IRMA

An Open-Source Incident Response & Malware Analysis Platform

Alexandre Quint **Guillaume Dedrie** Fernand Lone Sang

{aquint, gdedrie, flonesang}@quarkslab.com

Hack.lu
Luxembourg

October 24, 2014

About IRMA

- **IRMA**: **I**ncident **R**esponse & **M**alware **A**nalysis
- Customizable multi-analysis engine platform

Co-Funded Open-Source Project

AIRBUS GROUP
GRAND DUCHY OF LUXEMBOURG
GOVCERT.LU
DCNS
DE LA RECHERCHE À L'INDUSTRIE
cea
QUARKSLAB
INNOVATIVE SECURITY
orange™

Yet Another Malware Analysis Platform ?

virustotal

CAMAL
COSEINC Automated Malware Analysis Lab

AVCaesar

malwr

Metascan® online
SCAN A FILE, HASH OR IP ADDRESS FOR MALWARE

Objectives & Differences

Customizable open-source multi-analysis engine

Objectives & Differences

Customizable open-source **multi-analysis engine**

- Antivirus engines
- ... but not only !

Objectives & Differences

Customizable **open-source** multi-analysis engine

- Antivirus engines
- ... but not only !

- You can install it on **your network**
- You can **modify it** to be like in your dreams

Objectives & Differences

Customizable open-source multi-analysis engine

- Antivirus engines
- ... but not only !

- You can install it on **your network**
- You can **modify it** to be like in your dreams

- You can add your **own analysis engines**
- You can gather and get **only** information **relevant to you**
- You can display them **the way you want it to be displayed**

Objectives & Differences

Customizable open-source multi-analysis engine

When installed on **your network**

- your (confidential) files stays in your network
- you **keep control** over submitted files

History of the Project – Started in 2013, November

Initial release – v1.0.0 – 2014, June

- support for python-friendly packages
- fully NoSQL database

History of the Project – Started in 2013, November

Initial release – v1.0.0 – 2014, June

- support for python-friendly packages
- fully NoSQL database

Previous version – v1.0.4 – 2014, August

- support for Debian packages
- added analysis result formatters on frontend

History of the Project – Started in 2013, November

Initial release – v1.0.0 – 2014, June

- support for python-friendly packages
- fully NoSQL database

Previous version – v1.0.4 – 2014, August

- support for Debian packages
- added analysis result formatters on frontend

Current version – v1.1.0 – 2014, October

- migration to hybrid SQL and NoSQL database
- removal of Redis-server (backend for asynchronous job)
- drop of Debian packages for automation scripts

Available Analysis Engines

8 anti-viruses analyzers

- Clam Antivirus
- Comodo CAVL
- Eset Nod-32
- McAfee VirusScan
- FProt
- Sophos
- Kaspersky
- Symantec

1 file hash database analyzer

- NIST's National Software Reference Library (NSRL)

1 executable file analyzer

- PE Static File Analyzer (borrowed from Cuckoo Sandbox)

1 external analyzer

- VirusTotal Report search from a hash

Useful Links

- Homepage: `http://irma.quarkslab.com`
- Code Repositories:
  - `https://github.com/quarkslab/irma-frontend`
  - `https://github.com/quarkslab/irma-brain`
  - `https://github.com/quarkslab/irma-probe`
  - `https://github.com/quarkslab/irma-ansible`
- Twitter: `@qb_irma`
- `IRC: #qb_irma@freenode`

Useful Links

- Homepage: `http://irma.quarkslab.com`
- Code Repositories:
  - `https://github.com/quarkslab/irma-frontend`
  - `https://github.com/quarkslab/irma-brain`
  - `https://github.com/quarkslab/irma-probe`
  - `https://github.com/quarkslab/irma-ansible`
- Twitter: `@qb_irma`
- IRC: `#qb_irma@freenode`

Got Interested ? Download and follow these slides:

`http://irma.quarkslab.com/hacklu/irma-slides.pdf`
`http://irma.quarkslab.com/hacklu/irma-cheatsheet.pdf`

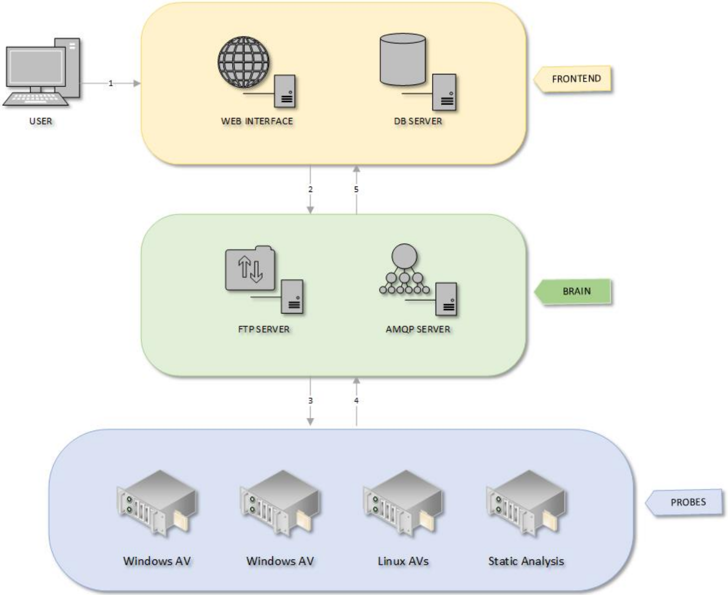
Outline

1. IRMA Installation
2. IRMA Internals
3. Activating Analyzers
4. Developing Analyzers
5. Customizing the API output
6. Conclusion & Future Work

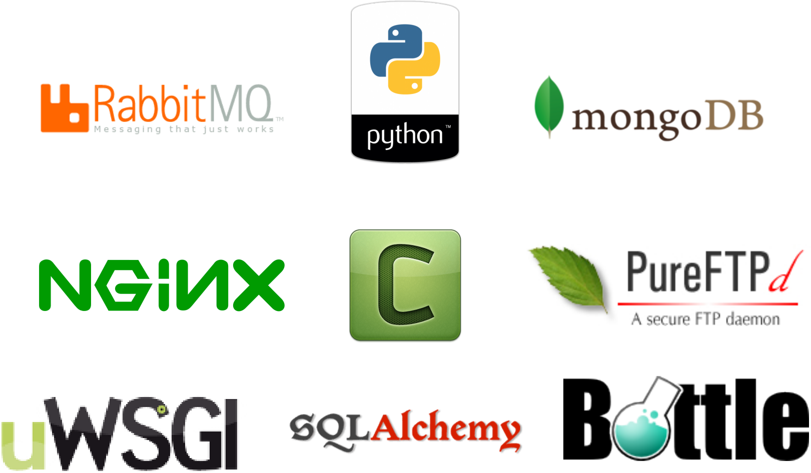
Outline

1. IRMA Installation
2. IRMA Internals
3. Activating Analyzers
4. Developing Analyzers
5. Customizing the API output
6. Conclusion & Future Work

IRMA as Three Parts System



Technologies Underneath



IRMA and Automation tools: a love story (1)

- easy installation of a complete IRMA appliance
- handle all dependencies
- configure all settings for frontend/brain/probes at once
- windows probes currently not supported

IRMA and Automation tools: a love story $^{(2)}$

son, can you help me to install a program?
...
Yes mom...
What your mom see:
WHAT REALLY HAPPENS:
NEXT, NEXT, NEXT, ACCEPT, INSTALL... FINISH

IRMA Quick Install (1)

Requirements

VirtualBox

download: `https://www.virtualbox.org/wiki/Downloads`

Vagrant (>= 1.5)

version: `vagrant --version`

download: `https://www.vagrantup.com/downloads.html`

Ansible (>= 1.6)

version: `ansible --version`

installation: `pip install ansible`

IRMA Quick Install (2)

Clone repositories

```
$ git clone --recursive --branch hacklu \
      https://github.com/quarkslab/irma-ansible.git
```

Fetch provisioning dependencies

```
$ ansible-galaxy install -r galaxy.yml -p roles
```

Create virtual machines and provision them

```
$ vagrant --no-provision
$ vagrant provision
# Or, simply ...
$ vagrant up
```

IRMA Quick Install (3)

Vagrant & Ansible magic results to 2 virtual machines:

allinone contains the brain, the frontend and several analyzers

IP address: 172.16.1.30

Analyzers: 3 anti-viruses: ClamAV, Comodo, McAfee

myprobe contains analyzers, but none of them is activated

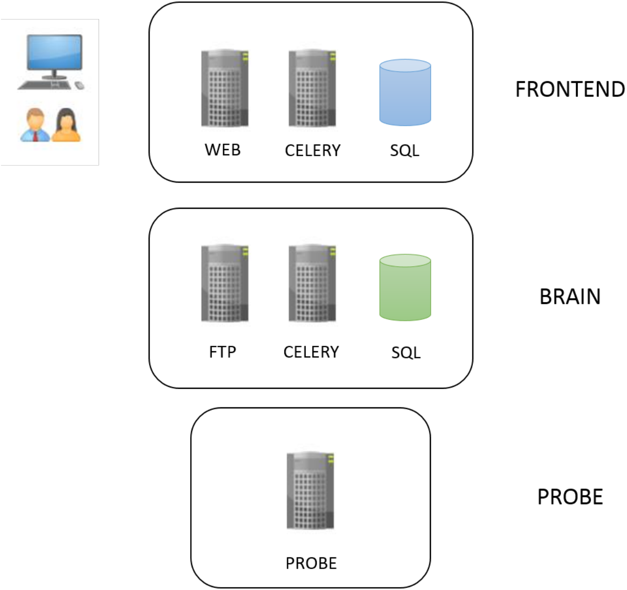
IP address: 172.16.1.42

Analyzers: a skeleton to develop new analyzers

Outline

1. IRMA Installation
2. IRMA Internals
3. Activating Analyzers
4. Developing Analyzers
5. Customizing the API output
6. Conclusion & Future Work

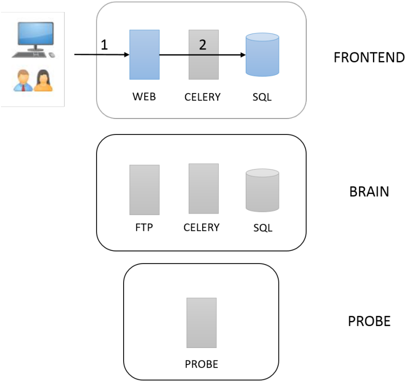
Scan workflow



File submission

First a user upload one or more files to the Web client:

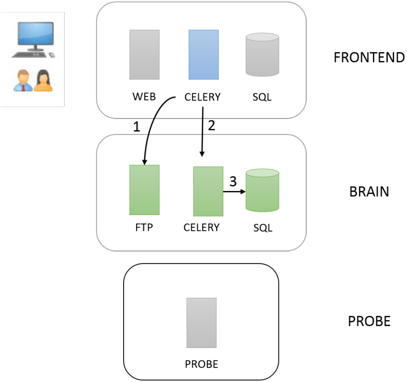
1. files are posted to the bottle API
2. scan record is created in the database file is stored on disk



Scan launch

Then the scan is launched:

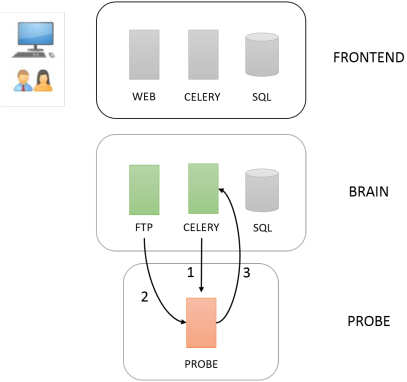
1. files are uploaded on Brain's FTP asynchronously
2. a meta job is launched on Brain celery
3. a scan record is created in Brain SQL



Probe dispatching

Each probe has its own celery queue

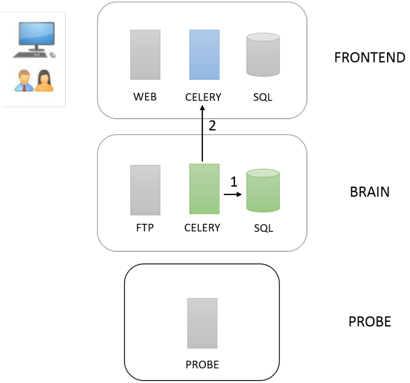
1. One sub job per probe per file is created
2. receives a FTP link to download the file, download and process it
3. returns a JSON formatted message to the brain



Brain collects results

Each probe results are tracked for scan progress

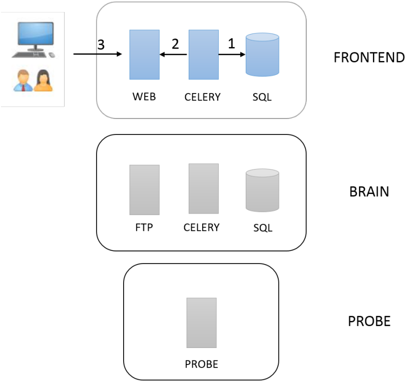
1. finished jobs are tracked with brain sql database
2. results are forwarded to frontend



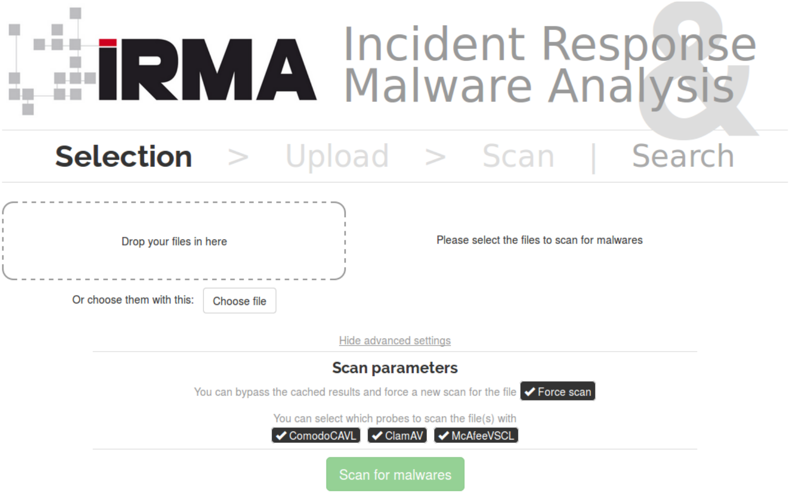
Frontend receives results

Each probe results forwarded by brain is stored in database

1. probe results are stored in NoSQL and linked in SQL
2. probe raw results are kept
3. formatters could filter raw results for cleaner response



Screenshots (1)



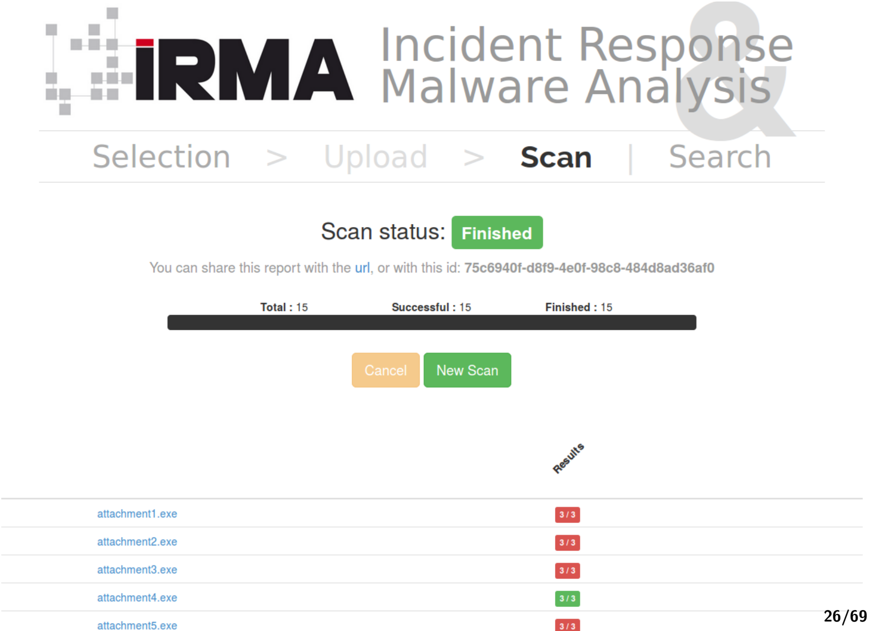
Please select the files to scan for malwares

Scan parameters

You can bypass the cached results and force a new scan for the file ✓ Force scan

You can select which probes to scan the file(s) with ✓ ComodoCAVL ✓ ClamAV ✓ McAfeeVSCL

Screenshots (2)



IRMA
Incident Response & Malware Analysis
Selection > Upload > Scan | Search

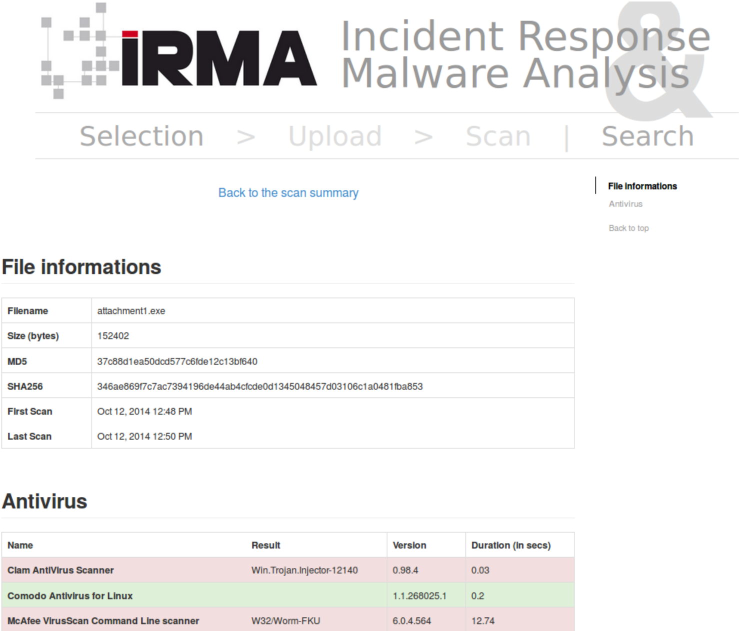
Scan status: Finished

You can share this report with the url, or with this id: 75c6940f-d8f9-4e0f-98c8-484d8ad36af0

Total : 15 **Successful :** 15 **Finished :** 15

	Results
attachment1.exe	3 / 3
attachment2.exe	3 / 3
attachment3.exe	3 / 3
attachment4.exe	3 / 3
attachment5.exe	3 / 3

Screenshots (3)


iRMA
Incident Response & Malware Analysis
Selection > Upload > Scan | Search
File informations
Antivirus
Back to top

Back to the scan summary

File informations

Filename	attachment1.exe
Size (bytes)	152402
MD5	37c88d1ea50dcd577c6fde12c13bf640
SHA256	346ae869f7c7ac7394196de44ab4cfcde0d1345048457d03106c1a0481fba853
First Scan	Oct 12, 2014 12:48 PM
Last Scan	Oct 12, 2014 12:50 PM

Antivirus

Name	Result	Version	Duration (in secs)
Clam AntiVirus Scanner	Win.Trojan.Injector-12140	0.98.4	0.03
Comodo Antivirus for Linux		1.1.268025.1	0.2
McAfee VirusScan Command Line scanner	W32/Worm-FKU	6.0.4.564	12.74

Outline

1. IRMA Installation
2. IRMA Internals
3. Activating Analyzers
4. Developing Analyzers
5. Customizing the API output
6. Conclusion & Future Work

Activating more analyzers (1)

son, can you help me to install a program?
...
Yes mom...
What your mom see:
WHAT REALLY HAPPENS:
NEXT, NEXT, NEXT, ACCEPT, INSTALL... FINISH

Activating more analyzers (2)

Connect to “allinone” Virtual Machine

```
$ vagrant ssh allinone
```

Login as irma and go to irma-probe code

```
$ sudo su irma
$ cd /opt/irma/irma-probe/current/
```

Activating more analyzers (3)

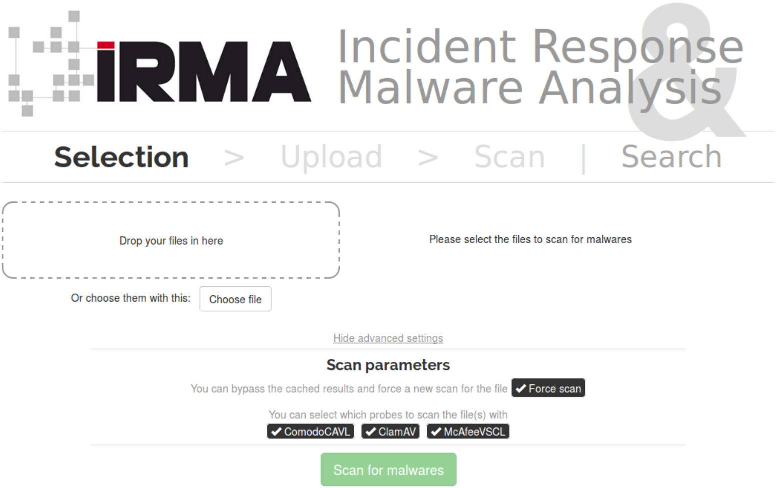
Install Module Dependencies

```
$ venv/bin/pip install -r \
    modules/metadata/pe_analyzer/requirements.txt
$ venv/bin/pip install -r \
    modules/external/virustotal/requirements.txt
```

Restart the Celery worker

```
$ sudo supervisorctl restart probe_app
```

Activating more analyzers (4)



IRMA Incident Response & Malware Analysis

Selection > Upload > Scan | Search

Drop your files in here

Please select the files to scan for malwares

Or choose them with this: Choose file

Hide advanced settings

Scan parameters

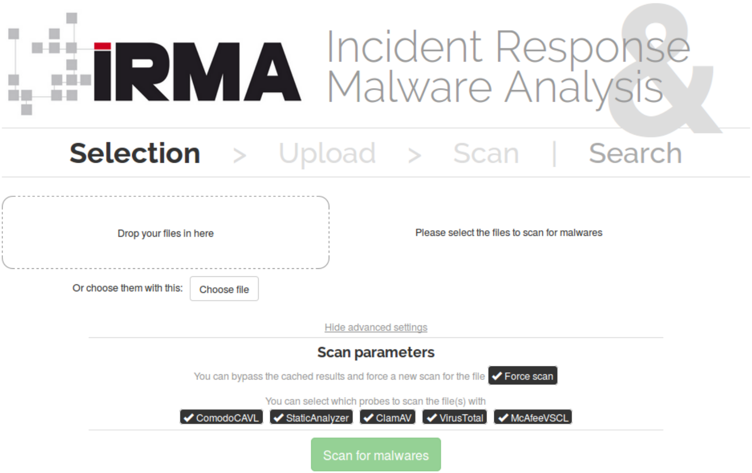
You can bypass the cached results and force a new scan for the file ✓ Force scan

You can select which probes to scan the file(s) with

✓ ComodoCAVL ✓ ClamAV ✓ McAfeeVSCL

Scan for malwares

Activating more analyzers (5)



Please select the files to scan for malwares

Or choose them with this:

Scan parameters

You can bypass the cached results and force a new scan for the file ✓ Force scan

You can select which probes to scan the file(s) with

✓ ComodoCAVL ✓ StaticAnalyzer ✓ ClamAV ✓ VirusTotal ✓ McAfeeVSCL

Magic Underneath

Analyzers are self-discovered

```
$ tree -L 1 /opt/irma/irma-probe/current
/opt/irma/irma-probe/current
|-- config
|-- docs
|-- lib
|-- modules
|-- probe
+-- tools
```

Magic Underneath

Analyzers are self-discovered

```
$ tree -L 1 /opt/irma/irma-probe/current
/opt/irma/irma-probe/current
|-- config
|-- docs
|-- lib
|-- modules
|-- probe
+-- tools
```

Analyzers are found by scanning the `modules` directory at startup, which contains analyzer plugins.

Magic Underneath

Analyzers are self-discovered

```
$ tree -L 1 /opt/irma/irma-probe/current
/opt/irma/irma-probe/current
|-- config
|-- docs
|-- lib
|-- modules
|-- probe
+-- tools
```

Analyzers are found by scanning the `modules` directory at startup, which contains analyzer plugins.

Analyzers are self-registered

1. Check for plugin dependencies
2. when registration is successful, analyzer is activated
3. a celery queue for each registered plugin is created

IRMA Modules

Example of Module

```
$ cd /opt/irma/irma-probe/current/
$ tree -L 1 modules/metadata/pe_analyzer
modules/metadata/pe_analyzer
|-- __init__.py  ←———  mandatory to have a python module
|-- pe.py
|-- plugin.py
+-- requirements.txt
```

IRMA Modules

Example of Module

```
$ cd /opt/irma/irma-probe/current/
$ tree -L 1 modules/metadata/pe_analyzer
modules/metadata/pe_analyzer
|-- __init__.py  ←——————  mandatory to have a python module
|-- pe.py  ←———————————  standalone custom analyzer module
|-- plugin.py
+-- requirements.txt
```

IRMA Modules

Example of Module

```
$ cd /opt/irma/irma-probe/current/
$ tree -L 1 modules/metadata/pe_analyzer
modules/metadata/pe_analyzer
|-- __init__.py  ←———   mandatory to have a python module
|-- pe.py  ←————————    standalone custom analyzer module
|-- plugin.py  ←————    glues the analyzer with the celery worker
+-- requirements.txt
```

IRMA Modules

Example of Module

```
$ cd /opt/irma/irma-probe/current/
$ tree -L 1 modules/metadata/pe_analyzer
modules/metadata/pe_analyzer
|-- __init__.py ←——— mandatory to have a python module
|-- pe.py ←————————— standalone custom analyzer module
|-- plugin.py ←————— glues the analyzer with the celery worker
+-- requirements.txt ←—— python dependencies to be installed
```

IRMA Modules

Example of Module

```
$ cd /opt/irma/irma-probe/current/
$ tree -L 1 modules/metadata/pe_analyzer
modules/metadata/pe_analyzer
|-- __init__.py  ←——  mandatory to have a python module
|-- pe.py  ←——————  standalone custom analyzer module
|-- plugin.py  ←————  glues the analyzer with the celery worker
+-- requirements.txt ←— python dependencies to be installed
```

Why is it recommended to adopt this structure ?

- Allows to modularize code
- Allows existing code-reuse and integration

Example of Plugin (1)

```
$ cat modules/custom/skeleton/plugin.py
[... python imports ...]

class SkeletonPlugin(PluginBase):

    #  plugin metadata
    _plugin_name_ = "Skeleton"
    _plugin_author_ = "<author name>"
    _plugin_version_ = "<version>"
    _plugin_category_ = "custom"
    _plugin_description_ = "Plugin skeleton"
    _plugin_dependencies_ = []

    @classmethod
    def verify(cls):
        raise PluginLoadError("Skeleton plugin is not
            meant to be loaded")
```

Example of Plugin (2)

```
    # probe interfaces
    def run(self, paths):
        response = PluginResult(
                          name=type(self).plugin_name,
                          type=type(self).plugin_category,
                          version=None
                       )
        try:
            started = timestamp(datetime.utcnow())
            response.results = "results here"
            stopped = timestamp(datetime.utcnow())
            response.duration = stopped - started
            response.status = 0
        # Handle analysis errors
        except Exception as e:
            response.status = -1
            response.results = str(e)
        return response
```

IRMA Plugin 101 (1)

`PluginBase` Class

- Used for analyzers discovery
- Handles self-registration

IRMA Plugin 101 (1)

PluginBase Class

- Used for analyzers discovery
- Handles self-registration

Declare dependencies

- Use one of the helpers:
  BinaryDependency, PlatformDependency, FileDependency, FolderDependency, ModuleDependency
- or, define a verify() classmethod:

```
@classmethod
def verify(cls):
    raise PluginLoadError("Describe the error")
```

IRMA Plugin 101 (2)

`ProbeResult` class

name the name of the probe

type the category of the probe

version the version of the probe

platform the platform on which the probe is executed

duration duration in seconds

status return code (< 0 is error, > 0 is context specific)

error `None` if no error else the error string

results Probe results

Outline

1 IRMA Installation

2 IRMA Internals

3 Activating Analyzers

4 Developing Analyzers

5 Customizing the API output

6 Conclusion & Future Work

Workflow to Develop Custom Analyzers (1)

1. Create a standalone module
2. Test our standalone module
3. Create a plugin to wrap our module
4. Test the plugin for our module
5. Integrate it to IRMA

TrID Analyzer (1)

TrID is an utility designed to identify file types from their binary signatures. While there are similar utilities with hard coded logic, TrID has no fixed rules. Instead, it's extensible and can be trained to recognize new formats in a fast and automatic way.

Download and Install on Debian

```
$ sudo dpkg --add-architecture i386
$ sudo apt-get update
$ sudo apt-get install libc6-i386 libncurses5:i386
$ mkdir modules/metadata/trid/
$ cd modules/metadata/trid/
$ curl http://mark0.net/download/trid_linux.zip -O
$ unzip trid_linux.zip
$ rm *.txt
$ chmod a+x ./trid
```

TrID Analyzer $^{(2)}$

Updating Definitions

```
$ curl http://mark0.net/download/tridupdate.zip -O
$ unzip tridupdate.zip
$ python tridupdate.py
```

TrID Analyzer $^{(2)}$

Updating Definitions

```
$ curl http://mark0.net/download/tridupdate.zip -O
$ unzip tridupdate.zip
$ python tridupdate.py
```

```
$ ./trid /bin/bash

TrID/32 - File Identifier v2.11 - (C) 2003-11 By M.Pontello
Definitions found:  5391
Analyzing...

 49.7% (.) ELF Executable and Linkable format (Linux)
     (4025/14)
 49.4% (.O) ELF Executable and Linkable format (generic)
     (4000/1)
  0.7% (.CEL) Lumena CEL bitmap (63/63)
```

Writing a module for TrID (1)

```
# python imports





class TrID(object):

    # Helper run_cmd(cmd, args)
    # :cmd: str, command to be executed
    # :args: list, arguments for the command
    # :return: tuple (retcode, stdout, stderr)
    @staticmethod
    def run_cmd(cmd, *args):
```

Writing a module for TrID (1)

```
# python imports
import re, os, sys
from subprocess import Popen, PIPE
from os.path import dirname, abspath, join, exists


class TrID(object):

    # Helper run_cmd(cmd, args)
    # :cmd: str, command to be executed
    # :args: list, arguments for the command
    # :return: tuple (retcode, stdout, stderr)
    @staticmethod
    def run_cmd(cmd, *args):
```

Writing a module for TrID (1)

```
# python imports
import re, os, sys
from subprocess import Popen, PIPE
from os.path import dirname, abspath, join, exists


class TrID(object):

    # Helper run_cmd(cmd, args)
    # :cmd: str, command to be executed
    # :args: list, arguments for the command
    # :return: tuple (retcode, stdout, stderr)
    @staticmethod
    def run_cmd(cmd, *args):
        cmdarray = [cmd] + args if isinstance(args, list) else
            [args]
```

Writing a module for TrID (1)

```
# python imports
import re, os, sys
from subprocess import Popen, PIPE
from os.path import dirname, abspath, join, exists


class TrID(object):

    # Helper run_cmd(cmd, args)
    # :cmd: str, command to be executed
    # :args: list, arguments for the command
    # :return: tuple (retcode, stdout, stderr)
    @staticmethod
    def run_cmd(cmd, *args):
        cmdarray = [cmd] + args if isinstance(args, list) else
             [args]
        pd = Popen(cmdarray, stdout=PIPE, stderr=PIPE)
```

Writing a module for TrID $^{(1)}$

```
# python imports
import re, os, sys
from subprocess import Popen, PIPE
from os.path import dirname, abspath, join, exists


class TrID(object):

    # Helper run_cmd(cmd, args)
    # :cmd: str, command to be executed
    # :args: list, arguments for the command
    # :return: tuple (retcode, stdout, stderr)
    @staticmethod
    def run_cmd(cmd, *args):
        cmdarray = [cmd] + args if isinstance(args, list) else
             [args]
        pd = Popen(cmdarray, stdout=PIPE, stderr=PIPE)
        stdout, stderr = map(lambda x:  x.strip()
            if x.strip() else None, pd.communicate())
```

Writing a module for TrID $^{(1)}$

```
# python imports
import re, os, sys
from subprocess import Popen, PIPE
from os.path import dirname, abspath, join, exists


class TrID(object):

   # Helper run_cmd(cmd, args)
   # :cmd: str, command to be executed
   # :args: list, arguments for the command
   # :return: tuple (retcode, stdout, stderr)
   @staticmethod
   def run_cmd(cmd, *args):
       cmdarray = [cmd] + args if isinstance(args, list) else
           [args]
       pd = Popen(cmdarray, stdout=PIPE, stderr=PIPE)
       stdout, stderr = map(lambda x:  x.strip()
          if x.strip() else None, pd.communicate())
       return (pd.returncode, stdout, stderr)
```

Writing a module for TrID $^{(2)}$

```
# Helper get_trid_path() to locate trid binary
@staticmethod
def get_trid_path():




# Helper to parse TrID output results
def check_analysis_results(self, paths, results):
    [...]

# Module entry point, return trid results
def analyze(self, paths):

```

Writing a module for TrID $^{(2)}$

```
# Helper get_trid_path() to locate trid binary
@staticmethod
def get_trid_path():
    current_dir = dirname(abspath(__file__))



# Helper to parse TrID output results
def check_analysis_results(self, paths, results):
    [...]

# Module entry point, return trid results
def analyze(self, paths):
```

Writing a module for TrID (2)

```
# Helper get_trid_path() to locate trid binary
@staticmethod
def get_trid_path():
    current_dir = dirname(abspath(__file__))
    trid = join(current_dir, ’trid’)


# Helper to parse TrID output results
def check_analysis_results(self, paths, results):
    [...]

# Module entry point, return trid results
def analyze(self, paths):
```

Writing a module for TrID (2)

```
# Helper get_trid_path() to locate trid binary
@staticmethod
def get_trid_path():
    current_dir = dirname(abspath(__file__))
    trid = join(current_dir, 'trid')
    return trid if exists(trid) else None

# Helper to parse TrID output results
def check_analysis_results(self, paths, results):
    [...]

# Module entry point, return trid results
def analyze(self, paths):
```

Writing a module for TrID (2)

```
# Helper get_trid_path() to locate trid binary
@staticmethod
def get_trid_path():
    current_dir = dirname(abspath(__file__))
    trid = join(current_dir, ’trid’)
    return trid if exists(trid) else None

# Helper to parse TrID output results
def check_analysis_results(self, paths, results):
    [...]

# Module entry point, return trid results
def analyze(self, paths):
    results = self.run_cmd(self.get_trid_path(), paths)
```

Writing a module for TrID (2)

```
# Helper get_trid_path() to locate trid binary
@staticmethod
def get_trid_path():
    current_dir = dirname(abspath(__file__))
    trid = join(current_dir, ’trid’)
    return trid if exists(trid) else None

# Helper to parse TrID output results
def check_analysis_results(self, paths, results):
    [...]

# Module entry point, return trid results
def analyze(self, paths):
    results = self.run_cmd(self.get_trid_path(), paths)
    return self.check_analysis_results(paths, results)
```

Writing a module for TrID (3)

```
# Helper to parse trid output results
def check_analysis_results(self, paths, results):

    # check stdout


        # iterate through lines

            # find info with pattern matching






            # create entries to be appended to results


        # uniformize retcode
```

Writing a module for TrID (3)

```
# Helper to parse trid output results
def check_analysis_results(self, paths, results):
    retcode, stdout, stderr = results
    # check stdout


      # iterate through lines

          # find info with pattern matching





          # create entries to be appended to results


      # uniformize retcode
```

Writing a module for TrID $^{(3)}$

```
# Helper to parse trid output results
def check_analysis_results(self, paths, results):
    retcode, stdout, stderr = results
    # check stdout
    if stdout:

        # iterate through lines

            # find info with pattern matching




            # create entries to be appended to results


        # uniformize retcode
```

Writing a module for TrID $^{(3)}$

```
# Helper to parse trid output results
def check_analysis_results(self, paths, results):
    retcode, stdout, stderr = results
    # check stdout
    if stdout:
        results = []
        # iterate through lines

            # find info with pattern matching





            # create entries to be appended to results


        # uniformize retcode
```

Writing a module for TrID $^{(3)}$

```
# Helper to parse trid output results
def check_analysis_results(self, paths, results):
    retcode, stdout, stderr = results
    # check stdout
    if stdout:
      results = []
      # iterate through lines
      for line in stdout.splitlines()[4:]:
          # find info with pattern matching





          # create entries to be appended to results


      # uniformize retcode


```

Writing a module for TrID (3)

```
# Helper to parse trid output results
def check_analysis_results(self, paths, results):
   retcode, stdout, stderr = results
   # check stdout
   if stdout:
     results = []
     # iterate through lines
     for line in stdout.splitlines()[4:]:
        # find info with pattern matching
        match = re.match(
           r’\s*(?P<ratio>\d*[.]\d*)[%]\s+’
           r’[(](?P<ext>[.]\w*)[)]\s+’ # extension
           r’(?P<desc>.*)$’, # remaining line
        line)
        # create entries to be appended to results


     # uniformize retcode
```

Writing a module for TrID (3)

```
# Helper to parse trid output results
def check_analysis_results(self, paths, results):
    retcode, stdout, stderr = results
    # check stdout
    if stdout:
      results = []
      # iterate through lines
      for line in stdout.splitlines()[4:]:
          # find info with pattern matching
          match = re.match(
              r’\s*(?P<ratio>\d*[.]\d*)[%]\s+’
              r’[(](?P<ext>[.]\w*)[)]\s+’ # extension
              r’(?P<desc>.*)$’, # remaining line
          line)
          # create entries to be appended to results
          if match:
              results.append(match.groupdict())
      # uniformize retcode
```

Writing a module for TrID $^{(3)}$

```
# Helper to parse trid output results
def check_analysis_results(self, paths, results):
    retcode, stdout, stderr = results
    # check stdout
    if stdout:
      results = []
      # iterate through lines
      for line in stdout.splitlines()[4:]:
          # find info with pattern matching
          match = re.match(
              r’\s*(?P<ratio>\d*[.]\d*)[%]\s+’
              r’[(](?P<ext>[.]\w*)[)]\s+’ # extension
              r’(?P<desc>.*)$’, # remaining line
          line)
          # create entries to be appended to results
          if match:
              results.append(match.groupdict())
      # uniformize retcode
      retcode = 0 if results else 1
      if not results:  results = None
      return retcode, results
```

Testing our wrapper for TrID

```
$ venv/bin/pip install ipython
$ venv/bin/ipython
Python 2.7.3 (default, Mar 13 2014, 11:03:55)
Type "copyright", "credits" or "license" for more
    information.

IPython 0.13.1 -- An enhanced Interactive Python.
?          -> Introduction and overview [...]
%quickref  -> Quick reference.
help       -> Python’s own help system.
object?    -> Details about ’object’, use [...]

In [1]: from trid import *;
In [2]: module = TrID();
In [3]: module.analyze(’/bin/bash’)
[...]
```

Writting the TrID plugin for IRMA (1)

```
# python imports
import re, os, sys, logging
from os.path import dirname, abspath, join
from datetime import datetime
from lib.common.utils import timestamp
from lib.plugins import PluginBase, FileDependency
from lib.plugin_result import PluginResult


class TrIDPlugin(PluginBase):

    # metadata
    _plugin_name_ = "TrID"
    _plugin_author_ = "IRMA (c) Quarkslab"
    _plugin_version_ = "1.0.0"
    _plugin_category_ = "metadata"
    _plugin_description_ = "Plugin for file type"
```

Writting the TrID plugin for IRMA (2)

```
# dependencies
_plugin_dependencies_ = [
    # trid binary
    FileDependency(
        join(dirname(abspath(__file__)), 'trid'),
        help='Make sure you have downloaded trid
            binary'
    ),
    # trid definitions
    FileDependency(
        join(dirname(abspath(__file__)), 'triddefs
            .trd'),
        help='Make sure to have downloaded trid
            definitions'
    ),
]
```

Writting the TrID plugin for IRMA $^{(3)}$

```
    def __init__(self):
        module = sys.modules[’modules.metadata.trid.trid’].
            TrID
        self.module = module()

    def run(self, paths):
        results = PluginResult(name=type(self).plugin_name,
                               type=type(self).
                                   plugin_category,
                               version=None)
        # launch file analysis
        try:
            started = timestamp(datetime.utcnow())
            results.status, results.results = self.module.
                analyze(paths)
            stopped = timestamp(datetime.utcnow())
            results.duration = stopped - started
        # handle exceptions
        except Exception as e:
            results.status = -1
            results.error = str(e)
        return results
```

Testing TrID plugin

```
$ venv/bin/python -m tools.run_modules
usage: run_module.py [-h] [-v] {TrID,ClamAV} filename [
    filename ...]
run_module.py: error: too few arguments
```

Testing TrID plugin

```
$ venv/bin/python -m tools.run_modules
usage: run_module.py [-h] [-v] {TrID,ClamAV} filename [
    filename ...]
run_module.py: error: too few arguments
```

```
$ venv/bin/python -m tools.run_modules TrID /bin/bash
{'duration': 0.2646839618682861З,
 'error': None,
 'name': 'TrID',
 'platform': 'linux2',
 'results': [{'desc': 'ELF Executable and Linkable format (
     Linux) (4025/14)',
              'ext': '.',
              'ratio': '49.7'},
             {'desc': 'ELF Executable and Linkable format (
                 generic) (4000/1)',
              'ext': '.O',
              'ratio': '49.4'},
             {'desc': 'Lumena CEL bitmap (63/63)',
              'ext': '.CEL',
              'ratio': '0.7'}],
 'status': 0,
 'type': 'metadata',
 'version': None}
```

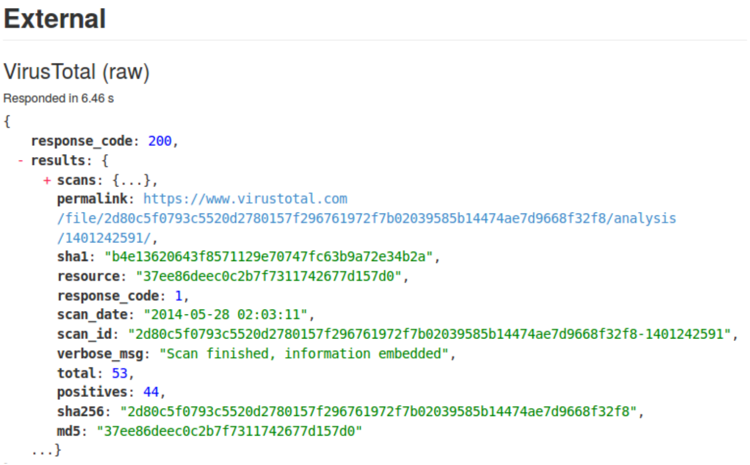
Adding it to your analyzers

```
$ sudo supervisorctl restart probe_app
$ sudo supervisorctl
probe_app        RUNNING   pid 2732, uptime 0:06:07
supervisor> help

default commands (type help <topic>):
=====================================
add    clear  fg        open  quit
avail  exit   maintail  pid   reload
remove  restart   start   stop  update
reread  shutdown  status  tail  version

supervisor> tail -f probe_app
```

Outline

1. IRMA Installation
2. IRMA Internals
3. Activating Analyzers
4. Developing Analyzers
5. Customizing the API output
6. Conclusion & Future Work

Analyzers results can be rough



External

VirusTotal (raw)

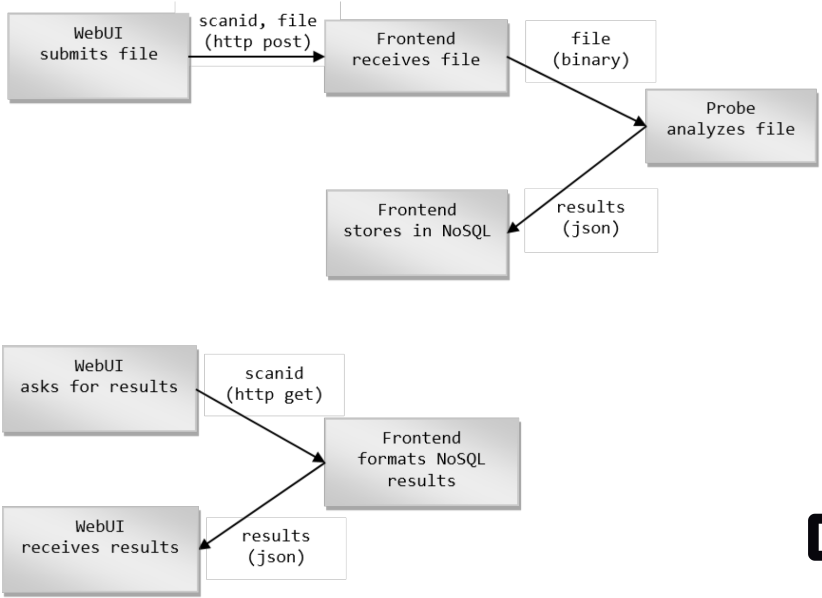
Responded in 6.46 s

```
{
    response_code: 200,
  - results: {
      + scans: {...},
        permalink: https://www.virustotal.com
        /file/2d80c5f0793c5520d2780157f296761972f7b02039585b14474ae7d9668f32f8/analysis
        /1401242591/,
        sha1: "b4e13620643f8571129e70747fc63b9a72e34b2a",
        resource: "37ee86deec0c2b7f7311742677d157d0",
        response_code: 1,
        scan_date: "2014-05-28 02:03:11",
        scan_id: "2d80c5f0793c5520d2780157f296761972f7b02039585b14474ae7d9668f32f8-1401242591",
        verbose_msg: "Scan finished, information embedded",
        total: 53,
        positives: 44,
        sha256: "2d80c5f0793c5520d2780157f296761972f7b02039585b14474ae7d9668f32f8",
        md5: "37ee86deec0c2b7f7311742677d157d0"
    ...}
```

Introducing Formatters

- Probes should not filter their output
- Formatters will be called on every results query
- They should only do filtering / light processing on raw results
- Same dynamic discovery method as probes

Results workflow



WebUI submits file

scanid, file (http post)

Frontend receives file

file (binary)

Probe analyzes file

results (json)

Frontend stores in NoSQL

WebUI asks for results

scanid (http get)

Frontend formats NoSQL results

results (json)

WebUI receives results

Architecture of the irma-frontend

```
$ tree -L 1 /var/www/prod.project.local/current
/var/www/prod.project.local/current
+-- config
+-- docs
+-- extras
+-- frontend
+-- lib
+-- web
```

Frontend API

```
$ tree -L 1 /var/www/prod.project.local/current/frontend
/var/www/prod.project.local/current/frontend
|-- api
+-- cli
+-- controllers
+-- helpers
+-- __init__.py
+-- models
+-- tasks.py
```

Frontend API

```
$ tree -L 1 /var/www/prod.project.local/current/frontend
/var/www/prod.project.local/current/frontend
|-- api
+-- cli
+-- controllers
+-- helpers
+-- __init__.py
+-- models
+-- tasks.py
```

```
$ tree /var/www/prod.project.local/current/frontend
/var/www/prod.project.local/current/frontend
|-- helpers
|   |-- formatters
|       |-- antivirus
|       +-- external
|           +-- virustotal
+-- [...]
```

Formatter registration

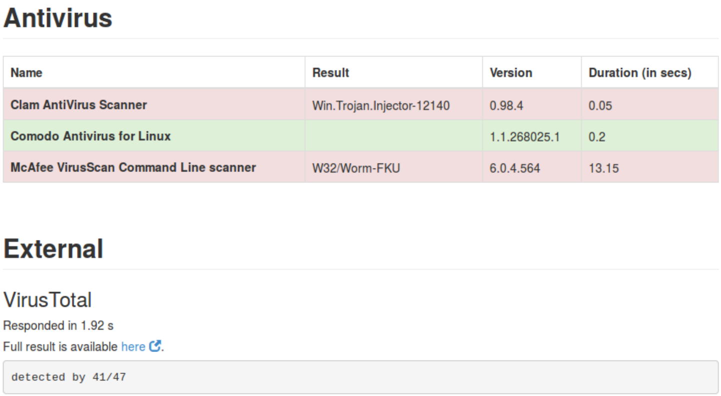
Each formatter:

- declares what type of results it could handle
- receives a copy of raw results and returns the formatted version

Formatter Example: Virustotal

```
@staticmethod
def can_handle_results(raw_result):
    expected_name = VirusTotalFormatterPlugin.plugin_name
    expected_category = VirusTotalFormatterPlugin.
        plugin_category
    return raw_result.get('type', None) == expected_category
         and \
        raw_result.get('name', None) == expected_name

@staticmethod
def format(raw_result):
    [...]
    status = raw_result.get('status', -1)
    vt_result = raw_result.get('results', {})
    if status != -1:
        av_result = vt_result.get('results', {})
    if status == 1:
        # get ratios from virustotal results
        nb_detect = av_result.get('positives', 0)
        nb_total = av_result.get('total', 0)
        raw_result['results'] = "detected by {0}/{1}" \
                                "".format(nb_detect,
                                    nb_total)
```

Formatter Example: Virustotal



Antivirus

Name	Result	Version	Duration (in secs)
Clam AntiVirus Scanner	Win.Trojan.Injector-12140	0.98.4	0.05
Comodo Antivirus for Linux		1.1.268025.1	0.2
McAfee VirusScan Command Line scanner	W32/Worm-FKU	6.0.4.564	13.15

External

VirusTotal

Responded in 1.92 s

Full result is available here ↗.

```
detected by 41/47
```

Outline

1. IRMA Installation
2. IRMA Internals
3. Activating Analyzers
4. Developing Analyzers
5. Customizing the API output
6. Conclusion & Future Work

Conclusion

- Install IRMA
- Discuss IRMA Internals
- Activate more analyzers
- Develop a custom analyzers
- Customize the API output

Future Work

- Adding more submitters to IRMA (mail, desktop, etc.)
- Add support for database sharing and anonymization
- Automatic retrieval of malware samples
- Data-mining on the database

Future Work

- Adding more submitters to IRMA (mail, desktop, etc.)
- Add support for database sharing and anonymization
- Automatic retrieval of malware samples
- Data-mining on the database

There is still a lot of work to do, and lots of ways for improvement

Roadmap (next release)

- Search and statistics on malware database
- Add support for new probes (sandbox, more AV, ...)
- Provisioning for windows probes

Roadmap (next release)

- Search and statistics on malware database
- Add support for new probes (sandbox, more AV, ...)
- Provisioning for windows probes

Feel free to give feedbacks and to submit your craziest ideas
they will be integrated to the roadmap

Authors and Contributors (1)

IRMA authors:

- David Carle – davounet – QUARKSLAB
- Bruno Dorsemaine – Ipecheur – Orange Group IS&T
- Guillaume Dedrie – guillaumededrie – QUARKSLAB
- Fernand Lone-Sang – kamino – QUARKSLAB
- Alexandre Quint – ch0k0bn – QUARKSLAB

IRMA Contributors (thanks for feedbacks !):

- Jean-Paul Weber
- ... and may be you ?

Authors and Contributors (2)

Join the Community or Contact us

GitHub

```
https://github.com/quarkslab/irma-frontend
https://github.com/quarkslab/irma-brain
https://github.com/quarkslab/irma-probe
```

```
@qb_irma
```

#irc

```
#qb_irma@freenode
```

Join the Community or Contact us

GitHub

https://github.com/quarkslab/irma-frontend
https://github.com/quarkslab/irma-brain
https://github.com/quarkslab/irma-probe

@qb_irma

#irc

#qb_irma@freenode

You need our expertise to develop a specific feature ?

irma-dev@quarkslab.com

Questions ?

QUARKSLAB
INNOVATIVE SECURITY

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com