# Real-time Forensics through Endpoint Visibility

Peter Kieseberg[1], Sebastian Neuner[1], Sebastian Schrittwieser[2], Martin Schmiedecker[1], and Edgar Weippl[1]

[1] SBA Research, Vienna, Austria,
[2] Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks, St. Pölten University of Applied Sciences, Austria,

**Summary.** In the course of the last years, there has been an established forensic process in place known by every investigator and researcher. This traditional process is regarded to produce valid evidence when it comes to court trials and, more importantly, it specifies on a very precise level how to acquire a suspects machine and handle the data within. However, when new technologies come into play, certain constraints appear: Having an incident in a network containing thousands of machines, like a global corporate network, there is no such thing as shutting down and sending an investigation team. Moreover, the question appears: Is this an isolated incident, or are there any other clients affected?
In order to cover such questions, this paper compares three tools aiming at solving them by providing real-time forensics capabilities. These tools are meant to be deployed on a large scale to deliver information at any time, of any client all over the network. In addition to a feature comparison, we deployed these tools within a lab environment to evaluate their effectiveness after a malware attack, using malware with pre-selected features in order to allow for a more precise and fair comparison.

## 1 Introduction

Through several years of accumulated practical experience and academic research, forensic investigators were able to establish a standardized and well-known routine for digital investigations [10, 3]. This is especially important, since relying on a common ground is critical for forensic investigations that have to back a legal trial, in order to provide soundness to claims of both sides, the defendant as well as the prosecutor (for different reasons obviously). However, there are forensic investigations that do not have the convenient features of physical access, sufficient investigation time or close to unlimited storage capacity. In times of fast growing storage capacities and even commodity hardware bringing more than two terabytes to the end-user as a USB stick, the well-established forensic process has to be re-invented. A first step to this new process has been shown by Neuner et al., who opted for a whitelisting approach that allows to exclude already known files from the acquisition process [18]. However, this still relies on the assumption that the computers which have to be investigated are physically accessible and are already (or at least can be) shut down.

Large companies such as Google, Facebook and Mozilla are challenged with the downsides of those standardized approaches. Like many other companies, they experienced several incidents [2], however they suffer from the problem of scales, having to investigate on thousands and thousands of computers. Relying on the established forensic approach and turning every computer off, making a 1:1 hard drive copy and so forth, is not only unfeasible in reality, but would cost millions of Dollars every hour [15]. Thus the three mentioned companies are developing solutions called *real-time forensic tools*, namely *Google's GRR Rapid Response* (GRR) [16, 8], Facebooks *osquery* [12] and *Mozillas Investi-Gator* (MIG) [17]. In this paper we compare these three tools with respect to their feature set and capabilities. Furthermore, we evaluate their effectiveness in a scenario where a presumable administrator detected multiple infections on different machines. Main questions answered include whether it is possible to detect the infections if the malware features are known and whether it is possible to detect every infected machine. More precisely, the contributions of our work are as follows:

– We survey the current state-of-the-art forensic approach with respect to real-time forensics.
– We compare the three real-time forensic tools regarding their features and applicability.
– We evaluate their effectiveness for a successful attack with a known malware.

The rest of the paper is structured as follows: Section 2 provides the needed background information to this work and also offers insights into the related work. Section 3 provides an overview on the three selected real-time forensic tools, but also discusses alternatives in the open-source sector as well as other commercial tools. Section 4 describes the methodology and the evaluation details of our work. This section furthermore describes the lab setup used for the evaluation as well as the selected malware and its features. Section 5 outlines the results of the evaluation. Section 6 discusses the limitations of our approach and future work in the direction of live forensics. Finally, Section 7 summarizes and concludes our paper on real-time forensics.

## 2 Background and Related Work

Forensics in a traditional sense is a standardized process - standardized in academic work [4] and by the National Institute of Standards and Technology (NIST) for law enforcement organizations such as the U.S. Department of Justice [19]. This process ensures that the investigator is carrying out reproducible steps in order to acquire a suspects data. Figure 1 [20, 21] shows a typical illustration of the process.

However, the process requires the suspect to have its data stored on a manageable number of devices, preferable only possessing low storage capacities. As shown in related work [10], storage is a very limiting factor during the acquiring process, since several copies have to be made for each device. These copies

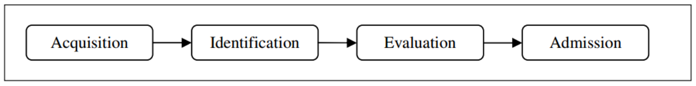


**Fig. 1.** Flow of a traditional forensic process.

include the actual working copy for the investigator, a backup copy in case the working copy is tampered and, in some cases, a copy directly sent to the client (e.g. the court). Having all these copies means a high recoverability against data loss, however, this also means that the investigator needs huge amounts of storage capacities and enough computing power to process the data for investigative tasks. This problem was already predicted before in 2010 by Garfinkel [10] and since then discussed in academic work. One suggestion in 2016 by Neuner et al. [18] is the utilization of file whitelisting of known files to reduce both, the required capacity and the required process power. Additionally there is not only academic work describing the traditional forensic process, but also suggestions by the NIST [13]. These suggestions for acquiring data include a graceful shutdown, once the volatile memory (e.g., RAM) is acquired.
Considering a typical suspect having one computer, several hard disks and a mobile phone, this traditional forensic process works well in practice. But considering modern storage techniques, like distributed storage (cloud storage), the standardized process mentioned above will not work in every detail [11]. Considering large companies such as Google, Facebook and Mozilla having an incident within their infrastructure of tens of thousands of clients, shutting down every (probably) affected computer will not work without causing huge costs to the infrastructure provider. Therefore companies like those three developed frameworks, called real-time forensic tools which do not require shutting down the client, but are able to copy important data over the network to a centralized station for further investigation. Considering an infected client, these real-time forensic tools are able to scan all clients in range for infection details to find other infected clients, with some of those frameworks being able to directly access the client and prevent further spreading of the malware, e.g. by disabling certain network interfaces. On the one hand, this approach is definitely considered tampering with the data on the client, however, on the other hand this approach is fast and does not affect the clients (or the networks) up-time. Certain frameworks (e.g. Googles GRR) are able to produce AFF4 images of the clients, which could be considered as a starting point for a forensic standard targeting live environments using real-time forensic tools. Nevertheless, it should not be unmentioned that carrying out real-time forensics is at no point compliant with any standardized forensic process as it is currently demanded by court.

## 3 Real-Time Forensic Tools

In contrast to traditional tools as outlined in Section 2 real-time forensic tools do not work upon the standardized forensic process. To tackle huge amounts of data

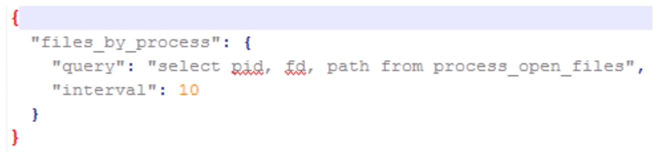
**Table 1.** Capabilities of real-time forensic tools

| | | osquery | MIG | GRR |
|---|---|---|---|---|
| File interaction | Read access on files | ✗ | ✓ | ✓ |
| | Client write access | ✗ | ✗ | ✗ |
| | File timelining | ✓ | ✗ | ✓ |
| Endpoint statistics | Host statistics (e.g. uptime) | ✓ | ✓ | ✓ |
| | Process listing | ✓ | ✓ | ✓ |
| | Connected users | ✓ | ✓ | ✓ |
| Network statistics | Users | ✓ | ✓ | ✓ |
| | Connected machines (IP) | ✓ | ✓ | ✓ |
| | Connected machines (MAC) | ✓ | ✓ | ✓ |
| Endpoint monitoring | Windows registry | ✓ | ✗ | ✓ |
| | Linux packages | ✓ | ✓ | ✗ |
| | Memory inspection (userland memory) | (✓) | ✓ | ✓ |
| Agent compatibility | Windows | ✓ | ✓ | ✓ |
| | Linux | ✓ | ✓ | ✓ |
| | MAC OSX | ✓ | ✓ | ✓ |
| | Embedded Devices (e.g. switches) | ✗ | ✓ | ✗ |
| Digital evidence acquisition | AFF4 | ✗ | ✗ | ✓ |

in real-time, without turning off the suspected computer, several tools have been developed by various companies [23]. In this section we provide insights into the three tools selected for evaluation, including an illustration of their capabilities with table 1 providing a compact overview.

### 3.1 osquery

Osquery was first released by Facebook in October 2014 as a simple way for extracting properties from a life system that can be helpful in a forensic investigation. It currently targets Linux, Ubuntu, CentOS, FreeBSD and OSX and was very recently extended to the Windows world [22]. The main idea behind osquery lies in providing an abstraction layer between the analyst and the operating system internals, allowing querying of information like changes in the file system, loaded kernel modules, information on processes and users, from a database-like structure. For this, all information is abstracted as so-called "tables" that follow the same syntax as SQLite tables and can be queried using SQL-commands.
Basically, there exist two ways of invoking osquery: Using an interactive shell called *osqueryi*, or configuring the *osqueryd* daemon. The osqueryi shell is completely stand-alone and typically used for prototyping, as well as ad-hoc analysis of the system. The osqueryd daemon on the other hand is used for structured and regular analysis of key features of the system, e.g. the list of running processes or changes in the file system. It is primarily configured by a scheduler, where defined queries are executed regularly. The daemon provides means for aggregation of these results over time and generates logs, thus can be used to easily show changes on the operating system level.
The main configuration work is done using so-called *query schedules*, SQL-style definitions of the data to be retrieved, including an interval definition for the recurring execution of the retrieval. Several queries can be packed together in so-called *packs* that allow for more fine-grained options on the logging, as well as the use of predefined packs for specific cases, including specific malware. Fig-



```
{
  "files_by_process": {
    "query": "select pid, fd, path from process_open_files",
    "interval": 10
  }
}
```

**Fig. 2.** A query schedule for osquery.

ure 2 shows a simple query schedule for retrieving all files opened by processes. The interval was set to 10 seconds, i.e. the daemon checks for these variable every ten seconds, events that take place in between will not be recorded and are lost for the analysis. Contrary to e.g. GRR, osquery is meant to be executed permanently to monitor changes on the fleeting aspects of the operating system, it is not capable of actually analyzing the actual content of files.

### 3.2 GRR

The Google GRR Rapid Response (GRR) was first announced by Cohen et. al. in 2011 and intended to handle Google's internal infrastructure regarding remote live forensics [5]. Basically, GRR is a Python agent that is installed on the clients to be managed. The GRR front-end servers, that are under the control of a system administrator (sysadmin), receive the messages sent by the GRR agents. This sysadmin initiates a so-called "flow" via the front-end server on the agents. This message contains code that is executed on the agents, which are requested to return the required information to the front-end server for aggregation and evaluation. The concept of "hunts" on the other hand describes massive amounts of flows, targeting a huge number of agents.

Most of the "basic" capabilities are built into GRR, such as file interactions, live memory analysis and endpoint monitoring. Strong points of GRR are on the one hand the possibility to manage the agent live by using an IPython shell that is capable to run on all major operating systems (Windows, Linux, OSX). On the other hand, GRR offers the possibility to extract forensic evidences using the open-source file format *Advanced Forensics File Format 4* (AFF4) [6].

### 3.3 MIG

To tackle problems like accidental private key pushes to github in a large environment (such as every computer and every server owned by Mozilla) Julien Vehent proposed Mozillas own real-time forensic tool, the Mozilla InvestiGator (MIG) [17], in 2015. MIG is written in GO and compiled into a statically linked binary for easy sharing and easy deployment. Although the binary has to be installed as a root service, activated MIG modules are locked down in terms of requested privileges. For secure communication between the clients on which a MIG agent is installed and the MIG master, Rabbit MQ is used to exchange PGP signed JSON messages. The underlying architecture is shown in Figure 3.

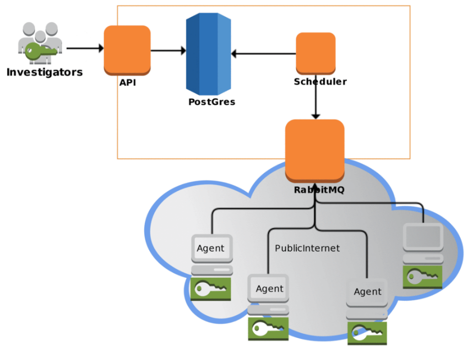


**Fig. 3.** Architecture of the Mozilla InvestiGator[1].

As soon as the agents are finished working on the tasks requested by the master, the results are sent back to the investigators and stored in a postgreSQL database. Table 1 outlines capabilities of MIG not mentioned here. The developers of MIG, besides various other features, managed to deploy MIG agents on rather restricted embedded systems like switches. This, on the one hand, adds a large amount of additional systems to be managed and analyzed, but on the other hand creates the possibility to monitor and protect these kinds of systems.

### 3.4 Commercial Solutions

Besides open-source real-time forensic tools there are also commercial tools available. This includes Mandiant's MIR, Encase Enterprise, as well as the real-time forensic tool of F-Response. These frameworks could not be evaluated due to the limited availability of the software, e.g. demo versions, however, even if a demo would be available for all of these commercial frameworks, they are typically limited and therefore cannot be compared to fully fledged open-source solutions.

## 4 Methodology

### 4.1 Lab Setup

Figure 4 depicts the setup of the lab environment used for the evaluation of the real-time forensic tools. As a first step (1), the control panel prepares the malware that is subsequently sent to the virtual machines. The malware can be chosen based on a range of pre-classified features (see Section 4.3 for details on the selection for our work). Step (2) initializes the VMs for a first use.

In our case this includes the installation of the operating system Windows 7 Service Pack 1 (64 bit) for GRR and Windows 10 Pro (64 bit) for MIG and

[1] Image source: `http://mig.mozilla.org/doc/.files/mig_workflow.gif`. Accessed: 13.09.2016

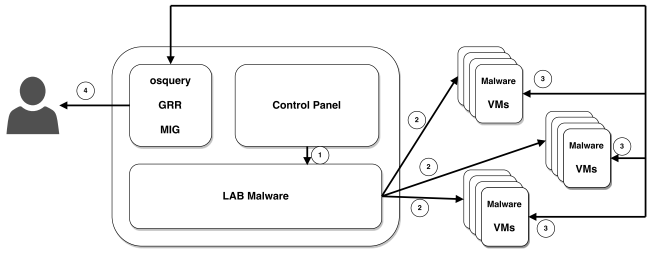


**Fig. 4.** The lab setup used for evaluating the real-time forensic tools.

osquery, as well as an agent corresponding to all three real-time forensic tools we are evaluating. In step (2) the malware is loaded onto the machines, enabling certain types of malware to infect the virtual machine at boot time or at time of the start of the operating system, respectively. Step (3) is bi-directional: The real-time forensic tools are polling for data, which results in data sent to the infected virtual machines. The way the data is sent (methods, protocols used) depends on the communication techniques of each real-time forensic tool. As soon as the data is available for each tool it is evaluated and made available for the investigator in step (4).

## 4.2 Malware Sample Selection

Based on the methodology described in Section 4, the malware used for evaluation was selected on the following feature set:

Feature (F1), Process spawning: Malware is often running as processes in the background in order to carry out their malicious activities. However, the names of those processes are often either publicly known or easy to spot [14].

Feature (F2), Persistence: Certain kinds of malware persists themselves on the system, either somewhere on the filesystem but also e.g. in the registry. Persistence ensures the malware staying on the system after a reboot, as well as the possibility to restart the malware process after manual termination [1].

Feature (F3), Network connection: Processes that start outgoing, as well as accept incoming connections without any user interaction, are often malware [24]. Outgoing connections can indicate data that is being exfiltrated or the establishing of a connection to a botnet server (Command and Control server) [9]. Incoming connections can indicate patching of the malware or dropping additional payload on the attacked system [7].

Therefore, the following samples of malware have been selected based on the feature list above: Sample (S1) containing the banking trojan *retefe*, a malware that installs a root CA on the infected machine and starts to intercept e-banking connections, sample (S2) containing the *Locky* ransomware that encrypts files on the user's hard disk for asking for ransom for the decryption key, as well as sample (S3) containing the *Win32.Viking* worm.

Feature *F1* is fulfilled by all of the three samples *S1*, *S2* and *S3*. Each of them

spawns several processes, some running in the background in order to carry out the malicious behavior. These processes include notoriously dangerous executables like "powershell.exe", "certutil.exe" and "tor.exe". All malware samples persist themselves on the system, more precisely the file system, fulfilling feature *F2*. Finally, feature *F3* is also fulfilled by all three samples to a varying degree: While the banking trojan does open several connections, the Win32.Viking worm works much more stealthy.
Sample S2, the Locky ransomware, was also chose, because it possesses a specialty: Contrary to other malware like ebanking trojans, ransomware stays hidden only for a specific time, until enough user files (or even the whole disk, depending on the actual malware) have been encrypted. Then the malware actually informs the user in order to make him/her pay the ransom. Thus, the detection capabilities evaluated in our scenarios are evaluated with respect to the "dormant" ransomware, i.e. the ransomware before or during the encryption phase, since its presence afterwards, in the ransom phase, is detected trivially.

### 4.3 Evaluation

The goal of the evaluation was to study the behavior and detection possibilities of the three live forensic tool-kits under real-life conditions. To this end we selected three malware examples and tested them on a system. Furthermore, we had a look on the capabilities of the different tool-kits and extrapolated their typical applicability in real-life scenarios.
For the evaluation, we infected a running system, with each malware separately, in order to get a good comparison of the results. While this evaluation yields good results for the detection of malware with known or at least expected feature, it does yield the problem that many artifacts are quite typical for the malware in question. Thus, we concentrated on utilizing the artifacts for detection that are more uncommon, like changes to system routines, changes to specific keys in the registry, or spawning of suspicious processes.

## 5 Results

In this section we provide a comparison of the analyzed tools with respect to our research scenario and outline major differences, as well as shortcomings based on the three malware samples selected before.

### 5.1 osquery

Osquery mainly targets the monitoring of operating system internals, i.e. it is a constant monitor of the system state and does not target the reconstruction of deleted files. Regarding the banking trojan retefe, this helps detection in case of continuous monitoring through the osqueryd daemon. Here, the following artifacts could be found that identified this malware. It has to be noted though that the malware does generate many more artifacts, we reduced our analysis to those issues that possess high significance. This also implies that we did not concentrate on artifacts that can be the result of arbitrary other programs running on the respective machine like memory usage or checking for needed third party software:

– For file interaction, osquery is capable to detect the changes done to the file system on a pure metadata level. The malware generates and changes several files in the AppData directories for Microsoft Office and Tor, using file names like "Microsoft.Win32.Task Scheduler.dll", as well as the TOR-AppData.
– On the endpoint statistics level, it created various process, the most notorious including instances of "powershell.exe", "certutil.exe" (for adding the root CA) and "tor.exe".
– Regarding the network level, it generates various connections to the outside world, which can be detected by constant querying of the respective interfaces using osquery.
– On the endpoint monitoring level, there is a change happening to the Windows registry, deleting and recreating a specific key "HStartupItem" for MS Office and creating several other keys. Furthermore, a new root CA is installed, also resulting in the respective changes in the registry. Altogether, the process changes the registry which can be detected by osquery.

For the Locky ransomware, we detected the following artifacts that indicate an infection with malicious software:

– On the file level, it generates an executable in the system32 directory of Windows, as well as a file containing decryption instructions. Furthermore, as soon as the ransomware process is started, in starts accessing and updating files and changing their names to the ".locky" suffix.
– On the endpoint statistics level, it generates some processes, with "cmd.exe" being the most notable.
– Regarding the network level it does some DNS-lookups and downloads executable code during the infection. This of course is only visible in osquery in case the malicious code is not already downloaded before. Furthermore, it opens a connection to the well-known Locky distribution site "greenellebox.com". In addition, it uses a known web browser user agent for HTTP communication, which can be filtered using osquery.
– On the endpoint monitoring level, while of course activity was shown that can be attributed to the infection, there was nothing outstanding recorded that enabled us to identify the infection with Locky with a high certainty, while, of course, the randomly generated key in the registry was visible and could be a starting point for further investigations.

Regarding the Win32.Viking worm, we detected the following artifacts that indicate an infection with malicious software:

– On the file level, it generates the dll-file "FastUserSwitchingCompatibility.dll" in the system32-directory, as well as deletes a file in this directory. Furthermore, it generates a randomly named file in the root directory (typically "c:").

- While only spawning a few processes, these include several instances of "reg.exe" for modifying the registry, as well as a (changed) instance of Internet Explorer.
- On the network level, this malware is invisible to osquery, as no direct network connections are opened, but the (modified) Internet Explorer is used for hiding the communication.
- On the endpoint monitoring level, the malware makes changes to the registry by adding a new key and creating a Windows Service pointing to the executable "FastUserSwitchingCompatibility.dll".

## 5.2 GRR

The main benefit of GRR is its capability to check actual file content and search for strings that can be attributed to known malware samples. Furthermore, it still allows for file timelining and looking for changed files in the overall OS structure. This also holds true for the analysis of running processes. Still, the typical idea of GRR, in contrast to osquery, does not lie in the permanent observation and monitoring of the system looking for changes that might hint at an infection, but more on analyzing a system suspected for an infection already having taken place. Regarding our first sample, the banking trojan retefe, the following artifacts can be detected:

- Regarding changes to the file system, GRR is capable to detect the changed files in the AppData directories for Microsoft Office and Tor. Furthermore, the docx-document used for infection contains several deviations to typical docx-files like irregular field values in the summary information. It also contains a stream with embedded javascript code. This is especially valuable, as it helps to reveal the actual source of the infection.
- GRR is capable of detecting the processes spawned by the malware, still, since GRR is typically used as ad-hoc tool in the course of an investigation and not as constant system monitoring, it might miss most of these processes.
- The same holds true for the networking level. Since a banking trojan is meant to be active regularly in order to intercept the e-banking connections, GRR can be used for detection.
- The same holds true for the connections on the network level, especially since relevant information on the connection parameters can be extracted from the infected file, thus giving a valuable hint on what to look for.
- Finally, GRR is perfectly capable on extracting the changes that happened to the registry, the recreated "HStartupItem" key, as well as the root CAs.

For the Locky ransomware, the capabilities to check the actual file content are especially valuable. Furthermore, the following artifacts were be used for detecting this infection:

- GRR is capable of detecting the files generated by the malware, especially the executable in the system32 directory of Windows. Furthermore, since the timeline of the files is accessible by GRR, arbitrarily changed files become

visible. In addition, the file system can be checked for files that should be readable (e.g. Office files), but only contain gibberish, hinting at encrypted data. Furthermore, the statistics also reveal the suffix changes. In addition, specific URLs can be found in the documents, as well as a dropped file, where the content does not match the file extension.
- While the encryption is taking place, GRR is capable of detecting the respective processes, especially running an executable with a randomly generated name from the local temporary directory (e.g. "b7uG0vk9g4qsBc5Z.exe").
- Locky contacts the distribution site "greenellebox.com" which can be detected using GRR during the connection. Furthermore, GRR could detect the known web browser user agent used for HTTP, in case Locky communicates during the investigation, still, the ransomware is typically limiting itself to small amounts of communication.
- GRR is also capable to see the randomly generated key in the registry, still, we found it rather hard to detect Locky solely by this artifact, especially in the presence of the much more distinctive artifacts on the file level.

Also with respect to the Win32.Viking worm, the capability to search for the content inside files helped a lot:

- The generated dll-file in the systems32-directory can be detected easily. Furthermore, it generates an executable with a random name in the root directory "C:" that contains search strings for anti-malware evasion.
- GRR was capable of detecting the spawning of the "reg.exe" command for editing the registry, if this is done during the investigation.
- While in theory GRR should be capable to see the network channel opened by using Internet Explorer, we were not able to detect this in our example environment using GRR.
- GRR is perfectly capable to detect the changes to the registry by adding a new key and creating a Windows Service pointing to the executable "FastUserSwitchingCompatibility.dll".

## 5.3 MIG

The MIG framework proposed by Mozilla, like GRR, is used in ad-hoc investigations and not for permanent monitoring. Furthermore, like GRR, it is also capable to provide read access to the actual content of files. Still, since the main goal was to tackle the problem of accidental pushes of information, it does not allow for file timelining, somewhat limiting the detection capabilities compared to GRR. Thus, in this section, we will mainly outline the differences to GRR.
With respect to the *retefe* banking trojan, the following artifacts could be observed in the lab environment:

- While it is possible to analyze the actual contents of the files, the detection of actually changed files is harder due to missing file timelining. Still, the detection is possible, especially when routinely looking for the ill-formatted docx-files.

- One major drawback for the detection, is the incapability to access the Windows registry, as the tool misses the recreated "HStartupItem" key, as well as the root CAs. This information is especially valuable, as it is (i) far more specific for this malware, (ii) typically not the effect of an user error (like badly formatted docx-files could be) and (iii) very simple to spot.

Still, even though the Windows registry could not be accesses, MIG is capable to detect the malware based on the other characteristics. For the Locky ransomware, the picture looks almost the same:

- Having no file timelining seems a bit problematic for getting the best picture on the changes taking place in the overall file system, still, we were able to detect the malware,
- Again, having access to the randomly named keys in the registry would add to the analysis.

For the Win32.Viking worm, the following artifacts were especially useful:

- Since the filename that is generated is known, and the malware generates an executable holding search strings for anti-malware evasion, we were able to detect it using MIG.
- Again, accessing the Windows registry would have helped a lot finding the Windows Service created by the malware.

It must be noted though that MIG is the only product of the three evaluated approaches that can be used for embedded devices, thus possesses a feature that must be taken into account as especially interesting in other real-life scenarios.

## 6 Limitations and Future Work

In terms of limitations of our evaluation, there is clearly the limited number of deployed clients. This accompanies one of our targets for future work: For follow-up work we plan to contact companies such as Google, Facebook and Mozilla to share their insights after several months (or even years) of deployment and execution of those real-time forensics tools. This would provide data from real-world deployments rather than from a lab environment, allowing to answer research questions like statistics on the typical time between detection and cleanup of a certain incident, or on the most commonly experienced incidents, and so on. However, the lab environment is indispensable without having access to the company data.
In case we do not get access to the requested data, a fallback plan is to imitate a large network by deploying thousands of cloud instances, running the real-time forensic tools. This would also provide insights into the long-term applicability of the evaluated tools. Additionally, this would bring shed light onto the effectiveness of the tools, e.g. in terms of time: How long does it take from detection until cleanup of a given incident?
Among further future work planned, we also intend to deploy more open source

real-time forensics tools (or at least freeware tools) on the cloud instances mentioned. This would extend the insights on different frameworks, but would raise the problem of increasingly unmaintained frameworks.

## 7 Conclusions

In conclusion, all the tools reviewed in this work were able to detect the samples, still the artifacts most probably used seem to differ. In the selected examples, especially MIG's incapability to check the Windows registry was noted, as this would offer a lot of additional capabilities. Still, it must be noted that MIG is capable of dealing with embedded systems, which is an additional benefit worth noting. From the point of view of usage, the use of osquery differs quite a lot from GRR and MIG: While GRR and MIG are made to be used during an investigation, i.e. at a specific point of time after e.g. an infection was suspected, osquery, while offering this capability too, is typically configured to automatically monitor the system based on different attributes and artifacts that are prepared to be queried like tables. Still, on the other hand, it does not offer the user the possibility to check actual data on the file system, especially reconstructing deleted files and checking for search strings inside suspicious files.
In conclusion, we would recommend to use a mixed approach by having the osquery daemon permanently monitoring a selection of artifacts, especially the process list, changes to the file system and changes to the Windows registry, as well as using either MIG or GRR for getting into the issue of file checking in case new and suspicious file generation or changes are detected by the monitoring. For choosing between GRR and MIG, this mainly depends on the system at hand. In case of a Windows system, GRR outperforms MIG due to its capabilities of file timelining and accessing the Windows registry. On the other hand, in case of a more complex system structure including embedded systems or low-end hardware, MIG is simply capable to generate a much more complete picture, as information from these sources can be incorporated into the analysis.

## Acknowledgements

The financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged.

## References

1. Syed Nasir Alsagoff. Malware self protection mechanism. In *2008 International Symposium on Information Technology*, volume 3, pages 1–8, 2008.
2. Eric Auchard. Major security breaches found in google and yahoo email services. Accessed: 13.09.2016.

3. B. Carrier. *File system forensic analysis.* Addison-Wesley Professional, 2005.
4. Eoghan Casey. *Digital evidence and computer crime: Forensic science, computers, and the internet.* Academic press, 2011.
5. MI Cohen, Darren Bilby, and Germano Caronni. Distributed forensics and incident response in the enterprise. *digital investigation*, 8:S101–S110, 2011.
6. Michael Cohen, Simson Garfinkel, and Bradley Schatz. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. *digital investigation*, 6:S57–S68, 2009.
7. Paolo Milani Comparetti, Guido Salvaneschi, Engin Kirda, Clemens Kolbitsch, Christopher Kruegel, and Stefano Zanero. Identifying dormant functionality in malware programs. In *IEEE Symposium on Security and Privacy.* IEEE, 2010.
8. Flavio Cruz, Andreas Moser, and Michael Cohen. A scalable file based data store for forensic analysis. *Digital Investigation*, 12:S90–S101, 2015.
9. David Dittrich and Sven Dietrich. Command and control structures in malware. *Usenix magazine*, 32(6), 2007.
10. Simson L Garfinkel. Digital forensics research: The next 10 years. *digital investigation*, 7:S64–S73, 2010.
11. Hong Guo, Bo Jin, and Ting Shang. Forensic investigations in cloud environments. In *Computer Science and Information Processing (CSIP), 2012 International Conference on*, pages 248–251. IEEE, 2012.
12. Facebook Inc. osquery performant endpoint visibility. Accessed: 13.09.2016.
13. Karen Kent, Suzanne Chevalier, Tim Grance, and Hung Dang. Guide to integrating forensic techniques into incident response. *NIST Special Publication*, 2006.
14. Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *USENIX security symposium*, pages 351–366, 2009.
15. Polly Mosendz. Lets calculate how much money facebook just lost during todays outage. Accessed: 13.09.2016.
16. Andreas Moser and Michael I Cohen. Hunting in the enterprise: Forensic triage and incident response. *Digital Investigation*, 10(2):89–98, 2013.
17. Mozilla. Mig: Mozilla investigator. Accessed: 13.09.2016.
18. Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. Effectiveness of file-based deduplication in digital forensics. *Security and Communication Networks*, 2016.
19. National Institute of Standards, Technology (NIST), and United States of America. Forensic examination of digital evidence: A guide for law enforcement. 2004.
20. Mark Pollitt. Computer forensics: An approach to evidence in cyberspace. In *Proceedings of the National Information Systems Security Conference*, volume 2, pages 487–491, 1995.
21. Mark M Pollitt. An ad hoc review of digital forensic models. In *Systematic Approaches to Digital Forensic Engineering, 2007. SADFE 2007. Second International Workshop on*, pages 43–54. IEEE, 2007.
22. Ty Sereyvathana. Osquery: Cross-platform, lightweight, and performant host visibility. In *7th Annual Open Source Digital Forensics Conference (OSDFCon)*, 2016.
23. Meir Wahnon. awesome-incident-response: all-one-tools. Accessed: 13.09.2016.
24. Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 116–127. ACM, 2007.