IRhelper Documentation

Release alpha

etz69

Jan 22, 2018

Contents:

A play POC tool for first quick analysis of memory images for fun and learning!

IRHelper is meant to be a simple tool for automating as much as possible the tasks an analyst would perform when acquiring a memory dump. It was inspired by the SANS Windows Forensic Analysis poster which provides steps to perform during DFIR assuming both disk and memory dumps are available.

https://digital-forensics.sans.org/community/cheat-sheets

https://digital-forensics.sans.org/media/poster-windows-forensics-2016.pdf

- Step 01: Prep Evidence/Data Reduction
- Step 02: Anti-Virus Checks
- Step 03: Indicators of compromise search
- Step 04: Automated memory analysis
- Step 05: Evidence of persistence
- Step 06: Packing/Entropy check
- Step 07: Review event logs
- Step 08: Super timeline examination
- Step 09: By-Hand memory analysis
- Step 10: By-Hand 3rd Party Hash lookups
- Step 11: MFT Anomalies
- Step 12: File time anomalies
- Step 13: If you got malware then Hurrey !!! If not look deeper!

IRHelper will cover for you the following steps which normally would be run By-Hand!

- Step 09: By-Hand memory analysis
- Step 10: By-Hand 3rd Party Hash lookups
- **Bonus**: Try to extract other information which would normally be found on the disk

The Bonus part is only best effort as data might be paged so we wont have enough information to extract what we want.

So the high level objectives of IRHelper would be:

- To extract as much information as possible from a memory dump and present it to the user in well presented/readable format
- To enable even novice users to be able to use it and extend it
- Learn while you play with it (Python/Memory analysis/Writing Volatility plugins)
- Integrate with other tools which can help in your decision making progress
- Focus on what matters
- Experiment with new techniques of detecting suspicious patterns
- Utilize as much as possible the existing volatility plugins
- Have fun running memory analysis!

Volatility is an amazing and very powerful tool for performing memory analysis. However for the novice user it does come with some drawbacks.

- Has to decide which plugins to run and what information is valuable for further analysis

- Requires scripting or development/coding skills to take full advantage of it
- Difficult to keep track of the different information and where to go next
- To detect some obvious patterns or leads can take long time
- Different OS versions require from the user to know by heart many OS internals

CHAPTER 1

Similar tools

- Volutility
- VolatilityBot
- Evolve
- DAMM

DAMM and VolatilityBot have similar objectives to irhelper. However DAMM is not easy to extend unless the user knows how to write volatility code and does not provide standard report at the end of the analysis and integrations with 3rd party tools. VolatilityBot was discovered after started writing irhelper :) although they have quite different objectives. IRHelper is also not meant to be very production code !

CHAPTER 2

IRHelper concepts

1. *Information extraction*: IRHelper modules used to extract as much information as possible from the memory image and store them for later processing. Usually sqlite is best option.
2. *Analysers*: Analyser modules or code is used to run data analysis on the extracted information. Analysers can combine multiple information and logic to provide some results or indicators which are not readily available to the original plugin or would take multiple steps by hand to construct

CHAPTER 3

Prerequisites

There are some 3rd party tools which are required to run the different modules. One basic one is Volatilty. Volatility has to be available in your path otherwise it will not be found from the modules and and you have to specify the full path. Currently volatility 2.5 was used for the development of the current code.

Other tools are used such as:

- Exiftool
- ClamAV
- RegRipper

Note: You might encounter problems with matplotlib. In which case disable from the settings file.

CHAPTER 4

Install

Installation is quick and easy:

```
git clone https://github.com/etz69/irhelper.git
cd irhelper
virtualenv venv
source dev/bin/activate
pip install -r requirements.txt
```

CHAPTER 5

Directory structure

The directory structure of the project is as follows:

```
irhelper.py
This is the main program to execute

vol_plugins
Contains custom or contrib unofficial Volatilty plugins

templates
Contains the report template

dump
Directory to dump code or artifacts from memory for further analysis

modules
Contains all irhelper modules

export
This is where you will find your shiny report !

docs
This amazing documentation !
```

CHAPTER 6

Usage

```
(venv)fsck:irhelper dxl$ python irhelper.py -h

usage: irhelper.py [-h] [-p [PROFILE]] [-r [RISK]] [--cache] [--debug]
                   [--initdb] [--hash] [--vt] [--osint] [-v]
                   reportTemplate memoryImageFile

  ;)( ;
 :----:
C|====|
 |    |
 `----'

The IR helper python tool!

positional arguments:
  reportTemplate        Report template to use
  memoryImageFile       The memory image file you want to analyse

optional arguments:
  -h, --help            show this help message and exit
  -p [PROFILE], --profile [PROFILE]
                        Volatility profile (Optional)
  -r [RISK], --risk [RISK]
                        Risk level to show processes (default 2)
  --cache               Enable cache
  --debug               Run in debug
  --initdb              Initialise local DB
  --hash                Generate hashes
  --vt                  Check VirusTotal for suspicious hash (API KEY
                        required)
  --osint               Check C1fApp for OSINT of ip/domain (API KEY required)
  -v, --version         show program's version number and exit
```

CHAPTER 7

Run

To run irhelper just point to the image you want to analyse:

```
python irhelper.py --initdb --debug templates/report.html image_samples/conficker.img
DEBUG: Cleaning DB file
No cache, normal run

---------------------------------------
Gathering image initial information
---------------------------------------

---------------------------------------
KDBG: 0xf80002e400a0
DTB: 0x187000
KUSER_SHARED_DATA: 18446734727860715520
id: 1
Number of Processors: 1
KPCR for CPU 0: 18446735277665033472
AS Layer1: AMD64PagedMemory (Kernel AS)
Image date and time: 2012-04-06 21:28:39 UTC+0000
Image local date and time: 2012-04-06 17:28:39 -0400
PAE type: No PAE
Image Type (Service Pack): 1
Suggested Profile(s): WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
AS Layer2: FileAddressSpace (irhelper/image_samples/conficker.img)
---------------------------------------

0) Win7SP0x64
1) Win7SP1x64
2) Win2008R2SP0x64
3) Win2008R2SP1x64
```

CHAPTER 8

Final Report

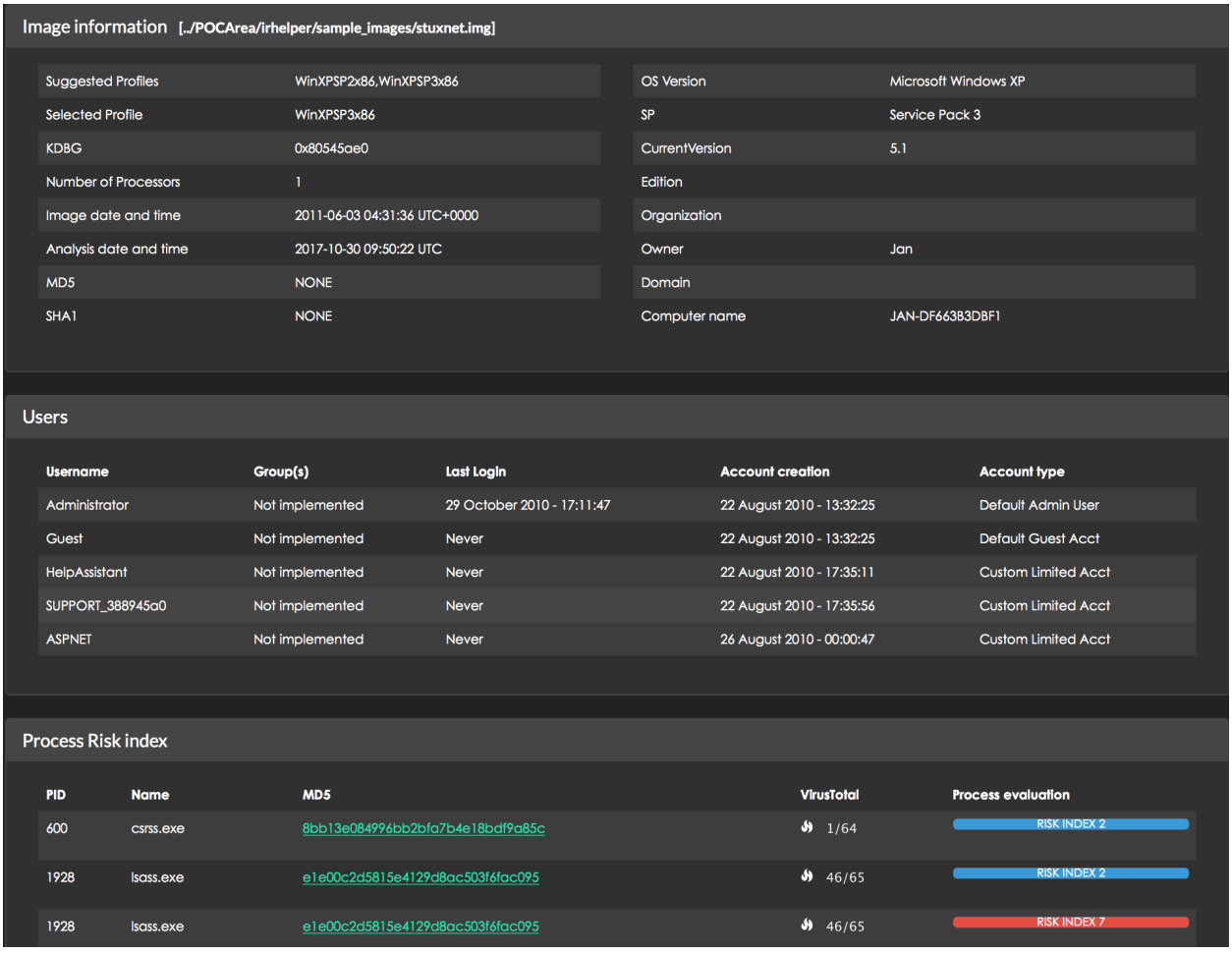

Image information [../POCArea/irhelper/sample_images/stuxnet.img]

Suggested Profiles	WinXPSP2x86,WinXPSP3x86	OS Version	Microsoft Windows XP
Selected Profile	WinXPSP3x86	SP	Service Pack 3
KDBG	0x80545ae0	CurrentVersion	5.1
Number of Processors	1	Edition	
Image date and time	2011-06-03 04:31:36 UTC+0000	Organization	
Analysis date and time	2017-10-30 09:50:22 UTC	Owner	Jan
MD5	NONE	Domain	
SHA1	NONE	Computer name	JAN-DF663B3DBF1

Users

Username	Group(s)	Last Login	Account creation	Account type
Administrator	Not implemented	29 October 2010 - 17:11:47	22 August 2010 - 13:32:25	Default Admin User
Guest	Not implemented	Never	22 August 2010 - 13:32:25	Default Guest Acct
HelpAssistant	Not implemented	Never	22 August 2010 - 17:35:11	Custom Limited Acct
SUPPORT_388945a0	Not implemented	Never	22 August 2010 - 17:35:56	Custom Limited Acct
ASPNET	Not implemented	Never	26 August 2010 - 00:00:47	Custom Limited Acct

Process Risk index

PID	Name	MD5	VirusTotal	Process evaluation
600	csrss.exe	8bb13e084996bb2bfa7b4e18bdf9a85c	1/64	RISK INDEX 2
1928	lsass.exe	e1e00c2d5815e4129d8ac503f6fac095	46/65	RISK INDEX 2
1928	lsass.exe	e1e00c2d5815e4129d8ac503f6fac095	46/65	RISK INDEX 7

CHAPTER 9

Contributing - We need you !

There are different ways you can contribute

- Write documentation
- Write code
- Report bugs

CHAPTER 10

Development

You can also run each module on its own while testing:

```
python modules/cmds/vol_imageinfo_module.py run image_samples/conficker.img␣
↪WinXPSP3x86

Python version: 2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)]

DEBUG: _cache: True
DEBUG: PLUGIN_DIR: /tmp/irhelper/vol_plugins/
DEBUG: _VOLATILITY_PROFILE: WinXPSP3x86

--------------------------------------
Gathering image initial information
--------------------------------------

DEBUG: ['vol.py', '--cache', '-f', 'image_samples/conficker.img', 'imageinfo', '--
↪output=sqlite', '--output-file=results.db']
DEBUG: Child process pid: 38001
Volatility Foundation Volatility Framework 2.5
{
    "status": true,
    "message": "",
    "cmd_results": {
        "KDBG": "0xf80002e400a0",
        "DTB": "0x187000",
        "KUSER_SHARED_DATA": "18446734727860715520",
        "id": 1,
        "Number of Processors": "1",
        "KPCR for CPU 0": "18446735277665033472",
        "AS Layer1": "AMD64PagedMemory (Kernel AS)",
        "Image date and time": "2012-04-06 21:28:39 UTC+0000",
        "Image local date and time": "2012-04-06 17:28:39 -0400",
        "PAE type": "No PAE",
        "Image Type (Service Pack)": "1",
```

```
        "Suggested Profile(s)": "WinXPSP2x86, WinXPSP3x86 (Instantiated with␣
↪WinXPSP2x86)",
        "AS Layer2": "FileAddressSpace (/tmp/irhelper/image_samples/conficker.img)"
    }
}
```

Logging

For logging purposes there are three methods used:

```
debug()
err()
print_header()
```

And the standard print!

Database

Various DB (sqlite) utils can be found in the modules.db.DBops

New module development

Edit cmd_processor.py and add your module as a method in the Modules() class. For example you can follow the skeleton module. All you have to do is return the standard result dict and create the appropriate section in jinja style (and bootstrap) in the templates/report.html file.

For example we want to create a new module to capture the command line executed by the user. This will be done by running volatility with the cmdscan plugin, store the results in the sqlite and finally return a dictionary with the cmds executed.

We will name our module vol_cmdline_module.py and place it inside modules/cmds/

Add our module class name in the cmd_processor.py:

```
def vol_cmdscan(self, **kwargs):
    '''
    Run cmdscan and record the command execution output

    Args:
        project (project): the project

    Returns:
        dict: Returns standard module response dict
    '''

    if 'project' in kwargs:
        _project = kwargs['project']

        #The module method to run
        ircmd.vol_cmdline_module.vol_cmdscan(_project)
        #Retrieve the results
        return ircmd.vol_cmdline_module.get_result()
```

```
    else:
        raise ValueError("Project info is missing")
```

Finally copy the skeleton_module.py in the new file and adjust!

CHAPTER 11

Research

Step 1: Prep evidence and data reduction

action: Hash lists from NSRL **description:** Download known MD5 hashes from NSRL for minimizing the false positives **references:**

- https://www.nsrl.nist.gov/Downloads.htm
- http://nsrlquery.sourceforge.net

feature:

Step 02: AV Checks

action: Run AV scans **description:** Run AV scan on extracted executables and dlls. Download yara rules and search on the different memory artifacts. ClamAv also supports yara **references:**

feature:

Step 03: IOC search

action: Search for IOCs **description:** Download yara rules and search on the different memory artifacts. ClamAv also supports yara **references:**

- https://github.com/Yara-Rules/rules
- https://malwareconfig.com/stats/

feature:

Step 04: Automated memory analysis

action: Automate daunting tasks for memory analysis **description:** Currently this is work in progress **references:**

feature:

Step 06: Packing/Entropy check

action: Calculate the density (entropy) of specific filetypes (exe and dll)

description: Files with low entropy than normal (what is normal?) may be packed executables which may lead you to potential malware on the system. The tool we select to carry out the scan is DensityScout! We will also try standard entropy with python implementation (slower) and slightly different than DensityScout and other approaches to detect packing Most likely files with "entropy" less than 0.1 (DensityScout) we can bring to the attention of the analyst. However in a default Windows installation we can see that there several legitimate files below 0.1 . This technique is likely to produce false positives.Here we can use outliers

references:

- https://www.cert.at/downloads/software/densityscout_en.html
- https://github.com/bridgeythegeek/regentropy
- https://github.com/dchad/malware-detection

feature: Packing entropy information of extracted files

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

CHAPTER 13

Module documentation

class modules.utils.helper.**Project**(*settings_path*)

Project class for all related data and methods of the project. This is the main class we have to load at the start of the project. It contains the necessary values for most of the project details such as the profile , directory locations, flags for features. It also provides several methods to provide access globally to the standard vars

clean_db()

Deletes the DB, cache and all files from dump dir

get_plugins_dir()

Returns the plugin directory for our custom plugins

get_root()

Return the root directory of the project. This is defined in the settings.py file and it is mandatory

init_db(*db_name*)

Set the DB name to be used from now on

@db_name (str): the db name

static load_properties()

Load the settings.py file

class modules.cmd_processor.**CommandProcessor**

class modules.cmd_processor.**Modules**

Simple command processor for adding new modules and retrieving results All modules return a dict which is of the following format:

```
result = {'status': True, 'message': '', 'cmd_results': ''}
```

`status`: If the module completely fails set this to False

`message`: A descriptive message , usually to show why it failed

`cmd_results`: This is usually a dict containing all the data which will be put in the report template

vol_cmdscan(***kwargs*)

Run cmdscan and record the command execution output

Args: project (project): the project

Returns: dict: Returns standard module response dict

vol_getosversion(***kwargs*)
Reads registry keys and tries to identify OS version information

Args: project (project): the project

Returns: dict: Returns standard module response dict

vol_imageinfo(***kwargs*)
Retrieves basic image info such as the type, profiles, KDBG etc..

Args: project (project): the project

Returns: dict: Returns standard module response dict

vol_malfind_extend(***kwargs*)
Run malfind and analyses the output. ToDo ML for asm

Args: project (project): the project

Returns: dict: Returns standard module response dict

vol_netscan(***kwargs*)
Runs different modules to discover network connectivity

Args: project (project): the project

Returns: dict: Returns standard module response dict

vol_pslist(***kwargs*)
Get as much as possible process information and dump pslist binaries to disk. This module will also run `exiftool`

Args: project (project): the project

Returns: dict: Returns standard module response dict

vol_regdump(***kwargs*)
Dumps SAM registry and tries to extract user information

Args: project (project): the project

Returns: dict: Returns standard module response dict

class `modules.db.DBops.`**DBOps**(*db*)

clean_db(*db*)
Deletes the sqlite file (cleans the db)

@db: Target db name (file)

get_all_rows(*table_name*)
Retrieve all rows from a table

@table_name: the table name

insert_into_table(*table_name*, *row*)
Insert data into a table

@table_name: the table name

@data: Data is an array containing list of dictionary items in the form of columnName:value

new_table(*table_name*, *table_fields*)
Create a new db table

@table_name: the table name

@table_fields: Table fields is a dict containing the name of the column as the key and the data type as the value {'id':'integer','name':'text','path':'text'}

new_table_from_keys(*table_name*, *table_keys*)
Create a new db table based on table keys with default type text

@table_name: the table name

@table_fields: Table fields is a dict containing the name of the column as the key and the data type as the value {'id':'integer','name':'text','path':'text'}

patch_table(*table_name*, *column_name*, *column_type*)
Add a column to an exisitng table

@table_name: the table name

@column_name: the new column name

@column_type: The type of the new column

sqlite_query_to_json(*query*)
Execute a query and return all results in json format

@query (str): A string which describes the query for sqlite. Complex queries with filters do not work always

table_exists(*table_name*)
Check if a table exists

@table_name: the table name

update_value(*table_name*, *column_name*, *value*, *key_name*, *key*)
Update a value in a table

@table_name: The table name

@column_name: The column name

@value: The new value

@key_name: The key name you want to filter on

@key: The key value you want to filter on

Python Module Index

m

Index