Alternative scalable HIDS with investigation capability

Hosed based intrusion detection system without hashing

Bachelor thesis

In this thesis, I show how a host-based intrusion detection system can be built for scalability. More specifically, I show how the sleuth kit can be used to create an intrusion detection system that does not rely on hashing, but on filesystem attributes. This way, it gains speed, which enables us to take the risk not to calculate hashes.

Degree programme:	BSc in Computer Science
Specialisation:	IT Security
Author:	Julian Stampfli
Tutor:	Dr. Bruce Nikkel
Expert:	Armin Blum
Date:	June 11, 2019

Berner Fachhochschule | Haute école spécialisée bernoise | Bern University of Applied Sciences

Abstract

Many tools exist to help a company to protect itself against cyber attacks. One type of such a tool is called an intrusion detection system (IDS). Those are created to detect attacks that somehow got through other measures and infected one or many hosts in the network. One type of IDS is called network-based intrusion detection system (NIDS). They operate on a network level and analyze the incoming and outgoing traffic for anomalies. Those anomalies usually signal an intrusion. When they find such an intrusion, they usually generate an alert for a system administrator or security professional to analyze.

There is also another type of IDS called host-based intrusion detection system (HIDS). They operate directly on the host and try to find attacks there. They are more effective at finding intrusions that are dormant and don't do anything for some time. They mostly operate on the file system and sometimes go beyond that. HIDS have been quite effective at finding intrusions on file basis in the past by creating cryptographic hashes of hashes and comparing them to previous executions. However, as file sizes have been growing, they began to struggle to execute within a short time frame. Calculating a hash is seen as the most reliable way to find changes to the file system, and with more data, it is taking increasingly long to calculate them. The situation has grown out of proportions because the time to scan now takes so long, that the intrusion detection system can't reliably find intrusions within a useful timespan.

In my thesis, I try to show a different solution to this problem. I created a host-based intrusion detection system that works at a file basis but does not calculate any hash. Instead, it finds intrusions by evaluating the file system attributes like modification time and permissions. This approach is risk-based because it is less reliable, but by increasing the speed, the host can be scanned multiple times more often than if hashes get calculated. I am using an open source forensic investigation tool called the sleuth kit (TSK). It offers much functionality for file system analysis and works on most operating systems. With this tool, I can extract the file system attributes reliably and fast, without touching the files themselves.

There is another advantage that I hope to give with my system. Forensic investigators usually struggle to reliably create a timeline of what happened on a file system after an intrusion. This timeline is essential because it can lead them to find out what exactly happened and often can make future intrusions harder. Here I want to help as well. Other than the other HIDS, my system stores all the executions. This way, an investigator can look at this history and sees how the attack started. One nice side-effect of that is that my system is very flexible. After changing something on the host, the system automatically adjusts.

With my tool, system administrators and forensic investigators have another option to tackle intrusion detection. Taking the risk-based approach can lead to many fast detections that otherwise would not be detected in time. Additionally, investigators have more data at their disposal to investigate incidents and learn valuable knowledge of the attacks and attackers.

Acknowledgements

Firstly, I want to tank my tutor Bruce Nikkel and for his outstanding guidance and ongoing support during this project.

Next, I want to thank Amir Blum for supervising this work as my expert and for giving valuable feedback at the start of the project.

Further, I want to thank the open source community behind The Sleuth Kit (TSK) and pytsk3 without whom this project would not have been possible. Especially Joachim Metz who is very active in the pytsk3 community and helped me find the solution to one of my issues.

Contents

Abstract **i**

Acknowledgements **iii**

1. Introduction **1**

2. Technical Background **3**

2.1. Definitions 3

2.2. Proposed solution - FIDS 8

2.3. Scope 9

3. Results **11**

3.1. System Architecture 11

3.2. Application Flow 20

3.3. Examples 23

3.4. Timeline Creation 29

3.5. Security 30

4. Discussion **37**

5. Conclusion **39**

5.1. Future Work 39

Declaration of authorship **41**

Glossay **43**

List of figures **45**

List fo tables **47**

List fo code listings **49**

APPENDICES **51**

A. Project Management **51**

A.1. Goal 51

A.2. Workpackages 52

A.3. Planning 52

A.4. Validation of Workpackages 52

B. Implementation Decisions **57**

B.1. Programming Language 57

B.2. Language for Configuration File 58

B.3. Database Management System 59

B.4. Output / Alerting . . . 59

C. Journal **61**
C.1. Week 01 18.02.-24.02. . . . 61
C.2. Week 02 25.02.-03.03. . . . 61
C.3. Week 03 04.03.-10.03. . . . 62
C.4. Week 04 11.03.-17.03. . . . 62
C.5. Week 05 18.03.-24.03. . . . 63
C.6. Week 06 25.03.-31.03. . . . 64
C.7. Week 07 01.04.-07.04. . . . 64
C.8. Week 08 08.04.-14.04. . . . 65
C.9. Week 09 15.04.-21.04. . . . 65
C.10. Week 10 22.04.-28.04. . . . 66
C.11. Week 11 29.04.-05.05. . . . 67
C.12. Week 12 06.05.-12.05. . . . 67
C.13. Week 13 13.05.-19.05. . . . 68
C.14. Week 14 20.05.-26.05. . . . 69
C.15. Week 15 27.05.-02.06. . . . 70
C.16. Week 16 03.06.-09.06. . . . 71
C.17. Week 17 10.06.-16.06. . . . 72
C.18. Week 18 17.06.-23.06. . . . 72

D. Meeting Notes **75**
D.1. First Meeting (28.02.) . . . 75
D.2. Second Meeting (07.03.) . . . 75
D.3. Third Meeting (21.03.) . . . 76
D.4. Expert Meeting (28.03.) . . . 77
D.5. Fourth Meeting (11.04.) . . . 77
D.6. Fifth Meeting (02.05.) . . . 78
D.7. Sixth Meeting (20.05.) . . . 78
D.8. Seventh Meeting (07.06.) . . . 79
D.9. Eight Meeting (12.06.) . . . 79

E. User Documentation **81**

F. Content of CD-ROM **83**

1. Introduction

Attacks on computer systems have become highly prevalent. This situation resulted in the creation of many tools that support the securing of said computer systems. Those tools have different purposes and are grouped into different families. Some of those families are responsible for keeping any malicious activity outside of the network while others try to find out whenever something manages to come through. Those types of system are named intrusion detection system (IDS). An IDS generally works by tracking and evaluating some form of activity from hosts and networks and then tries to find anomalies. They operate by comparing the current activity to some configuration which usually contains a form of a whitelist and blacklist. Found anomalies are then alerted or logged such that investigators and system administrators can evaluate the anomaly and depending on the result of the evaluation, take action. **hidsnids**

There are two flavors of IDS, one, which deals with data from the network, named network-based intrusion detection system (NIDS), and another which evaluates data from the host system, named host-based intrusion detection system (HIDS). NIDS are used to detect unusual behavior of the network like communication from hosts which usually don't communicate or suspiciously large amounts of data that is transferred. HIDS, on the other hand, are used to detect anomalies on a host. This process works by detecting changes like which processes are started or when the file system is changed. Both types of IDS have various advantages, and they should be used in conjunction for best results. **hidsnids**

This thesis is about the writing of a HIDS by analyzing changes on the file system through file integrity monitoring (FIM). As already mentioned, a HIDS operates on the host machine and detects anomalies by comparing resources available on the host. For FIM a HIDS usually calculates a cryptographic hash function for each file and compares them to previous executions. If the output of the cryptographic hash function changes, then the file has been altered. If this alteration is unexpected and detected as an anomaly by comparing it to a configuration, it is alerted. This approach has one weakness. The calculation of a cryptographic hash functions takes time.

Additionally, to calculate the hash function, the HIDS needs to read the whole content of the relevant files. Because those files are stored on some storage medium, it takes even more time for the whole process of reading the file from the file system and calculating the hash **hash:slow**, **hash:speed**. This issue is, from a historical perspective, not relevant, as it was efficient enough that the entire file system could be hashed in a small amount of time. However, storage media grew, and with it, the amount of data on a server **bruce:imaging**. With that, the calculation of cryptographic hash functions needs more time. So much indeed, that traditional HIDS can't scan big systems within a valuable amount of time anymore.

There is another way. Instead of reading the entire file and calculating the hashes for each, the file system already contains some information that can be valuable for FIM. The file system contains much metadata. Things like modification date, permissions, and file size. Accessing this type of information is cheap, and it can be achieved without touching any contents within the file. This data can be used for FIM and finally, to find anomalies **inode**. While not as reliable as hashing, the speed gain can justify sacrificing a small fraction of reliability. This risk has to be considered because finding an intrusion fast can be the difference between an attempted intrusion and a successful attack with possible exfiltration of data.

Another advantage that this HIDS has is the built-in support for forensic investigations. Sometimes it is not sufficient to find intrusions, because every IDS can miss some intrusions which then need manual investigation. For those investigations to be successful, more data is better because it makes it easier to reconstruct the attack. This way, they can evaluate what has happened and assess the risk. Most HIDS solutions don't offer much help in this regard. The solution developed within this thesis, however, has this idea of a forensic investigation built-in.

2. Technical Background

In this chapter, I provide some technical introduction to relevant topics. Further information is available in the linked resources.

2.1. Definitions

2.1.1. Filesystem

Storage media takes many forms. There are the traditional Hard Disk Drives (HDD) and newer Solid State Drives (SSD). Both are built based on different assumptions and using different technologies. There exist more storage media like Compact Discs (CD), Digital Versatile Discs (DVD), Universal Serial Bus (USB) flash drives and more. Storage media operates on blocks of data, which are typicaly 512 bytes or 4 Kilobyte (KB) in size. **bruce:imaging**

This creates a problem when an Operating System (OS) want to use a storage medium to store data. The OS generally wants to store files which can be any size from some bytes up to several Gigabyte (GB). File systems were created that the OS does not need to be concerned with data blocks. They create a layer of abstraction for the storage medium and offer the OS a simple and standard way to access files. Another benefit of file systems is that they store metadata such as timestamps, permissions, and attributes for each file. This data helps the OS to identify relevant data.

The file system has it's own datastructure. It uses so called inodes to store the metadata and uses the blocks of the storage medium to store the files themselves. Which metadata is available heavily depends on the file system used. There are some standard metadata, that almost all file system support, but there are also more metadata that only certain file system offer **bruce:imaging**.

2.1.2. Cryptographic Hashing Function

A hash function is, in general, a function which takes inputs of unspecified size and generates an output of a fixed size. Hash functions are used widely in computer science. One examples are programming and databases where they are used to access certain data within a data structure easily. By design, hash functions have collisions, meaning that multiple inputs generate the same output. This statement must be true if you consider the unlimited input size and limited output size. If there are more inputs than outputs, there must be at least one output value that is assigned to multiple inputs. For data storage and other use cases this is not a huge problem because collisions can be handled, and if the hash function has good distribution, collisions are unlikely. Additionally, in many systems, a hashing function with weak collision resistance is deliberately chosen because it is usually faster to execute. **hash:noncrypto**, **hash:slow**

In a cryptographic context, this tradeoff cannot be taken. Two big factors play into why not. Firstly, in cryptography, hashes are often used as an assurance that the content of some data has not changed. If collisions are easy to find, the data can be altered in ways that result in the same hash, meaning that the hash no longer fulfills the use case. Additionally, monetary incentive can be big. If a collision can be provoked, even if challenging, data can be changed. This data could be a legal document or a bank transaction, neither of which we want to change. For those reasons, a cryptographic hash function needs to be highly collision resistant. The drawback of collision resistant algorithms over simpler hashing functions is the number of operations needed for the calculation, while current hashing functions are performant and secure, they still take some time for much data **crypto**.

One cryptographic hash function is called Secure Hashing Algorithm - 256 (SHA-256). SHA-256 is a standard published by the National Institute of Standards and Technology (NIST). It creates a 256-bit output and has not yet been successfully attacked **sha**, **crypto**. This hashing algorithm is used in the implementation of the HIDS to make sure that the configuration file has not changed between multiple runs.

2.1.3. HIDS

A HIDS works by detecting changes on the local host. It does that by looking at files, processes, configuration, logs, or other indicators. In this thesis, I focus on FIM. It is important to note that the other origins mentioned are also a valuable source of information. Any of those might have Indicators of Compromise (IoC). Especially running processes and the configuration can hold relevant data. However, this implementation only covers file-based information due to the lack of resources.

In a HIDS that uses FIM, all the valuable information comes from unexpected changes to the file system. Many hosts do not have any changes on the file system except for software updates. Additionally, the file system usually contains too much information to cover in a handwritten configuration. Thus, the easiest approach is to compare the current state of the file system to previous ones. If it has changed significantly or in an unexpected way, something foul might have happened or might still be happening. The negative side effect of this approach is the false positives that come from legitimately changing the host. Those legitimate changes could be new versions of the webpage that is running, newly updated configuration or updated ssh keypairs. However, those changes are scheduled, and the alerts can then be quickly checked and acknowledged.

A good HIDS should be able to handle such valid changes without change on the configuration or other changes on the system. Additionally, it should be able to find intrusions reliably and in a timely fashion. Maybe, it is too late if the HIDS finds an intrusion a week after the infection. To detect changes on the files reliably prominent HIDS calculate a hash of the files and compares that hash to previous runs. If a cryptographic hash function is used, each file always generates a unique hash which can not be forged. The main drawback of using cryptographic hash functions is that they take a long time to compute for significant amounts of data. Some implementations thus offer to use non-cryptographic hash functions. The obvious drawback of this approach is that a collision can be generated and such a file can be altered or replaced without the HIDS noticing.

In the following sections, I present three of the most used HIDS, Tripwire, Aide, and Samhain. They built the main competition of the integration developed in this thesis.

Tripwire

In 1992 the first HIDS named tripwire was created and publicly released as a free tool. In 1997 the creator of tripwire then created the company Tripwire Inc. and bought the naming rights for tripwire. The free version was monetized, and they released new versions of tripwire **Tripwire:Impl**, **Tripwire:company**. Now they mostly market to enterprise and industrial customer and have multiple products for securing computer systems and networks. **tripwire**

From the documentation that they provide to potential customers, it is unclear how their product works exactly. This approach makes sense because this way, they can keep competitors at bay. However, they have some information about how their product works to attract new customers. They claim to find changed data in real time. Additionally, they seem to be able to know what part of the documents changed. They then use another product of theirs called ChangeIQ to determine if the change was malicious or not. **tripwire:fim:datasheet**, **tripwire:true:intent**

For this approach to work, they need to create a big database with the content of each file. Then tripwire probably is hooked into the kernel to get notified about changes to the file system in real time. This approach has benefits over hashing and over checking attributes because it is faster than the former and more reliable than the latter. However, for that to work, the system needs to be running at all times. Additionally, not every OS supports this kind of kernel level information. I could not find out how they find intrusions should they ever be shut down and restarted at a later time or on OS on which this information is not available.

What has to be said, tripwire does not directly compete with my solution. Tripwire is a commercial product with integration into monitoring and fancy user interfaces. They have much financial incentive to continuously improve their product and have high integration with an extensive suite of security tools aimed at big cooperations. I encourage people working for this type of company to have a look at tripwire. However, because it mainly focuses on those big corporate customers, it might be too expensive for smaller teams. My HIDS is more focused on smaller companies, teams within large companies, and participants of the open source community and is entirely free.

Even though the direct comparison might not even be relevant, my system has some advantages that I want to list here. It does not need a large amount of storage. For tripwire to work, it needs to store files entirely or partially to compare them to previous states accurately. This file storage might not be a problem for small files, but it does take much space if many machines are scanned or the amount of small files is extensive. My system only saves metadata, which is considerably less space.

Additionally, my system does not need to run at all times. This approach reduces the load, which is especially essential on high-performance machines, particularly if they have a lot of read-write operations. One final advantage is the fact that my system is open source. With that, any potentially interested party can look at the source code and see if the product is appealing to them. Additionally, they can check the source code for security issues and can find out exactly how the system works. This information is not available for tripwire customer. They have to trust the company behind it.

AIDE

Aide is an open source alternative to tripwire. It was created after tripwire went commercial. As aide is open source, it is easier to find information about how it works, which I detailed below. **aide:totherescue**, **aide:github**

When aide is first executed, it generates a reference database. This database contains all the information for each file that is within a path of interest. Depending on the config it contains more or less information, including cryptographic hash functions. Each subsequent execution then compares the files found to this initially created database. The database needs to be updated manually if the system has a scheduled change. It generates a log of all the changes and alerts all of the changes at once. Both the configuration and the database are usually stored on the file system that is scanned. **aide**, **aide:doc**

Aide can compare multiple cryptographic hash functions and non-cryptographic hash functions and some file system metadata. It runs on many modern Unix system **aide** and uses Pretty Good Privacy (PGP) keys to sign their releases. It has a strong community behind it, but only offers a comand line interface (CLI) and has no fancy user interface and is written in the programming language C. **aide:github**

Aide also features a rich configuration. It is based on rules. There are some preconfigured rules for each hashing function and for the checked file system metadata. Those rules can then be combined in new rules to create custom settings. Finally, the configuration offers the possibility to specify paths which should be scanned with the rules that should be applied. This way, different directories can be scanned with different settings. **aide:conf**

Samhain

Samhain is another open source HIDS. In comparison to aide it has a company backing it which offers support for customers. However, other than tripwire it is not necessary to use their services to run samhain. For that, they have a lot of publicly available documentation and community support. **samhain:company**

Samhain has a big focus on centralized management of the HIDS. All hosts run the system and report their findings to a single instance which is used for management and monitoring. The hosts also get the configuration from the server. For scanning, it has two modes. Firstly, it can scan the system similarly to aide by going through all the files and mostly generating hashes. However, it can also create a hook in the Linux kernel to find changes live. It uses both approaches to find intrusions. Contrary to aide samhain offers more functionality than FIM and can check open ports, processes and more.

Compared to my system, it can leverage the kernel integration to find intrusions faster. However, it still relies on hashing for scanning parts of the file system, especially on non-Linux systems. This part of samhain is improved by my system. Overall, samhain offers more functionality than FIM, which my system currently does not. For kernel integration, it needs to be running at all times. As already discussed in the tripwire section, this can be a drawback.

2.1.4. NIDS

Compared to a HIDS, a NIDS can be used to detect intrusions from multiple hosts at the same time. NIDS are used to analyze anomalies on the network. An anomaly on the network might be an unexpected session from a server which usually doesn't communicate with the internet or unusually large or frequent traffic. Certain intrusions can be tracked by analyzing the content of the packages. Some forms of intrusions are easier to detect on the network. Especially if they extract much data or are very aggressive at spreading within a network. The main advantage here is that multiple sources can be combined on the network level. Another advantage of NIDS is that current malware almost always contains some network communication **Malware:Behaviour**, **nids**.

However, a NIDS doesn't only have advantages. Certain kinds of attacks can only be detected on the network with much difficulty. For instance, a corrupted file upload which allows uploading of any file type can hardly be found by checking the network traffic. For best results, both types of IDS can be used in conjunction to detect attacks.

In this thesis, NIDS are not in the center. The implementation doesn't look at network traffic or mimic other NIDS functionality. However, as mentioned above, I highly encourage everyone to use an appropriate NIDS to improve the chance to find an intrusion.

2.1.5. The Sleuth Kit

TSK is an open source toolkit used to investigate disk images. It is based on The Coroner's Toolkit (TCT) **tct** and contains multiple command line interfaces and an Application Programming Interface (API) for various purposes **tsk**, **tsk:about**. It is mainly written in the programming language C, runs on most OS and is used to analyze numerous file systems. It is heavily used in forensic investigations to find deleted or corrupted files and for other information gatherings on a disk image.

It contains many tools which are used for specific purposes. Those include analysis of partitions, blocks, inodes, filenames, journaling file system, timelines, and more. In this thesis, I am mainly concerned with the tool that operates on filenames. This tool is called fls, which I described in the next section.

fls

Fls can be used to access directories, files, and their attributes. With it, the directories can be displayed recursively, and for each file, the attributes can be printed to the console. **tsk:fls**

This tool also can be accessed via the API and is used in the HIDS implementation to retrieve the directories, files, and attributes. The process of how this is achieved is explained in section 3.2.1.

2.1.6. Python

Python was the programming language of choice for this product. Python offers many advantages over other languages. Some of those are:

- Platform independence
- Good community support in the forensic community
- Active library for TSK named pytsk3
- Small overhead
- Easy for prototyping
- Easy to read

This decision was not made very lightly. Other programming languages were considered. For more information on those refer to appendix section B.1.

2.1.7. pytsk3

Pytsk3 is the library mentioned above that creates bindings for python to the API of TSK. As TSK, this library is open source and is still active. It is hosted on github and offers most of the functionality of the TSK API. Pytsk3 is used extensively in the scanner part of the implementation. For further information refer to section 3.2.1.

2.2. Proposed solution - FIDS

The main principle behind a HIDS is the detection of changed files. As already discussed in section 2.1.3, most tools rely on the calculation of hashes. This process is generally a good approach since changes can be found very reliably; however, as already mentioned, it can drastically hinder the performance of the HIDS. Sadly the actual performance lost cannot be clearly stated, as it heavily depends on what hardware is in use. However, considering the computational overhead of calculating a cryptographic hash function it is clear that the time it takes grows with bigger file systems. As storage medium has grown from a Megabyte (MB) to Terrabyte (TB) so has the requirement to store more data. Calculating hashes over such big systems is not viable as it can take a very long time. **hash:slow**, **hash:veryslow**, **hash:speed**

The name of this proposed solution is the forensics-based intrusion detection system (FIDS)

2.2.1. Time issues

In this thesis, I propose a different approach, the FIDS. Ignoring the hashing and the file content and focus on the file system metadata instead. This approach should be sufficient to find intrusions and thanks to not calculating hashes, the FIDS can execute more often. If a conventional HIDS might take several days to complete a run on a file system with multiple TB of data, the FIDS would take some minutes to hours. It could then be run multiple times a day and find new and changed files within a short amount of time. It is possible that the FIDS misses some changes if the attacker can change all the relevant metadata before the system rechecks the file. However, this risk is small in comparison to not finding an intrusion for multiple days. It is also possible to scan highly critical sections of a system with a traditional HIDS and the rest with the FIDS. This way, one has both advantages. The whole system can be checked in a timely fashion with the FIDS, and for certain smaller parts of the system, a general HIDS can detect changes by using cryptographic hash functions like SHA-256.

The FIDS uses TSK via pytsk3 to extract the file system metadata. The main advantage that this gives is the interoperability with various file systems. Additionally, by directly accessing the attributes from the file system, no metadata is changed. The files themselves are never touched. Furthermore, it can also be used to get the files and attributes of an image that has been extracted or of a virtual machine.

2.2.2. Investigation capabilities

Another improvement upon existing HIDS, is that investigation capabilities are built in from the start. Traditional HIDS are good at finding intrusions and alerting them. However, they don't offer much support for the investigation of the incident. They don't have information beyond what was configured. This lack of information makes sense if cryptographic hash functions have to be calculated. If something was incorrectly configured, an investigator does not have any useful output of those systems to gain further knowledge. They generate alerts and not much beyond that.

The FIDS stores all the available metadata for each scanned file for every run. This way, an investigator can use this database to gain more information about how the attacker proceeded. He can look at the changes of permissions, modification of files, and more, even if the system did not find any intrusions. It is also possible to generate a timeline out of this data to form an expansive view on what happened when on the file system.

2.2.3. Flexibility

Aide works by comparing each execution to the initial one. This approach is practical as it detects one intrusion multiple times. However, it generates many messages, and users are then less likely to take them seriously because of false positives. Additionally to that, after a legitimate change to the system, aide always generates alerts until the initial run is reset. This behavior can lead to undetected intrusions a short time after an upgrade, which is often used by attackers to gain a foothold because of the high number of alerts created by the upgrade. Thus, it is possible that an intruder can gain access shortly after an update which is alerted by aide but ignored because of the false positives. Aide is only used as an example here, other HIDS might work similarly and would give the same result.

The FIDS does not have this issue, at least not as strongly. As all the data gets collected for each run, it makes sense to compare each run to the previous. This approach has the benefit of legitimate changes being considered to be acceptable one run after they happened. This acceptance results in overall fewer messages, which increases the importance of each. If it is required that the system should always compare against one specific run, this could also be done. The FIDS only needs to be configured to not check against the latest, but a specific, predefined run. Only the configuration then needs to be changed to update this predefined run.

2.3. Scope

In this project I create a HIDS which uses TSK to detect changes named FIDS. It covers the three main changes discussed in section 2.2. The FIDS uses a SQL database to store the executed runs. It includes user documentation and this thesis documentation. It also contains the creation of a timeline for investigation purposes mentioned in section 2.2.2.

Out of scope are extensive alerting functionality and extensive example configurations for commonly used OS and tools. Furthermore, any big data analysis of the runs, while very interesting, are also out of scope.

3. Results

In this chapter, I will talk more about the implementation details of the FIDS. I will also look into the security of this implementation and what measures can be implemented to improve it.

3.1. System Architecture

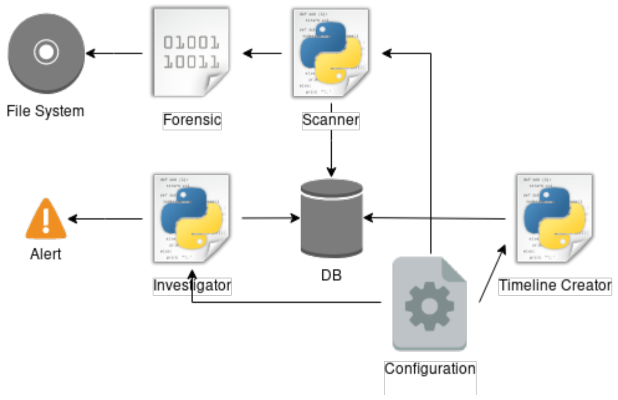

Figure 3.1.: System Architecture

As seen in figure 3.1, the FIDS contains two main components, the scanner and the investigator, a database and has connections to some libraries. It is split into scanner and investigator way to gain flexibility, mainly so that both components can be executed independently. This independence has the advantage that the scanner and the investigator don't need to be run on the same system. Also, it means that previous results of the investigator can be recreated by running the investigator on previously captured data. Additionally, if someone is only interested in the results of the scanner, the investigator doesn't need to be run. The scanner is explained in detail in section 3.2.1 and the investigator in section 3.2.2.

Besides the scanner and the investigator which were produced in this thesis, there are five other components which build the environment of the HIDS. There is the file system which contains the data in which we are interested. As already mentioned, this could be a mounted device that is directly accessible through the operating system. It could also be run on the disk images of virtual machines or previously created disk images. To run it on a previously created disk image might be interesting if used on backup images if backups are made by creating disk images.

Then there is the forensic component. This component is the combination of TSK and pytsk3. The main purpose of this component is the abstraction of the file system into python classes, which can then be used within the scanner. This abstraction is important to be compatible with multiple different file systems. TSK offers much different functionality; the FIDS is only using the utility that is used for the fls command described in section 2.1.5.

This leaves the configuration and the database. Both of which are also part of this thesis. The database is used to store the data of each run, and the configuration is used to change the behavior of the FIDS. The configuration is explained in depth in section 3.1.1 and the database in section 3.1.2.

There is another component that is not part of the HIDS functionality. The FIDS contains a component that is used to create a timeline. This component is special because it is designed for use by forensic investigators. It is explained in section 3.2.3.

3.1.1. Configuration

The configuration file is provided in YAML Ain't Markup Language (YAML). YAML is a human-friendly data serialization language with support for many programming languages. Its main advantage over other languages for configuration files is the readability and the ease to extend already existing configuration files. There were more reasons why YAML is used documented in appendix section B.2.

Database Configuration

The configuration consists of three parts. The first part is the database configuration. The current implementation supports only SQLite, as this is the most basic and easy to use format. Additionally, it doesn't require any connection to an external entity. For more information on why SQLite was chosen see appendix section B.3. For the configuration, only the filename is required as seen in listing 3.1. This configuration defines where the data will be sent to and read from in the scanner and investigator parts, respectively. As both parts need access to the database, this part of the configuration is in a separate part. It is possible and already prepared to extend the system to use more Database Management Systems (DBMS). However, this is not part of the scope of this thesis.

Listing 3.1: SQLite Configuration

```yaml
sqlite:
        filename: fids_db3.db
```

Scanner Configuration

The second part of the configuration is the part that defines the scanner. An example config can be seen in listing 3.2. It consists of one main key, which is named scan. If this config entry is missing, the scanner part of the HIDS is not executed by default. Beneath this top key it contains the following subkeys:

image_path Path to the file system that is used for the scan.

scan_paths List of paths to scan. Paths are scanned recursively. "/" thus scans all available paths.

ignore_paths Paths that should not be scanned. Can be any directory. The recursion stops once this path is reached and does not continue downwards. Practical if certain directories are not interesting for intrusion detection.

Listing 3.2: Scanner Configuration

```
scan:
        image_path: /dev/nvme0n1p1
        scan_paths:
                [
                        "/",
                        "/nonExisting",
                ]
        ignore_paths:
                [
                        "/temp/"
                ]
```

Investigator Configuration

The investigator configuration is similar to the scanner configuration in the way that it contains a top-level node called investigator. If this is missing the investigator part does not start. As can be seen in listing 3.3 it is a lot more complicated than the scanner configuration.

same_config This configuration specifies whether a changed config should result in an alert. Defaults to True.

validation_run This can define a run that is used for validation. If empty the second last one will automatically be used.

Rules Rules are ways to create templates on how changed files should be found. Rules are explained more through below.

Investigations Investigations define which paths and which files should be checked for intrusions. Investigations are explained more through below.

Listing 3.3: Investigator Configuration

```
investigator:
        same_config: True
        validation_run:
        rules:
                - name: php
                  rules:
                        - m
                        - i
                        - l
                        - n
                        - a
                  equal:
                        - meta_creation_time
                        - meta_size
                - name: logs
                  greater:
                        - meta_size
                  equal:
                        - meta_creation_time
        investigations:
                - paths:
                        - '/etc'
                  whitelist_inverted: false
                  fileregexwhitelist:
                        - '*.php'
                  rules:
                        - php
                - fileregexwhitelist: '/*'
                  fileregexblacklist: '*evilfile*'
                  blacklist_inverted: false
                  rules:
                        - logs
```

Rules

Rules represent templates that are later used in the investigations to find anomalies. Rules can be based upon other rules to extend them. Recursive rules are allowed, they simply extend each other. An example of the rule configuration can be seen in listing 3.3, the fields are explained below.

name Each rule has a name. If two rules with the same name are defined, the behavior might be inconsistent. The name is used to extend rules.

rules The rules which are used as a basis for this rule. They are referenced by name.

greater The properties which are allowed to grow between the runs. Greater also includes equal.

equal The properties that are supposed to stay equal during all runs.

Additionally, there are some predefined rules. The aide configuration heavily influenced which rules got predefined. The preconfigured rules are listed in listing 3.4.

Listing 3.4: Preconfigured Rules

```
rules:
        - name: p
          equal:
                - meta_mode
        - name: ftype
          equal:
                - meta_conten
        - name: i
          equal:
                - meta_addr
        - name: l
          equal:
                - meta_link
        - name: n
          equal:
                - meta_nlink
        - name: g
```

```
            equal:
                - meta_gid
        - name: s
          equal:
                - meta_size
        - name: m
          equal:
                - meta_modification_time
                - meta_modification_time_nano
        - name: a
          equal:
                - meta_access_time
                - meta_access_time_nano
        - name: c
          equal:
                - meta_changed_time
                - meta_changed_time_nano
        - name: S
          greater:
                - meta_size
```

Additional to those, there are some rules which do not directly check the file system metadata. There are three of them which change the behavior of the investigator. They must be configured on the investigation level. They are directly referenced by the investigator to specify if the related events should result in an alert or not.

file_rename_ok Usually a changed filename leads to an alert. This behavior is deactivated by setting this rule.

new_files_ok Usually a new file leads to an alert. This behavior is deactivated by setting this rule.

deleted_files_ok Usually a deleted file leads to an alert. This behavior is deactivated by setting this rule.

Investigations

The investigation configuration defines the objects which are scanned for intrusion, as well as how they are scanned. They contain rules and some configuration on what should be scanned. An example of the investigation configuration can be seen in listing 3.3, the fields are explained below.

paths For investigations different paths can be defined. This setting changes the behavior in such a way that only the specified path are investigated. Useful if certain paths need different observations.

rules The rules which are used within this investigation is based on are defined here. They are referenced by name.

fileregexwhitelist The regex whitelist works similarly to the paths config. Only files are analyzed that match this regex. It is parsed before the blacklist.

whitelist_inverted This configuration is to invert the whitelist to match all files that are not covered in the regex.

fileregexblacklist The regex blacklist is used to detect files based on their filename. If their path with filename results in a match with this regex it is alerted. Especially useful for finding unexpected files in locations that have frequent changes.

blacklist_inverted This configuration is to invert the blacklist to match all files that are not covered in the regex.

3.1.2. Database

The database is another component that is shared between the scanner and the investigator. As already mentioned, multiple DBMS could be used in conjunction with the FIDS. In this section, I do not focus on the DBMS used but rather use general examples only using SQL. The database consists of five relations. Firstly, I explain some reoccurring types, how they are stored, and what they represent after that, I explain the relations.

Some ideas are not specific to a relation. Most relations contain timestamps and Universaly Unique IDentifier (UUID). How they are stored can be seen below:

Timestamps To save space, timestamps are stored in the UNIX timestamp representation.

UUID IDentifier (ID) are stored as UUID in their hexadecimal representation.

Enum TSK defines many enumerations. Most of those enums have a number representation. To save space, this representation is stored in the database. The enums and their representation can be viewed in the TSK API reference. **tsk:file:header**

FIDS_RUN

The run relation is relatively simple. It contains an ID. This ID is used in the other relations as well to create a link to the run. Besides that, it contains a SHA-256 hash of the configuration that it was used. It then contains a start and end timestamp which represent the times when it started and when it ended. A run that is not yet completed only contains the start timestamp. The relation definition is shown in listing 3.5

Listing 3.5: Fids Run Table Definition

```
CREATE TABLE FIDS_RUN(
        id varchar(32),
        config_hash varchar(64),
        start_time int,
        finish_time int,
        PRIMARY KEY(id)
);
```

FIDS_ERROR

Each execution has the possibility of creating errors. Those errors are stored in this table. It is rather simple as well. Additionally to the run ID it contains an ID for the error as well. Next, it contains a description and a location where the error occurred. The table definition is shown in listing 3.6

Listing 3.6: Fids Error Table Definition

```
CREATE TABLE FIDS_ERROR(
        run_id varchar(32),
        id varchar(32),
        description text,
        location varchar(255),
        PRIMARY KEY(run_id, id)
);
```

FIDS_FILE

The file relation contains most of the information. Has an ID additional to the ID of the run. Then it has the path in which the file is located and all the metadata about the file. Most attributes start with 'meta' or 'name'. This naming is a reference to TSK which has separate structs for meta and name information about each file. I kept the naming of TSK. Thus more information about each attribute can be found on the API reference of TSK. There are also two indices, one on the inode address and one on the combination of path and file. Those are required to reduce the execution time of the investigator **tsk:file:struct**. The table definition is shown in listing 3.7.

Listing 3.7: Fids File Table Definition

```
CREATE TABLE FIDS_FILE(
        run_id varchar(32),
        id varchar(32),
        path text,
        meta_addr int,
        meta_access_time int,
        meta_access_time_nano int,
        meta_attr_state int,
        meta_content_len int,
        meta_content_ptr int,
        meta_creation_time int,
        meta_changed_time int,
        meta_creation_time_nano int,
        meta_changed_time_nano int,
        meta_flags int,
        meta_gid int,
        meta_link int,
        meta_mode int,
        meta_modification_time int,
        meta_modification_time_nano int,
        meta_nlink int,
        meta_seq int ,
        meta_size int ,
        meta_tag int ,
        meta_type varchar(255) ,
        meta_uid int ,
        name_flags int ,
        name_meta_addr int ,
        name_meta_seq int ,
        name_name int ,
        name_size int ,
        name_par_addr int ,
        name_par_seq int ,
        name_short_name int ,
        name_short_name_size int ,
        name_tag int ,
        name_type varchar(255) ,
        PRIMARY KEY (run_id , id)
);
CREATE INDEX inode ON FIDS_FILE(meta_addr);
CREATE INDEX fullpath ON FIDS_FILE(path , name_name);
```

FIDS_FILE_ATTRIBUTE

In TSK each file can contain multiple attributes. Those attributes are stored in this table. It contains the ID of both run and file and an additional one for the attribute itself. The attributes contain flags, a name and a type. The flags and the type enums called 'TSK_FS_ATTR_FLAG_ENUM' and 'TSK_FS_ATTR_TYPE_ENUM'. More information available in the TSK API reference **tsk:attr:struct**, **tsk:file:header**. The table definition is shown in listing 3.8

Listing 3.8: Fids File Attribute Table Definition

```
CREATE TABLE FIDS_FILE_ATTRIBUTE(
        run_id varchar(32) ,
        file_id varchar(32) ,
        id varchar(32) ,
        flags int ,
        tsk_id int ,
        name varchar(255) ,
        name_size int ,
        at_type varchar(255) ,
        PRIMARY KEY (run_id , file_id , id)
);
```

FIDS_FILE_ATTRIBUTE_RUN

Attributes can contain multiple data runs. Those data runs are represented in this relation. It contains the ID from run, file and attribute and one for the data run. It then has a block address and a lenght.

More information can again be found in the TSK API reference. **tsk:attr:run:struct** The table definition is shown in listing 3.9

Listing 3.9: Fids File Attribute Run Table Definition

```sql
CREATE TABLE FIDS_FILE_ATTRIBUTE_RUN(
        run_id varchar(32),
        file_id varchar(32),
        attribute_id varchar(32),
        id varchar(32),
        block_addr int,
        length int,
        PRIMARY KEY(run_id, file_id, attribute_id, id)
);
```

3.1.3. Shared

There are other shared parts in the source code. Mainly the model of the system. It contains multiple classes that define the types with one type for each relation. They can parse TSK objects and database rows. This way, both the scanner and the investigator can work with the same classes. Except for parsing, they don't have any functionality.

3.2. Application Flow

The main application flow consists of first reading the config file. FIDS used the default config path if not overwritten by passing the attribute. Then it starts the scanner if appropriate. After that, it starts the investigator if required. If any of the two configurations do not exist, the FIDS does not execute the respective component. If both are missing, it does neither scan nor investigate.

By passing the timeline flag, the HIDS-flow gets ignored. Instead, the FIDS creates the timeline and finishes the run.

3.2.1. Scanner

The scanner component is the first component of the HIDS path. It is responsible for getting all the information from the file system for the configured paths. It has multiple stages.

1. Initialization
2. Scan
3. Error Logging
4. Storing
5. Error Logging
6. Finalizing

In the initialization phase, the scanner creates a run object. The scanner immediately saves the run to the database. This way, users know just by looking at the database if there are any yet incompleted runs. This data also creates the first opportunity to look for inconsistencies. If there are a lot of started runs, something suspicious might be going on. It also creates the hash of the configuration and already saves it to the database.

The scan phase has more steps to it. First it creates the pytsk3 object called 'img_info'. Pytsk3 returns that when it receives the path to the disk image. Then the important 'fs_info' object is created by passing the img_info object. This way, the scanner has access to the file system. The scan starts by calling 'open_directory_rec' for each scan path. This function keeps track of all directories it already traversed into as to avoid circular loops and unnecessary steps. Then it checks if the path is in the ignored paths. It then iterates over all objects in the directory.

For all entries, it then checks if it is a valid entry with the required attributes to continue. Afterward, it checks if it is a directory. For all directories, the function first checks if it already visited the directory and if not it calls itself with the new directory as the parameter. If the entry is a file, it is parsed into a python object and then stored into a local list of found files. It also saves any errors that occur by creating an error object and storing them in a list of errors.

In the error logging phase, the scanner stores all errors that have occurred in the scan process to the database by iterating over them and saving one by one. Even if the scanner crashes when storing the files for some reason, it already persisted the errors.

The file saving phase occurs after the first error logging phase. Here the files are saved like the errors by iterating over them and saving one by one. Should any additional error occur while saving the files, the scanner adds them to a new list of errors.

There is a second error logging phase. In this phase, the errors that occurred while saving the files get stored into the database. The functionality is the same as for the first error logging phase.

In the finalizing phase, the run object gets updated, and the end timestamp added. It is then updated on the database. At this point, the scanner finished. It has written all the appropriate files from the file system and all errors that occurred to the database. The run object on the database has a fitting start and end timestamp.

3.2.2. Investigator

The investigator component is responsible for finding intrusions. As discussed in section 3.1.1 it does so with predefined rules and investigations. To explain how the investigator works, I split it into different steps as with the scanner.

1. Parsing configuration
2. Fetching runs
3. Fetching files
4. Investigation
5. Report generation

First, the investigator needs to parse the configuration. The investigator configuration is more intricate than the other configurations. Because of that, parsing needs more steps. The FIDS first flattens the alerts by expanding the 'greater' and 'equals' list appropriately to the sub-rules. Afterward, it expands the investigations by adding the configuration from the alerts directly to the investigations. This way, it is easier to access them in the following steps. The lists are continuously updated not to contain duplicates.

The investigator then fetches all run ID and selects the correct ones by finding the most recent finished runs. If the 'validation_run' property is set, it takes that one as the reference instead of the second most recent one. The investigator always takes the most recent for validation.

After the runs are acquired, the investigator retrieves the files by executing the SQL query shown in listing 3.10, which joins the file table with itself on the inode address or path and name. This way, the investigator can make good use of the indices that exist on this table. After the files get fetched, they are parsed and put into a list of tuples. This process takes some time for a large number of files.

Listing 3.10: SQL Querry for files

```sql
SELECT * FROM FIDS_FILE first
    LEFT JOIN FIDS_FILE second
        on (first.meta_addr=second.meta_addr or
            first.path=second.path and
            first.name_name=second.name_name)
    WHERE first.run_id = ? and second.run_id=?
        or second.run_id is null;
```

After that, the investigator starts the comparison. It first checks the configuration hashes of the two runs. Then it goes through each file and checks if it needs checking in any investigation by validating the path and regex whitelist. Then, it validates if the file has been renamed, created or deleted. It then checks the blacklist for a match. Next, it checks all the ‘equal’ and ‘greater’ attributes. For each check, it creates a detection error if necessary.

Ultimately, the alerts are used to generate a report. It then prints that report to the standard output and is meant to be used for further processing.

3.2.3. Timeline creator

The timeline creator part is quite straightforward. It first retrieves all the data for all files from the database. It then goes through each and prints a list with the body file format. This way, the output of the timeline creator can be passed to the ‘mactime’ utility, which parses the input automatically and creates an ASCII timeline. The only special part is the retrieval of the files. Instead of storing them in a list, the system stores them in a set. This way duplicated files with the same attributes are not considered twice. The set uses a non-cryptographic hash function for duplication detection.

3.3. Examples

In this section, I give some general use cases and the relevant configuration. I mainly use the Apache web server as an example. There are some assumptions on the directories and contents for both scenarios. Additionally, I assume that the file system is accessible through '/dev/sda1'.

For additional help, there is also a user-documentation in appendix section E

3.3.1. Webpage hosted in Apache

For those examples, I work under the assumption that the Apache web server is installed and has an 'htdocs' directory that is located in '/var/www/'. I assume that there is a webpage running with some PHP files. It also has HTML, CSS, javascript, and image files in a static folder. Then it has a log directory where the webpage itself stores all the logs and an upload directory where the user can upload all kind of files. 4 types of files are watched, some more and some less closely. I give Example Configurations each type. For the scanner, the configuration in listing 3.11 is appropriate.

Listing 3.11: Example Scanner Configuration

```

sqlite:
    filename: apache.db

scan:
    image_path: /dev/sda1
    scan_paths:
        [
            "/var/www/htdocs/",
        ]
```

PHP files

The web server has some PHP files in different directories. Those are used for various purposes, but they should never change, except for an update to the webpage. All of them lie in some subdirectory of the 'htdocs' directory. Thus they all are covered in the scan from the configuration in listing 3.11.

The tricky part is that we have multiple different PHP files in different directories. The configuration should cover that even if the exact location and folder structure changes a bit and new folders are added. As the scanner automatically scans recursively, the exact folder structure does not matter. The files are found when they are in the 'htdocs' directory. As the files should not change but might be accessed at any time, watching everything should be what we want. Especially noteworthy are the modified, changed, and created times. Also important are size, path, name, and inode.

This much for the rule, the investigation is then relatively easy. There is not a particular path that has to be watched since the scanner was configured only to scan 'htdocs'. By using the same database, the investigator only validates this data as well. Important though is the whitelist. There is an upload directory in which we expect changes. Thus a file name regex of '.php$' successfully narrows the files to only the required ones.

An example investigator configuration is listed in listing 3.12. For this configuration, I did not use the preconfigured rules.

Listing 3.12: Example PHP File Configuration

```
sqlite:
    filename: apache.db

investigator:
    same_config: True
    rules:
        - name: equal_but_accessible
        equal:
            - meta_creation_time
            - meta_creation_time_nano
            - meta_modification_time
            - meta_modification_time_nano
            - meta_changed_time
            - meta_changed_time_nano
            - meta_size
            - path
            - meta_addr
            - name_name
        greater:
            - meta_access_time
        investigations:
            - fileregexwhitelist: '.php$'
              rules:
                - equal_but_accessible
```

Upload directory

As already mentioned, there is an upload directory. Creating a correct configuration for this directory is more challenging than for the PHP files since any files might get uploaded and deleted from there all the time. The critical part here is that no one can upload a PHP file. If this was possible, an intruder could create any mischief. This intrusion already gets caught by the rule from listing 3.12 because it already scans for PHP files in the whole directory. It is still viable to create an additional rule that only catches those types of intrusions. Maybe this investigation is then executed more often than the other. Additionally, this configuration also checks for additional properties and alerts a PHP file in the upload directory until it has been deleted

For the rule we need to add the special rules 'new_files_ok' and 'deleted_files_ok' since both are valid in this directory. Then generally greater access and modification times are ok. For the investigation, a file regex blacklist is added. Additionally only the upload directory needs to be searched. With that, valid PHP files in other directories are easily excluded. The example configuration is shown in listing 3.13

Listing 3.13: Example Upload Directory Configuration

```
    sqlite:
        filename: apache.db

    investigator:
        same_config: True
        rules:
            - name: upload
              greater:
                - meta_modification_time
                - meta_changed_time
                - meta_access_time
            investigations:
                - fileregexblacklist: '.php$'
                  paths: [
                    "/var/www/htdocs/upload/"
                  ]
                  rules:
                    - upload
                    - new_files_ok
                    - deleted_files_ok
```

HTML, CSS, Javascript and images

The webpage also has some static context, namely HTML, CSS, Javascript, and image files. Those are in a directory called 'static'. Those files are like the PHP files; they should not change unless someone creates an update to the page. In this static folder, only those files are acceptable, in the configuration, this results in a blacklist for those file types which is negated. This way, files that don't equal these file types are alerted. Additionally, the previously used rule named 'equal_but_accessible' should do the trick to find modified files. An example configuration is shown in listing 3.14

Listing 3.14: Example Static Files Configuration

```
    sqlite:
        filename: apache.db

    investigator:
        same_config: True
        rules:
            - name: equal_but_accessible
              equal:
                - meta_creation_time
                - meta_creation_time_nano
                - meta_modification_time
                - meta_modification_time_nano
                - meta_changed_time
                - meta_changed_time_nano
                - meta_size
                - path
                - meta_addr
                - name_name
              greater:
                - meta_access_time
        investigations:
            - fileregexblacklist: '.(js|css|png|ts|jp(e)?g|htm(l)?)$'
              blacklist_inverted: true
              paths: [
                "/var/www/htdocs/static/"
              ]
              rules:
                - equal_but_accessible
```

Log directory

With that, the upload directory remains. Log files usually grow in size until they reach a certain limit. Then they stay static, and a new file is created, which then again grows. To create a configuration for that, it is important to check the size for growing. Additionally, all the timestamps should also only grow. One important detail is the 'file_rename_ok' rule as the name of rolled over log files might change. Also, the 'new_files_ok' rule needs to be added. The example configuration is available in listing 3.15

Listing 3.15: Example Log Directory Configuration

```
    sqlite:
        filename: apache.db

    investigator:
        same_config: True
        rules:
            - name: logs
              equal:
                - meta_creation_time
                - path
                - meta_addr
              greater:
                - meta_modification_time
                - meta_changed_time
                - meta_size
                - meta_access_time
        investigations:
            - paths: [
                "/var/www/htdocs/logs/"
              ]
              rules:
                - logs
                - file_rename_ok
                - new_files_ok
```

Complete configuration

The partial configurations can then be combined into one bigger configuration. Some rules can be reused, which makes everything a bit easier. The complete configuration is shown in listing 3.16.

Listing 3.16: Complete Example Configuration

```
sqlite:
    filename: apache.db

scan:
    image_path: /dev/sda1
    scan_paths:
    [
        "/var/www/htdocs/",
    ]

investigator:
    same_config: True
    rules:
        - name: upload
          greater:
            - meta_modification_time
            - meta_changed_time
            - meta_access_time
        - name: equal_but_accessible
          equal:
            - meta_creation_time
            - meta_creation_time_nano
            - meta_modification_time
            - meta_modification_time_nano
            - meta_changed_time
            - meta_changed_time_nano
            - meta_size
            - path
            - meta_addr
            - name_name
          greater:
            - meta_access_time
        - name: logs
          equal:
            - meta_creation_time
            - path
            - meta_addr
          greater:
            - meta_modification_time
            - meta_changed_time
            - meta_size
            - meta_access_time
    investigations:
            - fileregexwhitelist: '.php$'
              rules:
                - equal_but_accessible
            - fileregexblacklist: '.php$'
              paths: [
                "/var/www/htdocs/upload/"
              ]
              rules:
                - upload
                - new_files_ok
                - deleted_files_ok
            - fileregexblacklist: '.(js|css|png|ts|jp(e)?g|htm(l)?)$'
              blacklist_inverted: true
              paths: [
                "/var/www/htdocs/static/"
              ]
              rules:
                - equal_but_accessible
            - paths: [
                "/var/www/htdocs/logs/"
              ]
              rules:
                - logs
                - file_rename_ok
                - new_files_ok
```

3.3.2. SSH private keypairs

Another example for the FIDS is the tracking of private and public key pairs. Usually, the web server reads them when it is started. Afterward, it caches them in the memory, and the actual files should not be reread. This situation causes an updated access time to be an IoC of a possible intrusion. For this scenario, I assume that the key pair is located in the directory '/var/www/.ssh/'. An example configuration is shown in listing 3.17.

Listing 3.17: Example SSH Keypair Configuration

```
sqlite:
    filename: ssh.db

scan:
    image_path: /dev/sda1
    scan_paths:
    [
        "/var/www/.ssh/",
    ]

investigator:
    same_config: True
    rules:
            - name: equal_not_accessible
              equal:
                - meta_creation_time
                - meta_creation_time_nano
                - meta_modification_time
                - meta_modification_time_nano
                - meta_changed_time
                - meta_changed_time_nano
                - meta_access_time
                - meta_access_time_nano
                - meta_size
                - path
                - meta_addr
                - name_name
        investigations:
            - rules:
                - equal_not_accessible
```

3.4. Timeline Creation

The Timeline creator works by taking the database output of the scanner and transforming the file relation into the body format used by tools like mactime. This way, the FIDS can create an output which can be parsed by many tools with ease. For creating a timeline the FIDS offers a command line flag -m. For the configuration only the sqlite filename is needed, as shown in listing 3.4

```
sqlite:
    filename: apache.db
```

3.5. Security

A HIDS is a system that tries to improve the security of a host by detecting intrusions. Thus, it is essential to evaluate the security of such a system. This evaluation is the goal of this section. First, I show some limitations of the FIDS. It has some downsides, and it is necessary to look at them. After that, I describe some risk associated with running the FIDS. Those risks might be associated with all HIDS. Then I give some attack scenarios. Those include some scenarios where the FIDS is better suited than other HIDS, but also some that explicitly attack the limitations and risks. Lastly, I give examples of how to circumvent those attack scenarios. Some might be defeated if we change some small things, while some cannot be avoided work.

3.5.1. Limitations

There are some limitations that I list below. I reference those in the attack scenarios and detection sections.

Changing Attributes

As already mentioned multiple times, the proposed system doesn't use cryptographic hash functions for the detection of intrusions. This approach can be an issue as it means that changed files might not be found. Attributes can be changed, which could lead the FIDS to miss an intrusion. **chaning:times**, **changing:attributes**

It is possible to change the file system attributes. However, it is not trivial to change all of those attributes. The changing of some leaves traces and for some attributes root access is required. Additionally, the attacker would first need to remember the times which are correct such that he can correctly reset them.

Lastly, it is possible that the attacker does not change all attributes or that he sets them incorrectly, such that the FIDS can still discover the changes. Some changes to the attributes leave other traces, such as kernel-level logs. Those can be evaluated by an appropriate tool to find this kind of intrusion.

Non file based intrusion

Another limitation to any HIDS that operates on FIM is intrusions that are not file-based, which are getting more popular even as advanced persistant threat (APT). Many of the biggest threats for web applications, according to Open Web Application Security Project (OWASP), are not creating any files on the host **owasp**. There are also other attacks which don't need to use files if they can hijack an already started process. Those attacks are not detectable because the system only works with FIM.

Those kind of intrusions are getting more frequent and as already said, can't be discovered by only checking the file system. However, they leave other traces. They often leave logs in the application because for most of the OWASP top ten, many trial and error is required. APTs are more challenging to find. They leave traces as well in the form of weird network behavior. Because to achieve persistence, they need to redownload the malware after a host has been rebooted. This traffic leads to a perfect opportunity for a NIDS

Intrusion in non watched location

When the scanner is started it is fed with which paths to scan and which to ignore. If an intrusion injects a file in a place that is not watched, it cannot be found. This limitation comes from an incorrect configuration.

For non-watched locations, it is as if the host would not run a HIDS. Maybe the intrusion will at some point, write or read something in a watched location, but otherwise, the threat can go unnoticed for a long time. I recommend that at least the scanner part of the HIDS on the whole system. Then the attacks can at least be discovered by analyzing the timeline. Additionally, it is recommended to run the investigator on the whole system as well, but maybe with a lower frequency.

Preexisting intrusions

The system can only detect intrusion by unexpected changes. If it starts on an already running host, it is possible that this host is already infected. This infection cannot be seen by the system, as long as the files from the intrusion stay consistent.

Preexisting intrusions are hard to detect if they don't alter their files. Luckily, malware behaves like a typical software project. At some point, it needs to update its executable. It might also read or write other files that seem suspicious. Additionally, if the FIDS is deployed with new machines as well as to the already running machines, those are going to detect the intrusion as soon as the already infected hosts try to infect the newly deployed ones.

Shut down FIDS

The system relies on the running of the scanner. On a host, this would probably be done by creating a cronjob. If this can be disabled, then the scanner will no longer run, and no further intrusions are detected. Additionally, the investigator could also be stopped. This situation would lead to the same issue.

Should the scanner be shut down, then the investigator will no longer find any intrusions. This can be detected if the database is checked on new entries from another source. This is easiest done when an external database is used or when the SQLite file is copied from the host to an external machine at some points. Also, if logs are configured to be written after each execution, it can be suspicious if those entries are missing or are showing to be evaluating the same runs over and over again.

This limitation is not highly likely, because the intruder first needs to get administrative access to the host to be able to shut down the cronjob. This is not always easy to achieve, and once the attacker has administrative access, he can do whatever he wants on the system.

3.5.2. Risks

Aside from the limitations, there are some risks with running this system. As with the limitations, I give some comment to each risk and talk about how to evaluate them.

Running unknown code

As for any program, it is always a risk associated with the execution of code that is downloaded from the internet. It is possible that it was modified or that it is exploitable. However, this is an open source program. This fact makes it easier to evaluate the code. The same would need to be done for all the dependencies, more specifically for pytsk3 and TSK. This is possible but requires much work. However, the open source community is generally well trusted, which might be enough. To protect from alterations to the tool that has been downloaded, hashes can be calculated and the PGP signature can be validated. Additionally, the system behaves very straightforward. It scans the file system and then writes to a database, then reads from the database and creates alerts. This functionality can be monitored. Should the system do something else, it might either have a bug or is being abused.

File access

Any HIDS that uses FIM needs access to the files. Depending on the data that is stored on the host, this might be a cause for concern. The system has full access on any of those files. This cannot be circumvented. What can be done is using only the scanner on this system. Let it write the database file. Then let the investigator run on the data from another host. This way, the system can be monitored to make sure it doesn't do anything else except for writing the database.

Root access

The system needs access to the disk image which is mounted. To gain access to that, it needs to run with administrative privileges. With those permissions, the system could theoretically do anything on the host. The best way would be to limit the amount of time that it is executing with those privileges. To do that the scanner and the investigator can be executed in different processes where only the scanner has administrative access. The investigator only needs to read the database and send alerts, for neither it needs administrative privileges.

Exploiting of the system

As with any system, it is possible that an attacker can exploit the FIDS. This way he can run arbitrary code on the host. This risk is higher than for most applications since the FIDS runs with administrative privileges.

The best way to limit that risk is, again to limit the parts that run with administrative privileges — this way, the attacker needs to exploit the scanner, which is harder. It is important to note that the FIDS overall does not have a large attack surface. The only data that is read is metadata from the file system. There are no open ports, sockets, web connections, or files being read. It also does not need to be running at all times. This already limits the attack surface significantly.

3.5.3. Attack Scenarios

In this section, I describe several attack scenarios. I only explain what the intruder does and how he does it. I might give some description of the host and which processes are running. I also list which limitations or which risks the attacker uses. In section 3.5.4 I explain how those attacks can be avoided or detected and what is needed for the detection. Finally, in the discussion, I give a summary of how my system stands against other systems like Aide.

Classic Intrusion with persistance

The traditional way to gain a foothold on a host is to write a file on the file system. The attacker then executes the file remotely, and the host is compromised. How exactly the attacker was able to upload a file is not relevant for this scenario. The attacker does not do anything special with the file and leave it where it lands. He does not change any attributes or uses any other tactic to hide.

This attack exploits none of the risks and limitations as this is the kind of attack that the system is made to find.

Intrusion with persistance, attacker changes all metadata

The attacker changes a file or creates a file on the host and then resets the attributes in such a way that the system can't find anything wrong.

Risks and limitations:

- Changing Attributes
- Intrusion in non watched location

Intrusion without persistance to use as intermediate host

The attacker exploits a running process. He uses the host to gain access to other machines in the network and does not read or write anything from or to the host.

Risks and limitations:

- Non file based intrusion

Intrusion without persistance to exfiltrate data

As in scenario named 'Intrusion without persistance to use as intermediate host' the attacker gains access through an exploit of an already running process. However, in this scenario, he does read files of interest, for instance, private keys used for encryption.

Risks and limitations:

- Partially: Non file based intrusion

Intrusion without persistance but as APT

This scenario is similar to the previous two. Here the attacker exploits a process that is already running. He then goes on to use this process to do things that are uncharacteristic for this process. He does neither read nor write files from the host. His goal is to stay hidden for as long as possible. Additionally, he wants to reinfect the host whenever the process is restarted.

Risks and limitations:

- Non file based intrusion

Attacker exploits FIDS to gain administrative privileges

In this scenario, the attacker can somehow exploit the FIDS to gain administrative privileges. The most likely way is to give it a different configuration or change the code that is already running on the system.

Risks and limitations:

- Non file based intrusion
- Root access
- Exploiting of the system

Attacker changes the code maliciously

The attacker changes the code that the user downloads. He injects his malicious code in the version. The victim then executes the FIDS and the attacker can take over the host.

Risks and limitations:

- Preexisting intrusions
- Root access
- Exploiting of the system

3.5.4. Attack Detection

To detect the attacks described in Attack Scenarios, there are multiple things one can do. First, I give some information about how the specific scenarios can be detected if possible. Then there are some general recommendations on what can be done to increase the security of the system or the host.

Classic Intrusion with persistance

This attack scenario is the easiest. The FIDS detects this kind of intrusion if the configuration is correct. Compared to other HIDS, this kind of attack is found faster, because the system can executes faster.

Intrusion with persistance, attacker changes all metadata

The FIDS can only find this kind of attack if the attacker takes longer than the system has to scan. It is possible that the FIDS discovers it while the attacker has not yet changed all the attributes. Additionally, it is possible that the attacker did not change all the related attributes. In both cases, the system finds the intrusion and alerts it. Should the attacker be fast enough, the system does not find the changed file. Then it depends on how well the attacker hides. If he does not do anything on the file system, it is possible that this kind of intrusion goes unnoticed for a long time by the FIDS.

Here a NIDS or a different HIDS that evaluates different parts of the host come into play. A HIDS running on processes or ports could find this intrusion. Also, a NIDS can detect the intrusion because the attacker needs to communicate to or from the compromised host at some point. FIDS is not suited to find this kind of attack on its own.

Intrusion without persistance to use as intermediate host

Any HIDS that uses FIM cannot detect this type of attack. To still detect this kind of attack a NIDS can be used. Because the attacker most likely wants to gain access to other hosts in the network. When there is a NIDS installed, it should be able to detect this traffic and alert them. Another way would be to deploy an additional HIDS that does not use FIM. This way, it would find the intrusion and can alert it.

Intrusion without persistance to exfiltrate data

This type of attack is visible for a NIDS. Large amounts of data are transferred in a way, that might not be usual. This can result in alerts from a NIDS. However, the FIDS can detect this kind of attack as well. As the attacker reads the data, it changes the modified timestamp, which can be watched using the FIDS

Intrusion without persistance but as apt

This type of intrusion is not possible to detect by a HIDS that works with FIM, except the attacker reads or writes files. A NIDS is needed which can detect the intrusion because after the infected process is restarted, another host reinfects it. This is the kind of traffic a NIDS should be able to detect.

Attacker exploits HIDS to gain administrative privileges

For the FIDS, this attack is not detectable, because the attacker changes the system itself. The best way to protect against that is to monitor what it does carefully. The scanner part should only be allowed to read data from the disk image and write it to the database file. The investigator should be executed separately without admin privileges. This way, the attack surface is diminutive. The only apparent means for the exploit is the scanner config. To still find the intrusion, the FIDS might already have written the hash of the configuration to the database. By checking the hash, the attacker could be found.

Attacker changes the code maliciously

This attack is not detectable from the system, because firstly, the system itself was changed and behaves abnormally. Moreover, because when the system was executed, the intrusion already happened. To protect from this form of attack, the system administrator should check the hash of the HIDS. Additionally, the PGP signature needs to be checked. This way, an administrator can make sure that the package which is installed has not been altered. Further, the source code can be checked and compiled by the system administrator himself. This would take more work, but the possibility for malicious inclusions are fewer.

3.5.5. General defense

Generally, it makes sense to use a NIDS in conjunction with any HIDS. Some intrusions are more suitable for detection on the host level and others on a network level. Both make sense, so both should be used. Additionally, it makes sense to use an additional HIDS which is specialized in detecting intrusions on the process, or network activity level. This way, the host is monitored fully and intrusions can be even better detected. For the FIDS specifically, it makes sense to run the scanner and the investigator separately.

4. Discussion

The designed and implemented FIDS works differently than other HIDS that rely on FIM. Many will argue that because it does not generate cryptographic hash functions, it is insecure. This is not true, because finding intrusions always takes risks. If a conventional HIDS finds more intrusions but takes a long time to scan the whole system, it is not useful. Finding an intrusion days or even weeks after it happened might not be interesting. The attacker then has a long time to either hide, move on or extract all the information he is interested in. Finding intrusions in a timely fashion is very important and this is where my implementation shines. Additionally, even if it didn't find the intrusion, the database can be very helpful for forensic investigators to find out what the attacker did. This might help finding similar attacks in the future by adapting the configuration or by chaning other components. The argument this system makes is not even to be the one and only. In its current form this does not make sense. I hevily advise people to use a NIDS or a HIDS with other focusses. I also advise people to use a HIDS with FIM which uses cryptographic hash functions for highly critical parts of the system. However, if only a hash based HIDS is used, then many attacks will be found to late, or not at all. However, this risk based approach is not perfect. The decision to not rely on hashing results in a considerably weakened reliability. An attacker can change the file system attributes back to the original value **changing:attributes**.

During my thesis I created the system and I tested it by modifying data. However, the system has not yet been used in a productive environment and has not yet detected live intrusions. This means that it can not be said without doubt if it would work. The main reason why it has not been tested is that the time ran out. The system would needs to be tested in a live environment where attacks naturally happen. Sadly, I did not have access to such a system.

The FIDS uses TSK to analyze file systems. Thus, it is reliant on the support of TSK for the different file systems. While TSK supports many file systems, it does not support all of them. Some file systems are not yet supported which means that the FIDS cannot directly evaluate them. It is possible to extend pytsk3 to support more file systems but this is not yet easilz doable using FIDS.

However, the FIDS can be used for intrusion detection and for investigation. The output of the timeline creator can be very valuable for an investigator, because it gives a more detailed view on what happened on the file system than what is available by using the current state of the host. With the FIDS it is possible to create a timeline which can be viewed further back without loosing accuracy.

5. Conclusion

In this thesis the goal was to create a HIDS which finds intrusions in big file system fast and helps with forensic investigation. I created the FIDS which uses TSK to check file system metadata for FIM. This way it has low execution times compared to systems that use cryptographic hash functions. By taking the risk based aproach of not having the reliance from hashing, I gained a lot of speed. This speed can be used to find intrusions faster and decrease the reaction time.

To help with investigations, the FIDS contains a component which produces the output needed for timeline creation. Timelines are an important tool in forensics to find out what happened at what times. Specifically, with the timeline output from the FIDS, investigators can validate which files changed at what times.

5.1. Future Work

It would make sense to extend this HIDS to extend past the file system. It would be great if the scanner can also scan the processes and network connections. Not only would this data be important to finding intrusions by the system, but it could also give investigators much more information. Information which they are currently mostly missing.

The system should also be extensively tested. For this it needs to run for a prolonged time in a productive system. The output and the found intrusions would then need to be compared to other system to gain important information about how fast and how many intrusions are found using this system.

As already implied, it would be nice for the system to integrate with other DBMS. This way it could cover more usecases easier.

One other path that could be improved on would be the investigator. Currently it operates only on a configuration which someone must write. It would be great to autonomously analyse the scanner output and find anomalies on them. This could be done if a lot of data has already been collected on many different types of hosts. The types could then be grouped and an algorithm could detect similar patterns to find simmilar intrusions.

Declaration of primary authorship

I hereby confirm that I have written this thesis independently and without using other sources and resources than those specified in the bibliography. All text passages which were not written by me are marked as quotations and provided with the exact indication of its origin.

Place, Date:	Biel, June 11, 2019
Last Name, First Name:	Julian Stampfli
Signature:	....

List of Figures

3.1. System Architecture 11
A.1. Milestones 54

List of Tables

A.1. Workpackages . . . 53
A.2. Milestones . . . 53
A.3. Workpackages Evaluation . . . 55

B.1. Comparison of programming languages . . . 58
B.2. Comparison of configuration languages . . . 58
B.3. Comparison of DBMS . . . 59
B.4. Comparison of output and alerting functions . . . 60

C.1. Week 01 . . . 61
C.2. Week 02 . . . 61
C.3. Week 03 . . . 62
C.4. Week 04 . . . 62
C.5. Week 05 . . . 63
C.6. Week 06 . . . 64
C.7. Week 07 . . . 64
C.8. Week 08 . . . 65
C.9. Week 09 . . . 66
C.10. Week 10 . . . 66
C.11. Week 11 . . . 67
C.12. Week 12 . . . 68
C.13. Week 13 . . . 68
C.14. Week 14 . . . 69
C.15. Week 15 . . . 70
C.16. Week 16 . . . 71
C.17. Week 17 . . . 72
C.18. Week 18 . . . 73

Listings

3.1. SQLite Configuration . . . 12
3.2. Scanner Configuration . . . 13
3.3. Investigator Configuration . . . 13
3.4. Preconfigured Rules . . . 14
3.5. Fids Run Table Definition . . . 16
3.6. Fids Error Table Definition . . . 17
3.7. Fids File Table Definition . . . 17
3.8. Fids File Attribute Table Definition . . . 18
3.9. Fids File Attribute Run Table Definition . . . 19
3.10. SQL Querry for files . . . 21
3.11. Example Scanner Configuration . . . 23
3.12. Example PHP File Configuration . . . 24
3.13. Example Upload Directory Configuration . . . 24
3.14. Example Static Files Configuration . . . 25
3.15. Example Log Directory Configuration . . . 26
3.16. Complete Example Configuration . . . 27
3.17. Example SSH Keypair Configuration . . . 28

APPENDICES

A. Project Management

A.1. Goal

Before the start of the project the following main goal was defined:

Building of an HIDS that detects unauthorized or unusual behaviour on the file system. Compared to traditional HIDS file system integrity checking, it should scale with a lot of data and have the possibility to be used for investigation (retain historic data) built in from the start.

A.1.1. Sub goals

From this primary goal, the following sub goals were defined.

Scanning

The system is capable of scanning the file system for certain properties. The search is done by leveraging the sleuthkit tools. Thus the system is capable of interpreting the results from sleuthkit. It will further analyze them and decide on what to do with the results. Especially importance is given to the finding of differences.

Recording

The system records all findings. Including new, changed and deleted files in comparison to an earlier point in time. This recording enables the use of investigation as the evolution of the data can be viewed at any time. This data can also be used for machine learning algorithms to detect anomalies that are out of the scope of this thesis.

Evaluation

The system is capable of evaluating the results by applying predefined rules. Those rules can be adjusted by configuring the system.

It is thinkable that the system analyzes the recordings and makes decisions based on the historical behavior of the specific host and behavior from different similar hosts. This approach is not part of this thesis as it requires much historical data that is not present at the time of this thesis.

Alerting

The system is capable of being run continuously. This capability enables it to find anomalies automatically. The system can report those anomalies by creating alerts. It allows configuration of these alerts.

Scaling

The run of the system on a big file system completes in an appropriate amount of time. This speed allows the finding of anomalies that appeared recently. Additionally, it allows the storing of more states of the system which results in a her probability of capturing short-lived anomalies for future investigations.

A.2. Workpackages

From those goals the workpackages in table A.1 were defined. For a better overview they are assigned to categories. The categories are Architecture, Implementation, Validation and Administrative. Architecture is about defining how the system will look like and how it should work. Implementation is the effective implementation work for getting the system to run, this includes configuration and coding. Validation is about testing of the system. Administrative is everything that deals with project management and other workpackages that don't directly influence the system but need to be done.

The ID is a combination of the first letter of the category and a unique index. Administrative is shortened to D due to the conflict with Architecture.

The priority is a value of high, medium and low.

The workpackages are chrononically ordered. Meaning they should be worked on in approximately the order that they are given.

A.3. Planning

For the planning of this project the following milestones were created. Each coveres multiple workpackages. The mapping can be seen in table A.2. The milestones can also be seen in figure A.1. There they are displayed with an assumed and actual finish date.

A.4. Validation of Workpackages

At the beginning of this thesis, workpackages were defined. In this section I will validate which were implemented, which were cancelled and which new ones were added.

Table A.1.: Workpackages

ID	Short description	Prio	Category
D00	Setting up LaTeXdocument	h	Admin.
D01	Define workpackages and set deadlines	h	Admin.
A00	Research other HIDS and the sleuthkit tools	h	Archit.
A01	Decide on a Programming Language	h	Archit.
I00	Setup the developer environment	h	Impl.
I01	Add the ability to scan the whole system using sleuthkit	h	Impl.
A02	Decide which database connectors should be used	h	Archit.
I02	Add one database connector	h	Impl.
I03	Implement a recording functionality	h	Impl.
A03	Decide how the rules should be defined	h	Archit.
I04	Add template rules and ability to parse them	h	Impl.
I05	Add functionality to parse output according to rules	h	Impl.
V00	Verify that the system runs on a big file system	h	Valid.
I06	Add functionality of repeated scans	m	Impl.
A04	Define which alerting methods make sense	m	Archit.
I07	Add alerting functionality using one method	m	Impl.
V01	Verify the functionality of the software by changing the system	h	Valid.
V02	Verify the functionality of the software by running it on an infected system	m	Valid.
V03	Verify the alerting of the software by running it on an infectable system	m	Valid.
I08	Add multiple database connectors to different systems	m	Impl.
I09	Add multiple alerting methods	m	Impl.
A05	Define how to protect system and configuration from tampering	l	Archit.
I10	Implement software hardening	l	Impl.
D02	Finish user documentation	m	Admin.
D03	Finish project documentation	h	Admin.
D04	Create project presentation	h	Admin.
D05	Create project poster	m	Admin.
D06	Create project video	l	Admin.

Table A.2.: Milestones

ID	Short description	Workpackages
00	Setup	D00, D01, A00, A01, I00
01	Initial functionality	A02, I01, I02, I03
02	Rules	A03, I04, I05, V00
03	Alerting	A04, I06, I07
04	Exhaustive testing	V01, V02, V03
05	Usability	I08, I09, D02
06	Software Hardening	A05, I10
07	Presentation	D02, D03, D04, D05, D06

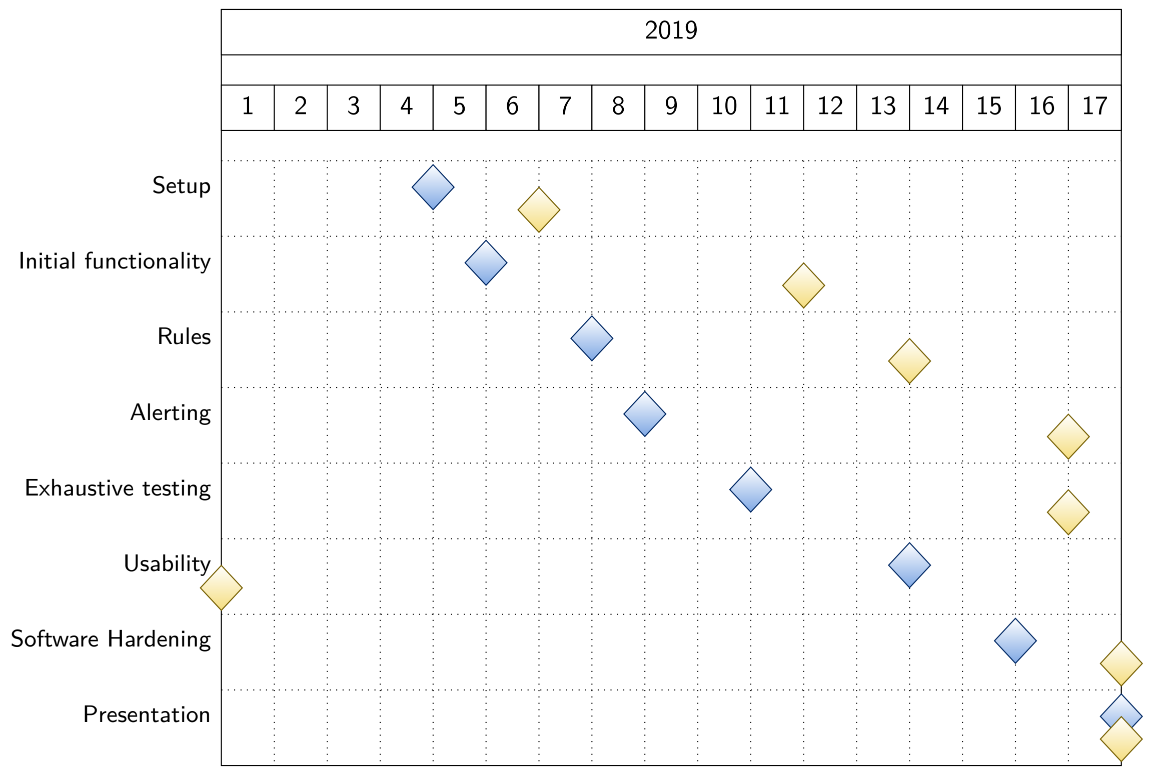

Figure A.1.: Milestones

Table A.3.: Workpackages Evaluation

ID	Short description	Status
D00	Setting up LaTeXdocument	Done
D01	Define workpackages and set deadlines	Done
A00	Research other HIDS and the sleuthkit tools	Done
A01	Decide on a Programming Language	Done
I00	Setup the developer environment	Done
I01	Add the ability to scan the whole system using sleuthkit	Done
A02	Decide which database connectors should be used	Done
I02	Add one database connector	Done
I03	Implement a recording functionality	Done
A03	Decide how the rules should be defined	Done
I04	Add template rules and ability to parse them	Done
I05	Add functionality to parse output according to rules	Done
V00	Verify that the system runs on a big file system	Done
I06	Add functionality of repeated scans	Cancelled
A04	Define which alerting methods make sense	Done
I07	Add alerting functionality using one method	Done
V01	Verify the functionality of the software by changing the system	Done
V02	Verify the functionality of ...	Cancelled
V03	Verify the alerting of the s...	Cancelled
I08	Add multiple database connectors to different systems	Cancelled
I09	Add multiple alerting methods	Cancelled
A05	Define how to protect system and configuration from tampering	Done
I10	Implement software hardening	Cancelled.
D02	Finish user documentation	Done
D03	Finish project documentation	Done
D04	Create project presentation	Done
D05	Create project poster	Done
D06	Create project video	Done
A06	Define Timeline Creator	New & Done
I11	Add Timeline Creator	New & Done
I12	Make rules for investigator depend on other rules	New & Done

B. Implementation Decisions

For this thesis there were some things that had to be defined during the thesis. Some decisions were already made before the thesis began. Those were mainly to only focus on the file system and to use TSK. Most other things were deliberately left open, so that they could still change if need be. The biggest decisions, which are listed here, were the programming language, the language and format of the configuration, the DBMS and the output. Those are described in the following sections.

Each of the decisions were made using a table with the criteria and the candidates. Each candidate has a ranking of 1-5 for each criteria where 1 is the lowest and 5 is the highest ranking. The rankings are then summed up and the highest candidate was chosen.

B.1. Programming Language

For the programming language any general purpose language would be applicable. I focus mostly on Python, Java, Go and C/C++. Those languages were chosen to further look into based on my knowledge, fit for the job and interest. There are some criteria on which I evaluated them listed below.

- Fit for the system
- Investigation community acceptance / community resources
- Personal experience and interest
- Learning potential

Those categories were not chosen at random. Each plays an important part in a successful thesis. The fit for the system is a generally important part. Some languages like C# don't really fit the system because it has historically been a windows only language. Although this is less true now, but because of that I excluded that specific language.

The acceptance of the investigation community has a direct link to the amount of community resources available. It is important that there exists a library that can be used to access the TSK API. If this was not the case I would need to create the bindings in the thesis which would be a lot of extra work. Additionally, if I use an exotic language, the chance that the system is actually used is smaller, because there will be no community backing it.

Personal experience and interest plays a role in any software development project. If the developer has no interest or experience in a language it takes longer until anything is completed. This is doubly true if the developer works alone.

Learning potential is also important. The developer, me, is more engaged if he can learn something while writing the code and thus is faster. This might be counterintuitive, as he needs more time because there are some unknowns. However, challenging oneself and learning is a big part of any developer.

In table B.1 I evaluated the mentioned languages on those attributes. Python achieves the highest score with 21. This is mainly because on the one hand it has a lot of community support and an active library to interact with TSK. Additionally, I already have some experience in it but not to much. This way starting off is easier and I can still learn a lot.

While Go and C/C++ interest me and would be a rather good fit, I have nearly no experience in either of them. Additionally, go doesn't have any community resources and bindings for TSK. For C/C++ what mostly brought me to give them such low rankings is the fact, that they are very easily exploitable.

With Java I have a lot of experience, also it has a reasonably good fit and comunity. However, the motivation to do yet another java project was what lead me against this language. Additionally, the investigation community currently really likes python and not java.

Table B.1.: Comparison of programming languages

Name	Fit	Community	Experience	Interest	Learning	Total
Python	4	5	4	4	4	21
Go	5	1	1	4	5	16
C/C++	3	3	2	3	5	16
Java	4	3	5	1	1	14

B.2. Language for Configuration File

The language of the configuration file is at the same time less and more important than the programming language. For the developer it is less important. As long as it is managable it is ok. It needs good integration with the programming language and then everything would be fine. On the other hand, for the user it is more important. He does not care which language the software was written in. However, he might need to change the configuration file relatively often.

I tried to bring both needs together. The categories that I used here are integration with python, readability, writeability and prevalence. The read and writeability is the amount of work that a user has to put in to read or write a configuration. This includes the likelyhood of syntax errors and the overall awkwardnes of the language. The prevalence is about how well know the configuration language is. This is important because it increases all three other parts.

For the languages that are evaluated I chose YAML, JavaScript Object Notation (JSON), ini and eXtensive Markup Language (XML). Those are the recommended languages for python. In table B.2 I evaluated each language for each attribute.

YAML emerges victorious. This is mostly because it was made to be human readable. Other than XML and JSON. It is also easily extensible and does not require much syntax knowledge. XML and JSON contain a lot of additional symbols that need to be defined correctly. For configuration that is mainly for humans this is a big drawback. The INI format could also work. However, this format is not widely used.

Table B.2.: Comparison of configuration languages

Name	Integration	readability	writeability	prevalence	Total
YAML	4	5	5	5	19
JSON	4	5	3	5	17
XML	3	5	1	3	12
INI	5	3	3	1	12

B.3. Database Management System

The DBMS used has more requirements than the programming and configuration languages. There are many competeing DBMS which all would be viable. This decision is focussed on the primary DBMS used in this implementation, meaning the one that will be used in the thesis. The system should afterwards be extended to support many, if not most, of the different DBMS to give the user the possibility to configure the system the way that makes the most sense for them.

The criteria for the first DBMS are simplicity, performance, support for prototyping, attack surface and usability on different systems. The DBMS analyzed all need to be open source or at least free of royalties. Chosen were SQLite, MariaDB and PostgreSQL. As a note, only relational databases were chosen because of my experience.

In table B.3 the results can be seen. SQLite has won mainly because it has a high simplicity and good support for prototyping. Additionally, it also works on systems that need to be extremely encapsulated because of the data they store. This way it is harder to exfiltrate data from those high security systems. As said before, more DBMS should be added to the system after the thesis ends. Other DBMS have benefits especially in the performance, but SQLite was chosen as dbms for the implementation in this thesis.

Table B.3.: Comparison of DBMS

Name	Simplicity	Performance	Prototyping	Attack Surface	Usability	Total
SQLite	5	2	5	5	5	22
PostgreSQL	3	5	3	3	2	16
MariaDB	3	4	3	3	2	15

B.4. Output / Alerting

The output or alerting is simmilar to the DBMS. There is not one correct choice. Multiple different ways need to be implemented if the system should become popular. This decision was mostly about the way of alerting that was implemented in this thesis. After the completion, most of the alerting methods should be supported, but it does not make sense to implement many of them during the thesis to not sway to far from the main topic.

The criteria for the output are extensibility, complexity for implementation, complexity for configuration and support for low configuration. Extensibility is mostly about how easily can it be extended into another form of alerting. Complexity is mostly about how much time is needed to implement it and what requirements does it require on the host system. For instance, for sending mails, an smtp server needs to be configured. The support for low configuration goes into how the type of alerting can be used if the user did not change the configuration a lot, or at all. This is true for when he just installed the system and wants to try it out.

The candidates for output and alerting are console output, rest interface, email and writing to a log file. The results can be seen in table B.4. Interestingly, output to the console has won easily. The biggest advantage from that is that this output can be handed over to any other application or script. This means that all the other candidates can be done by writing a script for them. This increases the extensibility enormly. Also, the complexity for both implementation and configuration is extremely low. A good console output is required for prototyping anyway, using it as alerting by letting the user handing it over to another tool seems to be the most sensible approach. Still, in the future additional alerting functions should be built in.

Table B.4.: Comparison of output and alerting functions

Name	**Extensible**	**Complex Impl**	**Complex Conf**	**Support no conf**	**Total**
Console Output	5	5	5	5	20
Log File	2	5	4	4	15
Rest Interface	4	3	1	1	9
Email	1	3	3	1	8

C. Journal

In this section I list a journal of my work with aproximate hours I worked on the thesis with the total that I have worked until then. Those numbers are not extremely exact, as I did not use a time measurement tool. I am mainly listing challenges I faced and decisions I made. I will also list the meetings I had with Bruce Nikkel (BN) and Armin Blum (AB) as well as a short meeting summary. I will also list the workpackages that were worked on and, if applicable, completed. Additionally I will also list milestones that were reached, postponed or cancelled.

There is one section per week where I worked on the thesis. Each section starts with a table that is running. The columns are Completed Workpackage (CWP), Reached Milestones (RM), Hours Worked (HW), Total Hours Worked (THW) and whether there was a meeting or not.

C.1. Week 01 18.02.-24.02.

Table C.1.: Week 01

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes

Tasks

In the first week I mainly gathered information about the thesis itself. On the 18.02. was the official kickoff event from the Berner Fachhochschule - Bern University of Applied Sciences (BFH). I also started gathering more information about HIDS. I scheduled a meeting with BN for the 01.03.

Problems

I did not face any problems in the first week.

C.2. Week 02 25.02.-03.03.

Table C.2.: Week 02

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes

Tasks

This week the unofficial kickoff with BN took place. We met in Bern and talked a lot about how we want to manage the project. The full meeting notes are available in section D.1. I also took some time after the meeting for postprocessing of the meeting. Then I started by creating the initial draft of the LATEXdocumentation by taking the BFH template and scaling it down to my needs. I also created a rough draft of the high level requirements and created a github repository.

Problems

The LATEXdocument was initially to complex and I did not understand all the packages. After some trial and error and reinstalling of packages I was able to get it to run.

C.3. Week 03 04.03.-10.03.

Table C.3.: Week 03

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes

Tasks

The main task that I did this week was the problem statement and the requirements that I needed to prepare for the second meeting. This is not in the workpackages. Then I needed to prepare for the meeting and postprocess it again. The meeting notes are in section D.2. Overall, I continued gathering information about HIDS, file system and NIDS.

Problems

I had no problems in week three.

C.4. Week 04 11.03.-17.03.

Table C.4.: Week 04

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No

Tasks

This week I wanted to improve the LATEXdocumentation. Additionally, I started working on the work-packages and milestones.

For non project management tasks, I started researching other HIDS that are available. Mainly I focussed on tripwire and aide. I also started researching TSK and its API

Problems

- Splitting the work into actionable workpackages
- Table formating in LATEX
- Milestone formatting in LATEX
- Almost no publicly available information on tripwire.

C.5. Week 05 18.03.-24.03.

Table C.5.: Week 05

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes

Tasks

With the knowledge of TSK I was able to search for libraries that support the TSK API for the different languages that I wanted to evaluate and then I decided the language. Before the meeting, I decided on the programming language, Python. The full reasoning is available in section B.1.

I also had to prepare for the meeting this week. The outline is shown in section D.3.

Problems

- Finding good TSK libraries was not possible for all languages
- Tutorial on Go was harder than anticipated.

Table C.6.: Week 06

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes

C.6. Week 06 25.03.-31.03.

Tasks

This week I started by contacting AB. We then decided for a meeting in the same week. I then used time to prepare for said meeting. Additionally, I started to create a local developer environment with the correct libraries and tools. With that I completed Milestone 00. Then I had the meeting with AB which needed a lot of postprocessing. The meeting notes are available in section D.4.

Problems

- I needed a lot of time to research certain uncertanties that appeared after talking with AB

C.7. Week 07 01.04.-07.04.

Table C.7.: Week 07

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No

Tasks

This week I realized that I am already behind schedule. I started by going through the goals and rework some of them to increase the clarity. I also rushed into trying to get the initial functionality going. This was not as easy as I anticipated and I lost a lot of time trying to get the output of pytsk3 that I needed. This was tough and I frequently checked the source code of pytsk3 to find solutions.

Problems

- I was not able to find the correct way to use pytsk3 to walk through the file system

C.8. Week 08 08.04.-14.04.

Table C.8.: Week 08

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes

Tasks

This week I started frustrated because I was not able to implement the file walking ability. However, after getting to it with a fresh mind I was able to find the solution by finding a new source of documentation. I was glad that I could find it. However, I still could not get all the information that I needed out of the pytsk3 or rather I did not yet know how to get to all the data.

Then I started preparing for the meeting with both AB and BN.

Problems

- I could not get to all the data.

C.9. Week 09 15.04.-21.04.

Tasks

This week was a very low week. I was frustrated by the inability to get pytsk3 to work the way I wanted and didn't have much motivation to work on the documentation. I took it slow and would regret it in the following weeks.

Problems

- I had issues with motivation.

Table C.9.: Week 09

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No

C.10. Week 10 22.04.-28.04.

Table C.10.: Week 10

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No

Tasks

This week I knew that I had to finish the file scanning ability. Especially after slacking off the week before I started researching and trying until I found the correct search query which lead me to the documentation of TSK API for the file struct. I felt very stupid because I had not found it before. With this documentation I could make sense of the output of pytsk3 and I was able to create an appropriate python class with the most important fields of a TSK file.

Problems

- At first I could not find the correct documentation.
- I struggeled with the datatypes of how the permissions were stored before I realized that it is a base 10 representation of the base8 permissions.
- Debugging was quite a challenge since the code needs root access to connect to the local harddrive.

C.11. Week 11 29.04.-05.05.

Table C.11.: Week 11

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No
11	A02, I02, I03	01	32	200	Yes

Tasks

After feeling the relief from last week and the pressure from being rather far behind I really wanted to finish Milestone 01. I did this by creating the database, a configuration to connect to the database and then storing the results of the file walk. For both the configuration and the database I had to first define which I wanted to use. For the configuration it was clear pretty fast that YAML seems the appropriate choice. For the DBMS I was more unsure. I first wanted to implement both, SQLite and postgres. However, I could not bring postgres to run on my developer environment. Considering the time and the other functionality, I decided to drop postgres and focus on SQLite.

Besides that, I also had another meeting. I really wanted to show my first prototype and thus I forced throguh all hardships. I also had other things to prepare for the meeting, but finally I was able to gain valuable feedback for my prototype. The notes for this meeting are listed in section D.6

Also, I was able to fix the debugging challenge. I simply created an image of an usb stick on which I then run the code. This had two advantages. Firstly, it did not need root access. Secondly, it was faster since the stick was smaller than the harddisk.

Problems

- Postgres would not run on local machine.
- Database schema needed some trial and error until it worked properly.

C.12. Week 12 06.05.-12.05.

Tasks

In this week I wanted to implement the changes proposed by BN. I did this by first adding all the information to the python class and then adding it to the database. This actually required some trial and

Table C.12.: Week 12

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No
11	A02, I02, I03	01	32	200	Yes
12	-	-	20	220	No

error since it was not always easy to find the correct values. Also, I did it twice, since I forgot to push it to the github repo and then was not sure if I already did it on the second machine.

Problems

- I did the work twice because I forgot to push it on my home pc.
- The USB Stick Image workaround did not work for everything as the file system on it did not contain all required attributes.
- Sometimes I had issues with the Python module handling. I was not always able to import the classes correctly.

C.13. Week 13 13.05.-19.05.

Table C.13.: Week 13

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No
11	A02, I02, I03	01	32	200	Yes
12	-	-	20	220	No
13	A03, I04, I05, V00	02	32	252	No

Tasks

In this week I was able to get a lot of momentum. Firstly I created a new directory for the system. This way the thesis documentation and the HIDS were split. Then I looked deeper into python modules and how to manage the imports properly. By doing that I was able to fix some issues of files sometimes not getting imported correctly. Also, I found some more information in the TSK API which I had not seen before. I added this information. Also, I added the errors that were thrown as to not lose information. I then followed with a bigger testing period on my local machine. This resulted in frustratingly many errors, which as I later found out were a direct result of how my filewalking process worked. I was able to solve all those errors and then added the first draft of finding intrusions. This config was harder to draft, but I came up with a implementation that would cover many usecases. I also implemented a first draft of the investigator, which for the moment takes a very long time.

In project management view I created a first draft of the thesis documentation and prepared for the meeting that would be on the start of next week.

Problems

- The system took about 15 minutes to complete on my local machine which resulted in long waiting periods
- The investigator config was rather hard to design
- The investigator took extremely long itself

C.14. Week 14 20.05.-26.05.

Table C.14.: Week 14

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No
11	A02, I02, I03	01	32	200	Yes
12	-	-	20	220	No
13	A03, I04, I05, V00	02	28	252	No
14	-	-	16	268	Yes

Tasks

In this week I started writing in the documentation by creating the first draft of the introduction.

However, I mainly focussed in bringing the scanning time down. I tried many approaches and wanted to find out where the issue lies. After clarifying with the main developer of pytsk3 it was clear that a direct filewalk on the disk image was not possible. However, on the same day I found a slightly tweaked version of my implementation that would result in an incredible performance boost. I no longer would use the directory paths but the directories that I already had. This way the runtime of the scanner went from about 15 minutes on my home machine to about 30 seconds. Sadly, I only realized that after a longer time. At first I had the investigator running as well as the scanner. This resulted in extremely long runtime. After I deactivated the investigator I realized how much faster it had become.

Problems

- By running both the scanner and the investigator I did not initially see the performance boost of the scanner.
- pytsk3 does not support a direct file walk.

C.15. Week 15 27.05.-02.06.

Table C.15.: Week 15

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No
11	A02, I02, I03	01	32	200	Yes
12	-	-	20	220	No
13	A03, I04, I05, V00	02	28	252	No
14	-	-	16	268	Yes
15	D05	-	40	308	No

Tasks

This week I started focussing on the documentation. So far this was not a priority for me. I took all week to come up with a version of the documentation that would be viable as a first draft. I also created the Poster in this week.

Problems

Besides some LATEXissues everything went fine this week. (Appart from the obvious issue that I should have started with this much earlier...)

C.16. Week 16 03.06.-09.06.

Table C.16.: Week 16

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No
11	A02, I02, I03	01	32	200	Yes
12	-	-	20	220	No
13	A03, I04, I05, V00	02	28	252	No
14	-	-	16	268	Yes
15	D05	-	40	308	No
16	A03, I04, I05, V00, I06, A04, I07, V01	03, (04)	40	348	Yes

Tasks

This week I focussed on both. First I wanted to bring the HIDS to an appropriate end. I did this by first redefining the rules and investigations to better match what is required. I adjusted them several times until they resulted in the final version that is available now. Additionally, I reworked the way the investigator finds relations between the files. By passing that task to the database I could gain increadible performance increases. I also verified the system by letting it run and then changing files. This way I reworked milestone 02, implemented milestone 03 and partially implemented milestone 04.

In the documentation I also changed a lot. I added many chapters while improving most already created ones. I also created the abstract and added that to the book tool from the BFH.

Problems

This week was very productive and I did not really have any big issues. After so much time working with pytsk3 and TSK I was able to quickly maneuver through the code. Also changing the milestone 02 was not that big after I realized what the biggest issues were.

In short, I did not really have issues.

C.17. Week 17 10.06.-16.06.

Table C.17.: Week 17

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No
11	A02, I02, I03	01	32	200	Yes
12	-	-	20	220	No
13	A03, I04, I05, V00	02	28	252	No
14	-	-	16	268	Yes
15	D05	-	40	308	No
16	A03, I04, I05, V00, I06, A04, I07, V01	03, (04)	40	348	Yes
17	A05, I10, D03, D04	06, 07	40	388	Yes

Tasks

This week I needed to finish both, thesis and the system.

For the system, I implemented the python packaging guide. This way the system is available through the python packaging system. I also added a PGP signature with a new private public keypair. This way the software is at least somewhat hardened.

For the documentation. I finished it. Also I finished the presentation and held it on the day after handing this documentation in. Hopefully it went well.

Problems

C.18. Week 18 17.06.-23.06.

Tasks

It is actually hard to describe what I do this week, because as of writing this it is still in the future. What I need to do is prepare for the bachelor thesis defense and actually holding that. I assume to take 16 hours for preparation and for the defense itself.

Besides that I also need to create a video of this thesis.

Table C.18.: Week 18

Week	CWP	RM	HW	THW	Meeting
01	-	-	8	8	Yes
02	D00	-	12	20	Yes
03	-	-	12	32	Yes
04	D01, A00	-	24	56	No
05	A01	-	24	80	Yes
06	I00	00	12	92	Yes
07	-	-	32	124	No
08	-	-	20	144	Yes
09	-	-	4	148	No
10	I01	-	20	168	No
11	A02, I02, I03	01	32	200	Yes
12	-	-	20	220	No
13	A03, I04, I05, V00	02	28	252	No
14	-	-	16	268	Yes
15	D05	-	40	308	No
16	A03, I04, I05, V00, I06, A04, I07, V01	03, (04)	40	348	Yes
17	A05, I10	06, 07	40	388	Yes
18	-	-	16	404	Yes

Problems

- I have no experience in video editing and creation. Thus, I expect this task to be quite challenging.

D. Meeting Notes

In this chapter I will list the meeting notes that detail what was talked about in the official meetings.

D.1. First Meeting (28.02.)

People

- BN
- Julian Stampfli (JS)

Talking Points

- Programming language
- Contact frequency and tools
- Workplace
- Expert which was then unknown

Next Steps

- JS
  - Creating of rough, high level requirements
  - Setup of Github
- BN
  - Working on slides
  - Arrangement of production server

D.2. Second Meeting (07.03.)

People

- BN
- JS

Talking Points

- First contact with AB by BN
- Requirements
- LATEXdocument used

- Aide configuration
- Abstract
- DBMS
- Software Hardening
- Alerting
- Only the bare minimum of project management work

Next Steps

- Validating programming language
- Cleaning LaTeXfile
- Create Workpackages
- Prioritize Workpackages
- Create Milestones
- Set deadlines to milestones

D.3. Third Meeting (21.03.)

People

- BN
- JS

Talking Points

- Who will contact AB
- Which TSK commands to use
- Milestones and tasks
- Size of the project (is it too big)
- Order of work packages
- Programming language
- Deamon vs scheduled task
- Timestamps and database schema
- Timezones

Next Steps

- Contact with AB
- Start with Milestone 00-01
- Find a good name

D.4. Expert Meeting (28.03.)

People

- AB
- JS

Talking Points:

- Getting to know each other
- Architecture of system
- Security concerns
- Goals
- Some unclear points
- Execution time
- Meeting with BN

Next Steps

- Clarify goals
- Create clear architectural picture

D.5. Fourth Meeting (11.04.)

People

- AB
- BN
- JS

Talking Points

- Security concerns
- Filesystems
- Querying the database
- Performance comparison
- Architecture
- Getting to know each other

Next Steps

- Work on Milestone 01

D.6. Fifth Meeting (02.05.)

People

- BN
- JS

Talking Points

- Validation of first prototype
- Error handling
- Storing of everything fo every run
- Adding all information from TSK
- Production server

Next Steps

- Improve prototype
- Work on milestone 02

D.7. Sixth Meeting (20.05.)

People

- BN
- JS

Talking Points

- Speed
- Poster
- Intrusion detection
- Layout of LATEXdocument

Next Steps

- Solve speed issue
- Improve intrusion detection
- Create poster
- Start working on documentaion

D.8. Seventh Meeting (07.06.)

People

- BN
- JS

Talking Points

- Speed issue
- Poster
- Status of documentation
- Details of how the documentation should be structured
- Status of the system

Next Steps

- Improve details of the system
- Add forensic output to system for demonstration purposes
- Add Samhain HIDS to documentation
- Improve and finish the documentation
- Create presentation

D.9. Eight Meeting (12.06.)

People

- BN
- JS

Talking Points

-

Next Steps

-

E. User Documentation

F. Content of CD-ROM

Content of the enclosed CD-ROM, directory tree, etc.