslog (experimental)
	prefix	`string`	irma-brain:	prefix to append to rsyslog entries
	debug	`boolean`	False | enable Debug log	
	sql_debug	`boolean`	False | enable SQL debug log	
celery_options	concurrency	*integer*	0	number of concurrent workers (0 means nb of cores)
	soft_time_limit	`integer`	300 (sec)	time limit before task soft interrupt
	time_limit	`integer`	1500 (sec)	time limit before task is killed
broker_brain	host	`string`		hostname for the RabbitMQ server
	port	`integer`	5672	port for the RabbitMQ server
	vhost	`string`		virtual host configured for brain
	username	`string`		username used for brain on the RabbitMQ server
	password	`string`		password used for brain on the RabbitMQ server
	queue	`string`		queue to poll new tasks on the RabbitMQ server
broker_probe	host	`string`		hostname for the RabbitMQ server
	port	`integer`	5672	port for the RabbitMQ server
	vhost	`string`		virtual host configured for probes
	username	`string`		username used for probes on the RabbitMQ server
	password	`string`		password used for probes on the RabbitMQ server
	queue	`string`		queue to poll new tasks on the RabbitMQ server

Continued on next page

Table 2 – continued from previous page

Section	Key	Type	Default	Description
broker_frontend	host	string		hostname for the RabbitMQ server
	port	integer	5672	port for the RabbitMQ server
	vhost	string		virtual host configured for frontend
	username	string		username used for frontend on the RabbitMQ server
	password	string		password used for frontend on the RabbitMQ server
	queue	string		queue to poll new tasks on the RabbitMQ server
sqldb	dbms	string	sqlite	dbapi engine
	dialect	string		sqlalchemy dialect
	username	string		database username
	password	string		database password
	host	string		database host
	dbname	string	/var/irma/db/brain.db	database name
	tables_prefix	string		database tables prefix
ftp	protocol	string	“sftp”	choose File Transfer Protocol (“sftp” or “ftps”)
ftp_brain	host	string		hostname for the FTP server
	port	integer	21	port for the FTP server
	auth	string	“password”| SFTP authentication method (“password” or “key”)	
	key_path	string	sftp private key absolute path	
	username	string	username used by probe on the FTP server	
	password	string		password used by the probe on the FTP server
interprocess_lock	path	string	/var/run/lock/irma-brain.lock	Concurrency file lock
ssl_config	activate_ssl	boolean	False	Enable RabbitMQ ssl
	ca_certs	string		RabbitMQ SSL certs
	keyfile	string		RabbitMQ SSL keyfile
	certfile	string		RabbitMQ SSL certfile

Generate a SQLite database for scan tracking

You could easily generate the user database by running the following command. The path of the database is taken from the configuration file and the folder where the database is going to be stored must be created beforehand.

Note: The default path for the database is /var/irma/db/ make sure it exists before creating user database.

```
$ cd /opt/irma/irma-brain/current/
$ ./venv/bin/python -m scripts.create_user
usage: create_user <username> <rmqvhost> <ftpuser>
```

(continues on next page)

(continued from previous page)

```
      with <username> a string
           <rmqvhost> the rmqvhost used for the frontend
           <ftpuser> the ftpuser used by the frontend
example: create_user test1 mqfrontend frontend
```

To create an entry in the database for the frontend named `frontend` and which uses the `mqfrontend` virtual host on the RabbitMQ server, simply run the following commands:

```
$ ./venv/bin/python -m scripts.create_user frontend mqfrontend frontend
```

Note: There is a limitation due to SQLite. The folder where the database is stored, plus the database file must be writable by the user running the worker:

```
$ sudo chown irma:irma /var/irma/db/brain.db
$ sudo chmod a+w /opt/irma/irma-brain
```

4.2.3 Probe configuration

Configuration

The configuration file is `config/probe.ini` located in the installation directory.

Note: We recall in the following the meaning of each field in `config/probe.ini`:

Section	Key	Type	Default	Description
log	syslog	`integer`	0	enable rsyslog (experimental)
	prefix	`string`	irma-probe:	prefix to append to rsyslog entries
celery_options	concurrency	*integer*	0	number of concurrent workers (0 means nb of cores)
	soft_time_limit	`integer`	300 (sec)	time limit before task soft interrupt
	time_limit	`integer`	1500 (sec)	time limit before task is killed
broker probe	host	`string`		hostname for the RabbitMQ server
	port	`integer`	5672	port for the RabbitMQ server
	vhost	`string`		virtual host configured for probes
	username	`string`		username used for probes on the RabbitMQ server
	password	`string`		password used for probes on the RabbitMQ server
	queue	`string`		queue to poll new tasks on the RabbitMQ server
ftp_brain	host	`string`		hostname for the FTP server
	port	`integer`	21	port for the FTP server
	auth	`string`	“password”| SFTP authentication method (“password” or “key”)	
	key_path	`string`	sftp private key absolute path	
	username	`string`		username used by probe on the FTP server
	password	`string`		password used by the probe on the FTP server

4.3 SSL settings

SSL is available for 5 services:

- for an https connection with an nginx configuration;
- for RabbitMQ;
- for PostgreSQL with an authentication by certificate;

In the nominal case, enabling SSL for at least one of these services generates a PKI made of a root CA and, for each mechanism, an intermediate CA and some other stuff. Every CA and https certificate requires an openssl configuration file. These files are set in the appropriate directory in `./extras/pki/conf`: `root.config` at the root, a `<service>/ca.config` in the `https`, `rabbitmq` and `psql` directories and configuration file corresponding to https clients in `https` directory. The configuration files are copied in the corresponding directories during their generation.

The PKI is generated in the infra directory `./infras/<infra-name>/pki` where `<infra-name>` is the ansible variable `infra_name` in `group_vars/all.yml` (defaults to "Qb"). The PKI is described in `infras/<infra-name>/<infra-name>-infra.yml`. During the provisioning, ansible updates the PKI (or creates it) according to this file. To erase the PKI, delete the infra directory first.

4.3.1 HTTPS

Enable HTTPS

To enable SSL on the frontend server, edit `group_vars/all.yml` with:

```
frontend_openssl: True
nginx_https_enabled: True  # require frontend_openssl
nginx_https_client_enabled: True  # require nginx_https_enabled
```

Note: HTTPS and HTTP connections can operate at the same time.

Note: `nginx_https_enabled` [required] activates the server's certificate verification. `nginx_https_client_enabled` [optional] activates the client's certificate verification.

Generate certificates

The crypto objects for an https connection are generated in `infras/<infra-name>/pki/https`. By default, these are:

- a CA (key, certificate, chained certificate, database and CRL);
- a server (key, certificate, chained certificate);
- a client (key, certificate, chained certificate).

```
$ tree infras/Qb/pki/https
infras/Qb/pki/https/
├── ca
│   ├── 01.pem
│   ├── 02.pem
│   ├── ca-chain.crt
│   ├── ca.config
│   ├── ca.crt
│   ├── ca.key
│   ├── db
│   │   ├── ca.crl.srl
│   │   ├── ca.crl.srl.old
│   │   ├── ca.crt.srl
│   │   ├── ca.crt.srl.old
│   │   ├── ca.db
│   │   ├── ca.db.attr
│   │   ├── ca.db.attr.old
│   │   └── ca.db.old
│   └── https.crl
├── clients
│   ├── client-chain.crt
│   ├── client.config
│   ├── client.crt
│   ├── client.key
│   └── revoked
└── server
    ├── server-chain.crt
    ├── server.config
    ├── server.crt
    └── server.key
```

Add a client

To add a client:

- edit `infras/<infra-name>/<infra-name>-infra.yml` with:

```
---

infra:
  name: Qb
  https:
    clients:
      running:
        - name: client
        - name: new_client #there we indicate a the name of the new user
      revoked: []
```

- add an openssl configuration file `./extras/pki/conf/https/<client-name>.config` corresponding to the new user.
- provision with ansible: it copies the previous file in clients directory.

Revoke a client

To revoke a client:

- edit `infras/<infra-name>/<infra-name>-infra.yml` with:

```
---

infra:
  name: Qb

  clients:
    running:
      - name: client
    revoked:
      - name: bad_user # the user is now in revoked list and not in running list
```

- provision with ansible: it revokes the user with the user's CA and moves its stuff in `clients/revoked/`.

4.3.2 RabbitMQ

Enable SSL on RabbitMQ

To enable SSL in RabbitMQ, edit `group_vars/brain.yml` with:

```
rabbitmq_ssl: True
```

Note: If you are updating an already running no_ssl version, do the following on irma-brain RabbitMQ server:

```
$ sudo rabbitmqctl stop_app
$ sudo rabbitmqctl reset
$ sudo rabbitmqctl start_app
# create again the RabbitMQ vhosts, usernames and passwords:
$ sudo ./extras/scripts/rabbitmq/rmq_adduser.sh probe probe mqprobe
$ sudo ./extras/scripts/rabbitmq/rmq_adduser.sh brain brain mqbrain
$ sudo ./extras/scripts/rabbitmq/rmq_adduser.sh frontend frontend mqfrontend
```

Certificates generation

The crypto objects for RabbitMQ with SSL are generated in `infras/<infra-name>/pki/rabbitmq`. These are:

- a CA (key, certificate, chained certificate and database);
- a server brain (key, certificate);
- 3 clients for the entities frontend, brain and probe (key, certificate).

```
$ tree infras/Qb/pki/rabbitmq
infras/Qb/pki/rabbitmq/
├── ca
│   ├── 01.pem
│   ├── 02.pem
│   ├── 03.pem
│   ├── 04.pem
│   ├── ca-chain.crt
│   ├── ca.config
```

(continues on next page)

(continued from previous page)

```
│   ├── ca.crt
│   ├── ca.key
│   └── db
│       ├── ca.crt.srl
│       ├── ca.crt.srl.old
│       ├── ca.db
│       ├── ca.db.attr
│       ├── ca.db.attr.old
│       └── ca.db.old
├── clients
│   ├── brain-client.crt
│   ├── brain-client.key
│   ├── frontend-client.crt
│   ├── frontend-client.key
│   ├── probe-client.crt
│   └── probe-client.key
└── server
    ├── brain.crt
    └── brain.key
```

Note: In RabbitMQ case, only the CA needs a openssl configuration file.

4.3.3 Postgresql

Enable SSL on Postgresql

To activate SSL in PostgreSQL service, edit `group_vars/brain.yml` with:

```
postgresql_ssl: True
```

Generate certificates

The crypto objects for PostgreSQL with SSL are generated in `infras/<infra-name>/pki/psql`. These are:

- a CA (key, certificate, chained certificate, a CRL and database);
- a server (key, certificate);
- a client frontend (key, certificate).

```
$ tree infras/Qb/pki/psql
infras/Qb/pki/psql/
├── ca
│   ├── 01.pem
│   ├── 02.pem
│   ├── ca-chain.crt
│   ├── ca.config
│   ├── ca.crt
│   ├── ca.key
│   ├── db
│   │   ├── ca.crl.srl
│   │   ├── ca.crl.srl.old
```

(continues on next page)

(continued from previous page)

```
│   │   ├── ca.crt.srl
│   │   ├── ca.crt.srl.old
│   │   ├── ca.db
│   │   ├── ca.db.attr
│   │   ├── ca.db.attr.old
│   │   └── ca.db.old
│   └── psql.crl
├── clients
│   ├── frontend.config
│   ├── frontend.crt
│   ├── frontend.key
│   └── revoked
└── server
    ├── server.config
    ├── server.crt
    └── server.key
```

Revoke a client

To revoke a client:

- edit `infras/<infra-name>/<infra-name>-infra.yml` with:

```
---

infra:
  name: Qb

  psql:
    clients:
      revoked:
        - name: bad_user # bad_user is now in revoked list and no longer in␣
↪running list
```

- provision with ansible: it revokes the user with the user's CA and moves its stuff in `clients/revoked/`.

4.3.4 External PKI

It is also possible to use an external PKI for one or more of these services, for the root entity or the whole Irma's PKI. In this case, it is necessary to provide the corresponding cryptographic objects in PEM format. To specify which PKI's part are provided by an external PKI, edit `group_vars/all.yml` :

```
root_external: False
pki_rabbitmq_external: False
pki_https_external: False
pki_psql_external: False
```

By default, the automatic generation of the whole PKI is activated and all variables for external PKI are set to False.

External root

To use a external root, edit `group_vars/all.yml` with:

```
root_external: True
root_external_key: root_key.key
root_external_cert: root_cert.crt
```

Note: `root_key.key` and `root_external_cert` must contain the paths to respectively the key and the certificate of the external root entity.

The Irma's PKI will be generated with this external root as authority.

External HTTPS PKI

To use an external PKI for HTTPS and disable the automatic generation of a new one, edit `group_vars/all.yml` with:

```
pki_https_external: True
```

Provide the cryptographic objects and specify the paths editing `group_vars/frontend.yml`:

```
frontend_openssl_certificates:
  cert:
   src: https_server.crt
   dst: /etc/nginx/certs/{{ hostname }}.crt
  key:
   src: https_server.key
   dst: /etc/nginx/certs/{{ hostname }}.key
  ca:
   src: https_ca_cert.crt
   dst: /etc/nginx/certs/ca.crt
  chain:
   src: https_ca_chain.crt
   dst: /etc/nginx/certs/ca-chain.crt
  crl:
   src: https_crl.crl
   dst: /etc/nginx/certs/https.crl
```

Note: `frontend_openssl_certificates.cert.src` is the path to the server's certificate `frontend_openssl_certificates.key.src` is the path to the server's private key `frontend_openssl_certificates.ca.src` is the path to the CA's certificate `frontend_openssl_certificates.chain.src` is the path to the CA's certification chain `frontend_openssl_certificates.crl.src` is the path to the CRL

External RabbitMQ PKI

To use an external PKI for RabbitMQ and disable the automatic generation of a new one, edit `group_vars/all.yml` with:

```
pki_rabbitmq_external: True
```

Provide the cryptographic objects and specify the paths editing `group_vars/all.yml`:

```
rabbitmq_cacert : ca-chain.crt
rabbitmq_server_key : server.key
rabbitmq_server_cert: server.crt
rabbitmq_frontend_key: frontend-client.key
rabbitmq_frontend_cert: frontend-client.crt
rabbitmq_brain_key: brain-client.key
rabbitmq_brain_cert: brain-client.crt
rabbitmq_probe_key: probe-client.key
rabbitmq_probe_cert: probe-client.crt
```

Note: `rabbitmq_cacert` is the path to the CA's certification chain `rabbitmq_server_key` is the path to the server's private key `rabbitmq_server_cert` is the path to the server's certificate `rabbitmq_frontend_key` is the path to the frontend's private key `rabbitmq_fontend_cert` is the path to the frontend's certificate `rabbitmq_brain_key` is the path to the brain's private key `rabbitmq_brain_cert` is the path to the brain's certificate `rabbitmq_probe_key` is the path to the probes' private key `rabbitmq_probe_cert` is the path to the probes' certificate

External PostgreSQL PKI

To use an external PKI for PostgreSQL and disable the automatic generation of a new one, edit `group_vars/all.yml` with:

```
pki_psql_external: True
```

Provide the cryptographic objects and specify the paths editing `group_vars/sql-server.yml`:

```
postgresql_ssl_cert_src_path: server.crt
postgresql_ssl_key_src_path: server.key
postgresql_ssl_ca_src_path: ca-chain.crt
postgresql_ssl_crl_src_path: psql.crl
```

Note: `postgresql_ssl_cert_src_path` is the path to the server's certificate `postgresql_ssl_key_src_path` is the path to the server's private key `postgresql_ssl_ca_src_path` is the path to the CA's certificate chain `postgresql_ssl_crl_src_path` is the path to the CRL

4.4 Database migration

IRMA uses Alembic to manage and perform databases migration.

Note: Alembic is a useful tool to manage migration, but can't surpass local engine implementation of SQL. As `SQLite` doesn't manage schema modifications such as `ALTER_COLUMN`, the whole migration system of IRMA won't support it. The preferred database engine is `PostgreSQL`.

You can still use SQLite, but you will be on your own for migrations.

Warning: Please note that most of the manipulations on this can and sometimes will alter your data. If you are not sure about what you are doing, and even if you are sure, **make backup**.

4.4.1 Requirements

- Alembic package

4.4.2 Content

Database migrations are managed in the **frontend** and **brain** IRMA components.

The files/directories used are:

```
alembic.ini
extras/migration/
  +- env.py
  +- script.py.mako
  +- versions/
       +- <revision_1>.py
       +- <revision_2>.py
       +- ...
```

Note: All the commands below will assert to be executed on top of this file system, as Alembic needs the `alembic.ini` configuration file.

You could also use the `-c <path_to_conf_file>`.

4.4.3 Usage

Alembic manage a 'revision' for each database evolution. These revisions are used to upgrade or downgrade the database schema.

The command:

```
$ alembic current
```

... shows the current revision of the database.

The command to get the history of the latest alembic migrations is:

```
$ alembic history --verbose
```

Create database from scratch with Alembic

Configuration and creating database

Alembic will use the information in the `[sqldb]` section of the configuration files (respectively `config/frontend.ini` or `conf/brain.ini` for the repositories of the frontend or the brain components). Make sure they are accurate.

The database must already exist. This step is quite simple, the SQL command usually being:

```
sql$ CREATE DATABASE <db_name>;
```

Update your schema with Alembic

If you use a virtualenv, activate it. Then enter:

```
$ alembic upgrade head
```

Alembic applies each revision one after the other. At the end of the process, if no error occurs, your database should be updated.

Note: You can update the database one revision at a time, or up to a specific revision. See the *revisions* section for further information.

If you already have a database WITHOUT Alembic

Alembic stores its current revision number in database. If your database doesn't have this information, you are very likely to encounter errors when using Alembic, as it will try to create already existing tables.

The easiest solution is to destroy your database and go for a fresh install.

Although, if you don't want to lose your data, you could update the Alembic information manually.

You will need to:

1. Get the exact current Alembic revision of your database. Each migration file has a `Revision ID` in its header. Investigate the successive revisions to know which one matches your current database state.
2. Once you known your Alembic revision, run:

   ```
   $ alembic stamp <your_alembic_revision_number>
   ```

3. Your database is now synchronized with Alembic! You should be able to use Alembic to upgrade/downgrade your database now. Be aware that if the revision number you provided is false, you could encounter massive errors while attempting to upgrade/downgrade your database.

Generating a new revision

Creating a new revision can be done with the command:

```
$ alembic revision -m <revision_message>
```

This command produces a new `<hash>_<revision_message>.py` file in the `extras/migration/versions/` directory. This file contains two functions `upgrade` and `downgrade`, respectively used to upgrade the database to the revision, or downgrade from it. These two functions are empty and must be completed with the desired modifications (see the alembic documentation section ops).

A revision could be produced automatically, from database metadata defined in the IRMA SQL objects description through `sqlalchemy`, with the command:

```
$ alembic revision --autogenerate -m <revision_message>
```

These SQL objects are defined in:

- `frontend/models/sqlobjects.py` for the frontend,
- `brain/models/sqlobjects.py` for the brain.

Alembic scripts in IRMA repositories are already configured to use metadata defined in these files. You should be able to use the `--autogenerate` option without further modifications.

Note: IRMA configuration allows to prefix table names through configuration. Our revision files use the function `<frontend_or_brain>/config/parser.py:prefix_table_name` to generate table names rather than keeping alembic-generated plain string names. A good practice would be to keep using this function in revision files.

Warning: Alembic easily detects changes such as adding/removing columns, but could be blind on thin, inner modifications. Re-reading the auto-generated script is a strongly recommended step before actually performing the migration.

See the alembic documentation section autogenerate for more information.

Warning: Database modifications using `ALTER_COLUMN` (such as changing the type of a column) can't be performed on `SQLite` databases. Be aware of this limitation if you **absolutely** want to use migration scripts with this SQL engine.

Migrating between revisions

Once the revision is properly described, the migration is performed with:

```
$ alembic upgrade head
```

Alembic allows to migrate the database to any revision, relatively to the current revision or absolutely. Several examples:

```
$ alembic upgrade +4
$ alembic downgrade base
$ alembic upgrade <revision_number>+3
```

4.4.4 Tips and tricks

Note: Don't trust Alembic too much. It is nothing more than a tool, without any comprehension on the code. Cautiously read the revision scripts it generates.

Note: Database migration is hardly ever a painless step. Be sure to:

1. save your data before performing a migration,
2. test your application after the migration to ensure its compatibility with the new data schemes.

Note: With a PostgreSQL database, the Float type is tolerated but the real type name used by the database is Real. It means that SQL objects described in sqlalchemy with Float columns will be properly applied in database, but at each autogenerate revision, alembic will see Real type in database, against Float type in the code metadata, and so will perform each time a useless alter_column from Real to Float. This problem could be avoided (with PostgreSQL) by declaring Real instead of Float.

See this page for more information on PostgreSQL numeric types.

Note:

Alembic can't directly deal with many somehow complex operations, such as type migration with no trivial cast. In these cases, the operation must be manually described with a raw SQL command (which could be database-dependent).

For instance, alembic can't perform the migration from real to datetime:

```
> alembic.alter_column('table', 'column',
                       existing_type=sqlalchemy.REAL(),
                       type_=sqlalchemy.DateTime(),
                       existing_nullable=False)
```

... because of an error a column "column" cannot be cast automatically to type timestamp with time zone.

A proper migration for PostgreSQL would be (in Python):

```
> alembic.execute('ALTER TABLE "table" ALTER COLUMN "column" TYPE TIMESTAMP WITHOUT␣
↪TIME ZONE USING to_timestamp(column)')
```

And the reverse code to downgrade the migration could be:

```
> alembic.execute('ALTER TABLE "table" ALTER COLUMN "column" TYPE REAL USING␣
↪extract(epoch from column)')
```

Note: Rather than managing migrations directly with Alembic, we could generate SQL migration revision to be used directly on database with the command:

```
$ alembic upgrade <revision> --sql > migration.sql
```

Note: Deleting a revision *R* is simple:

- downgrade the database to the revision before *R-1* the revision you want to delete;
- if any, edit the script of the following revision *R+1* and update the down_revision variable to match the revision number of revision *R-1*;
- delete the script of the revision *R* you want to delete;
- upgrade your database.

The deleted revision want be applied any more.

CHAPTER 5

Technical description

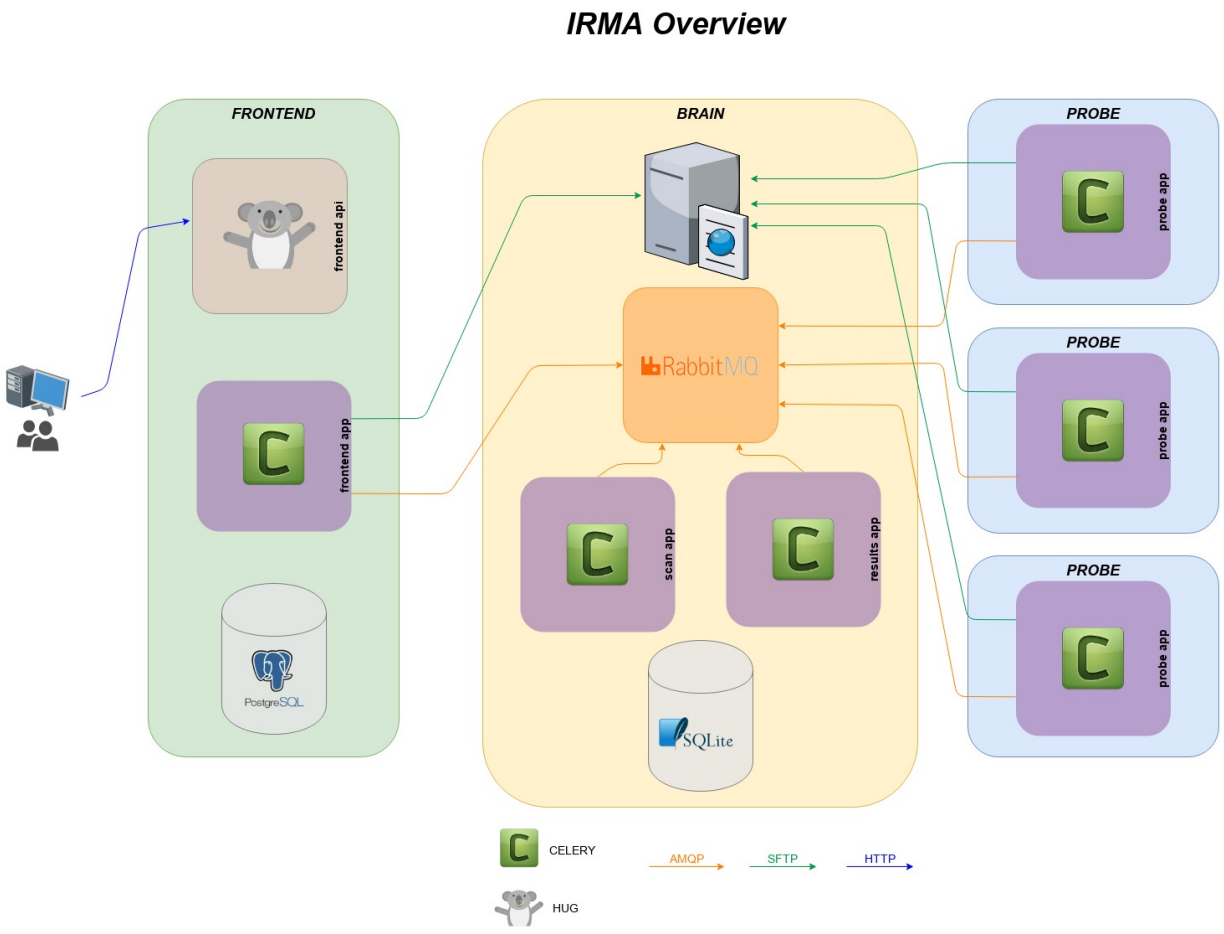
Each major component of the IRMA platform comes with their own python-based application. As the **Brain** is the nerve center of the whole platform, it is recommended to install it first before installing other components. One can then install either the **Frontend** or the **Probes** he wants.

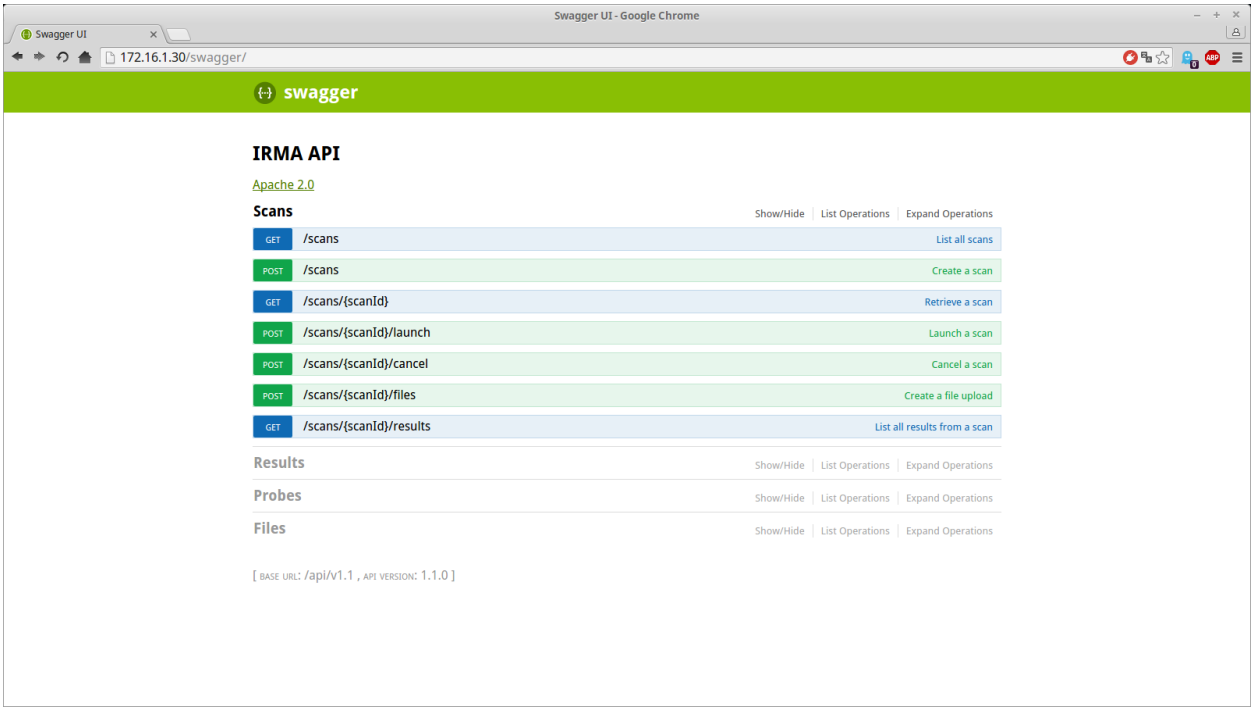
The IRMA entrypoint is the web API hosted on **frontend**. File results are stored in **PostgreSQL database**. All files transfers are done through FTP (**sftp server** on **brain**). All tasks are executed by celery applications that consumes their own task queue on **RabbitMQ server**. For further details give a look at **scan workflow** part

5.1 API documentation

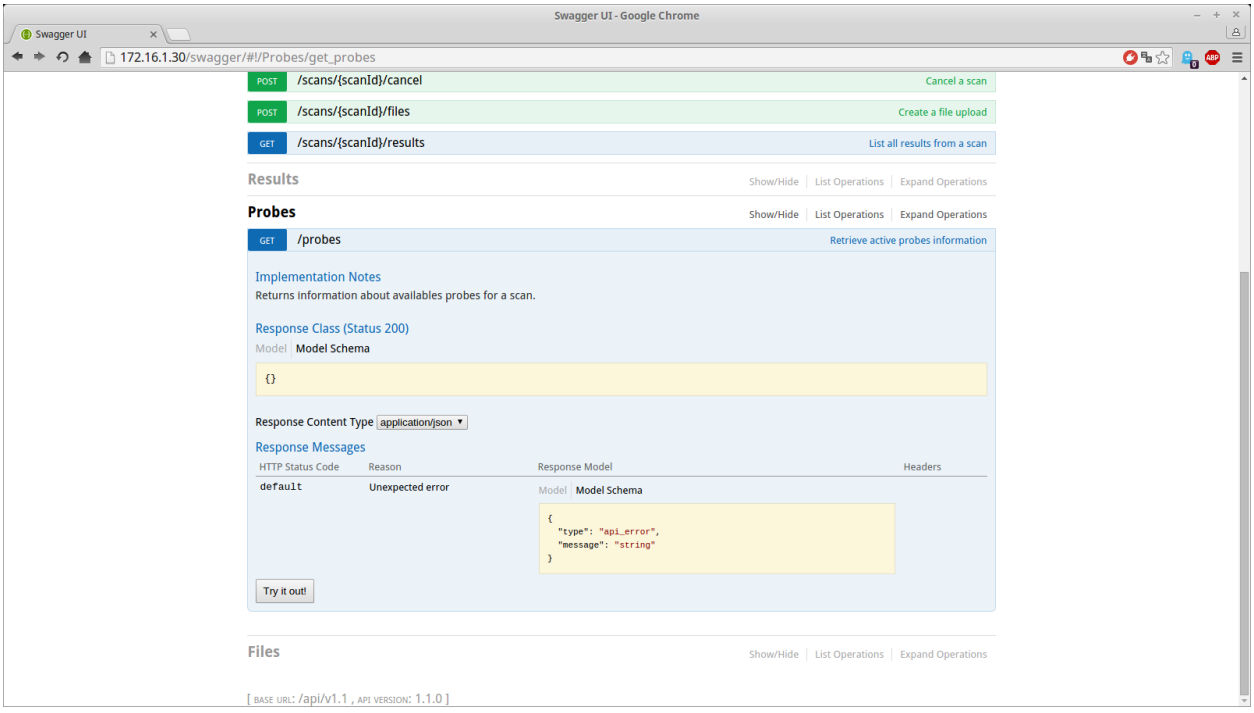
There is a dynamic documentation for IRMA API available on your instance

It allows you to read documentation but also try request and see server response.

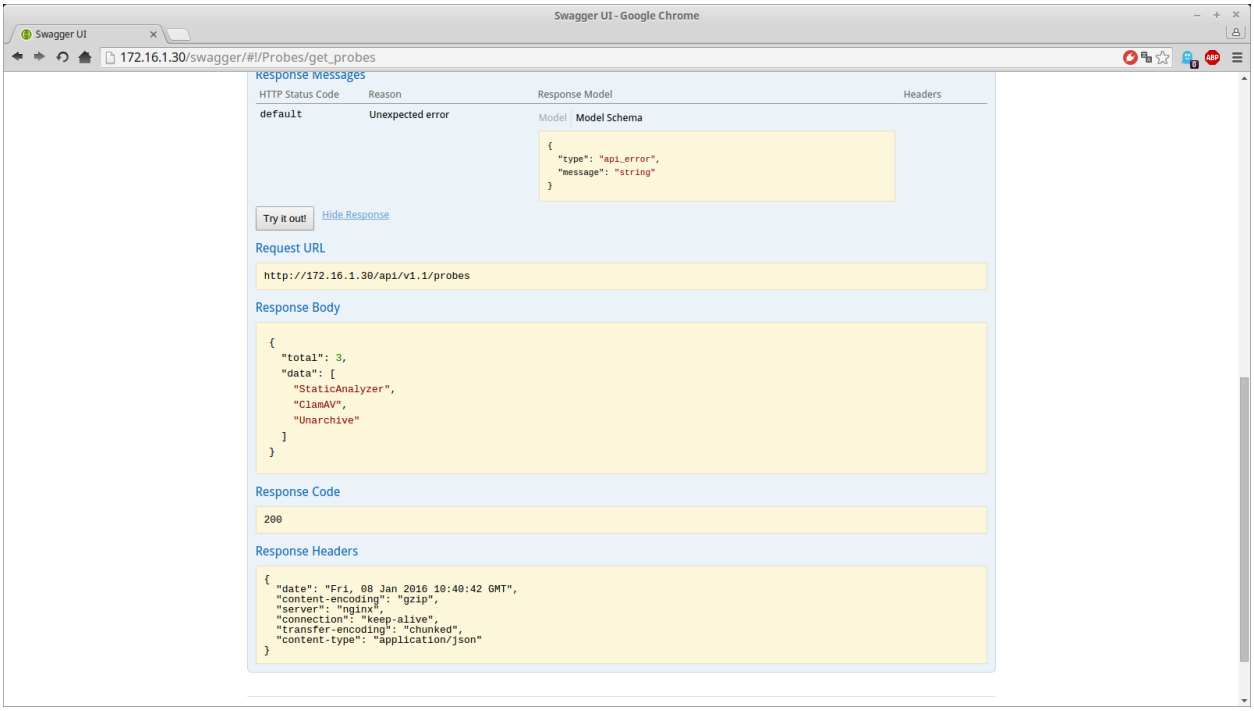

IRMA Overview
FRONTEND
frontend api
frontend app
PostgreSQL
BRAIN
RabbitMQ
scan app
results app
SQLite
PROBE
probe app
PROBE
probe app
PROBE
probe app
CELERY
AMQP
SFTP
HTTP
HUG



You could see detailed information about one specific API route:



and by clicking on the *Try it* button, see the server response:



5.2 Frontend

The **Frontend** handles scan submission to the **Brain**, stores the results of the scanned files. These results can be displayed through a web graphical user interface or via the command line interface.

5.2.1 Installation

The **Frontend** must be installed on a GNU/Linux system. With some efforts, it should be possible to run it on a Microsoft Windows system, but this has not been tested yet.

This section describes how to get the source code of the application and to install it.

5.2.2 Architecture

Let us recall first the inner architecture of the **Frontend**. It uses multiple technologies with each a specific purpose:

- A client through which a user submits a file and get the analysis results. There are two clients bundled in the repository: a web user interface and a command-line client.
- A python-based restful API, served by a NGINX web server and a uWSGI application server. It gets the results of a file scan by querying a database.
- A worker that will handle scan submission to the **Brain** and store the results of analyzes scheduled by the **Brain**. The worker relies on Celery, a python-based distributed task queue.
- A database server (PostgreSQL) is used to store results of analyzes made on each file submitted either by the web graphical interface or the CLI client.

5.3 Brain

The **Brain** is a python-based application that only dispatches analysis requests from different frontends[1] to the available **Probes**. Analyses are scheduled by the **Brain** on **Probes** through Celery, an open source task.

5.3.1 Installation

The **Brain** must be installed on a GNU/Linux distribution. With some efforts, it should be possible to run it on a Microsoft Windows system, but this has not been tested yet.

This section describes how to get the source code of the application for the **Brain** and to install it.

5.3.2 Architecture

Let us recall first the inner architecture of the **Brain**. It uses multiple technologies with a specific purpose each:

- a Celery worker that handles scan requests from **Frontends** and results returned by the **Probes**.
- a RabbitMQ server used by Celery as a backend and as a broker for task queues and job queues used to schedule tasks for **Probes** (for scan jobs) and the **Frontend** (for scan results).
- an SFTP server where files to be scanned are uploaded by **Frontends** and downloaded by **Probes**,

5.3.3 Nginx

In the **Frontend**, we use a nginx web server to serve the uWSGI application and the static web site that query the API in order to get results of scanned files and to present them to the user.

5.3.4 SQL server

The Frontend relies on a PostgreSQL database to keep track of all scans info.

5.4 Probe

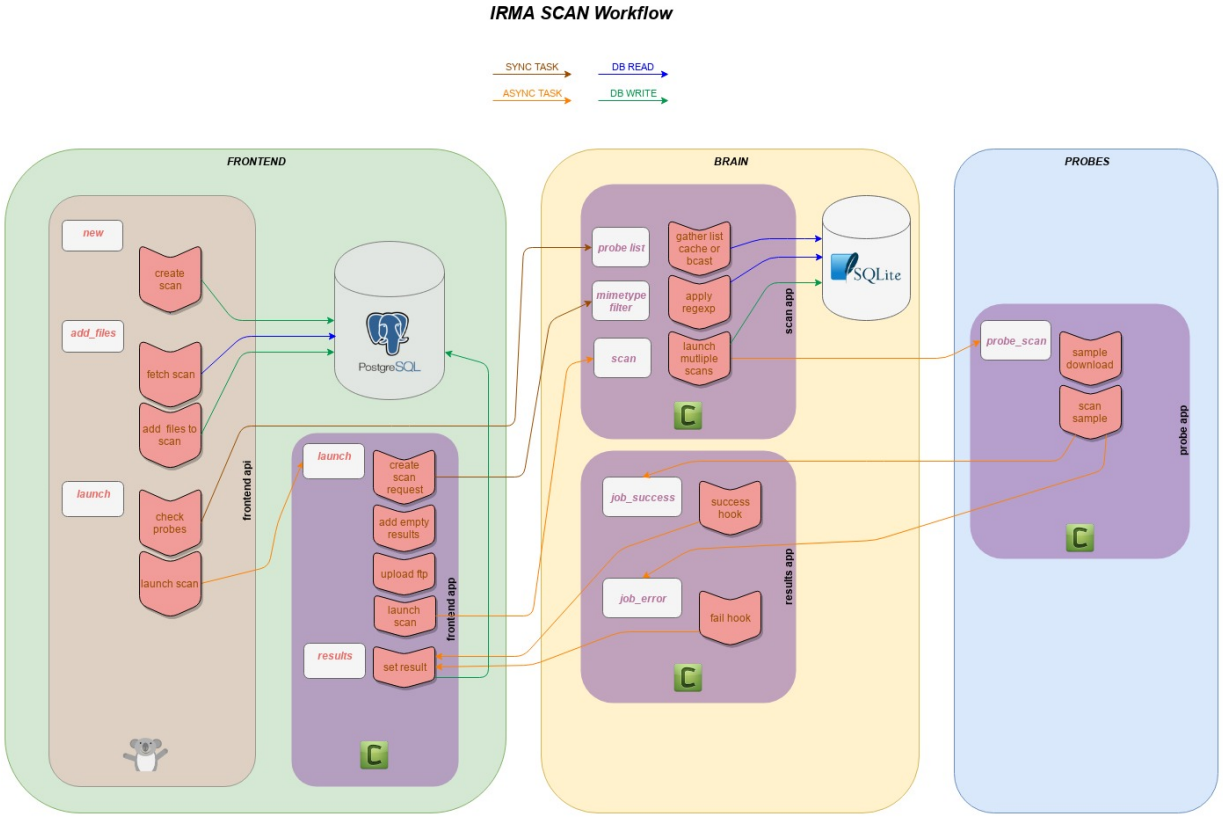
The **Probes** are python-based application that host a single or multiple analyzers. Each analyzer listens on a specific work queue and waits for an analysis to be scheduled by the **Brain** through Celery, an open source task framework for Python. Python version should be at least 3.4 on linux, 3.5 on windows.

5.4.1 Architecture

Probes are mainly Celery workers that handle scan requests from **Brain**

[1] This feature is not ready yet, we are currently working on its implementation.

5.5 Scan workflow



5.5.1 Frontend API Part (frontend_api/uwsgi+hug)

1. A new scan object is created in **PostgreSQL** database.
2. Files are uploaded to the WEB API, stored on Filesystem and registered in **PostgreSQL** database.
3. Scan is launched, an asynchronous task is launched on **Frontend** celery.

5.5.2 Frontend Celery Part (frontend_app/celery)

1. Used probes are filtered according to scan options (selected probes, mimetype filtering).
2. Empty results are created in **PostgreSQL** database (one per probe per file).
3. Each file is uploaded to **SFTP** server.
4. For each file uploaded a scan task on **Brain** is launched with the file probelist (according to scan option *force* some results could already be present).

5.5.3 Brain Celery Part (scan_app/celery)

1. A new scan object is created in **SQLite** database to track jobs (for canceling).

2. Each file is send for analysis in sent for analysis in every probe selected (each time a probe is available in IRMA, it registers itself to the brain and open a **RabbitMQ** Queue named with its probe name, probe list is retrieved by listing active queues).
3. Two callbacks are set on every probe scan tasks, one for success and the other for failure.

5.5.4 Probe Celery Part (probe_app/celery)

1. Scan task is received with a file id.
2. File is downloaded as temporary file.
3. File is scanned by the probe.
4. Results are sent back to **Brain** to one of the two callbacks set.

5.5.5 Brain Celery Part (result_app/celery)

1. successful results are marked as completed in **SQLite** database.
2. successful results are forwarded to **Frontend**.
3. error are marked as completed in **SQLite** database.
4. As there is no result, an error message is generated to tell the **Frontend** the particular job for the file and probe failed.

5.5.6 Frontend Celery Part (frontend_app/celery)

1. Results is received for each file and probe.
2. Results are updated in **PostgreSQL** database.
3. If scan is finished, a scan flush task is launched on **Brain** to delete files on **SFTP** server.

5.6 Functional Testing

Only available on *dev* environments

On the frontend, to launch the functionals tests:

```
$ cd /opt/irma/irma-frontend/current/web
$ npm run functional-tests
```

It will launch the Javascript implementation of Cucumber. Cucumber.js will take a file that contain test scenarios, written using the Gherkin language). For each steps of a scenario, an action is perform by Cucumber.js, like accessing a web page and typing some texts in a form. In order to do that, IRMA project uses Puppeteer, a software that can launch and control a Chromium instance in headless mode.

IRMA scenarios can be found in `frontend/web/tests/functionals/*.feature` and actions used to perform these steps are available in the file `frontend/web/tests/functionals/support/steps.js`.

When an error occured, you will get a screenshot of the page where the scenario ends. It will be available on the VM at `frontend/web/error.jpeg`.

5.6.1 Debug

Using the headless mode of Chromium, it will be difficult to debug if an error occured.

You can launch the test using a real Chromium instance on your host:

- You'll need NodeJS and NPM
- Install IRMA web interface devDependencies on your host

```
$ cd frontend/web
$ npm install --only=dev
```

- Update the `ROOT_URL` (see: `frontend/web/tests/functionals/support/steps.js`) variable to the location of your IRMA web url (for example: `const ROOT_URL="http://172.16.1.30"`) and toggle the `HEADLESS` variable to `false` (see: `frontend/web/tests/functionals/support/hooks.js`)
- Run the tests:

```
$ npm run functional-tests
```

You can also use the power of X11 Forwarding through SSH to see a real browser launching the tests on the VM and getting the result on the host, without having to install NodeJS:

```
$ vagrant ssh # to connect as vagrant user/superuser
$ sudo apt-get install xorg # to install a X11 server to launch Chromium
$ sudo sed -i "s/^X11Forwarding .*/X11Forwarding yes/" /etc/ssh/sshd_config
$ sudo systemctl restart sshd
$ exit # disconnect from vagrant user
$ vagrant ssh -- -l deploy -X # to connect as deploy with XForwarding enable
$ cd /opt/irma/irma-frontend/current/web
$ sed -i "s/^const HEADLESS = true;/const HEADLESS = false;/" tests/functionals/
↪support/hooks.js
$ npm run functional-tests
```

You should see an instance of a Chromium browser on your host machine, running the tests.

Take a look at the argument pass to the `puppeteer.launch()` function in `frontend/web/tests/functionals/support/hooks.js`. For example, by modifying the `SLOW_MOTION_DELAY` you can force Puppeteer to slow down its operations.

CHAPTER 6

Extending IRMA

6.1 Adding a new probe

6.1.1 Writing a Plugin for the probe

Note: To be a valid probe module, IRMA expects it to have a predefined structure. To save time, one can get a minimal working structure from the skeleton plugin. The new plugin is stored in the appropriate sub-directory of the directory probe/modules according to the type of the new probe (antivirus, metadata, external...).

For a probe that is not a antivirus

1. Copy the directory skeleton to the new module (appropriate localisation). Example with a module my_module with metadata type :

```
$ cp -r probe/modules/custom/skeleton/ probe/modules/metadata/my_module
```

2. If there are packages to install, specify them in the file requirements.txt. Otherwise remove the file
3. Adjust the file plugin.py according to the module :

- Adjust the class's name with the name of your probe
- Fill in the fields of the class :- _plugin_name_ = [the plugin name]
  - _plugin_display_name_ = [the field _name of the class of the probe]
  - _plugin_version_ = [the version number]
  - _plugin_category = [the type of the probe: IrmaProbeType.]
  - _plugin_description = [quick description]
  - _plugin_dependencies = [list of dependencies: platform, binary or/and file] => if used import from lib.plugins PlatformDependency, BinaryDependency or/and FileDependency

  - _mimetype_regexp = [mimetype corresponding]

4. Implement the functions corresponding to the type of the plugin

For an antivirus

In the case of an antivirus, it is a little different because an Antivirus class was created to avoid code's duplication. You can use the skeleton below:

plugin.py:

```
#
# Copyright (c) 2013-2018 Quarkslab.
# This file is part of IRMA project.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License in the top-level directory
# of this distribution and at:
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# No part of the project, including this file, may be copied,
# modified, propagated, or distributed except according to the
# terms contained in the LICENSE file.

from .skeleton import Skeleton

from ..interface import AntivirusPluginInterface
from irma.common.plugins import PluginMetaClass


class SkeletonPlugin(AntivirusPluginInterface, metaclass=PluginMetaClass):

    # =================
    #  plugin metadata
    # =================
    _plugin_name_ = "Skeleton"
    _plugin_display_name_ = Skeleton._name
    _plugin_author_ = "IRMA (c) Quarkslab"
    _plugin_version_ = "1.0.0"
    _plugin_category_ = "custom"
    _plugin_description_ = "Plugin skeleton"
    _plugin_dependencies_ = []
    _mimetype_regexp = None

    # ================
    #  interface data
    # ================

    module_cls = Skeleton

    # If needed, overload the `verify` classmethod in order to check your class
    # is instanciable. It should return if everything is alright, otherwise
    # raise an exception. By default it checks that the module's attribute
    # `self.scan_path` is an existing file (cf. `super()._chk_scanpath`)
    #
    # @classmethod
```

(continues on next page)

(continued from previous page)

```
    # def verify(cls):
    #     pass
```

The metaclass `PluginMetaClass` handles the registering of the plugin to a plugin manager. It also checks that the class is instanciable thanks to the `verify` method.

skeleton.py:

```
#
# Copyright (c) 2013-2018 Quarkslab.
# This file is part of IRMA project.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License in the top-level directory
# of this distribution and at:
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# No part of the project, including this file, may be copied,
# modified, propagated, or distributed except according to the
# terms contained in the LICENSE file.

import logging

# Choose the class you need to inherit from
from modules.antivirus.base import AntivirusUnix, AntivirusWindows

log = logging.getLogger(__name__)

# Inhererit from AntivirusUnix or AntivirusWindows according to your plateform
class Skeleton(Antivirus):
    name = "Skeleton for Antivirus"

    # ==================================
    #  Constructor and destructor stuff
    # ==================================

    def __init__(self, *args, **kwargs):
        # class super class constructor
        super().__init__(*args, **kwargs)

        # do your initialization stuff
```

The recipe is the same, the files with the corresponding module name and differents fields need to be updated. The attributes in `Antivirus._attributes` are meant to be defined by the instanciation. One can either:

- leave it blank, in this case the super class will assign it a default value (eg. `"unavailable"` for `self.version`);
- define it directly (eg. `self.scan_path = Path("/opt/skeleton/skeleton")`);
- define a function to be called to assign it (eg. `def get_scan_path(self): ...`), the super class will take care of calling it and handling exceptions.

6.1.2 Testing the new plugin

Before testing, module's necessary stuff (binaries, files, etc) must be provisioned to the VM.

```
$ cd ansible
$ vagrant rsync
$ vagrant ssh
$ sudo su deploy
$ cd /opt/irma/irma-probe/current
$ venv/bin/python -m extras.tools.run_module
```

This last command lists available modules.

Now, if the new module is available, its launching can be done:

```
$ venv/bin/python -m extras.tools.run_module my_module file
```

6.1.3 Automatic provisioning

Creating a new role

Create a new directory with this structure:

```
cd ansible
tree roles/quarsklab.my_module
roles/quarkslab.my_module/
+-- defaults
|   +-- main.yml
+-- tasks
    +-- main.yml
```

tasks/main.yml is the default entry point for a role containing Ansible tasks. In this file, write the instruction to install the module. Add the file tasks/update.yml to write the informations for the update if necessary. In defaults/main.yml it is usual to store default variables for this role. If there are particular instructions, for example how to obtain a licence for a antivirus, add a README file.

Invoking the module role

Modify playbooks/provisioning.yml : add the module

```
-name : my_module
 hosts: my_module
 roles:
 - { role: quarkslab.my_module, tags: 'my_module'}
```

If a task update was defined, add the module in playbooks/updating.yml :

```
-name : my_module
 hosts: my_module
 roles:
 - { role: quarkslab.module, tags: 'my_module', task_from : update}
```

Defining hosts

Modify the environment to add the new probe.

For example for the allinone_dev :

```
$ cat environments/allinone_dev.yml
[ ... snip ... ]
    virustotal:
      - brain.irma
    my_module:
      - brain.irma
    "probe:children":
      - clamav
      - comodo
      - mcafee
      - static-analyzer
      - virustotal
      - my_module
```

CHAPTER 7

Troubleshooting

7.1 Check Celery configuration

7.1.1 Celery Workers

Before going further, you should check that the python applications manages to communicate with the RabbitMQ server through Celery. To ensure that, from the installation directory, execute both Celery workers:

On GNU/Linux:

```
$ cd /opt/irma/irma-brain/current
$ ./venv/bin/python -m brain.scan_tasks


 -------------- celery@brain v3.1.23 (Cipater)
---- **** -----
--- * ***  * -- Linux-3.16.0-4-amd64-x86_64-with-debian-8.2
-- * - **** ---
- ** ---------- [config]
- ** ---------- .> app:         scantasks:0x7fbd7ee4c350
- ** ---------- .> transport:   amqp://brain:**@127.0.0.1:5672/mqbrain
- ** ---------- .> results:     amqp://
- *** --- * --- .> concurrency: 2 (prefork)
-- ******* ----
--- ***** ----- [queues]
 -------------- .> brain            exchange=celery(direct) key=brain


[2016-07-15 15:00:36,155: WARNING/MainProcess] celery@brain ready.
```

This worker is responsible for splitting the whole scan job in multiples job per probe per file.

```
$ cd /opt/irma/irma-brain/current
$ ./venv/bin/python -m brain.results_tasks
```

(continues on next page)

(continued from previous page)

```
 -------------- celery@brain v3.1.23 (Cipater)
---- **** -----
--- * ***  * -- Linux-3.16.0-4-amd64-x86_64-with-debian-8.2
-- * - **** ---
- ** ---------- [config]
- ** ---------- .> app:         resultstasks:0x7fa68f9aa590
- ** ---------- .> transport:   amqp://probe:**@127.0.0.1:5672/mqprobe
- ** ---------- .> results:     disabled://
- *** --- * --- .> concurrency: 2 (prefork)
-- ******* ----
--- ***** ----- [queues]
 -------------- .> results          exchange=celery(direct) key=results

[2016-07-15 14:59:01,799: WARNING/MainProcess] celery@brain ready.
```

And this worker is responsible for collecting and tracking results.

If your Celery worker does not output something similar to the above output, you should check twice the parameters in the application configuration file you are using.

7.2 Verifying RabbitMQ configuration

We can verify that the RabbitMQ server has taken into account our modifications with some commands:

7.2.1 Checking for vhosts

```
$ sudo rabbitmqctl list_vhosts
Listing vhosts ...
mqbrain
/
mqfrontend
mqprobe
mqadmin
...done.
```

If the defined virtual host are not listed by the above command, please execute once more the script.

7.2.2 Checking for users

```
$ sudo rabbitmqctl list_users
Listing users ...
probe   []
brain   []
frontend        []
...done.
```

If the defined users are not listed by the above command, please execute once more the script.

7.2.3 Changing password

If you do not remember the password you just typed, you can change it with `rabbitmqctl` command:

```
$ sudo rabbitmqctl change_password brain brain-rmq-password
Changing password for user "brain" ...
...done.
```

Restarting the service

You may want to restart the service. Thus, the following command can be done:

```
$ sudo invoke-rc.d rabbitmq-server restart
```

7.3 Check SFTP accounts

Try to login as `frontend` and upload a sample file in home dir (should raise an error as it is non writeable) then in `uploads` dir.

```
$ sftp frontend@localhost
frontend@localhost's password:
Connected to localhost.
sftp> put test
Uploading test to /test
remote open("/test"): Permission denied
sftp> ls
uploads
sftp> cd uploads/
sftp> put test
Uploading test to /uploads/test
test
↪100%   10     0.0KB/s   00:00
```

7.4 FTP-TLS accounts

Additionnally, if you have configured IRMA to use FTP-TLS, you can check whether the configured account is valid. On Debian, this can be done with the `ftp-ssl` package:

```
$ sudo apt-get install ftp-ssl
[...]
$ ftp-ssl <hostname of the brain>
Connected to brain.
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
220-You are user number 1 of 50 allowed.
220-Local time is now 18:55. Server port: 21.
220-This is a private system - No anonymous login
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 15 minutes of inactivity.
Name (brain:root): frontend-ftp
500 This security scheme is not implemented
234 AUTH TLS OK.
```

(continues on next page)

(continued from previous page)

```
[SSL Cipher DHE-RSA-AES256-GCM-SHA384]
200 PBSZ=0
200 Data protection level set to "private"
331 User probe OK. Password required
Password: frontend-ftp-password
230 OK. Current directory is /
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

7.5 Restful API

One can verify that the restful API is up and running by querying a specific route on the web server or by checking the system logs:

```
$ curl http://localhost/api/v1.1/probes
{"total": 9, "data": ["ClamAV", "ComodoCAVL", "EsetNod32", "FProt", "Kaspersky",
↪"McAfeeVSCL", "NSRL", "StaticAnalyzer", "VirusTotal"]}

$  sudo cat /var/log/supervisor/frontend_api.log
[...]
added /opt/irma/irma-frontend/current/venv/ to pythonpath.
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI master process (pid: 3943)
spawned uWSGI worker 1 (pid: 3944, cores: 1)
spawned uWSGI worker 2 (pid: 3945, cores: 1)
spawned uWSGI worker 3 (pid: 3946, cores: 1)
spawned uWSGI worker 4 (pid: 3947, cores: 1)
mounting frontend/api/base.py on /api
mounting frontend/api/base.py on /api
mounting frontend/api/base.py on /api
mounting frontend/api/base.py on /api
WSGI app 0 (mountpoint='/api') ready in 0 seconds on interpreter 0x99a3e0 pid: 3945␣
↪(default app)
WSGI app 0 (mountpoint='/api') ready in 0 seconds on interpreter 0x99a3e0 pid: 3946␣
↪(default app)
WSGI app 0 (mountpoint='/api') ready in 0 seconds on interpreter 0x99a3e0 pid: 3944␣
↪(default app)
WSGI app 0 (mountpoint='/api') ready in 0 seconds on interpreter 0x99a3e0 pid: 3947␣
↪(default app)
```

7.6 Logs

7.7 How to debug

7.7.1 Collect debug files

An Ansible playbook is available in order to gather logs and other useful files.

The playbook is `ansible/playbooks/collect_debug.yml` and it will allow you to retrieve on each host:

- IRMA Files (located on the multiples hosts);
- Systemd logs;
- Application logs (Nginx, RabbitMQ, PostgreSQL).

After running the playbook, all the files are available in the directory specified in the `debug_directory` variable of the playbook. The files are store in directories named after the host they where retrieve from (`<debug_directory>/<host_name>/<debug_files_or_directory>`). Most of the files are plain text but Systemd logs are using a binary format. To explore and read them, you'll need the `journalctl` command, for example:

```
$ journalctl -D debug/brain.irma/var/log/journal
```

7.7.2 Switch debug log on

Configuration file for frontend, brain and probe is located by default in the `config` folder and is named respectively `frontend.ini`, `brain.ini` and `probe.ini`.

To turn on debug log just add the following line:

```
[log]
syslog = 0
debug = 1
```

and restart all related applications.

To turn on SQL debug log (warning: its verbose) just add the following line:

```
[log]
syslog = 0
debug = 1
sql_debug = 1
```

and restart all related applications.

7.7.3 Debug a probe

Open a session on the probe machine and change directory to the irma-probe location. Try the run_module tool on a file to see what analyzer is detected and what is its output on a file.

```
$ sudo su deploy
$ cd /opt/irma/irma-probe/current
$ ./venv/bin/python -m extras.tools.run_module

[...]
usage: run_module.py [-h] [-v]
                     {Unarchive,StaticAnalyzer,ClamAV,VirusTotal} filename
                     [filename ...]
run_module.py: error: too few arguments
```

Here 4 probes are automatically detected. Now try one on a file:

```
$ ./venv/bin/python -m extras.tools.run_module ClamAV requirements.txt
{'database': {'/var/lib/clamav/bytecode.cvd': {'ctime': 1458640823.285298,
                                               'mtime': 1458640823.069295,
```

(continues on next page)

(continued from previous page)

```
                                          'sha256':
↪'82972e6cc5f1204829dba913cb1a0b5f8152eb73d3407f6b86cf388626cff1a1'},
         '/var/lib/clamav/daily.cvd': {'ctime': 1458640822.8932924,
                                       'mtime': 1458640822.6692889,
                                       'sha256':
↪'9804c9b9aaf983f85b4f13a7053f98eb7cca5a5a88d3897d49b22182b228885f'},
         '/var/lib/clamav/main.cvd': {'ctime': 1458640821.6972747,
                                      'mtime': 1458640813.9771628,
                                      'sha256':
↪'4a8dfbc4c44704186ad29b5a3f8bdb6674b679cecdf83b156dd1c650129b56f2'}},
 'duration': 0.0045299530029296875,
 'error': None,
 'name': 'Clam AntiVirus Scanner',
 'platform': 'linux2',
 'results': None,
 'status': 0,
 'type': 'antivirus',
 'version': '0.99'}
```

And check the output.

7.7.4 Debug Ansible Provisioning

To debug errors while provisioning (same goes with deployment) with following typical command:

```
$ ansible-playbook  --private-key=~/.vagrant.d/insecure_private_key --inventory-file=.
↪vagrant/provisioners/ansible/inventory/vagrant_ansible_inventory -u vagrant␣
↪playbooks/provisioning.yml
```

Example output:

```
TASK [Mayeu.RabbitMQ : add rabbitmq user and set privileges] ******************
[DEPRECATION WARNING]: Using bare variables is deprecated. Update your playbooks so␣
↪that the environment value uses the
full variable syntax ('{{rabbitmq_users_definitions}}').
This feature will be removed in a future release. Deprecation
warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
failed: [brain.irma] (item={u'vhost': u'mqbrain', u'password': u'brain', u'user': u
↪'brain'}) => {"failed": true, "item": {"password": "brain", "user": "brain", "vhost
↪": "mqbrain"}, "module_stderr": "", "module_stdout": "Traceback (most recent call␣
↪last):\r\n  File \"/tmp/ansible_wKXoO5/ansible_module_rabbitmq_user.py\", line 302,␣
↪in <module>\r\n    main()\r\n  File \"/tmp/ansible_wKXoO5/ansible_module_rabbitmq_
↪user.py\", line 274, in main\r\n    if rabbitmq_user.get():\r\n  File \"/tmp/
↪ansible_wKXoO5/ansible_module_rabbitmq_user.py\", line 155, in get\r\n    users =␣
↪self._exec(['list_users'], True)\r\n  File \"/tmp/ansible_wKXoO5/ansible_module_
↪rabbitmq_user.py\", line 150, in _exec\r\n    rc, out, err = self.module.run_
↪command(cmd + args, check_rc=True)\r\n  File \"/tmp/ansible_wKXoO5/ansible_modlib.
↪zip/ansible/module_utils/basic.py\", line 1993, in run_command\r\n  File \"/usr/lib/
↪python2.7/posixpath.py\", line 261, in expanduser\r\n    if not path.startswith('~
↪'):\r\nAttributeError: 'list' object has no attribute 'startswith'\r\n", "msg":
↪"MODULE FAILURE", "parsed": false}
```

You could first increase ansible verbosity by adding `-vvv` option (`-vvvv` on windows for winrm debug), it will help is the problem is linked to arguments.

```
$ ansible-playbook -vvv --private-key=~/.vagrant.d/insecure_private_key --inventory-
↪file=.vagrant/provisioners/ansible/inventory/vagrant_ansible_inventory -u vagrant␣
↪playbooks/provisioning.yml
TASK [Mayeu.RabbitMQ : add rabbitmq user and set privileges] ******************
task path: /home/alex/repo/irma-ansible/roles/Mayeu.RabbitMQ/tasks/vhost.yml:13
[DEPRECATION WARNING]: Using bare variables is deprecated. Update your playbooks so␣
↪that the environment value uses the full
variable syntax ('{{rabbitmq_users_definitions}}').
This feature will be removed in a future release. Deprecation warnings can be
disabled by setting deprecation_warnings=False in ansible.cfg.
<127.0.0.1> ESTABLISH SSH CONNECTION FOR USER: vagrant
<127.0.0.1> SSH: EXEC ssh -C -q -o ForwardAgent=yes -o Port=2222 -o 'IdentityFile="/
↪home/alex/.vagrant.d/insecure_private_key"' -o KbdInteractiveAuthentication=no -o␣
↪PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey -o␣
↪PasswordAuthentication=no -o User=vagrant -o ConnectTimeout=10 127.0.0.1 '/bin/sh -
↪c '"'"'( umask 77 && mkdir -p "` echo $HOME/.ansible/tmp/ansible-tmp-1468570550.09-
↪211613386938202 `" && echo ansible-tmp-1468570550.09-211613386938202="` echo $HOME/.
↪ansible/tmp/ansible-tmp-1468570550.09-211613386938202 `" ) && sleep 0'"'"''
<127.0.0.1> PUT /tmp/tmpiysJ6l TO /home/vagrant/.ansible/tmp/ansible-tmp-1468570550.
↪09-211613386938202/rabbitmq_user
<127.0.0.1> SSH: EXEC sftp -b - -C -o ForwardAgent=yes -o Port=2222 -o 'IdentityFile=
↪"/home/alex/.vagrant.d/insecure_private_key"' -o KbdInteractiveAuthentication=no -o␣
↪PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey -o␣
↪PasswordAuthentication=no -o User=vagrant -o ConnectTimeout=10 '[127.0.0.1]'
<127.0.0.1> ESTABLISH SSH CONNECTION FOR USER: vagrant
<127.0.0.1> SSH: EXEC ssh -C -q -o ForwardAgent=yes -o Port=2222 -o 'IdentityFile="/
↪home/alex/.vagrant.d/insecure_private_key"' -o KbdInteractiveAuthentication=no -o␣
↪PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey -o␣
↪PasswordAuthentication=no -o User=vagrant -o ConnectTimeout=10 -tt 127.0.0.1 '/bin/
↪sh -c '"'"'sudo -H -S -n -u root /bin/sh -c '"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'echo BECOME-SUCCESS-
↪rbeeckncuxenewcwkayivqiwvarchlrd; LANG=fr_FR.UTF-8 LC_ALL=fr_FR.UTF-8 LC_
↪MESSAGES=fr_FR.UTF-8 /usr/bin/python /home/vagrant/.ansible/tmp/ansible-tmp-
↪1468570550.09-211613386938202/rabbitmq_user; rm -rf "/home/vagrant/.ansible/tmp/
↪ansible-tmp-1468570550.09-211613386938202/" > /dev/null 2>&1'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"' &&␣
↪sleep 0'"'"''
failed: [brain.irma] (item={u'vhost': u'mqbrain', u'password': u'brain', u'user': u
↪'brain'}) => {"failed": true, "invocation": {"module_name": "rabbitmq_user"}, "item
↪": {"password": "brain", "user": "brain", "vhost": "mqbrain"}, "module_stderr": "",
↪"module_stdout": "Traceback (most recent call last):\r\n  File \"/tmp/ansible_
↪Qo3lZl/ansible_module_rabbitmq_user.py\", line 302, in <module>\r\n    main()\r\n␣
↪File \"/tmp/ansible_Qo3lZl/ansible_module_rabbitmq_user.py\", line 274, in main\r\n␣
↪    if rabbitmq_user.get():\r\n  File \"/tmp/ansible_Qo3lZl/ansible_module_rabbitmq_
↪user.py\", line 155, in get\r\n    users = self._exec(['list_users'], True)\r\n␣
↪File \"/tmp/ansible_Qo3lZl/ansible_module_rabbitmq_user.py\", line 150, in _
↪exec\r\n    rc, out, err = self.module.run_command(cmd + args, check_rc=True)\r\n␣
↪File \"/tmp/ansible_Qo3lZl/ansible_modlib.zip/ansible/module_utils/basic.py\", line␣
↪1993, in run_command\r\n  File \"/usr/lib/python2.7/posixpath.py\", line 261, in␣
↪expanduser\r\n    if not path.startswith('~'):\r\nAttributeError: 'list' object has␣
↪no attribute 'startswith'\r\n", "msg": "MODULE FAILURE", "parsed": false}
```

In this particular case, verbose doesn't add much information as the problem is linked to ansible scripts. Let's go one level deeper so. Ansible output the temporary script executed on guest (highlighted in previous code block) but delete it just after execution. To further debug it we will set ansible to keep remote files and the debug session will now takes place inside the guest.

```
$ ANSIBLE_KEEP_REMOTE_FILES=1 ansible-playbook -vvv --private-key=~/.vagrant.d/
↪insecure_private_key --inventory-file=.vagrant/provisioners/ansible/inventory/
↪vagrant_ansible_inventory -u vagrant playbooks/provisioning.yml
```

(continues on next page)

(continued from previous page)

in debug log get the temporary ansible path to remote script:

```
/usr/bin/python /home/vagrant/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275/
↪rabbitmq_user
```

Log in to remote machine and go to the temporary ansible dir. Explode the compressed script and run it locallly:

```
$ vagrant@brain:~/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275$ ls
rabbitmq_user

$ vagrant@brain:~/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275$ python␣
↪rabbitmq_user explode
Module expanded into:
/home/vagrant/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275/debug_dir

$ vagrant@brain:~/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275$ ls debug_
↪dir/
ansible
ansible_module_rabbitmq_user.py
args

$ vagrant@brain:~/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275$ python␣
↪rabbitmq_user execute
Traceback (most recent call last):
  File "/home/vagrant/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275/debug_
↪dir/ansible_module_rabbitmq_user.py", line 302, in <module>
    main()
  File "/home/vagrant/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275/debug_
↪dir/ansible_module_rabbitmq_user.py", line 274, in main
   if rabbitmq_user.get():
  File "/home/vagrant/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275/debug_
↪dir/ansible_module_rabbitmq_user.py", line 155, in get
    users = self._exec(['list_users'], True)
  File "/home/vagrant/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275/debug_
↪dir/ansible_module_rabbitmq_user.py", line 150, in _exec
    rc, out, err = self.module.run_command(cmd + args, check_rc=True)
  File "/home/vagrant/.ansible/tmp/ansible-tmp-1468571039.87-134696488633275/debug_
↪dir/ansible/module_utils/basic.py", line 1993, in run_command
    args = [ os.path.expandvars(os.path.expanduser(x)) for x in args if x is not None␣
↪]
  File "/usr/lib/python2.7/posixpath.py", line 261, in expanduser
    if not path.startswith('~'):
AttributeError: 'list' object has no attribute 'startswith'
```

You could now add debug to source files and properly understand where the problem is. In our example case, it is an ansible problem related to module_rabbitmq_user present in 2.1.0.0 see github PR

CHAPTER 8

References

8.1 Disclaimer

IRMA is distributed as it is, in the hope that it will be useful, but without any warranty neither the implied merchantability or fitness for a particular purpose.

Whatever you do with this tool is uniquely your own responsibility.

8.2 License

IRMA source code is licensed under Apache License, version 2.0.

The full license text can be found below (*Apache License, version 2.0*).

8.3 Apache License, version 2.0

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.
```

(continues on next page)

(continued from previous page)

```
      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.
```

(continues on next page)

(continued from previous page)

```
   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.
```

(continues on next page)

(continued from previous page)

```
   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
      negligent acts) or agreed to in writing, shall any Contributor be
      liable to You for damages, including any direct, indirect, special,
      incidental, or consequential damages of any character arising as a
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS
```

8.4 Authors

IRMA is a project co-funded by the following actors:

- CEA DAM
- DCNS

- GOVCERT.LU (governmental CERT of Luxembourg)
- Airbus Group
- Quarkslab
- Orange Group IS&T

The PRIMARY AUTHORS are (and/or have been):

- Alexandre Quint - Lead Developer, Quarkslab
- David Carle - Quarkslab
- Guillaume Dedrie - Quarkslab
- Fernand Lone-Sang - Quarkslab

And here is an inevitably incomplete list of MUCH-APPRECIATED CONTRIBUTORS – people who have submitted patches, reported bugs, helped answer newbie questions, and generally made IRMA that much better:

- lpecheur
- y0ug
- mdeloitte

CHAPTER 9

Resources

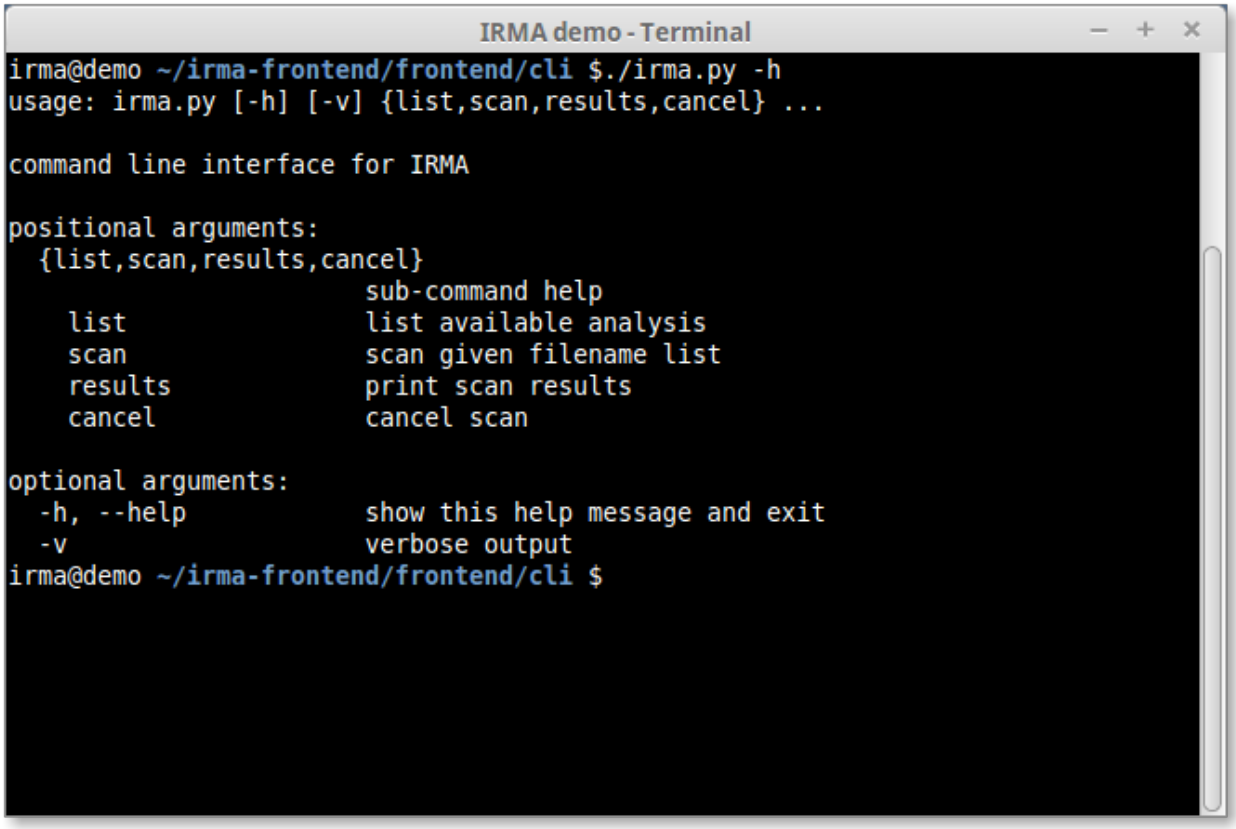
- Project website
- IRC (irc.freenode.net, #qb_irma)
- Twitter (@qb_irma)

CHAPTER 10

Screenshots

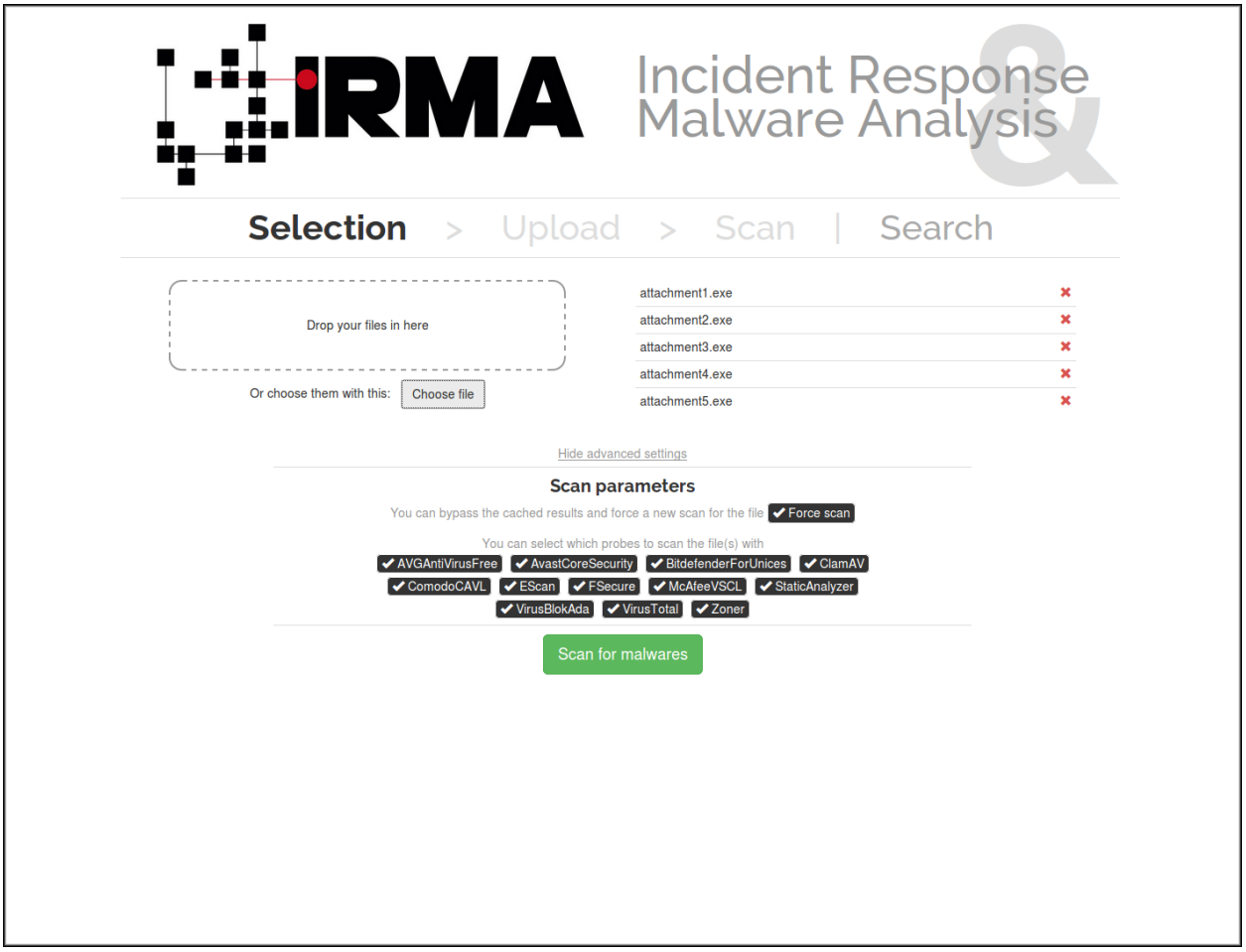
10.1 Command Line Interface

A sample script can be found in frontend repository. Add your own frontend address before testing it.



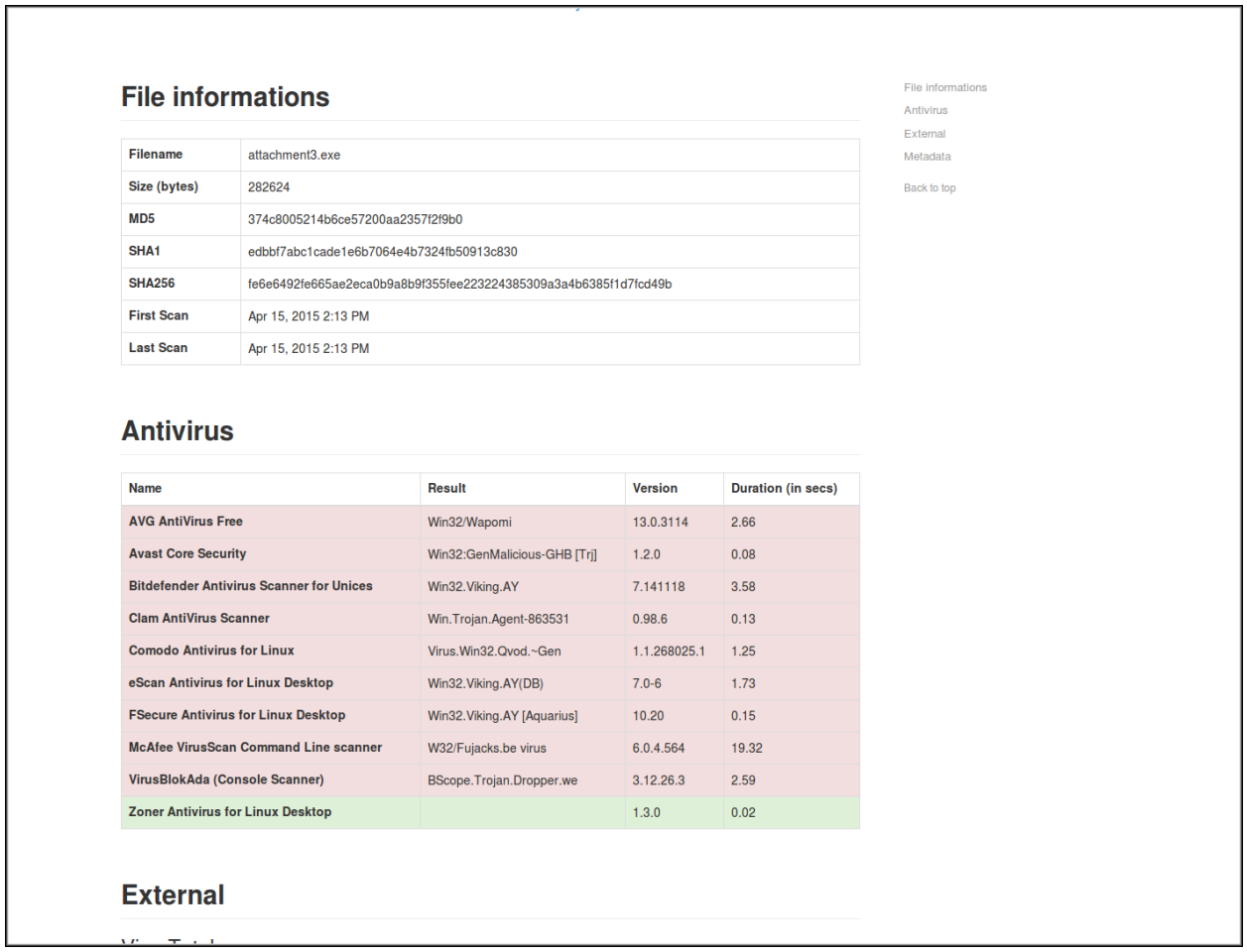
10.2 Web Interface

Some screenshots of the irma user interface shipped with frontend package.




IRMA
Incident Response
Malware Analysis
Selection > Upload > Scan | Search
The files are being uploaded...
Cancel


IRMA
Incident Response
Malware Analysis
Selection > Upload > Scan | Search
92%
Scan status:
Running
Scan Id:
1fc46d12-e940-46ff-9271-a80588dcea56
Tasks:
55 / 60
Cancel
New Scan
attachment1.exe
11 / 12
attachment2.exe
11 / 12
attachment3.exe
11 / 12
attachment4.exe
11 / 12
attachment5.exe
11 / 12



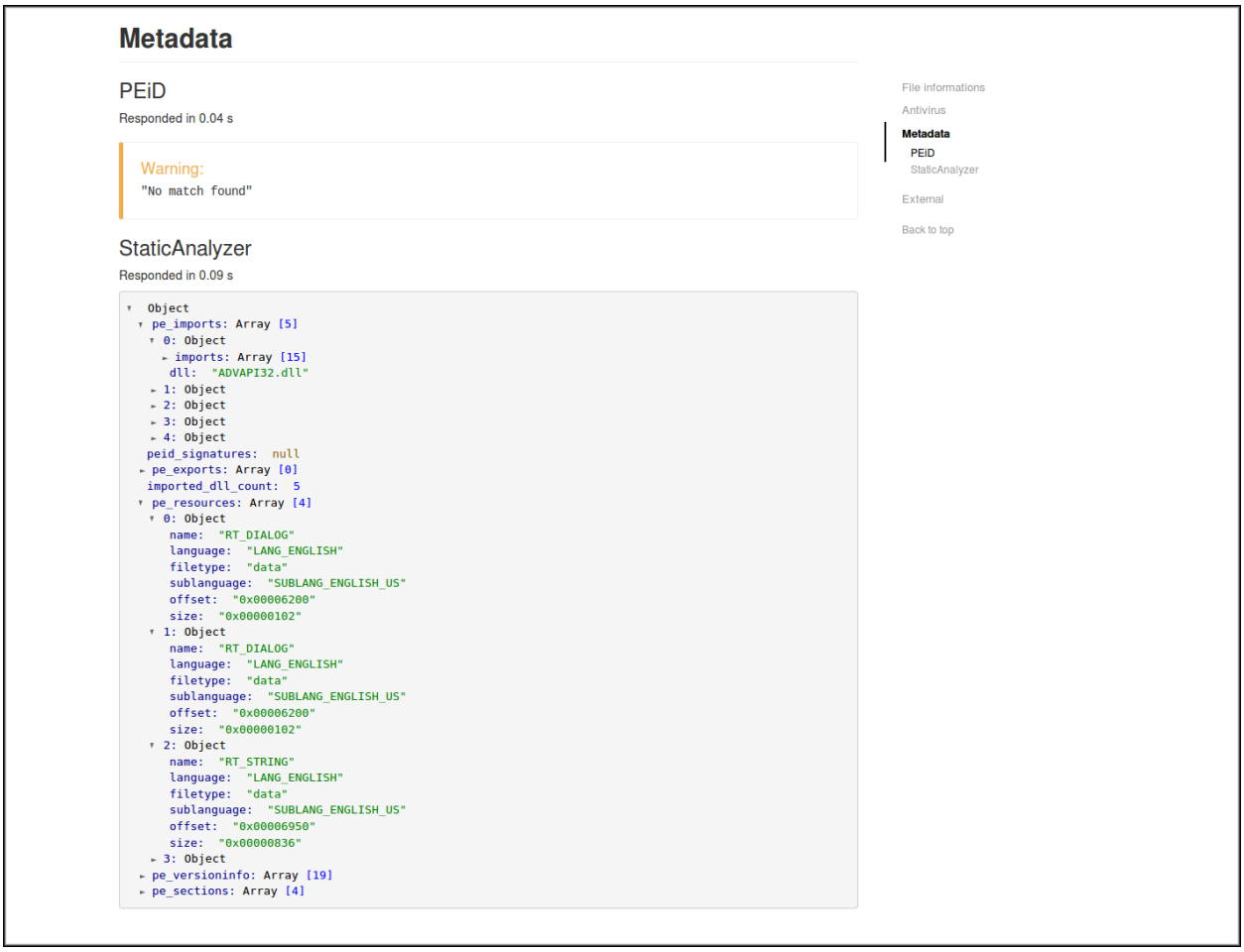
File informations

Filename	attachment3.exe
Size (bytes)	282624
MD5	374c8005214b6ce57200aa2357f2f9b0
SHA1	edbbf7abc1cade1e6b7064e4b7324fb50913c830
SHA256	fe6e6492fe665ae2eca0b9a8b9f355fee223224385309a3a4b6385f1d7fcd49b
First Scan	Apr 15, 2015 2:13 PM
Last Scan	Apr 15, 2015 2:13 PM

Antivirus

Name	Result	Version	Duration (in secs)
AVG AntiVirus Free	Win32/Wapomi	13.0.3114	2.66
Avast Core Security	Win32:GenMalicious-GHB [Trj]	1.2.0	0.08
Bitdefender Antivirus Scanner for Unices	Win32.Viking.AY	7.141118	3.58
Clam AntiVirus Scanner	Win.Trojan.Agent-863531	0.98.6	0.13
Comodo Antivirus for Linux	Virus.Win32.Qvod.~Gen	1.1.268025.1	1.25
eScan Antivirus for Linux Desktop	Win32.Viking.AY(DB)	7.0-6	1.73
FSecure Antivirus for Linux Desktop	Win32.Viking.AY [Aquarius]	10.20	0.15
McAfee VirusScan Command Line scanner	W32/Fujacks.be virus	6.0.4.564	19.32
VirusBlokAda (Console Scanner)	BScope.Trojan.Dropper.we	3.12.26.3	2.59
Zoner Antivirus for Linux Desktop		1.3.0	0.02

External


Metadata
PEiD
Responded in 0.04 s
Warning:
"No match found"
StaticAnalyzer
Responded in 0.09 s
▾ Object
▾ pe_imports: Array [5]
▾ 0: Object
▸ imports: Array [15]
dll: "ADVAPI32.dll"
▸ 1: Object
▸ 2: Object
▸ 3: Object
▸ 4: Object
peid_signatures: null
▸ pe_exports: Array [0]
imported_dll_count: 5
▾ pe_resources: Array [4]
▾ 0: Object
name: "RT_DIALOG"
language: "LANG_ENGLISH"
filetype: "data"
sublanguage: "SUBLANG_ENGLISH_US"
offset: "0x00006200"
size: "0x00000102"
▾ 1: Object
name: "RT_DIALOG"
language: "LANG_ENGLISH"
filetype: "data"
sublanguage: "SUBLANG_ENGLISH_US"
offset: "0x00006200"
size: "0x00000102"
▾ 2: Object
name: "RT_STRING"
language: "LANG_ENGLISH"
filetype: "data"
sublanguage: "SUBLANG_ENGLISH_US"
offset: "0x00006950"
size: "0x00000836"
▸ 3: Object
▸ pe_versioninfo: Array [19]
▸ pe_sections: Array [4]
File informations
Antivirus
Metadata
PEiD
StaticAnalyzer
External
Back to top