Automation of memory-based IOC analysis

Alberto Rico Simal

Kongens Lyngby 2015

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

The purpose of this thesis is to identify and implement an automation system over the Volatility platform, providing a way to analyze memory images from an array of hosts in an efficient manner, with the goal of detecting indicators of compromise (IoC) within them, among a set of predefined malware traits.

To achieve this, an organizational context is assumed, where there's an infrastructure of hosts within a typology, sharing therefore a set of security policies, allowing the IT security manager to effectively maintain and verify the compliance of the latter, as well as to respond to an incident in an efficient way.

Preface

This thesis was prepared at DTU Compute in fulfillment of the requirements for acquiring an M.Sc. in Engineering.

The thesis deals with the possibilities of automatization of Indicator of Compromise discovery, as well as ways of enforcing security policies for those hosts in the context of an organization, fostering the capabilities of Volatility, a memory analysis tool.

The thesis work here presented consists of the following parts:

- Introduction and State of the Art, ranging over the current possibilities regarding memory-analysis, IoC discovery, and automation.
- Design, where different possibilities for such system implementation are evaluated and criticized.
- Testing of the resulting system, rating the capabilities and results obtained in a staging environment.
- Conclusions that can be drawn from the presented results, as well as future work that can be envisioned from the final status of the work.

The presented work is original, and credit is given where due, especially to the Volatility Foundation team, which tool is used here under its GNU General Public License.

Lyngby, 26-June-2015

Alberto Rico Simal

Acknowledgements

I would like to thank my supervisor, Robin Sharp, for his guidance and insight, throughout the completion of this project.

I'd also like to express my thanks to the NordSecMob Consortium and its team, for the opportunity to take this master program under the Erasmus Mundus Master Course scholarship.

And, of course, to my family and friends, without whose support and love, this wouldn't have been possible.

Contents

Chapter 1

Introduction

1.1 Indicators of Compromise

Indicators of Compromise, when related to Computer Forensics are defined as "forensic artifacts of an intrusion that can be identified on a host or network"[12]. This relates to the traces inherent or related to the attack vector followed to compromise the host (e.g. an open network connection to a system under the attacker control).

Abbreviated as IoC, they can fall under very different categories, and they can be more or less descriptive of an attack, depending on the probability of such traits making an appearance under a not-compromised system status. This is, an IoC can be as simple as an unknown running process, in the case of a host where there's an explicit policy about those. In a system where a user can run different applications, we would need more information about every unknown running process, rather than deeming their existence as IoC.

For lack of better words, we can refer to such concepts as "resolution" and "scope". Resolution would describe the probability of an IoC pertaining to an actual attack, while scope defines how much of the attack behaviour can be captured in the IoC description (e.g, a black-listed device driver existence, as IoC, would have a high resolution, although might have a low scope rating, if

the device can use different drivers).

Due to the variability and difficulty to express such ratings due to malware variability, and developing the current project focused on an organizational scope, we'll focus on those IoCs that can be described through security policies.

1.2 APT - Advanced Persistent Threats

In counterposition to general web security threats, where vulnerabilities are exploited in mass through the usage of web-crawlers, and usually take place after public release of the vulnerability details, APT security threats are those targeting a specific entity, in a stealthy and continuous manner. They are defined as advanced, due to the sophisticated techniques used to achieve them, usually involving zero-day vulnerabilities (vulnerabilities not yet publicly known).

These are usually focused on targets where the reward for a successful intrusion outweighs the research and work needed, such as government systems, or companies with valuable or strategic electronic assets; and/or when an attacker needs a larger window between the attack and its consequent discovery and incident response.

1.3 Memory-based analysis, Volatility

The type of analysis chosen to gather information about the target system status, is memory-based, where an image of the running system will be taken and run through the automated discovery tool.

This provides us, technically, with all the data as for how the system is currently running, as well as residual artifacts of behaviors that executed sometime in the past, provided the image includes all physical address spaces[15, p. 57]. This is of importance for the detection of advanced persistent threats.

1.4 Automation

Collection and process of memory images will follow a client-server scheme, where a system will be receiving and queuing them through the analysis process.

The rationale behind this is that network bandwidth is ample within the organization scope, while individual hosts might be of limited processing power. This makes it logical to transfer large image files over the network, delegating the analysis to a server, or servers, where the image analysis can be scheduled and processed without disturbing the workload of the systems.[1]

Additionally, this scheme fits well into Incident Recovery situations, where a set of servers can be spawned aside, without running anything but a memory snapshot tool on the target systems.

1.5 Project plan

The time allocated for the development of this project is in accordance to the ECTS system, where 1 ECTS point equates to approximately 25 hours of work.[3] Given 30 ECTS points, 750 hours are planned for all the project phases, including:

- Research, state of the art, scope definition.
- Design.
- Implementation.
- Testing, evaluation and documentation.

The schedule was devised following the Gantt diagram in figure 1.1, for each of the 20 weeks for the project.

1.6 Summary

We have defined what an IoC is, as well as how they can be discovered through memory analysis, with a rough idea on how such system could be automatized.

Thus, we are ready now to describe the current state of the art of the field.

[1] Assumming those systems need to be continuously available, as in servers, or that the analysis will be scheduled more than once a day, discarding the nightly processing.

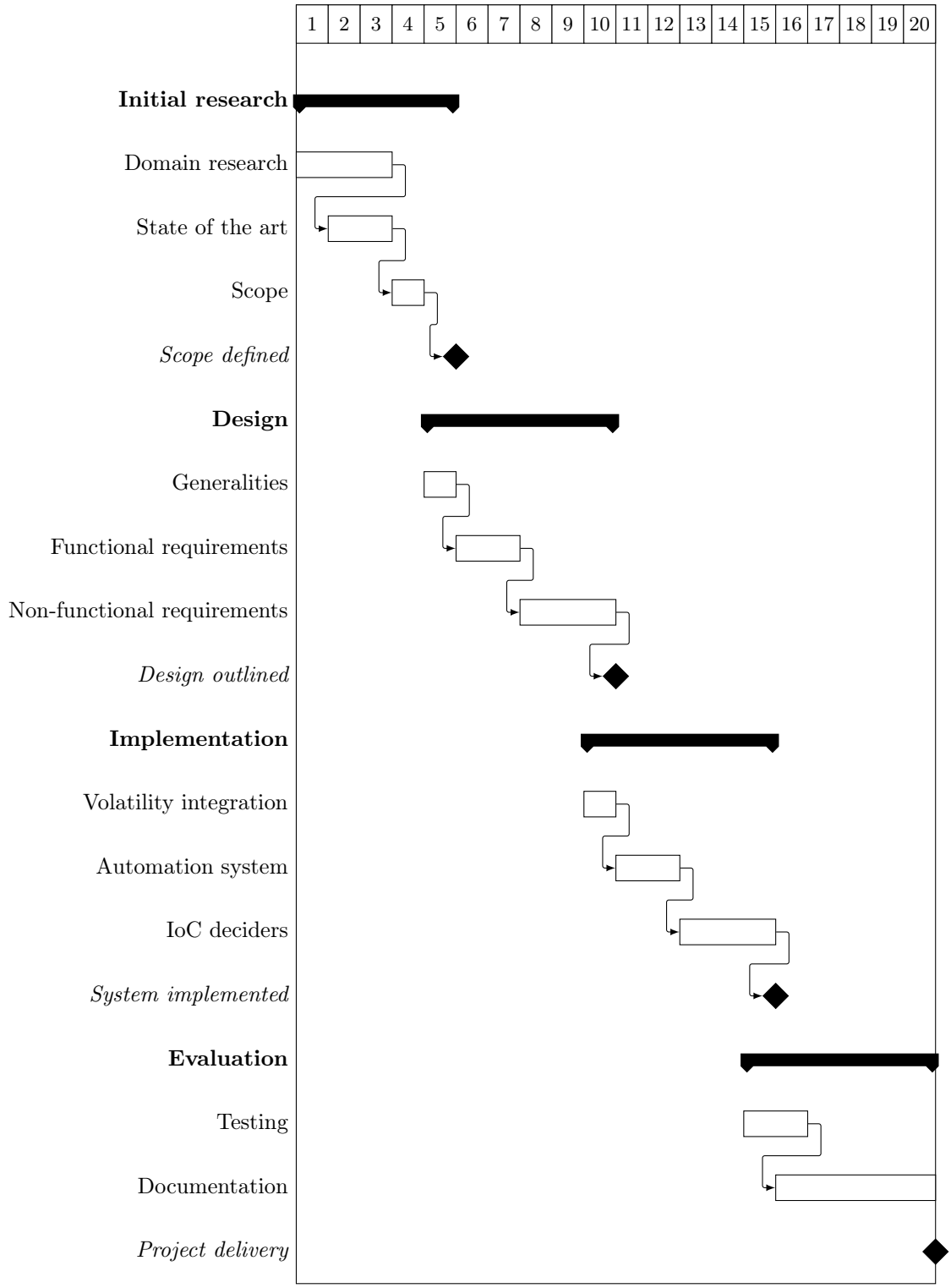

Figure 1.1: Project schedule

Chapter 2

State of the art

2.1 Automated memory analysis tools

The market for memory analysis tools, focused in security, is a niche area, with few representatives including:

- EnCase Forensic.[23]
- MAGNET IEF. [6]
- Mandiant Redline. [17]
- Volatility.[8]

All these fulfil the analysis requirements for IoC discovery (performing structure/artifact search through provided memory images), but have different characteristics that limit the possibilities regarding expansion or scripting, as required for a project as the present one.

The license with which the tools are published, as well as the availability of an API, possibility to integrate custom modules/scripts, or a command-line feature to customise searchs, are the features subject to comparison (see figure 2.1).

Tool	License	API	Custom mod.	Command-line
EnCase Forensic	Commercial	No	No	No
MAGNET IEF	Commercial	No	No	No
Mandiant Redline	Freeware (closed-source)	No	No	No
Volatility	GNU-GPL (open-source)	No	Yes	Yes

Figure 2.1: Memory analysis tools considered for the project, considering scripting/automation potential, or extension with custom modules

2.1.1 EnCase Forensic

The first one, part of the EnCase solution, is probably the most popular framework for forensics analysis in industry, providing as well the means to produce evidence in court, according to the developers claim that it "preserves data in an evidence file format (L01,Lx01 or E01, Ex01) with an unsurpassed record of court acceptance"[22].

Although open/raw formats are readable by this solution, the mentioned formats are proprietary, just as the product itself, limiting its possibilities for the present project.

2.1.2 MAGNET IEF

As for MAGNET IEF (standing for Internet Evidence Finder), it provides an intuitive interface for evidence gathering, focusing on the permanent storage, but providing as well with a "RAM Capture" tool, allowing us to run a limited artifact analysis on the resulting memory dump. Nevertheless, the same publishers refer[5] to Volatility and Mandiant Redline for further analysis.

2.1.3 Mandiant Redline

Mandiant Redline is closer to the kind of analysis required by this project, specifically dealing with IOCs, and using an open file format for the description of these, OpenIOC[10]. This enables security analysts to run tests over a batch of indicators of compromise, individually, per image.

2.1.4 Volatility

With Volatility, we find the first completely open solution in the field, allowing us to run different kinds of analysis that can uncover IoCs, over memory images. The offered analysis plugins are extendable, thanks to the open-sourced code[11], and although it doesn't offer an API, it's possible to develop one on top of the existing code (in Python), calling the plugins selectively and, therefore, allowing full automation[1].

2.2 IoC definition format

Mentioned before, the OpenIOC framework is the only currently open-source format, for IoC definition. It's based on XML, meaning that it's extendible, and that its usage is eased by the availability of XML parsers in most languages.

It follows an AND/OR tree structure, where individual traits within the IoC are specified, establishing a relationship between them and the relevance of their co-existance.

This seems ideal for checking of individual indicators of compromise; however, in the corporate scope, it doesn't cover the possibility of defining white or black lists for traits that have not been described before, but are part of a security policy.

Due to the project context, the definition of such lists would enable the resulting system to detect behaviours that haven't been described as malicious, but should trigger an alert due to policies or known preconditions, potentially enabling discovery of advanced persistent threats.

2.3 Manual analysis with Volatility

To understand how to automate IoC discovery, it's first relevant to expose how a manual analysis is performed, looking for a set of traits in a potentially compromised memory image.

[1] Please refer to "Implementation and testing" chapter for an overview about how integration between Volatility and our platform was achieved.

2.3.1 Example: finding Win32/PowerLoader

Here we'll start with a simple case—finding evidence of compromise by a typical Windows trojan. In this case, Win32/PowerLoader[2].

The intelligence that we have previously gathered is a set of traffic samples[4], where we can see that the trojan uses a hard-coded IP address to locate its relay server: 172.16.253.130. We can see such sample below:

```
2013-02-03 22:51:29.389714 IP 172.16.253.130.53 >
    8.8.8.8.53: 34738+ A? real-newslife[.]com. (35)
E..?.@5.5.+.^.real-newslife[.]com..
2013-02-03 22:51:29.389769 IP 172.16.253.130.53 >
    4.2.2.2.53: 34738+ A? real-newslife[.]com. (35)
E..?.A5.5.+.j.real-newslife[.]com..
2013-02-03 22:51:29.533563 IP 8.8.8.8.53 >
    172.16.253.130.53: 34738 1/0/0 A 213.57.77.220 (51)
E..Ohb5.5.;real-newslife[.]com.T‘9M.
2013-02-03 22:51:29.542478 IP 172.16.253.130.1067 >
    213.57.77.220.80: Flags [S], seq 2345406742, win
    64240, options [mss 1460,nop,nop,sackOK], length 0
E..0.C@-9M..+.P..p.(.
2013-02-03 22:51:29.564096 IP 4.2.2.2.53 >
    172.16.253.130.53: 34738 1/0/0 A 213.57.77.220 (51)
E..Ohc..5.5.;P=.real-newslife[.]com9M.
2013-02-03 22:51:29.732505 IP 213.57.77.220.80 >
    172.16.253.130.1067: Flags [S.], seq 294311925, ack
     2345406743, win 64240, options [mss 1460], length
    0
E..,hd.9MP.+..‘E
2013-02-03 22:51:29.732618 IP 172.16.253.130.1067 >
    213.57.77.220.80: Flags [.], ack 1, win 64240,
    length 0
E..(.D@-9M..+.P..P][..
2013-02-03 22:51:29.732932 IP 172.16.253.130.1067 >
    213.57.77.220.80: Flags [P.], seq 1:267, ack 1, win
     64240, length 266
E..2.E@,9M..+.P..P
POST /postnuke/blog.php HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows
    NT 5.1; InfoPath.1)
```

```
Host: real-newslife[.]com
Content-Length: 84
Cache-Control: no-cache
```

First, we want to check if any connection has been created towards that address. We use the "connections" Volatility plugin to obtain a list of those (sample abbreviated listing below):

```
18:26 (master) ~/Documents/Development/my-stuff/master
    -thesis/src/client$ volatility connections -f DTU-5
    A40BFDB6E6.raw

Volatility Foundation Volatility Framework 2.4

Offset(V)  Local Address  Remote Address        Pid
---------- --------------- --------------------- ---
0x8620d688 10.0.2.15:1195  2.16.63.24:80         2024
0x862a3458 10.0.2.15:1123  2.16.63.48:80         2340
0x8631c790 10.0.2.15:1179  2.16.63.24:80         2024
0x8659bcd8 10.0.2.15:1175  2.16.63.48:80         2024
...
0x86204440 10.0.2.15:1170  172.16.253.130:53     2340
...
0x86233528 10.0.2.15:1251  216.58.209.130:80     2024
0x8623a298 10.0.2.15:1227  50.31.164.166:80      2024
0x862ce6d8 10.0.2.15:1116  216.58.209.130:443    2808
0x86449988 10.0.2.15:1092  216.58.209.106:443    2808
```

The result yields the offset at which the connection structure is located, along with the connection vector (IP addresses and ports), and the PIDs of the processes that hold them.

At this point, we have detected that, in fact, the system is or has been connected to the address (an indicator of compromise in itself). Nevertheless, we now want to know which process established the connection.

From the entry, we see that the process ID is 2340. We can obtain a listing of

the processes, along their PIDs using the "pslist" plugin. Let's see the result below:

```
18:26 (master) ~/Documents/Development/my-stuff/master
    -thesis/src/client$ volatility pslist -f DTU-5
    A40BFDB6E6.raw

Volatility Foundation Volatility Framework 2.4

Offset(V)  Name                    PID   PPID   Thds
      Hnds   Sess  Wow64 Start
                                Exit
---------- -------------------- ------ ------ ------
    -------- ------ ------
    ------------------------------
    ------------------------------

0x867c6830 System                    4      0     54
       546 ------      0
0x86645740 smss.exe                368      4      3
        19 ------      0 2015-06-25 04:04:27 UTC+0000
0x8651f128 csrss.exe               584    368     11
       575      0      0 2015-06-25 04:04:28 UTC+0000
0x86595128 winlogon.exe            608    368     26
       554      0      0 2015-06-25 04:04:28 UTC+0000
0x86681798 services.exe            652    608     15
       273      0      0 2015-06-25 04:04:28 UTC+0000
0x86570da0 lsass.exe               664    608     27
       385      0      0 2015-06-25 04:04:28 UTC+0000
0x866aa8a0 VBoxService.exe         824    652      8
       105      0      0 2015-06-25 04:04:28 UTC+0000
0x866732e8 svchost.exe             868    652     20
       218      0      0 2015-06-25 04:04:28 UTC+0000
0x865623d8 svchost.exe             956    652      9
       270      0      0 2015-06-25 04:04:29 UTC+0000
0x86432508 svchost.exe            1048    652     84
      1457      0      0 2015-06-25 04:04:29 UTC+0000
0x863ce7a8 svchost.exe            1108    652     11
       102      0      0 2015-06-25 04:04:29 UTC+0000
0x863ae710 svchost.exe            1168    652     12
       183      0      0 2015-06-25 04:04:29 UTC+0000
0x863a6020 explorer.exe           1580   1528     14
       372      0      0 2015-06-25 04:04:30 UTC+0000
```

```
...
0x863a6020 cacax.exe                2430    1580      2
        1134        0        0 2015-06-25 17:51:47 UTC
    +0000
```

Here we identify the owner of the connection as "cacax.exe", according to the process ID. This gives us additional information to continue tracing the source—the PPID, the spawning parent process ID.

This concludes that the trojan was executed by the "explorer.exe", PID 1580 process (since we have launched this manually, simply double-clicking on the application executable).

2.3.2 Automatizing the process

To automate the analysis process, we need to decide:

- Input: intelligence/previous knowledge about the traits that are to be found.
- Sequences: processing and decision flow.
- Output: generated intelligence—whether evidence has been found, and useful information about it.

In the previous example, we have started from a single input (an IP address), and we have concluded that the infringing process has the image name "cacax.exe"—the output.

The sequence could be modeled as a flow chart (see Figure 2.2).

2.4 Summary

Different memory-analysis tools have been analysed, within the ones providing IoC discovery capabilities, and the automation possibilities for each have been evaluated.

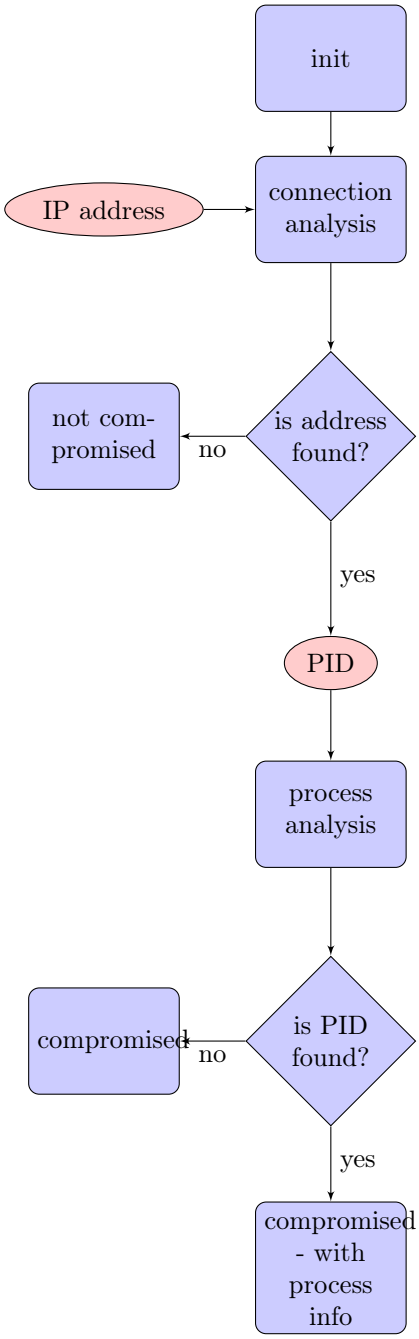

Figure 2.2: Connection analysis flowchart, when performing a manual analysis.

Volatility seems the natural tool with which to develop the present framework, thanks not only to its custom modules functionality, but also thanks to it being open-source, allowing us to build on top of it for a seamless integration.

With the example of a manual analysis, we can see how the automation could be designed, through generalisation of the flow chart.

Chapter 3

Design

3.1 Design decisions

3.1.1 Risk, probability and uncertainty

In IT Security, risk is defined as "the potential that a given threat will exploit vulnerabilities of an asset or group of assets and thereby cause harm to the organization. It is measured in terms of a combination of the probability of occurrence of an event and its consequence", according to the ISO27005 standard[14].

When detecting indicators of compromise of advanced persistent threats, although in certain cases it's semantically possible to determine the minimum impact caused by a particular discovered trait, the concept of probability doesn't apply—for the attack was already *possibly* conducted.

This discards the usage of a risk metric when performing analysis, but still allows us to account for the individual impact of each trait, qualitatively, instead of using a numerical value.

3.1.2 Target systems

Due to the nature of the APT, attacks are likely to take place through host systems, where human interaction gives a lead on bypassing certain security measures—such as foreign USB devices, where these ports can't be disabled due to the workflow, or specially crafted exploits wrapped in apparently harmless documents.

Focusing on the organizational context, we account for a widespread usage of Windows-based host systems[19]—making Windows the target OS of preference for the development of our system, being supported by Volatility existent plugins.

3.1.3 Target traits, Volatility plugins

The traits—or individual pieces of relevant information--should be chosen to cover the most significance when analysing for APT.

We shall focus on network activity, processes and privileges, as they're feasible to characterise and scope, as opposed to scouring for other particular traits that are only discoverable through heuristics.

3.1.4 Client-server architecture

The implemented system is envisioned to be deployed as a server within the scope network, with queuing capabilities, so that the hosts can submit their memory images on a scheduled basis—or manually, when deployed for incident recovery.

The server will then report to the system administrator or security analyst, notifying of discovered traits, if any.

3.2 Implementation requirements

3.2.1 Scope

The scope of the project should cover:

- Automation of IoC individual trait discovery, based on specific Volatility plugins.
- Intefacing/extending Volatility, to accommodate the automated calls to its plugins.
- Definition of black/white lists, and enforcement through automation, focused on APT discovery.

The requirements are here distinguished between functional and non-functional, where:

- Functional requirements refer to the actual capabilities of the framework to be implemented.
- Non-functional requirements detail how this capabilities will work, regarding technologies and methods to be followed and how they were constrained.

RFC 2119 is followed to indicate requirement level[1].

3.2.2 Functional requirements

3.2.2.1 Trait/IoC characterization

Due to the varied nature of the advanced persistent threats, the IoC characterization shall be expressed in the form of policies, known as:

- blacklists, when the traits should not appear within the target image, or
- whitelists, when the traits within the policy are the only ones allowed within the target image.

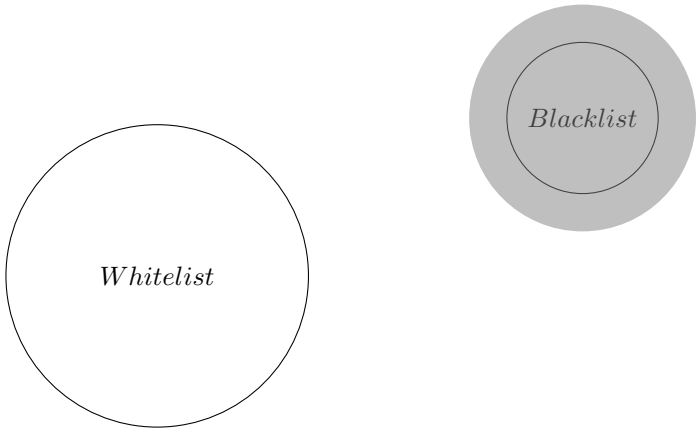

Figure 3.1: Disjoint sets of policies

Since both sets of traits are disjoint, their usage shall be mutually exclusive within each analysis—only one behaviour shall be chosen for a given policy/-analysis.

This differentiation is meant to help the establisment of policies, depending on the previous knowledge about the system—if this knowledge is partial, or the system is intended to find an IoC within a very specific area, blacklists shall be used, whereas a strictly defined system will benefit from whitelist usage.

3.2.2.2 Connection analysis

The framework shall analyse the current and past connections, looking for anomalous traits within those. These traits shall cover the originating IP address, as well as the ports involved in the connection.

Additionally, when an anomalous trait is discovered within a connection, the originating process identifier (PID) shall be extracted from its structure, and the corresponding process structure queried within the same image, and analysed, providing the user with information about the source of the infringing trait.

3.2.2.3 Running processes analysis

Current and past running processes shall be analysed, in search for anomalous spawns. These checks shall cover image names, as well as security ID (SID). SIDs are alphanumeric strings, used to enforce access control, unique for each trustee (or user), but following a well-known type of convention [18].

This analysis shall provide regular expresion matching for such SIDs.

3.2.2.4 Rootkit analysis

Volatility enables the user, using one of the its default plugins, to perform an automated search for rootkit software within an image. This includes detection of hooks within the System Service Dispatch Table (SSDT)—these hooks allow an attacker to capture system calls, and jump the resulting execution through a different part of code.

Such functionality shall be leveraged to provide the user with information of existing hooks, should they exist.

3.2.2.5 Drivers and devices analysis

Current corporate security policies tend to cover peripheral devices that are allowed for usage, if any, usually physically disabling USB ports. However, certain workstations provide USB as the only way of connecting a mouse and keyboard, enabling a user to exploit that port to withdraw confidential information from the organization systems, or as a vector to unknowingly introduce a piece of malware. Additionally, malicious virtual devices can be set in place as well.

By profiling which drivers can or can't be used in those systems, an automated system can detect anomalies as to which pieces of hardware, or virtual devices, are or have been present on a system.

The framework shall analyse in search for such devices, searching through the image device tree.

3.2.2.6 Automation

The analysis over a given image shall run unattended, its necessary parameters put in place before execution, and giving the user a detailed result of the analysis upon completion.

This means the system shall be able to gather information, and take decisions as to where to conduct further analysis, driven by the deciders set in place. These deciders shall be interchangeable and modular, for the end user to be able to modify behaviour in a simple way.

The inputs for the analysis shall be:

- The target image, in a compatible format,
- a set of signatures, containing the policies in a standard format,

The result shall be expressed in a structured, but human readable way, enabling both automated treatment and direct usage.

3.2.2.7 Extensibility

The framework shall take future developments into consideration, allowing the user to extend the framework to cover artifacts other than those exposed in the present work.

This shall follow a similar approach to Volatility, where different artifacts are integrated as modules, providing a well-known interface for further development.

3.2.3 Non-functional requirements

3.2.3.1 Analysis platform

The chosen platform for image analysis is Volatility, thanks to its open source, and modularity, that enables us to perform calls within its modules.

An interface shall be implemented for us to implement the discussed analysis, as well as for a future user to easily extend on those, or create her own.

3.2.3.2 Target systems

The chosen target for image analysis is the Windows system, due to popularity in an organisational context[19]. Nevertheless, deciders may be developed on top, that render the framework compatible with structures compatible with hosts other than Windows.

The image shall be gathered from the target, and fed to the framework, which shall be based on Python, meaning it shall be multiplatform.

3.2.3.3 Formats

The format of the input files shall be .RAW memory image files, although compatibility with different formats may be available, depending on current Volatility compatibility.

The format of the policy files shall be JSON, thanks to its easy compatibility with different tools, web, and human readability.

3.3 Summary

In this section, we have explored the overall design decisions for the implementation of the discussed framework, defining both the behaviour, and the interaction with the tool.

Chapter 4

Implementation and testing

4.1 Engine implementation

The engine is implemented keeping in mind the aforementioned extensibility requirement. It is conceived as a runner, taking a set of modules (named "deciders") that will check for traits (named "signatures"), using an initialized Volatility instance over the provided image (named "analyzer").

The deciders are simple classes that extend from the Decider class, and need only to implement a method that ascertains a Decision, describing whether matching traits have been found or not, along with a List of IndicatorOfCompromise objects, describing those.

The signatures are expressed following the JSON format, following the convention in figure 4.2.

Both deciders and signatures are placed in their respective source folders, where they will be automatically loaded from, on the framework initialization.

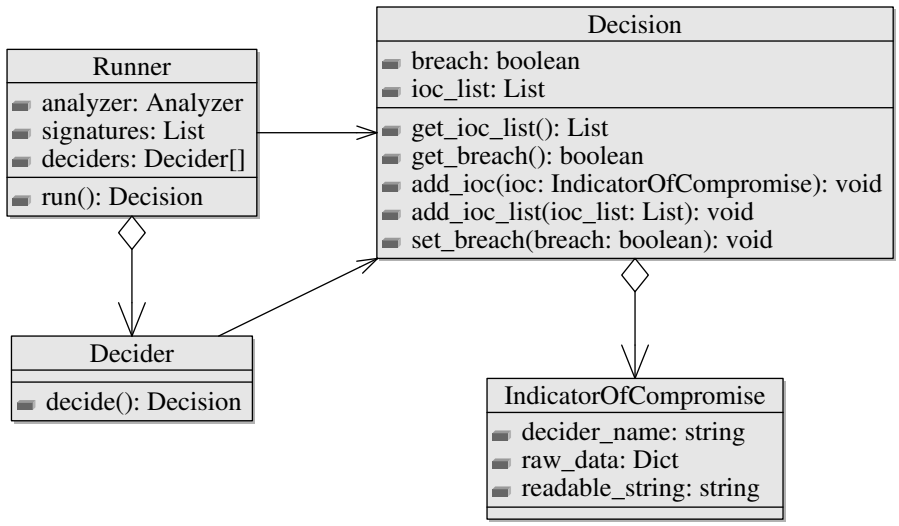

Figure 4.1: Framework class diagram

```
{
    "behaviour": "blacklist",

    "trait_name": [
        ...
    ],

    "another_trait_name": [
        ...
    ],
    ...
}
```

Figure 4.2: Default trait format,

4.1.1 Volatility integration

Volatility doesn't provide an API as such, but since it's open source, a make-do API was implemented for interacting with its modules.

The overall initialization code for the framework was analyzed, and extracted into an independent class. Here, the requested modules are imported and executed following the Volatility module loading code[9].

Due to the different types of data provided, there's no encapsulation for the results obtained for each module—this is delegated to the deciders algorithms.

The procedure followed when loading and calling the created platform API is:

1. The image is loaded onto memory:
   (a) Location config is set
   (b) Options are parsed (this causes the image to load)
2. Upon a received call:
   (a) The module is obtained from the available plugins from our Volatility installation.
   (b) The target object within the plugin is gathered—containing the "calculate" method.
   (c) The "calculate" method is executed. This returns a list of entities, each including the offset and data depending on the type of the plugin called (in the case of a process entity, it will contain the ImageName, the UniqueProcessId, ParentProcessId, number of threads, etc.)
3. The list is returned to be consumed by the decider. A decider will typically iterate over the list, finding relationships with the provided inputs (signatures).

4.1.2 Algorithm

The runner algorithm can be simplified as follows (please refer the the appendix "Extending the framework", for details about the decider implementation):

```
analyze_image():
```

```
worklist = clone(AVAILABLE_DECIDER_LIST)
decision = new Decision()

while (worklist.length > 0):
    decider = pop(worklist)
    decision = decision + decider.decide(ANALYZER, SIGNATURES)

return decision
```

4.1.3 Web interface

The framework creates a web server upon initalization, accepting file uploads from the hosts[1], with the image file compressed in ZIP format.

Running multithreaded, it concurrently analyzes the received images, acting as a black box to the submitting host—no information is provided on the JSON response, aside from a status message.

The information gathered is made available to the analyst through a different channel[2].

This interface is meant for the system administrator to be able to schedule host imaging and submission, while being in control of the gathered information.

4.1.4 Memory image acquisition

The framework user can use a memory imaging tool of her choice, as long as the resulting file is compatible with Volatility formats[7]. For the development and testing, MoonSols DumpIt[16] was used, for simplicity.

As of version 2.4, the one used in the project, the formats are:

- Raw/Padded Physical Memory
- Firewire (IEEE 1394)
- Expert Witness (EWF)

[1] Sample code for the client is provided.
[2] Implemented as e-mail notification, in this case.

- 32- and 64-bit Windows Crash Dump
- 32- and 64-bit Windows Hibernation
- 32- and 64-bit MachO files
- Virtualbox Core Dumps
- VMware Saved State (.vmss) and Snapshot (.vmsn)
- HPAK Format (FastDump)
- LiME (Linux Memory Extractor)
- QEMU VM memory dumps

4.2 Testing

For the evaluation of the present project, a set of proof of concepts were designed to test whether the IoC traits were exposed by the framework, or ignored otherwise.

The scope of this testing is limited by the fact that we are fostering functionality provided by Volatility, and it is merely meant to show an idea of the type of policy enforcement that can be achieved by the platform.

4.2.1 Environment

The running environment has been staged using Oracle VirtualBox, with the intention of constraining dynamically the characteristics of the system—namely, the available memory space, to test its effects in performance.

The relevant features for the environment are shown in figure 4.3.

4.2.2 Image file compression

The system allows for submission of zipped files for analysis. This is meant both to save storage space, in case an image archive is to be kept, as well as to decrease the transmission time, and lower the network usage—allowing for more frequent analysis.

OS	Microsoft Windows XP (32 bit)
OS version	Professional, 5.1, Build 2600.xpsp_sp3_qfe
RAM	256MB, 512MB and 1024MB
Host	MacOSX(10.10.3)/UNIX

Figure 4.3: Target system features, where the population and execution of the proofs of concept was conducted.

The structures in memory space are optimized for rapid access, rather than smaller footprint, creating redundancy (primarily due to unallocated areas) that makes for efficient compression of the memory image files. Using the populated snapshots for benchmarking, space savings of around 50% were achieved. This rate will be lower or higher depending on the actual memory usage.

4.2.3 Image population

It is important to introduce the concept of population, in order to obtain relevant results. We shall understand population, as the simulation of a regular working environment, through the execution of different processes, and user interactions with those.

To keep metrics relevant among different tests, and considering our results are entirely dependent on the memory structures present and past, this population needs to be:

- Randomized—this initial state must include sufficient user interactions, and running and killed processes, to create enough residue in the memory space, as an actual workstation would typically have.
- Reproducible—the initial state of the system must be the same for every test.

The randomization of the initial state was simulated manually, including the installation of an office automation suite, a Python platform, loading two different browsers with different sets of open URLs, along Flash-based websites. Interactions were performed on these applications and websites, performing daily tasks on them for a work day.

As for reprocibility, thanks to the usage of virtualization, it is achieved through snapshotting[20]. This ensures the perfect consistency of the initial state.

4.2.4 Benchmarking

4.2.4.1 Proofs of concept

To test for the accuracy of the analysis, a series of experiments were conducted, via PoC (proofs of concept). The approaches that were successfully tested for this, were:

Networking

- Web browsing to blacklisted addresses.
- Network configuration to use blacklisted proxies/DNS servers.
- Samples of code creating or accepting network connections.

This covers cases where a piece of software accesses sensitive addresses (such as the ones known to be in use as relay for a trojan malware package), or when a DNS server is changed for an unknown one (translating URLs into compromised IP addresses, where phising attacks can be conducted, for instance).

Processes

- Spawning blacklisted processes.
- User applications impersonating reserved process names (for security ID checking).

This checking enables discovery of processes executed out of policy, and checks for the SID to match a given regex, based on the well-known possible IDs—identifying processes owned by the user, admin or system.

Drivers

- Addition of an external USB hub.
- Rubber Ducky[13] driver presence.

Certain attacks take place through peripherals. A common case is the one created by the Rubber Ducky—it disguises itself as a keyboard, able to execute preprogrammed commands upon installation, with the same permissions as the user currently owns.

Since it has a particular footprint, using the uncommon "usbccgp" driver, its detection was included in this part.

Rootkit

- Detection of ZeroAccess/Sirefef presence (using ContagioDump samples[21]).

A common tecnique used by rootkits is attaching SSDT hooks. A test was performed to verify that we were able to capture those, along with the non-malicious system ones.

4.2.4.2 Analysis modeling

As discussed in the state of the art, when overviewing Volatility, a manual analysis can be modeled as a flow graph, along with inputs and outputs.

This serves as a high-level translation towards an algorithmic, automatic approach, and is an indication of how the deciders were implemented in the framework.

4.2.4.3 Networking analysis modeling

Input/Output

Input
Target memory image
List behaviour (black/white list)
IP addresses
Local ports
Remote ports
Output
The infringing connection, if any
If the spawning process is found, it will be included as evidence

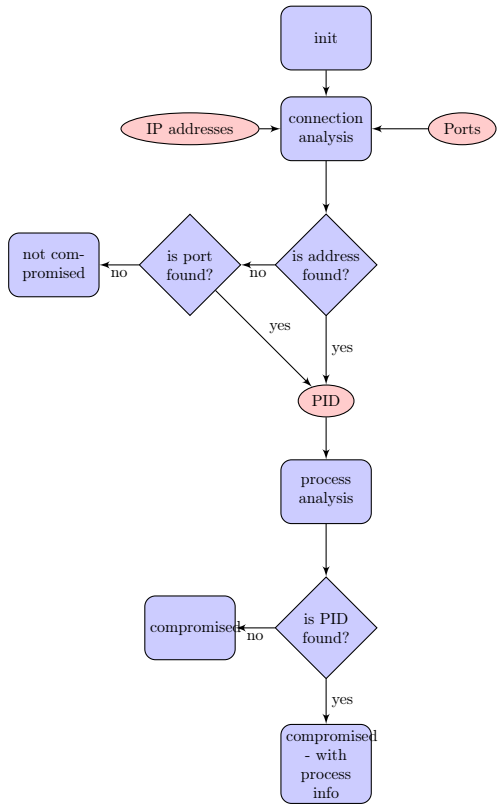

Figure 4.4: Connection analysis flowchart

4.2.4.4 Process analysis modeling

Input/Output

Input
Target memory image
List behaviour (black/white list)
Process names
Regular expressions for security IDs, along related processes
Output
The infringing processes
The processes with non-matching security ID

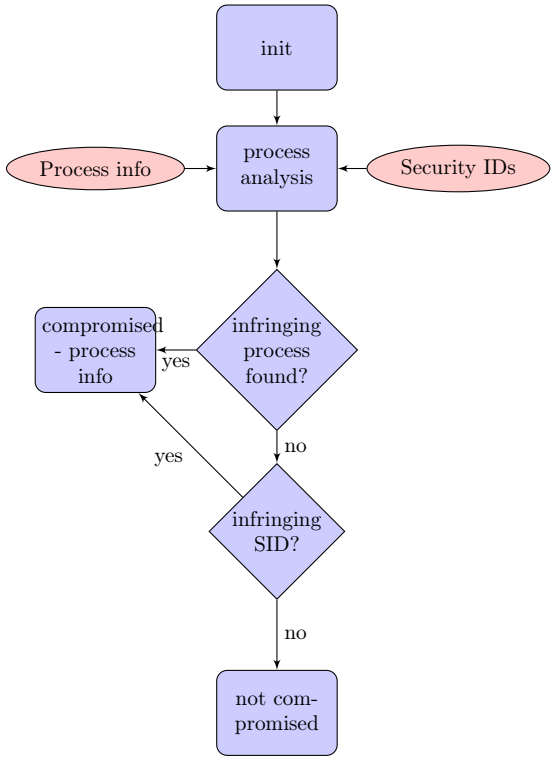

Figure 4.5: Process analysis flowchart

4.2.4.5 Driver analysis modeling

Input/Output

Input
Target memory image
List behaviour (black/white list)
Set of driver IDs
Output
The infringing driver list

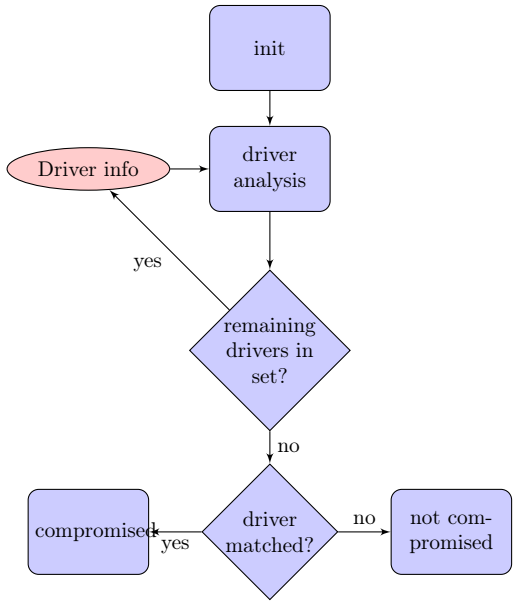

Figure 4.6: Driver analysis flowchart

4.2.4.6 Rootkit analysis modeling

Input/Output

Input
Target memory image
List behaviour (black/white list)
Set of SSDT hooks
Output
The infringing hooks

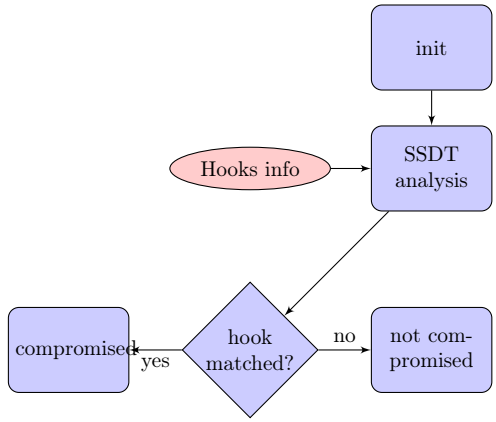

Figure 4.7: Rootkit analysis flowchart

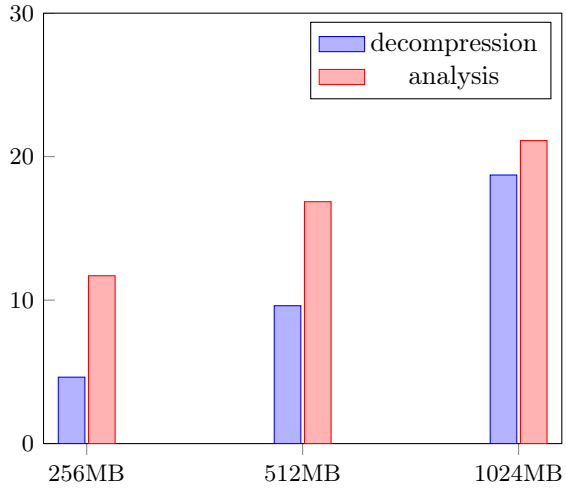

Figure 4.8: Plotted benchmark results, in seconds—compression time is distinguished from analysis time, for predicted growth differences

4.2.4.7 Performance

Performance is based on a single metric—timing. Using memory images from three different snapshots, for each memory size, the analysis time is computed. The framework server was deployed on a 3.06 GHz Intel Core 2 Duo machine, with 8 GB 1067 MHz, DDR3 SDRAM, providing 8.5 GB/s bandwidth[24] (relevant, due to the I/O intensive Volatility analysis).

Memory image creation time and network transmission grow linearly along memory space size, and therefore are not included in these results. Every image was timed three times, and the average time was taken.

The benchmarked results are exposed on figure 4.8[3].

4.2.4.8 Benchmark conclusions

The crafted PoCs were detected as expected, in most of the cases. In the case of the 256MB memory host, there were failures to detect certain previous

[3]Testing performed on the default populated images.

network connections. This is an expected behaviour, when memory is freed and reallocated due to low space available.

In the case of advanced attacks, such as the rootkit case, the success of the detection lies on the previous knowledge about the attack, since a specific set of traits needs to be known, in order to discard false positives.

Analysis time increases under less than linear growth, along image size.

This was at first unexpected due to types of sequential analysis conducted in the process, that are of quadratic, $\mathcal{O}(n^2)$ complexity.

Referring again to the network decider, the moment a infringing structure is detected, the process holding the connection is to be located through its PID, in order to get its image name—involving two sequential searches.

We can safely assume this is due to the way Volatility traverses the image files, using the available pointers to traverse linked structures when possible[15, p. 156] (when still allocated). With more analysis modules or deciders accessing deallocated structures, we can expect this to change.

As for the decompression, it takes a significative percentage of the time in the process, when the images are large enough. This decreases the incentive for compression, but it's up to the system administrator to optimize resources disabling or enabling compression.

4.3 Summary

The specifics about the platform implementation have been discussed, giving an overview over the overall decisions, and analyzing the performance when analysing images of different sizes (previously populated).

The reasoning behind the results is discussed in the benchmark conclusions.

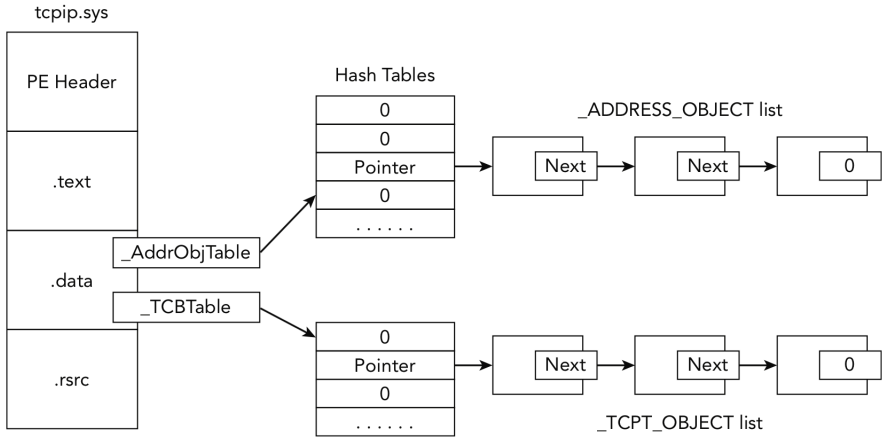

Figure 4.9: Socket and connection objects, structured as linked lists, as present in memory, illustrating how Volatility can traverse them in an efficient way[15, p. 315].

Chapter 5

Conclusions

5.1 Compliance with original goals

When first approaching the project, it was originally intended to design a platform that could discover advanced persistent threats, through indicators of compromise, in an automated way.

For this, the procedure followed by a security analyst was the target of our automation. Innovation posed two different challenges:

- An advanced persistent threat conventionally uses zero-day exploits, and is custom crafted, therefore it's extremely difficult to detect such threats without knowing its specific traits, or returning a very high false positive rate.

- It is already possible to detect IoC using a batch, such as the functionality provided by Mandiant Redline—again using previous knowledge, with a still small public library of IOC files[1].

[1] 336 available schemas, on iocbucket.com, as of June 2015

The lacking knowledge about the possible threats had to be garnered from a different source, and assumming an homogeneous environment such as the one APT are focused on—organisational systems—the solution was to define a set of lists that contained information about what a system should or shouldn't perform.

Therefore, the result could be described as a policy compliance tool, but with two key advantages:

- The analysis is external to the target host, and an archive of images can be kept in storage for a period of time, in case a new trait is discovered. This way, detection can take place retroactively
- Thanks to unallocated memory areas analysis, as provided by Volatility, it is possible to detect threats designed to have short interaction periods, as long as the are where the structure of interest lies hasn't been reallocated.

However, usage of memory images for automated analysis and policy enforcement is an unusual approach, due to weaknesses such as high resource usage, in terms of network, storage and processing. It is counterbalanced by the amount of information that can be obtained from the analysis, but it has a significant footprint on the systems.

The memory snapshots should be regarded as highly sensitive data, due to the memory imaging capturing data that is meant to be kept encrypted, hashed, or directly avoid its storage, such as passwords—temporarily present on RAM, once typed by the user.

5.2 Future work

The project has been designed with extendibility as a requirement. It is expected that a system administrator, or security analyst deploying this tool, will update such system with deciders that fit the organisation interests and resources to defend.

There are plenty of data structures that can be assessed through Volatility (currently with over 50 different implemented modules), so there's much room for improvement

Another interesting way to develop this work, would be complementing memory

analysis with storage analysis, checking for differences between loaded and stored modules, to check for modifications that can be evidence of an attack.

Appendix A

Deployment manual

A.1 Prerequisites

The main requirements for running the framework are:

- A working Python 2 installation,
- Volatility 2.4,
- SMTP server (for notification via e-mail)

Additionally, the platform uses the following Python plugins:

- Flask (server functionality)
- Werkzeug (id tools)

A.2 Obtaining the software

The software is publicly available on GitHub, available under GPL license, and can be obtained from https://github.com/AlbertoRico/volatility-o-matic.

You may obtain the package as a compressed folder, at:

```
https://github.com/AlbertoRico/vol-o-matic/archive/master.zip
```

Or using Git on a terminal:

```
$ git clone https://github.com/AlbertoRico/vol-o-matic.git
```

A.3 Server deployment

The first thing to do, prior to execution, is configuring the server with our own parameters (particularly, the e-mail address where we intend to receive the reports). This is located on the "config.json" file.

To run the server, simply execute the following, from the installation directory:

```
$ python server.py
```

At this point, an endpoint will be available on http://<host_ip_address>:<port>/upload

It will be expecting compressed (ZIP) memory images, submitted through multi-part HTTP POST. An example client is included, for illustration purposes.

A.4 Example of usage

To begin with, we start the Vol-o-matic server:

```
20:56 (master) ~/Documents/Development/my-stuff/master-thesis/src/server$ python server.py
Vol-o-matic 0.1 - Alberto Rico Simal - hello@albertori.co
Technical University of Denmark, 2015 - Under GNU GPL license

 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

We have previously obtained an image from a host that we want to analyse. We can use the included "submit_dump.py" script to submit it:

```
21:10 (master) ~/Documents/Development/my-stuff/master-thesis/src/client$ python submit_dump.py -f DTU-5A40BFDB6E6-
20150625-040558.raw -d http://localhost:5000/upload
Compressing dump file...
Submitting file for analysis...
Done.
21:12 (master) ~/Documents/Development/my-stuff/master-thesis/src/client$
```

Analysis will then take place on our server, submitting a report to the configured mail. In this case, we receive:

> -Breach report-
>
> Breach detected! - –Processes IoC– - cacax.exe - Blacklisted name - –Processes IoC– - cmd.exe - Pattern: Ŝ-1-5-21([
> s
> S])+-513$ - SID: S-1-5-21-1220945662-113007714-854245398-513 - –Connections IoC– - cacax.exe - 2408 - 0x81235828 - 108.162.232.205

Successfully identifying a trojan running on our stage system.

Bibliography

[1] Scott Bradner. Key words for use in rfcs to indicate requirement levels. 1997.

[2] Microsoft Security Center. Trojan:win32/powerloader. http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Trojan%3AWin32%2FPowerLoader#tab=1. [Online; accessed 25-06-2015].

[3] European Commission. European credit transfer and accumulation system (ects). http://ec.europa.eu/education/ects/ects_en.htm. [Online; accessed 25-06-2015].

[4] ComputerSecurity.org. Powerloader trojan downloader cacax.exe malware traffic sample. http://www.computersecurity.org/malware-traffic-samples/trojan-downloaders-malware-family-types/powerloader-trojan-downloader-cacax-exe-malware-traffic-sample/. [Online; accessed 25-06-2015].

[5] Magnet Forensics. Acquiring memory with magnet ram capture. http://www.magnetforensics.com/computer-forensics/acquiring-memory-with-magnet-ram-capture. [Online; accessed 25-06-2015].

[6] Magnet Forensics. Magnet ief, overview. https://www.magnetforensics.com/digital-forensics-software/internet-evidence-finder. [Online; accessed 25-06-2015].

[7] Google Code Volatility Foundation. Volatility, supported formats. https://code.google.com/p/volatility/wiki/FAQ21#What_memory_dump_formats_are_supported. [Online; accessed 25-06-2015].

[8] Volatility Foundation. Volatility foundation - releases. http://www.volatilityfoundation.org/#!releases/component_71401. [Online; accessed 25-06-2015].

[9] Volatility Foundation. Volatility vol.py code. https://github.com/volatilityfoundation/volatility/blob/master/vol.py. [Online; accessed 25-06-2015].

[10] The OpenIOC Framework. Openioc. http://www.openioc.org/. [Online; accessed 25-06-2015].

[11] GitHub. Github - volatilityfoundation/volatility. https://github.com/volatilityfoundation/volatility. [Online; accessed 25-06-2015].

[12] Will Gragido. Understanding indicators of compromise (ioc) part i. https://blogs.rsa.com/understanding-indicators-of-compromise-ioc-part-i/. [Online; accessed 25-06-2015].

[13] HakShop. Usb rubber ducky deluxe. http://hakshop.myshopify.com/products/usb-rubber-ducky-deluxe?variant=353378649. [Online; accessed 25-06-2015].

[14] ISO/IEC. Information technology – security techniques – information security risk management, 2011.

[15] M.H. Ligh, A. Case, J. Levy, and A.A. Walters. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory.* Wiley, 2014.

[16] MoonSols Ltd. Moonsols dumpit goes mainstream. http://www.moonsols.com/2011/07/18/moonsols-dumpit-goes-mainstream/. [Online; accessed 25-06-2015].

[17] FireEye Mandiant. Redline, accelerated live response. https://www.mandiant.com/resources/download/redline. [Online; accessed 25-06-2015].

[18] Microsoft. Well-known security identifiers in windows operating systems. https://support.microsoft.com/en-us/kb/243330. [Online; accessed 25-06-2015].

[19] NetMarketShare. Desktop operating system market share. http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0. [Online; accessed 25-06-2015].

[20] Oracle. Virtualbox snapshots. https://www.virtualbox.org/manual/ch01.html#snapshots. [Online; accessed 25-06-2015].

[21] Mila Parkour. Contagiodump samples. http://contagiodump.blogspot.dk/2012/12/zeroaccess-sirefef-rootkit-5-fresh.html. [Online; accessed 25-06-2015].

[22] Guidance Software. Encase enterprise overview. https://www.guidancesoftware.com/products/Pages/encase-enterprise/overview.aspx. [Online; accessed 25-06-2015].

[23] Guidance Software. Encase forensic v7. https://www.guidancesoftware.com/products/Pages/encase-forensic/overview.aspx. [Online; accessed 25-06-2015].

[24] Transcend. Ddr sdram data transfer rate. http://www.transcend-info.com/Support/FAQ-292. [Online; accessed 25-06-2015].