Seminar Report

Incident Response and Forensic Tools

Dominik Mann

MatrNr: 20412403

Supervisor: Arthur Wachtel

Georg-August-Universität Göttingen
Institute of Computer Science

March 26, 2023

Contents

List of Figures ii

Listings ii

List of Abbreviations iii

1 Introduction 1

2 Theoretical Incident Response 2
2.1 Threat Intelligence 2
2.2 Incident Response Plan 3

3 Practical Incident Response 4
3.1 Forensic Image 4
3.2 Memory Image 7
3.3 Timestamps in Digital Forensics 8
3.4 Hashset 9
3.5 Live Incident Script 10

4 Scaling live forensics 13
4.1 Velociraptor Deployment 13
4.2 Incident Response with Velociraptor 14

5 Conclusion 15

References 17

A Image creation A1

B Evidence A1

C Timestomping A3

D Hashsets A4

E YARA Rules A4

F Velociraptor A5
F.1 Installation A5
F.2 Artifacts A6

List of Figures

1 SANS IR Plan 4
2 Suspicious Files A7
3 Last SSH logins A9

Listings

1 Forensic Image Mounting 6
2 Forensic Image Creation A1
3 Output of `ss -anepo` here already filtered for truncation. It illustrates the benefit of filtering for IOCs in speeding up system analysis as well as the importance of understanding normal system behavior in such a large log. . A1
4 Output of `/var/log/secure` showing ssh login attempts. A1
5 Output of `unhide -v quick checksysinfo` showing a hidden process that executes `netcat` indicating a possibility for attackers to issue commands through the `netcat` connection. A1
6 Listing of all users in the superuser group. A2
7 Output of `lsof` here again filtered filtered for an IOC (can also be a malicious user with parameter `-u`). It shows the open files created by the `netcat` command. It additionally shows the location of the binary. A2
8 Output of `lsmod` showing the malicious rootkit *Diamorphine* (Output truncated). A2
9 Output of `modinfo diamorphine` showing metadata and location of the rootkit. A3
10 Output of `lsof | grep deleted` that shows that this process uses a deleted binary of `netcat` and is still running. A3
11 Output of `find /usr/bin/ -mtime -90` that shows all modified files in the last 90 days in directory /usr/bin. Among other things A3
12 Common timestomping methods to alter the timestamps of a file. A3
13 Commands to create a hashset of binaries. These commands can be repeated for the Rocky 9 operating system. A4
14 Exemplary YARA rule [30]. A5
15 Exemplary Velociraptor installation. A5
16 Exemplary Velociraptor artifact to detect suspisious files [31]. A6
17 Exemplary Velociraptor artifact to detect last logins per SSH [32]. A7
18 Exemplary live investigation after user account breach. A9

List of Abbreviations

AVML	Acquire Volatile Memory for Linux
BKA	Bundeskriminalamt
CA	Certificate Authority
CKC	Cyber Kill Chain
HPC	High-Performance Computing
IR	Incident Response
IOC	Indicators of compromise
VQL	Velociraptor Query Language

1 Introduction

Cyber crime is a critical issue with increasing number of cases. By shifting to hybrid office models following the COVID-19 pandemic, the attack surface was widened, and as a result, the number of cyber-attacks targeting organizations and corporations increased [1]. The FBI's Internet Crime Complaint Center reported that during the COVID-19 pandemic, complaints in the United States increased from 1,000 to 4,000 complaints per day [2]. In addition, the German Bundeskriminalamt (BKA) registered 85.960 new cybercrime cases in 2017 that rose to 124.137 cases in 2021, increasing by 44% alone in Germany [3]. A new study by the anti-virus specialist McAfee estimates the monetary loss by cybercrime to be $945 billion and an additional $145 billion for investment in cyber security measures, totaling in an estimate of over $1 trillion costs for the global economy [4]. Especially High-Performance Computing (HPC) clusters are particularly promising. In HPC clusters, hundreds of individual computers contribute to substantial computing performance, which allows cybercriminals to exploit these systems for malicious purposes, such as mining cryptocurrency or maintaining botnets. That these are not only theoretical considerations show the attacks on HPC clusters in Europe [5] and Norway [6]. While HPC systems are also susceptible to many of the same attack vectors used against conventional IT systems, they also possess their own vulnerabilities due to exotic hardware and software that are implemented in these systems [6]. Since HPC clusters are accessed by a large number of users and security patches are often delayed, they require extra attention when it comes to security.

Typical attack vectors for HPC systems involve users not protecting their passwords or using passwords that are easily exploitable, such as '*password123*'. A attacker may then attempt to escalate their privileges by triggering a local root exploit and elevating their privileges during the escalation. From this foundation, attackers may exploit other systems that are not open to the public network and attempt to conceal the path of their attack. Because of this, detection may be delayed and may only occure due to a significant difference between the behavior of hackers and that of the account owner [7].

The need for incident response (IR) plans is essential for providing fast and effective solutions to such occurrences. The implementation of a organized and well-established IR framework will assist in countering the steady trend of attacks. The framework should ensure that the organization's information assets are confidential, trustworthy, and available, as well as guide investigators in a predefined manner so that a standard procedure is established [8]. The purpose of this research is to address exactly this niche within the IT security field.

The report is based on the assumption that an attacker has already compromised a legitimate user account as a result of an attack. Therefore, the methods presented will be based on this scenario. However, IR may also start with endpoint protection alerts or the detection of security gaps in the environment. This work will give recommendations regarding IR procedure, highlight possible pitfalls and present appropriate tools for analysis. This study will not examine the actual digital forensic analysis process associated with investigating malicious assets, and will therefore only briefly touch on this subject when appropriate.

The paper starts with section 2 describing theoretical frameworks of attack and defense behaviour. Furthermore, section 3 presents the practical side of IR by introducing disk and memory image acquisition and the importance of timestamps in a forensic analysis.

The section also includes descriptions on the incident response script that was developed for this project. In section 4 we will talk about how incident response can be scaled to include characteristics of HPC environments. We conclude the paper with a brief summary in section 5.

2 Theoretical Incident Response

This section should present a theoretical foundation for incident response procedures and provide insight into the way defenders approach an incident. There will be two key concepts introduced in this article that are important in understanding the process of planning and executing IR playbooks.

2.1 Threat Intelligence

Prior to discussing practical applications, it is necessary to review the theoretical foundations of IR in order to gain a thorough understanding of why some practical procedures are carried out. First we will look onto the attackers side and introduce the *kill chain* before reviewing the defenders side and talking about the six-step IR plan.

To detect and mitigate intrusions, it is essential to understand the attacker's perspective so that a plan for upcoming events can be created and appropriate measures can be taken. Consequently, intrusion analysis approaches have been developed to map and illustrate attacker behavior during an attack. Over the years, three models have distinguished themselves from the rest, namely the *Diamond Model* [9], the *Cyber Kill Chain* (CKC) [10] and the *MITRE ATT&CK* framework [11]. In this work, we will focus on the CKC as it describes high-level adversary objectives that are easier understandable in an introduction into cybersecurity and IR without losing important information (for a deep level complementary see [11]). In practise however, combining the models can provide even deeper understanding of attacks and is commonly proposed by the authors of the models [9], [11].

Based on military doctrines, the CKC outlines the stages an attacker must pass through in order to accomplish their objectives. The CKC consist of the following individual steps:

1. **Reconnaissance**: Gather data on target organization (E-Mail, IP addresses, open ports, etc.).
2. **Weaponization**: Craft malicious payload and bundle it with backdoor.
3. **Delivery**: Delivering weaponized bundle to the victim through open security gaps.
4. **Exploitation**: Exploit a vulnerability to execute code in victim's system.
5. **Installation**: Installing malware and obtain credentials.
6. **Command Control**: Set up command channel for remote manipulation of system.
7. **Actions on Objectives**: Obtain ultimate objective and persistence.

Essentially, an intrusion is a process by which an aggressor develops a payload to breach a trusted boundary, establish presence within an environment, and then directly act upon that presence in order to achieve their objectives. Each step in the kill chain must be completed successfully by the adversary in order to penetrate the network. Deviation from these steps will result in a disruption on the achievement of the desired objective [10]. In contrast, a defender can use kill chain analysis to determine which information is available for a defensive response. In the event that analysts find indications that adversaries have achieved a certain stage, they may assume they have also completed the previous stages and search for indicators of compromise (IOC) in order to prevent future breaches. Additionally, incident responders have a guideline to follow when preparing for an incident. When receiving a report of a new attack method (such as OpenIOC[1] or MISP[2]), a defender is able to determine what kinds of IOCs or attack techniques would be reasonable to be produced or employed when attacking a specific system [12].

2.2 Incident Response Plan

Following our examination of the incident from the perspective of the attackers, we will move on to the perspective of the defenders.

Every security incident is unique in its nature and IR should adapt

accordingly. To give incident handlers a direction while investigating, the IR process should be guided by a general framework, yet it must remain flexible throughout. In practise, two IR plans stand out, the NIST Incident Handling Guide [13] and the SANS Incident Response Plan [5]. When compared side-by-side, NIST groups containment, eradication, and recovery from a security incident in one phase, which means that these three steps take place simultaneously, while SANS separates them. In this work, we will focus on the SANS IR Plan, as separating each phase into individual steps will provide a thoroughly understanding of the concept. Additionally, the concept incorporates the dependence among phases, as each step can have an impact on previous steps when new IOCs are encountered. As we will see, a recovered system is monitored and if malicious behavior persists, the system will be contained again [5].

As visualized in Figure 1, the response plan can be divided into six steps that follow a linear sequence. The definition of the steps are as following [5]:

1. **Preparation**: A team should be able to respond to incidents in a timely manner. Adequate tools and equipment are provided and sensitive assets are identified.

2. **Identification**: In this step, deviations from normal behavior are evaluated by gathering telemetry such as log files, error messages and other resources. When an incident is discovered, the IR team will be notified and the event is classified in its severity.

3. **Containment**: This phase is intended to limit the extent of current damage and prevent further damage from occurring by isolating affected servers from the network in the short term. Additionally, a forensic image of the system is created to preserve important evidence. In the long term, accounts and backdoors left over from the attack will be removed and security patches will be installed to all systems.

[1]https://www.mandiant.com/resources/blog/openioc-basics
[2]https://www.misp-project.org/

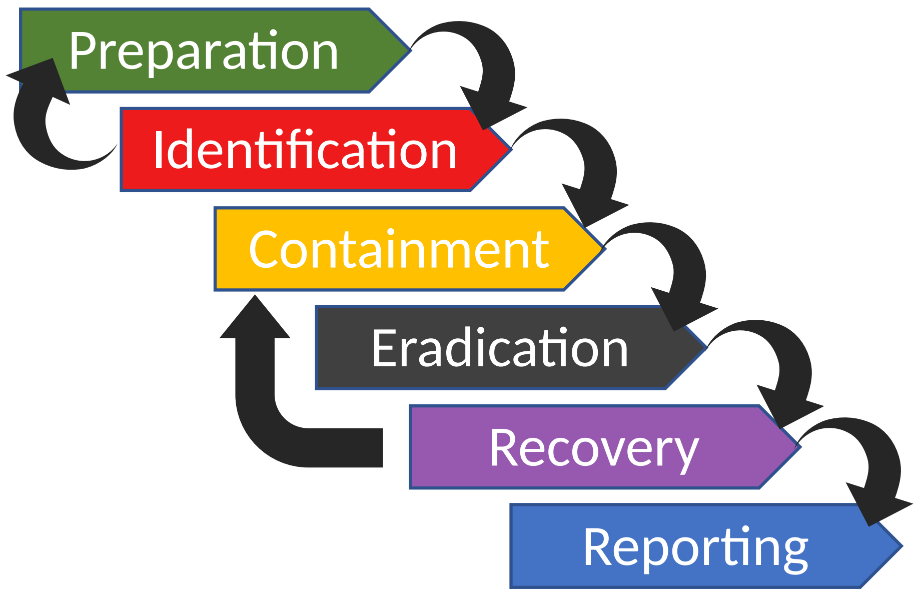


Figure 1: The SANS Incident Response Plan (based on [14])

4. **Eradication**: In this phase, the actual root cause of the malicious incident will be removed, as well as any other illicit content on the affected system. Moreover, improvements will be incorporated based on the learned attack vector. At this stage, a system should be fully operating again.

5. **Recovery**: During this phase, affected systems will be carefully reintroduced into the production environment. In order to prevent a recurrence of the incident due to same causes, tests and monitoring of affected systems are essential. If the system shows abnormal behavior again, it will be put back into the containment phase.

6. **Reporting**: Concluding, lessons learned from the incident are being discussed and documentation of the event is completed. An informative report should be able to answer the Who, What, Where, Why and How questions. Overall, the goal is to learn from incidents within an organization in order to improve security as well as serve as a reference for subsequent incidents.

3 Practical Incident Response

After examining the theoretical aspects of IR, we shall now discuss practical considerations of the IR process. As there are numerous characteristics that need to be taken into account, this section provides valuable insights into important IR characteristics and provides guidance through the incident investigation process.

3.1 Forensic Image

For incident response analysts, it is essential to have a solid understanding of the facets of forensic imaging. In order to ensure proper evidence handling, an analyst must be familiar with tools, techniques, and procedures. This will ensure that the evidence is handled properly and that the evidence can be trusted [12]. In order to preserve data in its original state, the hard drive of a system needs to be copied for analysis. It is advised to create a master copy and a working copy. Additionally, all steps taken to record a copy should be documented [15]. The incident handler may then analyze the disk without having to worry about metadata changes (timestamps, permissions, etc.) or that data may be altered. In such a scenario, the analyst can always go back to the master copy to restore it to its original state. In every scenario, the analyst should at least document the hash of the original disk and the hash of the copy. These two hashes should be identical and ensure that the disk content is not altered.

Luttgens et al. [16] distinguishes between three types of forensic images - the *physical image*, a *partition image* and a *logical image*. A physical copy of a disk is a bit-by-bit copy of every addressable allocation unit on the original disk. Therefore, a storage medium the same size or larger is required. Advantages of the physical image are the capturing of all data on a drive, consequently spaces like file slack space (unused space reserved for a file; it may contain residual data such as portions of deleted files) and unallocated space is captured. An examiner is able to access all these blocks, being able to recover deleted files in unallocated space or access hidden data in slack space. Although a physical disk image is nearly always preferred because of its all-encompassing nature, all three images have their purposes. Which image type is selected depends on the specific use case. A partition image refers to a bit-by-bit copy of a single partition of a disk that illustrates a subset of a physical image. When the indicators originate exclusively from a particular partition, it can provide significant time saving during the imaging process. A logical disk image is a simple copy of active data on a partition of a hard drive. It neglects spaces that are not occupied with data, and therefore the possibility of file carving is not provided. A logical image, on the other hand, allows the investigator to quickly scan the contents of a hard drive for files and folders relevant to the investigation and can save the investigator a substantial amount of time.

In practise, a hard drive should always be attached to the forensic workstation through some kind of write blocker, ideally a hardware write blocker, to prevent accidentally altering data while imaging. If the hardware is not available, the investigator should disable the auto mount option of the workstation. After connecting the hard drive with the workstation, the block device of the attached disk can be printed with the `dmesg` command. To ensure file integrity, it is advised to calculate the hash sum of the disk before and after creating a image. To create an image of the disk, several methods via software are available. Common command line tools are `dd`, `dcfldd`[3] and `dc3dd`[4]. The advantage of `dd` is that it is available on nearly every Unix system. Due to the fact that most Unix-based systems display hard disks and other hardware as files, it is possible to copy them forensically sound using the `dd` command [16]. Drawbacks of `dd` are the inability to calculate checksums of the image and the lack of user feedback during the image process. Therefore, enhanced versions of `dd` where developed by the U.S. Department of Defense Computer Forensics Laboratory (`dcfldd`) and the Defense Cyber Crime Center (`dc3dd`). They enable writing logs, using hashing algorithms, file splitting and provide progress bars to users. Listing 2 provides sample commands for creating an image with these tools. The hash of `/dev/sdb` is calculated for verification after image creation. The commands will produce images based on the disk provided in `/dev/sdb` with a block size of 4096 bytes per block. The two advanced methods will additionally calculate the `SHA-256` hash of the output image and automatically verify the integrity. `dd` and `dcfldd` additionally can be

[3]sourceforge.net/projects/dcfldd
[4]sourceforge.net/projects/dc3dd

provided with the `conv` argument, specifying that a corrupt block should be padded with nulls to preserve the full length of the original data and `noerror` ensures the continuity after such an error occurred. To created an image remotely, one can use `netcat` to open a connection between the compromised system and the workstation and executing the command.

As all of the aforementioned methods produce a raw image format, these images can be used with nearly all Unix and forensic tools, however, they are large and cumbersome to work with. To provide solutions to these drawbacks, additional image file formats were developed. A common file format used is the *EnCase Evidence File* or short `.e01`. Like `dd` and its descendants, it uses a physical bit stream of the data to produce an image. However, the file format supports compression and contains metadata that stores information about the drive type, operating system and timestamps of image creation. Moreover, a key feature is the *Cyclical Redundancy Check* that ensures file integrity by verifying each 64 KB block of data over the entire image and additionally supports `MD5` hashes [12]. This type of image can be easily created with the AccessData FTK Imager[5] or Guidance Software EnCase[6] that provide a GUI to guide through the process of creating an image.

Another option is to use a boot disk with a pre-installed operating system such as *CAINE*[7]. The system should be booted from the provided boot device. A target source (USB-Stick or another drive) for saving the image should be mounted as read-write. CAINE provides easy management of partitions with the pre-installed software `Mounter`. After that, the disk can be imaged with the software of choice (dd, dcfldd, Guymager, EnCase, etc.[8]; an extending description can be found in [17]).

To mount a raw image, it is important to remember that an image should always be mounted read-only. This is achieved by providing the `ro` mount parameter. Additionally, the `noload` parameter should be passed as the manual for mount states that a system may still try write to a device by replaying the journal if the filesystem is dirty. The `noload` option prevents this kind of behavior. In some cases, when the disk contains a complex partition table, an offset needs to be specified that is calculated by multiplying the block size with the starting value of the specific partition that needs to be analyzed (this can be located with `fdisk -l` or `mmls` from the Sleuthkit). Assuming that the a block is the size of 512 bytes and the partition to analyze starts at block 2048, a typical command can look like:

```
# 512*2048=1048756
$ mount -t ext4 -o ro,noload,loop,offset=1048576 image.dd /mnt/evidence
```

Listing 1: Forensic Image Mounting

As an alternative, the partitions can be mounted automatically as loop devices with the command `losetup -Pr image.dd` and then the loop device can be mounted to the desired mount point.

To mount an `.e01` image, the `ewfmount` tool from the `ewf-tools` package is needed. It mounts the `.e01` file automatically as read-only. As the file format acts as a container, the underlying raw image can be mounted with the former methods.

To analyze an image, no special forensic operating system is needed. However, distri-

[5]www.accessdata.com

[6]www.guidancesoftware.com/forensic

[7]https://www.caine-live.net/

[8]NIST provides a catalog for forensic tools (https://toolcatalog.nist.gov/search/index.php?ff_id=1)

butions like Kali Linux [9], CAINE, or SIFT[10] provide a range of preinstalled forensic tools and activated settings to ensure forensically sound operation, including auto-mounting disabled and extended file system support.

3.2 Memory Image

A disadvantage of focusing exclusively on offline acquisitions is the loss of volatile data that only exists in memory. The volatile data contains important evidence such as network connections, running processes, open files, login sessions, etc., all of which are lost when the system is shut down, particularly in HPC systems where the system often resides in the memory rather than on the hard drive. Especially systems compromised with malware or platforms such as MetaSploit, evidence of the malicious software is only contained within the memory. Moreover, the creation of a hard drive copy of every system takes time, however, in the case of an active incident time is a crucial factor. [12].

An important characteristic of memory acquisition is that alteration of RAM while attempting to create an image is unavoidable. Since the system is still running, data in the memory is constantly moving and changing. The act of simply looking at the contents of memory alters the contents of the memory [18]. In addition, executing a tool to collect the memory writes to the RAM. In Linux systems, memory can typically be accessed through the `/dev/mem` and `/proc/kcore` files. The first file contains raw physical contents of memory, the second contains the same contents but in the core file format [18]. The script that will later be introduced uses the open-source tool *Acquire Volatile Memory for Linux*[11] (AVML) written in Rust and released by Microsoft. AVML is capable to acquire a raw memory dump (in LiME format) in kernel space without knowing the target operating system or kernel a priori. It does so with the least possible interaction, however the LiME kernel module needs to be loaded in the kernel space which disturbs the system. It then tries to create dumps from `/dev/crash`, `/proc/kcore` or `/dev/mem`. AVML does also provide compression; however, when there is only one try available, it is recommended that the normal raw format should be used to maximize success rates. To analyse the memory image, frameworks like *Volatility*[12], *Rekall*[13] or *Project Freta*[14] can be used. Volatility uses profiles for operating systems where the memory image originated from and executes plugins to extract information. Rekall is a fork of Volatility and extends the framework by introducing automated detection of suitable profiles and constructs new profiles if required [19]. Project Freta is a complementary to AVML and is also developed by Microsoft. In contrast to the latter two frameworks, Project Freta provides a GUI and provides automated full-system memory inspection.

When a corresponding plugin exists, the memory can be analyzed as it would be on the system. However, finding relevant information can be difficult when confronted with large image dumps [18]. Based on Johansen's [12] borderline examination methodology, the following six items should be included in a memory analysis:

1. Identify rogue processes
2. Identify file handles and associated files of the rouge process

[9]https://www.kali.org/
[10]https://www.sans.org/tools/sift-workstation/
[11]https://github.com/microsoft/avml
[12]https://www.volatilityfoundation.org/
[13]http://www.rekall-forensic.com/
[14]https://www.microsoft.com/en-us/research/project/project-freta/

3. Dump suspicious process content and drivers for later analysis
4. Review network artifacts for active or listening malicious connection
5. Examine evidence of code injection
6. Check signs for rootkits

It is important to note that these steps are not set in stone, but should serve as a guideline for beginning an analysis of memory without further IOCs.

3.3 Timestamps in Digital Forensics

It is difficult for humans to perceive digital evidence. It is quite possible that important pieces of digital evidence may be overlooked by examiners who are not fully aware of how seemingly useless data can be converted into evidence of great value. When considered independently, timestamps are of little value, however, when linked to other events, they reveal their true potential, allowing investigators to form causal relationships between these events. Additionally, timestamps can act as an alternative if log files are corrupt or not available [20]. The amount of available timestamps depend on the implemented file system type. In Linux file systems, file system types like Ext3 and NFS use three types of timestamps, whereas Ext4 and Lustre support four different timestamps. The three shared timestamps include:

- **Access time**: Last time the file was read.
- **Modification time**: Last time the file's content has been modified.
- **Change time**: The Last time the file's inode has been changed (e.g. changing permission, ownership, file name, etc.).

The fourth timestamp that is newly introduced is the **Creation time** that refers to the time of creation of a file. These timestamps are stored as 32-bit integers each and represent the seconds since Unix epoch [21].

To parse the timestamps, the Unix tool `stat` or Sleuthkit's `istat` can be used. As `stat` does not always displays the creation time, `debugfs` can be utilized. When using `debugfs` the file system of the investigated file needs to be known. Entering the `debugfs` prompt with `debugfs -w </dev/sdX>` and executing the command `stat /path/to/file` will display all four timestamps. It is important to note that `debugfs` as well as `stat` does display the nanoseconds, however in the case of `debugfs` this not directly obvious. The nanoseconds can be obtained by converting the hexadecimal after the colon to decimal and dividing it by four. Comparing the value to the value of `stat` will reveal that it is in fact the nanosecond.

Timestamps can help analyzing the attackers behavior. When a malicious file has been identified on a system, timestamps can be used to search for new files that have been created during the same period of time. The same applies to instances when a malicious network connection has been detected. The creation of files at the time the connection occurred can be an indicator of malicious activity [19]. For that purpose, analysts can create timelines that illustrate a temporal image of the system's activity. Based on this motivation, two tools were created: *Plaso*[15] and Autopsy's timeline feature[16]. The general

[15] https://github.com/log2timeline/plaso
[16] http://wiki.sleuthkit.org/index.php?title=Timeline

idea behind these tools is to gather temporal data of each file and save it to into a *body file* format. Afterwards, the body file will be merged and sorted into a single file that can be viewed either in a CSV editor or with the *Timeline Explorer*[17] for further analysis.

A drawback of this kind of analysis is shown in Listing 12. If attacker gain administrative rights on a system, they are capable of altering the timestamps in their favor. Despite the fact that some attackers may not consider altering timestamps, modifying the access and modification times is relatively easy to accomplish using the `touch` command. An alteration of change and creation times is more advanced, and can assist in identifying inconsistencies if these were not modified. For example, there may be inconsistencies when the access time is older than the creation time, indicating malicious intend. Moreover, when a binary is replaced with a trojanized version, timestamps may be considerably later than expected (for example the system installation time) [19]. Inspecting all three/four timestamps of the relevant file, as well as related files, is advised.

3.4 Hashset

As already mentioned in subsection 3.2, preserving volatile data is important for the investigation. As creating and analyzing memory images again may take considerable time or may not be possible due to the lack of profiles in the analysis tools, an additional method is to collect the volatile data directly from the system. Moreover, it can help in deciding if a full hard drive duplication is necessary and therefore help triaging systems into relevant and non-relevant systems for the investigation. Although this collection can be conducted manually, it is recommended that this task is automated using a script in order to ensure uniformity and minimize changes made to the system [12]. Furthermore, to perform a profitable analysis, the analyst should know the normal behavior of the investigated system as to distinguish between suspicious artifacts and typical actions [13], [19].

According to Kent et al. [15], volatile data should be collected in the following order of importance:

1. Network connections
2. Login sessions
3. Contents of memory
4. Running processes
5. Open files
6. Network configurations
7. Operating system time

To collect these kind of information, a script was developed to ensure consistent and least-interfering contact with the system. Additionally, a hashset of known good binaries of the Rocky 8.7 and Rocky 9 operating systems where prepared. The hashset should ensure that no binaries were replaced by malicious replicates that trick the examiner into obtaining maliciously forged results. Despite the fact that a hashset is a good starting

[17]https://ericzimmerman.github.io/!index.md

point for verifying binary integrity, it should be kept in mind that changing one bit in a tool also changes the hash sum. Therefore, it is necessary to ensure that the same version of tools is used. If this cannot be complied with, the examiner needs to verify the output of different tools (e.g. `ps` vs. `pstree` vs. `ls /proc/pid`) to exclude the possibility of trojanized binaries. Additionally, the `PATH` environment variable should be observed if unexpected locations of binaries are present [19]. In the event that no hashsets are available, the incident handler may also consider setting up a new system (such as a virtual machine) with the same operating system version and updates. Afterwards, a custom hashset of binaries can be created with the `sha1sum` tool (or other hashing algorithms). To check the system, the command `'sha1sum -c <hashset_file>'` can be executed. The hashsets created for this research uses the `MD5` as well as the `SHA1` algorithm. Included were all binaries under `/sbin` and `/bin` as well as respective binaries under `/usr` and `/usr/local`. Additionally, a recursive search of all files with execution permission was conducted. Despite the fact that this may result in overlapping results, it will also reveal additional files that have not been included in the hashset. The procedure for creating a hashset can be found in Listing 13. To ensure integrity of the hashset, the respective hash for all hash files was calculated. The output is archived in a `tar.gz archive`. Additionally, it was digitally signed with the author's digital signature.

3.5 Live Incident Script

The script is a merged collection of already existing solutions [22]–[25] as well as suggestions from literature [12], [15] translated into a Bash script. Kent et al.'s [15] order of volatility was considered and implemented accordingly. However, network connections and configurations were merged into one category. Additionally, a persistence, system, logs, and files category was introduced, whereas the operating system time is collected during general system information gathering. Optionally, a memory image with AVML can be created. Moreover, the user has the choice to scan the system with the lite version of the compromise assessment tool *Thor*[18]. This tool rates elements based on numerous characteristics of malware described in YARA rules (see Appendix E for more information). It can also be used to scan mounted disk images in the `lab` mode. In addition, the user can also choose to collect HPC system specific information in the form of executed `SLURM` jobs on the node. Information about `JobID`, `Job State`, the `Nodelist` and additional details are gathered to support the analysis. In this manner, the investigator has the opportunity to identify further potentially compromised systems, since a user is able to connect to the worker node to which the `SLURM` job was issued, which otherwise would not be possible. Using this information, a follow-up investigation can be carried out to collect job scripts for a particular suspicious JobID to identify whether malicious intent is contained within them or if attackers executed any malicious files on the compute nodes.

To avoid changing timestamps, the file system that is mounted on the root file system needs to be remounted with the `noatime` option[19]. The command for this operations is: `'mount -o remount,noatime /dev/sdX'` where X is the used file system. It prevents the update of access timestamps while reading files during the collection. After the script finishes, the file system can be remounted with original options. Johansen [12] came up with a clever idea of how to execute a script in a safe environment. In his recommendation, a network share on a dedicated server should be set up with one directory for the script

[18]https://www.nextron-systems.com/thor-lite/

[19]Note however, that the `noatime` option has no effect on NFS mounts.

and one for storing the output. In order to access the share, new throwaway credentials should be created which have never been used for anything else within the organization before. The folder for the script was made read-only, the second folder for the output was made writable. Beside this suggestion, tools to collect volatile data should always be placed on a portable device along with all needed tools as to not pollute the target machine by copying files to the system. As well, all external tools should be statically linked binary files. Thus, no dependencies on other library functions are required [15].

The script starts with collecting network information. During an attack, attackers may open listeners on previously unutilized ports in an attempt to gain or reclaim access to a system. Additionally, attackers may try to redirect network traffic by forging wrong MAC-adresses and therefore creating a *man-in-the-middle-attack*. Besides this attempt, attackers can install rootkits in the attempt to try hide ports from queries on a running system. Hence, the script collects, among others, information such as network devices, lists of network connections, and ARP table caches. To detect hidden ports, the external tool `unhide-tcp` is employed. This tool identifies TCP/UDP ports that are listening but are not listed in `/bin/netstat` by brute-forcing all available TCP/UDP ports. Listing 3 shows a result of a `ss` output that displays a `netcat` connection listening on port 4444 for commands issued by a potential attackers. Having the knowledge that port 4444 is abnormal, this entry can indicate a compromise.

Next, log-in sessions are included in the collection. It collects various information of `SSH` logins to the system. The information includes login timestamps, a list of login attempts, and the origin of the login. Especially the origin of a login can be interesting as it can depict additionally compromised systems when the attacker tries to exploit the network with lateral movement. Listing 4 depicts login attempts taken from the `secure` log that registered numerous login attempts to the user *bad*. Since the login attempts are occurring rapidly, one may assume that this is a brute force attack. The attack has been successful in the end, as a login was registered in this scenario. Additionally, `SSH` and `SSHD` configurations are collected that may contain malicious entries to ensure reentry for an attacker.

Moreover, the script collects information about the memory. It queries the currently used memory and collects used swap space as well as information of the CPU. It mainly should be used for detecting abnormal behaviour of the system. As an example, an idle system that utilizes a large amount of memory may be an indication that hidden software, such as cryptocurrency mining, is executed.

Collecting information about processes is another important category. Many modern tools use 'memory only' injection methods, where malicious code is added to processes but is never written to disk. Given this fact, evidence of running malicious content can only be obtained in this state. Due to the fact that the system typically runs a variety of processes, it is recommended that a baseline or historical record of those processes is maintained. A more detailed examination should be conducted on processes that cannot be verified [19]. When examining a process, it is important to consider details such as the location of the executable, the type of child processes spawned by the process, as well as if the execution start time aligns with the expected time. Additionally, the script checks for process that were executed by deleted binaries (Listing 10) or if the processes runs from untypical places like `/tmp` or `/dev`. Moreover, `unhide` is utilized, just as in the case of network connection gatherings. In this setting, it tries to detect hidden and fake processes by comparing results from `/proc` vs. `/bin/ps`, as well as comparing information gather from `/bin/ps` with the information gathered by walking through the `procfs`. As a last verification step, `unhide` compares output from `/bin/ps` to information gathered from the `syscalls`. Listing 5 shows how `unhide` can detect a process hidden by the rootkit *Diamorphine*.

As an additional step, open files executed on the system are collected. It shows the user and process that opened a file. When having additional IOCs, one can filter the `lsof` output with the `-p` parameter for a process ID or with the `-u` parameter for filtering for a specific user. By using this information, one may be able to identify more malicious activities by an attacker or uncover hidden malicious executables.

As a last collection block, non-volatile data is collected that are further divided into subcategories. The first block collects general host information such as hostname, kernel version, last boot time, the `PATH` environment, date and time configurations, and so on. Quite interesting information is gathered regarding to all mounted devices, confirming that no forged options have been used. Additionally, *Loadable Kernel Modules* (LKMs) are collected. Listing 8 and Listing 9 show exemplary output for this kind of information. In this scenario, a malicious LKM could be detected that belongs to the rootkit *Diamorphine*. This rootkit has the ability to hide processes, files or directories and can give the user root access to the system. Additional indicators can be taken into account by investigating the Linux Kernel Modules under `/usr/lib/modules/$(uname -r)/kernel/` that are also gathered. Unknown `.ko` files indicate signs of a rootkit. Having this information, an analyst can access the persistence collection and check if a the unknown kernel module is loaded at boot time to create backdoors for attackers. To load the malicious LKM, a configuration needs to be created under `/etc/module-load.d` or `/etc/modprobe.d`. The configurations in these two directories will be collected and are available for analysis. Additionally the `crontab` and `cron` entries are collected as well as outputs of system commands that list startup items. Furthermore, processes instructed to start at boot time in location like `/etc/init.d`, `/etc/rc.d`, etc. are also collected.

To gain additional information in determining if the system should be further analyzed, the collection of logs under `/var/logs` and a collection of modified files (in the last 90 days) as well as key files (anacron, systemd, ssh, yum, etc.) is conducted. Additional to just simply collecting all log files, a search of binary code in log files is done. Moreover, files that try to evade detection by adding white space or names like `"..."` are extracted. A special attention is put on directories like `/tmp` and `/dev` where hidden files are uncommon [19]. Furthermore, `setuid` and `setgid` files are searched recursively as they allow to execute files with administrative rights without having root access. The collection of system and file information serves the purpose of having additional indicators when analyzing and finding relevant evidence in volatile data. The analyzer can quickly fetch the files that are interesting or substantiate findings without the need to create a whole disk image.

The last step is to gather information about the users and groups within the system. Attackers try to ensure regaining access to a system by elevating user rights or creating user accounts. Therefore, files like `passwd` or a list of users in the `sudo` group are gathered to give the analyzer the ability to verify the presence of unknown or reactivated user accounts or elevated privileges of legitimate accounts. Furthermore, Anson [19] notes that accounts that run daemon processes should be inspected, as they should not have the ability to log into the system interactively. Discovering such activity should be investigated.

Although the volume of data collected by this script is quite large, Kent et al. [15] assert that when in doubt it is always advisable to collect as much volatile data as possible because as soon as the computer is turned off, all opportunities to collect volatile information will be lost. To distribute the script, `clush` can transfer and execute the script on different nodes in parallel. The resulting outputs can than be transferred back onto the management node.

Concluding, an interesting tool should be mentioned. The *Unix-like Artifacts Collector*[20] (UAC) is a IR collection script that automates the collection of system information. The tool uses so called *artifact files* to make the collection highly customizable and extendable. An artifact files contains a set of rules (artifacts) that will be used by the collector. In turn, each artifact specifies the tool and parameters to be used. The tool additionally distinguishes between hash collectors that hash the searched asset and the command collector that collects output. Additionally, UAC can collect and store information into the bodyfile format that can directly be put into timeline analysis. Therefore, UAC provides the user with the ability to adapt to changing circumstances and extend the analysis when more information is collected.

4 Scaling live forensics

Traditionally, an analyst in digital forensics would create a full image of an infected device, as well as capturing the memory contents and other artifacts. The data is transferred to an analysis workstation, where it can be analyzed over the course of several days or hours [12]. In a HPC system environment, this type of analysis does not scale well when looking at possibly hundreds or thousands of infected systems. An alternative solution was presented in the form of a script that collects volatile data, however, execution of the script and a manual analysis must be conducted for each system separately. There must be assurance that the results from each system are received and can be assigned accordingly. To overcome this problem, the software *Velociraptor*[21] will be introduced. As a digital forensics and incident response tool, Velociraptor is used to collect disk-based artifacts, monitor a system, hunt for a specific threat and respond to an incident within the organization. Velociraptor tries to solve the problem of investigating numerous system simultaneously by executing commands on the live system and gathering the results in a central endpoint. It uses a modified form of the `SQL`, the Velociraptor Query Language (VQL) to analyze the data directly. This section shall explain the deployment and details of Velociraptor and how it can help speeding up incident responses in a HPC system environment.

4.1 Velociraptor Deployment

Meyer et al. [26] show that Velociraptor outperforms related software in incident response and forensics (Google Rapid Response, Cynet) in almost all categories in form of implemented features. It can be considered as a state-of-the-art live incident response tool that basically only needs one server in default configuration. The server acts as a front end that communicates with the clients and hosts the web-based admin GUI.

Each deployment relies on the static Velociraptor binary and on a unique configuration file which includes connection information and cryptographic keys for message passing. Since key material is unique to each deployment, one Velociraptor deployment cannot connect with another deployment ensuring security and integrity [27]. The configuration

[20]https://github.com/tclahr/uac
[21]https://docs.velociraptor.app/

creation process is guided and an exemplary installation can found in Listing 15. Velociraptor encrypts messages using certificates from a self-signed Certificate Authority (CA) that is generated during the initial configuration step. A signed CA is included in the client configuration and is used to verify all certificates that are required during communication. A second mode for certificate deployment is to issue certificates minted by `Let's encrypt`. The key idea behind Velociraptor is that a client maintains a persistent connection to the Velociraptor server and as soon as a task is scheduled by a user, the execution of this task is done on the client itself and the result will be send back to the server. This ensures the possibility to scale such infrastructure to thousands of clients without the need of a high performance server. Additionally, Velociraptor maintains server stability by employing three limits [27]:

- **Concurrency**: Controls how many clients are served simultaneously.
- **Load shedding**: The server accepts only connections up to a certain rate ('queries per second' (QPS) limit) and refuses any additional connections above the limit, causing clients to retry the connection at a later point
- **Task recruitment limits**: The server limits the rate at which endpoints are assigned to a task to spread responses coming back over time.

However, Velociraptor can nevertheless be deployed in a multi-server architecture when the client number increases or a lot of data needs to transferred. An individual server is designated as the master, while all other servers are designated as minions. While the Master acts as a bridge for all messages between the minions, the minions receive events from the Master and generate events from the clients output that they send to the Master. To distribute load evenly, a load balancer can be installed in front of the servers. Alternatively, multiple front end URLs can be provided in the client configuration that will be picked at random. To store and distribute results coming from the clients, the servers share files via a common NFS.

4.2 Incident Response with Velociraptor

Velociraptor utilizes VQL that uses the basic sentence structure of SQL to translate the output of clients into rows to display to the user. Due to VQL's flexibility, users are able to adapt to new IOCs easily by designing new query rules (which can also be shared with others) that can be issued to all endpoints. The strength of VQL, and where it distinguishes Velociraptor from manual collection with scripts, is that it can further filter results on the fly. For instance, a user may query every client to output the installed software of a system. The user may than additionally group and stack the outputs and give back the counts of the rows to quickly get an overview of the system installments and their respective system. This can be used for discovering suspicious software across the whole network without manually touching every system [27].

These VQL queries can be packed in so called *artifacts* that can be compared to the artifact files used by UAC mentioned in subsection 3.5. UAC artifact files can also serve as a template for writing VQL queries to extend existing artifacts and knowledge base. These artifacts are structured `YAML` files that contain the query with the definition of parameters and a name attached to it. The parameters allow the user to customize a query in the GUI without the need edit the underlying VQL query. A *hunt* is created when an artifact is executed to collect information from the endpoints. Hunting refers to the process by which an analyst actively searches for malware or suspicious activity, usually in response to threat intelligence or vulnerabilities that have been disclosed [27]. These hunts will execute on every client connected to the server until the expiration date is reached. Expiration dates are used to ensure that queries will be executed by clients who connect to the system at a later time. In addition to hunts, the user has the ability to execute arbitrary shell commands or custom VQL queries directly on the system using the endpoint's respective shell to allow for adhoc and individual investigation.

Listing 16 and Listing 17 display such artifacts that include VQL queries. Listing 16 collects file information about the name, size and its permissions. Suspicious files will than be declared to lie in `/home` or `/tmp` (default settings) and either start with a dot, are larger than a pre-specified size or have `setuid` permission. Parameters can also be changed in the GUI while creating a hunt to adapt the investigation as needed. Figure 2 shows the truncated Velociraptor result that was collected in a network of two clients. It can be noted that the Rocky9 machine contains a suspicious `netcat` binary that is hidden in the `/tmp` directory and has `setuid` permission. This binary can be used by normal users to establish connections (e.g. backdoors) to other machines without root access. These artifact can also be used for verifying the clean state of systems, as it can has the capability to detect particular malicious files on all systems. In addition to the predefined artifacts listed above, Velociraptor also provides predefined artifacts in the form of "file finders" which search files based on metadata based such as size, name, or timestamps. When conducting more sophisticated searches, hashes of files can be provided that are independent of changeable metadata and can be used to detect files that were for example renamed.

Additionally, Figure 3 presents the power of simultaneous investigation of the network. In this scenario, a malicious attacker connects from IP `10.0.3.7` to the Rocky9 machine by using a compromised account (here user `infected`). The attacker than successfully carved for credentials, enabling lateral movement in the network and connecting as `root` to the Rocky8 machine through the Rocky9 machine (IP `10.0.3.5`). This information can help understanding the attack path the attacker took and reveals additional compromised systems. Listing 17 shows that this information is generated by parsing the `wtmp` log and extracting the information to display it as rows in the GUI.

To be concrete, Velociraptor acts like a live incident script presented in subsection 3.5, however it utilizes the advantage that the commands are executed simultaneously on all systems in a network and gathers the results in an organized and readable manner. Moreover, by making use of artifacts, Velociraptor is extremely flexible and up-to-date by sharing knowledge within the IR community.

5 Conclusion

We have shown the importance of a proper IR framework for organizations in the event of a cyber attack. In order to guarantee the best possible defense mechanism, defenders first have to understand the attack vector of malicious activities in order to harden the system in every phase of an attack. As systems and software are exposed to various vulnerabilities, the investigator needs to have a broad knowledge about a systems normal behavior and current attack vectors. In addition, it is necessary to consider the pitfalls of forensic analysis in regard to the distortion of evidence. We argue that traditional forensic analysis may not be applicable due to the large number of systems that can potentially be compromised. In order to differentiate between compromised and non-compromised systems first, live incident response should be used, followed by traditional forensic analysis of the compromised systems. To minimize the chance of altering evidence, the forensic script that was developed for this project should be used. Nevertheless, the preferred method of conducting live incident response on HPC systems is to use Velociraptor, due to the simultaneous investigation procedure and its flexibility. By exploiting Velociraptor's flexibility, the analyst can investigate the network in an innovative manner and produce engaging results.

References

[1] I. Lella, M. Theocharidou, E. Tsekmezoglou, and A. Malatras, *Enisa threat landscape 2021*, 2015. DOI: 10.2824/324797.

[2] Internet Crime Complaint Center, *Internet crime report 2021*, 2021.

[3] Bundeskriminalamt, *Cybercrime: Cybercrime bundeslagebild 2021*, Wiesbaden, 2021.

[4] Z. M. Smith, E. Lostri, and J. A. Lewis, *The hidden costs of cybercrime*, 2020.

[5] P. Kral and C. Wright, *Incident handler's handbook*, 2021.

[6] M.-D. J. McLaughlin, W. A. Cram, and J. L. Gogan, "A high performance computing cluster under attack: The titan incident," *Journal of Information Technology Teaching Cases*, vol. 5, no. 1, pp. 1–7, 2015, ISSN: 2043-8869. DOI: 10.1057/jittc.2015.1.

[7] P. Korambath, *Cyber security in high-performance computing environment*, 2014.

[8] *Iso 27001 – annex a.16: Information security incident management*, 2013.

[9] S. Caltagirone, A. Pendergast, and C. Betz, *The diamond model of intrusion analysis*, 2013.

[10] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 80, 2011.

[11] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, *Mitre att&ck: Design and philosophy*, 2020.

[12] G. Johansen, *Digital Forensics and Incident Response*, 1st ed. Birmingham: Packt Publishing Limited, 2017, ISBN: 9781787285392.

[13] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, *Computer security incident handling guide: Recommendations of the national institute of standards and technology: Nist special publication 800-61 revision 2*, 2012. DOI: 10.6028/NIST.SP.800-61r2.

[14] *Assessing ir in the cloud using the sans ir model.* [Online]. Available: https://static.packt-cdn.com/products/9781800569218/graphics/Images/B16575_11_03.png.

[15] K. Kent, S. Chevalier, T. Grance, and H. Dang, *Guide to integrating forensic techniques into incident response: Nist special publication 800-86*, 2006.

[16] J. T. Luttgens, *Incident response & computer forensics*, 3rd ed. New York: McGraw-Hill Education, 2014, ISBN: 9780071798693.

[17] J. Boucher, *Imaging with caine*, 2021. [Online]. Available: https://www.caine-live.net/page8/CAINE%20Imaging%20Instructions%20(March%202021)%20-%20External.pdf.

[18] M. Sheward, *Hands-on incident response and digital forensics.* Swindon, UK: BCS Learning & Development, 2018, ISBN: 9781780174204. [Online]. Available: https://learning.oreilly.com/library/view/-/9781780174204/?ar.

[19] S. Anson, *Applied incident response.* Indianapolis, Indiana: Wiley, 2020, ISBN: 9781119560265.

[20] R. Koen and M. S. Olivier, "The use of file timestamps in digital forensics," *ISSA*, 2008.

[21] T. Göbel and H. Baier, Eds., *Anti-forensics in ext4: On secrecy and usability of timestamp-based data hiding*, vol. 24, 2018. DOI: `10.1016/j.diin.2018.01.014`.

[22] la3ar0v, *Tuxresponse*, 2019. [Online]. Available: `https://github.com/la3ar0v/TuxResponse`.

[23] rshipp, *Ir-triage-toolkit*, 2013. [Online]. Available: `https://github.com/rshipp/ir-triage-toolkit/blob/main/linux/run`.

[24] WithSecureLabs, *Linuxcatscale*, 2022. [Online]. Available: `https://github.com/WithSecureLabs/LinuxCatScale`.

[25] jgasmussen, *Linux-baseline-and-forensic-triage-tool*, 2022. [Online]. Available: `https://github.com/jgasmussen/Linux-Baseline-and-Forensic-Triage-Tool`.

[26] M. Meyer, G. Auth, and A. Schinner, "A method for evaluating and selecting software tools for remote forensics," *INFORMATIK*, pp. 867–878, 2021. DOI: `10.18420/informatik2021-074`.

[27] Rapid7, *Velociraptor - digging deeper!* 2023. [Online]. Available: `https://docs.velociraptor.app/`.

[28] EGI Computer Security and Incident Response Team, *Attacks on multiple hpc sites*, 2020. [Online]. Available: `https://csirt.egi.eu/attacks-on-multiple-hpc-sites`.

[29] N. Naik, P. Jenkins, R. Cooke, J. Gillett, and Y. Jin, Eds., *Evaluating Automatically Generated YARA Rules and Enhancing Their Effectiveness*, 2020. DOI: `10.1109/SSCI47803.2020.9308179`.

[30] *Yara in a nutshell*, 22.03.2023. [Online]. Available: `https://virustotal.github.io/yara/`.

[31] G. A. Iosif, *Linux.detection.anomalousfiles.* [Online]. Available: `https://docs.velociraptor.app/artifact_references/pages/linux.detection.anomalousfiles/`.

[32] Rapid7, *Linux.sys.lastuserlogin.* [Online]. Available: `https://docs.velociraptor.app/artifact_references/pages/linux.sys.lastuserlogin/`.

A Image creation

```
$ sha256sum /dev/sdb > sdb_hash.sha256

$ sudo dd if=/dev/sdb of=image.dd bs=4096 conv=sync,noerror

$ sudo dc3dd if=/dev/sdb of=image.dd bs=4k hash=sha256 hashlog=hash.log
   log=image.log progress=on

$ sudo dcfldd if=/dev/sdb conv=sync,noerror bs=4k hash=sha256 hashlog=
   hash.log of=image.dd
```

Listing 2: Forensic Image Creation

B Evidence

These examples display a simple and easy recognizable attack scenario, however they show the effectiveness of the script that collects these types of output. For illustration purpose, the Diamorphine rootkit[22] was utilized that was also used during real-life attacks against HPC clusters [28]. Additionally, Listing 4 was created using the `hydra` tool to perform a brute-force attack.

```
root@infected:~# ss -anepo | grep netcat
tcp     LISTEN   0        1      0.0.0.0:4444       0.0.0.0:*
   users:(("netcat",pid=8264,fd=3)) uid:1001 ino:93393 sk:35d <->
```

Listing 3: Output of `ss -anepo` here already filtered for truncation. It illustrates the benefit of filtering for IOCs in speeding up system analysis as well as the importance of understanding normal system behavior in such a large log.

```
Mar 21 15:57:42 infected sshd[5253]: Failed password for bad from
   10.0.3.6 port 56720 ssh2
Mar 21 15:57:42 infected sshd[5254]: Failed password for bad from
   10.0.3.6 port 56726 ssh2
Mar 21 15:57:43 infected sshd[5257]: Failed password for bad from
   10.0.3.6 port 56736 ssh2
Mar 21 15:57:43 infected sshd[5258]: Failed password for bad from
   10.0.3.6 port 56752 ssh2
Mar 21 15:57:43 infected sshd[5256]: Failed password for bad from
   10.0.3.6 port 56734 ssh2
Mar 21 15:57:43 infected sshd[5255]: Accepted password for bad from
   10.0.3.6 port 56732 ssh2
Mar 21 15:57:43 infected sshd[5255]: pam_unix(sshd:session): session
   opened for user bad by (uid=0)
```

Listing 4: Output of `/var/log/secure` showing ssh login attempts.

```
root@infected:~# unhide -v quick checksysinfo
Unhide 20130526
Copyright 2013 Yago Jesus & Patrick Gouin
License GPLv3+: GNU GPL version 3 or later
http://www.unhide-forensics.info
```

[22]https://github.com/m0nad/Diamorphine

```
NOTE : This version of unhide is for systems using Linux >= 2.6

Used options: verbose
[*]Searching for Hidden processes through  comparison of results of
   system calls, proc, dir and ps

Found HIDDEN PID: 5629
        Cmdline: "netcat"
        Executable: "/usr/bin/nc.openbsd"
        Command: "netcat"
        $USER=bad
        $PWD=/

[*]Searching for Hidden processes through sysinfo() scanning
```

Listing 5: Output of `unhide -v quick checksysinfo` showing a hidden process that executes `netcat` indicating a possibility for attackers to issue commands through the `netcat` connection.

```
root@infected:~# grep -Po '^sudo.+:\K.*$' /etc/group
user,bad
```

Listing 6: Listing of all users in the superuser group.

```
root@infected:~# lsof -p 8264
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
netcat  8264  bad  cwd    DIR    8,5     4096      2 /
netcat  8264  bad  rtd    DIR    8,5     4096      2 /
netcat  8264  bad  txt    REG    8,5    43664 525115 /usr/bin/nc.openbsd
netcat  8264  bad  mem    REG    8,5  2029560 526489 /usr/lib/x86_64-
   linux-gnu/libc-2.31.so
netcat  8264  bad  mem    REG    8,5   101320 526503 /usr/lib/x86_64-
   linux-gnu/libresolv-2.31.so
netcat  8264  bad  mem    REG    8,5    96728 533717 /usr/lib/x86_64-
   linux-gnu/libbsd.so.0.10.0
netcat  8264  bad  mem    REG    8,5   191472 526482 /usr/lib/x86_64-
   linux-gnu/ld-2.31.so
netcat  8264  bad   0u    CHR  136,1      0t0      4 /dev/pts/1
netcat  8264  bad   1u    CHR  136,1      0t0      4 /dev/pts/1
netcat  8264  bad   2u    CHR  136,1      0t0      4 /dev/pts/1
netcat  8264  bad   3u   IPv4  93393      0t0    TCP *:4444 (LISTEN)
```

Listing 7: Output of `lsof` here again filtered filtered for an IOC (can also be a malicious user with parameter `-u`). It shows the open files created by the `netcat` command. It additionally shows the location of the binary.

```
root@infected:~# lsmod
Module                  Size  Used by
diamorphine            16384  0
isofs                  49152  0
vboxsf                 81920  1
vboxvideo              36864  0
ttm                   106496  1 vboxvideo
drm_kms_helper        184320  1 vboxvideo
```

Listing 8: Output of `lsmod` showing the malicious rootkit *Diamorphine* (Output truncated).

```
root@infected:~# modinfo diamorphine
filename:       /lib/modules/5.4.0-28-generic/kernel/diamorphine.ko
description:    LKM rootkit
author:         m0nad
license:        Dual BSD/GPL
srcversion:     D2376C77D0B08B5067F9CFB
depends:
retpoline:      Y
name:           diamorphine
vermagic:       5.4.0-28-generic SMP mod_unload
```

Listing 9: Output of `modinfo diamorphine` showing metadata and location of the rootkit.

```
root@infected:~# lsof -p 8264 | grep deleted
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
netcat  8264  bad  txt    REG    8,5    43664 525115 /usr/bin/nc.openbsd
    (deleted)
```

Listing 10: Output of `lsof | grep deleted` that shows that this process uses a deleted binary of `netcat` and is still running.

`ls` was modified (truncated).

```
root@infected:/usr/bin# find /usr/bin/ -mtime -90
/usr/bin/
/usr/bin/VBoxClient
/usr/bin/logresolve
/usr/bin/mail
/usr/bin/rotatelogs
/usr/bin/etags
/usr/bin/emacsclient
/usr/bin/ls
```

Listing 11: Output of `find /usr/bin/ -mtime -90` that shows all modified files in the last 90 days in directory /usr/bin. Among other things

C Timestomping

Attackers may want to conceal their tracks by modifying timestamps of malicious files to blend into the system. This technique is knows as *timestomping.* A common method for timestomping is making use of the `touch` and `debugfs` command. In the following, the timestomping methods should be illustrated:

```
$ touch malicious_file -r legit_file # this will inherit the file's
   timestamps

$ touch -a -d '22 Jan 2024 11:09:23.194826432' malicious_file # changing
    the file's access time

$ touch -m -d '1 Feb 2023 10:23:23.199494912' malicious_file # changing
   the file's modification time

# changing the change time, assuming the files is located in /dev/sda1
$ debugfs -w -R 'set_inode_field /path/to/malicious_file ctime
   201001010101' /dev/sda1
$ echo 2 > /proc/sys/vm/drop_caches
```

```
# changing the creation time, assuming the files is located in /dev/sda1
$ debugfs -w -R 'set_inode_field /path/to/malicious_file crtime
   201001010101' /dev/sda1
$ echo 2 > /proc/sys/vm/drop_caches

$ stat malicious_file
  File: malicious_file
  Size: 0               Blocks: 0          IO Block: 4096   regular
   empty file
Device: 801h/2049d      Inode: 688100      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2024-01-22 11:09:23.194826432 +0100
Modify: 2023-02-01 10:23:23.199494912 +0100
Change: 2010-01-01 02:01:00.037994959 +0100
 Birth: 2010-01-01 02:01:00.212135631 +0100
```

Listing 12: Common timestomping methods to alter the timestamps of a file.

D Hashsets

```
md5sum /sbin/* | tee -a rocky8_hash.md5
md5sum /bin/* | tee -a rocky8_hash.md5
md5sum /usr/bin/* | tee -a rocky8_hash.md5
md5sum /usr/sbin/* | tee -a rocky8_hash.md5
md5sum /usr/local/sbin/* | tee -a rocky8_hash.md5
md5sum /usr/local/bin/* | tee -a rocky8_hash.md5
find / -xdev -type f -perm -o+rx -print0 | xargs -0 md5sum | tee -a
   rocky8_hash.md5

sha1sum /sbin/* | tee -a rocky8_hash.sha1
sha1sum /bin/* | tee -a rocky8_hash.sha1
sha1sum /usr/bin/* | tee -a rocky8_hash.sha1
sha1sum /usr/sbin/* | tee -a rocky8_hash.sha1
sha1sum /usr/local/sbin/* | tee -a rocky8_hash.sha1
sha1sum /usr/local/bin/* | tee -a rocky8_hash.sha1
find / -xdev -type f -perm -o+rx -print0 | xargs -0 sha1sum | tee -a
   rocky8_hash.sha1
```

Listing 13: Commands to create a hashset of binaries. These commands can be repeated for the Rocky 9 operating system.

E YARA Rules

Malware researchers can use YARA to identify and classify malware samples. It is possible to create detailed descriptions of malware families (or any other type of malware) using YARA based on textual or binary patterns of the malware. The patterns can be made of a series of strings but also more complex rules like wild-cards, case-insensitive strings, regular expressions, special operators, etc. A rule consists of three sections [29]:

- **Meta**: Description of the malware that is depicted.
- **Strings**: Patterns of Malware

- **Conditions**: Specifies the number of signatures/strings required matching with the target to declare the sample as malware.

An exemplary rule could be illustrated like the following:

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

Listing 14: Exemplary YARA rule [30].

Whenever one of the strings specified in the rule matches within a file, the rule declares that file as belonging to the *banker* malware.

F Velociraptor

F.1 Installation

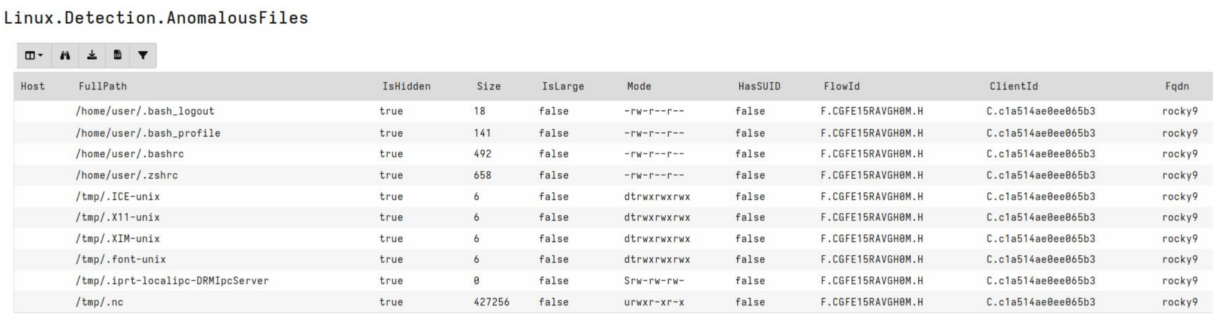
```
# Download binary + signature file (X defines the release version)
$ wget https://github.com/Velocidex/velociraptor/releases/download/vX.X.
    X-rc1/velociraptor-vX.X.X-rc1-linux-amd64-musl
$ wget https://github.com/Velocidex/velociraptor/releases/download/vX.X.
    X-rc1/velociraptor-vX.X.X-rc1-linux-amd64-musl.sig

# Verify binary
$ gpg --verify velociraptor-v*-rc1-linux-amd64-musl.sig
# Add key (take RSA key you obtained by --verify)
# gpg: key B22A7FB19CB6CFA1: public key "Velociraptor Team (Velociraptor
     - Dig deeper!  https://docs.velociraptor.app/) <support@velocidex.
    com>" imported
$ gpg --search-keys 0572F28B4EF19A043F4CBBE0B22A7FB19CB6CFA1 # -> press
    1
# OR
$ gpg --keyserver hkp://keys.openpgp.org:80 --recv 0572
    F28B4EF19A043F4CBBE0B22A7FB19CB6CFA1
# Optional: Verify key again
$ gpg --verify velociraptor-v*-rc1-linux-amd64-musl.sig

# Make velociraptor binary executable
$ sudo chmod +x velociraptor-v*-rc1-linux-amd64-musl

# Create config for server interactivly
$ ./velociraptor-v*-rc1-linux-amd64-musl config generate -i
# Follow steps and fill in the appropriate entries
# Server and Client yaml should be created
# Change the server IP in client config

# Create a .rpm package for server installation (CentOS)
$ ./velociraptor-v*-rc1-linux-amd64-musl --config server.config.yaml rpm
    server --binary velociraptor-v0.6.8-rc1-linux-amd64-musl

# OR to run without installing that creates test machine (machine acts
   as server and client simultaneously)
$ ./velociraptor-v*-rc1-linux-amd64-musl --config server.config.yaml gui

# rpm installation
$ sudo rpm -i velociraptor_*-rc1_server.rpm


# Check if installation successful
$ systemctl restart velociraptor_server.service
$ systemctl status velociraptor_server.service

# Create a GUI user
# Other user roles: reader, analyst, investigator, artifact_writer
$ ./velociraptor-v*-rc1-linux-amd64-musl --config server.config.yaml
   user add admin --role administrator

# To start a velociraptor client, copy the binary and the client config
   to the client system
# Can also create a rmp package for client and install it like above
$ ./velociraptor-v*-rc1-linux-amd64-musl --config client.config.yaml
   client -v
```

Listing 15: Exemplary Velociraptor installation.

F.2 Artifacts

```
name: Linux.Detection.AnomalousFiles

description: |
  Detects anomalous files in a Linux filesystem.

  An anomalous file is considered one that matches at least one criteria
   :

  - Hidden (prefixed with a dot);

  - Large, with a size over a specified limit; or

  - With SUID bit set.

type: CLIENT

parameters:
  - name: MaxNormalSize
    description: Size (in bytes) above which a file is considered large
    type: int
    default: 10485760
  - name: PathsToSearch
    description: Paths to search, separated by comma
    type: str
    default: "/home/**,tmp/**"

sources:
  - precondition: |
      SELECT OS
      FROM info()
      WHERE OS = 'linux'

    query: |
      SELECT Fqdn AS Host,
             FullPath,
             substr(str=Name, start=0, end=1) = "." AS IsHidden,
             Size,
             Size > MaxNormalSize AS IsLarge,
             Mode.String AS Mode,
             Mode =~ "^u" as HasSUID
      FROM glob(globs=split(string=PathsToSearch, sep_string=","))
      WHERE IsHidden OR IsLarge OR HasSUID
```

Listing 16: Exemplary Velociraptor artifact to detect suspisious files [31].

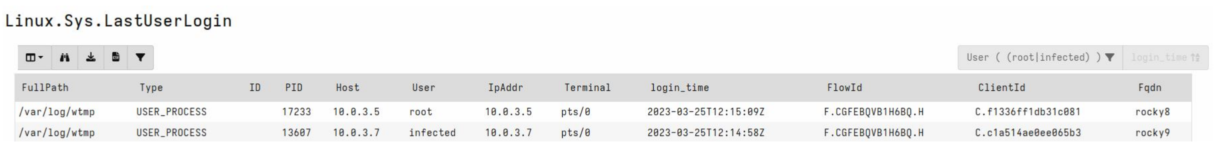

Linux.Detection.AnomalousFiles

Host	FullPath	IsHidden	Size	IsLarge	Mode	HasSUID	FlowId	ClientId	Fqdn
	/home/user/.bash_logout	true	18	false	-rw-r--r--	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/home/user/.bash_profile	true	141	false	-rw-r--r--	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/home/user/.bashrc	true	492	false	-rw-r--r--	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/home/user/.zshrc	true	658	false	-rw-r--r--	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/tmp/.ICE-unix	true	6	false	dtrwxrwxrwx	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/tmp/.X11-unix	true	6	false	dtrwxrwxrwx	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/tmp/.XIM-unix	true	6	false	dtrwxrwxrwx	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/tmp/.font-unix	true	6	false	dtrwxrwxrwx	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/tmp/.iprt-localipc-DRMIpcServer	true	0	false	Srw-rw-rw-	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9
	/tmp/.nc	true	427256	false	urwxr-xr-x	false	F.CGFE15RAVGH0M.H	C.c1a514ae0ee065b3	rocky9

Figure 2: Suspicious files found with Velociraptor's Artifact Linux.Detection.AnomalousFiles (truncated).

```
name: Linux.Sys.LastUserLogin
description: Find and parse system wtmp files. This indicate when the
             user last logged in.
parameters:
  - name: wtmpGlobs
    default: /var/log/wtmp*

  - name: MaxCount
    default: 10000
    type: int64

export: |
  LET wtmpProfile <= '''
       [
         ["Header", 0, [
           ["records", 0, "Array", {
               "type": "utmp",
               "count": "x=>MaxCount",
               "max_count": "x=>MaxCount"
           }]
         ]],

         ["timeval", 8, [
          ["tv_sec", 0, "int32"],
          ["tv_usec", 4, "int32"]
         ]],

         ["utmp", 384, [
          ["ut_host", 76, "String", {
           "length": 256
          }],

          ["ut_tv", 340, "timeval"],

          ["ut_id", 40, "String", {
           "length": 4
           }],

          ["ut_type", 0, "Enumeration", {
            "type": "short int",
            "choices": {
               "0": "EMPTY",
               "1": "RUN_LVL",
               "2": "BOOT_TIME",
               "5": "INIT_PROCESS",
               "6": "LOGIN_PROCESS",
               "7": "USER_PROCESS",
               "8": "DEAD_PROCESS"
             }
          }],

          ["ut_user", 44, "String", {
           "length": 32
          }]
        ]]
       ]
   '''

sources:
  - precondition: |
      SELECT OS From info() where OS = 'linux'
    query: |
      LET parsed = SELECT FullPath, parse_binary(
                   filename=FullPath,
                   profile=wtmpProfile,
                   struct="Header"
                 ) AS Parsed
      FROM glob(globs=split(string=wtmpGlobs, sep=","))

      SELECT * FROM foreach(row=parsed,
      query={
         SELECT * FROM foreach(row=Parsed.records,
         query={
           SELECT FullPath, ut_type AS Type,
              ut_id AS ID,
              ut_host as Host,
              ut_user as User,
              timestamp(epoch=ut_tv.tv_sec) as login_time
          FROM scope()
        })
      })
```

Listing 17: Exemplary Velociraptor artifact to detect last logins per SSH [32].



FullPath	Type	ID	PID	Host	User	IpAddr	Terminal	login_time	FlowId	ClientId	Fqdn
/var/log/wtmp	USER_PROCESS		17233	10.0.3.5	root	10.0.3.5	pts/0	2023-03-25T12:15:09Z	F.CGFEBQVB1H6BQ.H	C.f1336ff1db31c081	rocky8
/var/log/wtmp	USER_PROCESS		13607	10.0.3.7	infected	10.0.3.7	pts/0	2023-03-25T12:14:58Z	F.CGFEBQVB1H6BQ.H	C.c1a514ae0ee065b3	rocky9

Figure 3: Last SSH logins found with Velociraptor's Artifact Linux.Sys.LastUserLogin (truncated).

```
$ USERNAME="<username>"

# Find malicious users login
$ cat /var/log/secure | grep $USERNAME

# Collect bash history
$ cat /home/$USERNAME/.bash_history

# Find processes started by user
$ ps -aux | grep $USERNAME
# or
$ ps -u $USERNAME

# Examine processes started by user
$ export PID=1234 % <- PID of malicious user

$ kill -STOP ${PID}

# List files opened by the process:
$ lsof -np ${PID}

# Find network activity of process
$ netstat -tup | grep $PID

# Does the output of lsof correspond to /proc/{PID}/fd?
# Does the output of the netstat correspond to /proc/{PID}/fd

# Find further potential targets
$ sacct -X --user=$USERNAME -S 2023-01-01 -E 2023-03-31 --format=jobid,
    jobname,partition,account,alloccpus,state,exitcode,start,end,nodelist
```

Listing 18: Exemplary live investigation after user account breach.