NTNU
Norwegian University of Science and Technology

Automated triage of samples for malware analysis

Halvor Mydske Thoresen

Master in Information Security
Submission date: June 2017
Supervisor: Geir Olav Dyrkolbotn, IIK
Co-supervisor: Katrin Franke, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Acknowledgment

First of all I would like to thank my parents for their support and motivating me to actually finish this thing.
Thanks to my big brother Eirik for always being ahead and creating nearly impossible expectation that I can have fun trying to fulfill.
Big thanks to my grand parents for moral support.
Thank you to Geir Olav Dyrkolbotn and Katrin Franke for great supervision and providing an excellent office space to work in.
Many thanks to the the third party who got me access to the VirusTotal file feed, and experts to interview.
And lastly, thanks to Eirik Bue who shared a tiny office space with me every day for three months and kept me half sane.

H.M.T.

Abstract

As people continue to rely increasingly on information systems, the threat landscape will keep evolving.
To combat and defeat new threats we need good cyber threat intelligence.
Analysis of malicious software is a popular method of gathering such cyber threat intelligence.
This can be done both manually and automatically, with pros and cons on both sides.
One thing they have in common is that they both require quite a lot of resources, either if it is time or computational power.
A problem then arises when the analyst has more suspicious samples to analyze than available resources.
The analyst then has to triage and prioritize which samples to focus resources on.
This triage itself requires quite a lot of resources, which results in a vicious cycle.
This thesis proposes a methodology for automating such sample triage, resulting in a list of samples sorted by a score which indicates the probability of containing good cyber threat intelligence.
In order to prove and validate the methodology, a program was developed which followed most of the methods. However, a few limitation had to be considered.
Various tests was conducted on the program output in order to validate it. One of which included comparison between computer and human triage. The results were not absolute conclusive, but the study contributed to the needed research of improving efficiency and reduce resource requirements to perform analysis of malware. We have showed that generic rules can become useful if combined with feature values and weights, if used for the right purpose. With these weights and values, we also showed that it is possible to always get a relevant output of interesting samples by changing the values to fit the current situation.
And lastly, it was concluded that we there is a good possibility that replacing human triage with an automated system will indeed increase efficiency of the malware analysis process.

Contents

List of Figures

List of Tables

1 Introduction

This thesis will be introduced by a brief high level overview of the current situation to get an understanding of why the study is conducted.
Then the main problematic area in focus will be described in more detail for the reader to see where there is room for improvement.
This is followed by a short explanation of possible benefits and desired improvements
These sections all boils down to the resulting research questions, which will be answered throughout the thesis.
Finally, a description of limitations that had to be considered, and the rest of the thesis structure.

1.1 Topic covered

As society continues to evolve, we rely more and more on information systems for everything to function properly.
New technologies and software platforms are introduced every day to solve problems and increase efficiency in almost all aspects of our civilization.
However, this rapid growth in technology has also introduced a whole new and bigger threat landscape.
Information has always been one of the most valuable resources, but it has been quite challenging to get access both with and without authorization due to physical restrictions.
This is no longer the case, as almost all information is being stored digitally, and availability is at an all time high.
The amount and availability of information, together with this new threat landscape, makes it very lucrative for actors with malicious intent to steal for monetary profit or misuse the information to strengthen their agenda.
On this new digital battlefield, the gap between actors has been reduced, as there is almost a certainty that everyone has some vulnerability in their system. A single person can do just as much damage as a whole state, with the right tools and knowledge.
It has also proven over the years, that people have a hard time grasping the damage caused by a cyber attack, due to not leaving physical marks.
This all results in more and more people equipping their black hats, and joining the malicious side.

However, the amount of information security professionals with an intent to defend and protect does not seem to increase just as fast, which causes a lack of available workforce and resources to fight this digital battle.

This study try to improve efficiency and reduce the resources required to perform one of the most essential defensive method white hats has in their arsenal, gathering intelligence on the threat actors by analyzing their weapons.

Keywords
Malware analysis, reverse engineering, big data, weighted score, filtration, VirusTotal, Cyber threat intelligence, malware hunting

1.2 Problem description

A popular method of gathering cyber threat intelligence today is from analyzing and reverse engineering malware samples[1].
However, acquiring malware samples that has a high probability of providing new and good cyber threat intelligence can be quite challenging.
Most of the new samples of malware found today comes from infected machines discovered after an incident. How the infected machine gets discovered might be due to multiple other factors than the malware itself, resulting in a new sample without having detection coverage of it. Coverage, as in having detection rules for a large percentage of possible malicious software.
However, these kind of discoveries all happens after the fact, where it would be preferable to have detection of the malware before it infected the system.
In order for companies to increase their detection coverage, they rely on information sharing within their community and between parties. Where not only information from incidents are shared, but also threat intelligence gained from analysis of samples gathered from other sources. The activity of looking for malware samples is often refered to as malware hunting, and can be quite resource consuming.
There are multiple techniques used in order to hunt for malware, such as setting up honeypots or crawling forums and online communities.
However, one of the most popular techniques is to utilize services such as VirusTotal which provides access huge amounts of file samples that are uploaded from people all over the world.
Such a stream of files does of course contain a lot of already known malware together with huge amounts of benign samples, but occasionally someone uploads a file they find suspicious that has never been seen before. This is the golden egg that the malware hunter is after.
Finding these few samples in the midst of millions of others can be a huge chal-

lenge, as each and every sample must be analyzed in some way or another. Either statically, dynamically, automated, or manually. All of which has their pros and cons.
No matter how one decides to analyze the files, the main problem remains as the amount of available file samples is currently too large and continues to grow.

1.3 Justification, motivation and benefits

There is currently a lot of research and development conducted with the goal of getting ahead of threat actors and to keep information safe.
However, conventional security has always been a very protected subject often involving a lot of secrecy.
Information security is no different, as it has proven to be almost as profitable to provide protection of information as to stealing it.
Huge amounts of security research is therefor commercialized, which can limit the protection and security in the long run.
However, as the threat of cyber attacks has increased and become more globalized, this is starting to change. Information sharing and aggregation is becoming a huge aspect in the fight against threat actors.
But we are not yet at the level needed to provide adequate protection. This is partly due to the difficulties and cost of gathering good cyber threat intelligence.
By introducing more techniques to increase efficiency of malware analysis, the resources required and cost might be reduced, which could make it easier to justify increased sharing of information.

1.4 Research questions

Based on the described problem areas, this thesis will answer the following research questions in order to improve or enlighten the situation.

- How can we define what is an interesting file sample for manual analysis?
- How can human triage of file samples be replaced by an automated system?
- To which extent can such an automation help improve efficiency of malware analysis?
- How generic file sample features be utilized to find interesting samples to analyze?

1.5 Limitations

The proof of concept created alongside this thesis will be limited to only focus on 32bit exe files, this means that the given results will be limited by the same factors.

This limitation is not considered to have much of an impact, due to the fact that these files are some of the most represented in malicious software.
Extracting good general features proved to be quite challenging and time consuming to code, so the proof of concept is limited to a small set of features.
However, the methodology chapter will describe and discuss solutions without limitations.
The testing that requires human participants, will also be limited to a small set of data due to the time required to analyze the samples.
And lastly, this thesis is based on the interviews of security experts in Norway. Unfortunately we were not allowed to disclose their identities, where they are working, or the interviews themselves due to the possibility of publication. This has also lead to some tiny gaps of missing information. We will however still refer to these interviews, for more information please contact the author.

1.6 Thesis structure

The rest of this thesis will consist of the following structure.

Related wok: The field of malware analysis and cyber threat intelligence gathering is quite young, relatively speaking.
However, it is also growing at a very rapid rate. Which can make it hard to follow the development and gather an overview of the situation.
This chapter will give a brief introduction to the subject necessary to fully understand the decisions and reasoning during the rest of this thesis.

Methodology: A idealistic and theoretical methodology is proposed in order to answer the research questions.
This contains reasoning and explanation on why choices are made and how something should be done for the best results.
It also includes how to test and verify the proposed methodology.

Proof of concept: The proof of concept that was developed based on the methodology and with quite a few limitations, will be explained here.
This chapter also describes how the testing, validation, and verification, from the methodology chapter, was conducted.

Results: All the results gathered from the testing and verification of the proof of concept, findings during litterateur study, and interviews will be displayed and summarized in the results chapter. The gathered information from the proof of concept is described and analyzed in this chapter.

Discussion: The discussion chapter will discuss, reflect, and criticize what is described in previous chapters, the results, and the study itself.

Conclusion: The last chapter will contain a short summary of the thesis as well as what could be concluded based on the results, what benefits the research has given, recommendation for future work, and lastly a short round up.

2 Related work

In order to fully understand the problematic areas mentioned and how the proposed methodology can help reduce some of these problems, it is necessary to get some background on the topics mentioned.
The following chapter will describe relevant topics which is mentioned in the thesis, and previous techniques or methods used for hunting malicious software and gathering cyber threat intelligence, as well as provide description of terms used throughout the thesis.

2.1 Cyber Threat Intelligence

The term cyber threat intelligence(CTI) is used throughout this thesis, and is an umbrella term for all kinds of information regarding cyber threats and protection of intellectual property.
This can be information about attribution of threat actors, infrastructure used for malicious purposes, or tactics, techniques, and procedures(TTPs) of a threat actor.
However, it only become CTI after the relevant information has been collected and thoroughly evaluated, resulting in credible and reliable information.
CTI can be used in different areas of information security, such as strategic, operational, tactical, and technical.
Technical CTI is the form that is most relevant for this thesis and what most malware analysts is looking for.
It is often referred to as indicators of compromise(IoC). This is information such as known files and their signatures, network communication like emails or certain patterns, infrastructure like IPs and domains, or other behavioral traits.

2.1.1 Cyber kill chain

A kill chain is generally speaking a process to target and engage an adversary to create a desired effect.
It is a concept often used by the military during strategic operations.
The general military kill chain is find, fix, track, target, engage, assess.
This does not suit the cyber domain very well. This is why Lockheedmartin created own process called a cyber kill chain.
The cyber kill chain is specifically made for digital intrusions and the steps required for an adversary to successfully infiltrate the target.
While analyzing malware, it is usually a good idea to have the cyber kill chain in mind. This can make it easier to determine which step of the attack the sample is trying to complete, which can give indications on how to continue the analysis.
For example if the analyst classifies the sample as an installer, he can start looking for communication functionality and infrastructure details. [2]

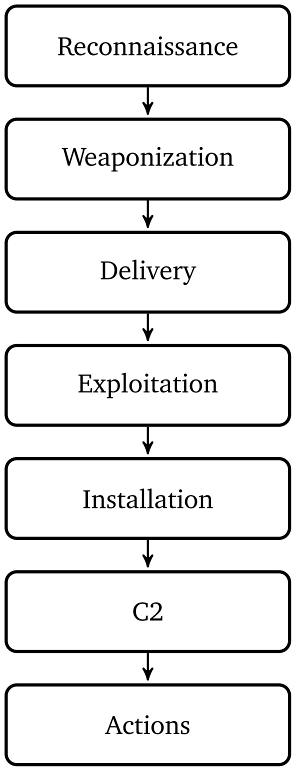

Figure 1: Cyber kill chain

2.2 Malware hunting

One of the core aspects of this thesis is to gather samples which is considered interesting for further analysis.
More often than not, this means samples which has a malicious nature.
The area of looking for and gathering malicious software, or malware as we will call it from now on, is often called malware hunting.
Hunting for malware can be a challenging task as they try to go unnoticed by design. However, there are multiple methods that can be used.

2.2.1 Incident response

An response team is often deployed after an attack has been detected, or a suspicious incident has occurred. Their task is usually to stop any ongoing malicious activity from further harming the target. To to this, they usually perform digital forensics on the targeted systems to figure out the attack vector and what is affected.
During the forensic work, it is quite usual to find either the malware itself or indi-

cators on what kind of malware was used and where it came from.
At this time, the forensic worker gathers everything that seems suspicious and out of place. These files is then often sent to further analysis by a malware expert in order to determine if the file, or sample as they are often refereed to during analysis, is in fact malicious.
If this is the case, then the analyst tried to gather as much information about the malware is possible, for attribution of the threat actor and how it functions to provide indicators of compromise, which can be used in intrusion detection systems to prevent further attacks.
Samples gathered from such incidents has a relatively high probability of being quite interesting, since there often is a reason for the response team to be called in the first place. However, not everyone is in a position to cooperate with a response or forensic team.

2.2.2 Honeypots

Another option get access to malware is to place yourself as bait, in order to trick a threat actor to attack you.
There are specific systems created for exactly this purpose, called honeypots.
While being attacked, one can monitor the system and extract any files used or left behind by the threat actor. These files are also quite easy to find as the whole system structure is synthetic and already known, so that any changes will stick out like a sore thumb.
When setting up a honeypot, it is important to know what kind of attacks or threat actors you wan to deceive. As the honeypot must be attractive enough to be worthwhile the attackers resources.
The following are examples on how different setups can result in quite different samples.
In order to gather large amount of small time malware, one would most likely want to set up a system targeted for automated attack bots. These bots scan the Internet for possible targets. If the honeypot has a heap of vulnerabilities and provides ease of access for the attacker, then it will be seen as a prime target.
However, if more sophisticated samples is desired, then the honeypot must be more tailored to the specific threat actor.
This requires extensive knowledge of what the targeted threat actor's desires and goals are.
If looking for malware created by state sponsored threat actors, or advanced persistent threats(APT), you might want to simulate something valuable such as a government hosted server with confidential information stored on it.

This might be challenging as most APTs probably have the resources and knowledge to detect honeypots.

2.2.3 Forums

A lot of threat actors and groups coordinates their efforts online in order to optimize their return of investment and to share knowledge.
By crawling popular forums, one can often find samples of malware or even the published source code.
Threat actors also tends to lease or sell their malware as products for profit, which also provides access to newly created malware if you have the resources to pay for it.

2.2.4 Data feeds

As mentioned, not every researcher or analyst has a collaboration with an incident response team Combined with the fact that not everyone is comfortable with the moral implications of honeypots or paying threat actors for malware, has resulted in researchers creating multiple platforms for sharing of already collected malware samples to share the difficulties of hunting for said malware.
Some of the biggest today is platforms such as Malwr[3], VirusShare[4], and VirusTotal[**?**], which is what this thesis utilizes.
However most of these only provide already analyzed known malware, which is great for researcher and developers, but not ideal for analysts looking for undiscovered cyber threat intelligence.

2.3 VirusTotal

VirusTotal is an exception to the statement about sources only providing known malicious samples.
However, sample sharing is not their main function.
VirusTotal is a service which allows users to freely upload and scan any files they want with over 50 different anti virus solutions. Some basic dynamic analysis is also done.
The system works by starting up multiple clean virtual environments with the anti virus solutions activated, then executing the given file on the machine.
The result is then an overview of which anti virus detected the sample, and what kind of malware it was detected as.
The virtual environment is also closely monitored during execution, which adds some behavioral information about the file such as attempted external communication or file creation, to the result report.
VirusTotal does not analyze the samples themselves.

Even though their main functionality is not strictly sharing of samples, they do allow users to download the files that are uploaded to the service.
However, its only really usable when looking for specific samples, since the platform receives millions of submissions every week. [5]

2.3.1 YARA

VirusTotal's solution to this problem is the YARA tool.
YARA is a framework for detailed searching through files based on various conditions set by the user.
These conditions is written in structures called Rules, and can look like the following example.

```
rule exampleName : example
{
    meta:
        description = "This is just an example"
        thread_level = 9001
    strings:
        $a = "This is"
        $b = {DE AD BE EF}
    condition:
        $a and b
}
```

They also allow users to keep a certain amount of rules active on the website, which results in an alert to the user if any file matching that rule is uploaded. The user can then download that file.
This is a very commonly used method for malware hunting.[6]

2.3.2 File feed

VirusTotal also provides a sevice they call the file feed.
This is a continuous stream of every sample uploaded to the service together with its result report, sent directly to the user, which is probably one of the largest pool of samples available. This service does not come cheap however.

2.4 Malware analysis

Main categories - Manual and automatex soure malware analysts handbook + andre bøker Automated analysis, sandboxing Akana (Android files) Binary Guard True Bare Metal Comodo Automated Analysis System and Valkyrie Deepviz Malware Analyzer Detux Sandbox (Linux binaries) EUREKA Malware Analysis Internet Service Joe Sandbox Document Analyzer (PDF, RTF and MS Office files) Joe Sandbox File Analyzer (EXE and DLL files) Joe Sandbox APK Analyzer (Android files) Malwr (also see MalwareViz) sandbox.pikker.ee VxStream Sandbox (Hybrid

Analysis) ThreatExpert ThreatTrack ViCheck

2.4.1 Techniques

Static analysis: A lot of interesting information can be found in static features such as the code itself, or metadata. However, most threat actors hide these pieces of information which makes it very complicated and hard to find. Due to this, a lot of resources such as time and expertise is required in order to analyze the static elements of malware.
Another downside is that these resources is required even to find out that there is no interesting information in the sample.
The upside however, is more control and a higher confidence on the information gathered than with dynamic analysis[7, 8].

Dynamic analysis: To bypass most of the techniques threat actors use, such as obfuscation by xor or cluttered code, to hide interesting information, we can execute the malware and analyze the actions it takes and traces it leaves.
In order for the malware to function it has to unpack itself and establish communication, and there is very few ways of preventing someone from monitoring that.
However there are some problems with dynamic analysis. There is a higher possibility of false negatives and information going unnoticed. There is a lot of reasons for that to happen. One of the most occurring is code in the malware that analyses the environment it is being run in and will detect if a virtual machine is trying to run, or if the network is simulated[9, 8].

Both of these can be done manually or automatically, with varying results.
Most malware in the world would not be classified as very advanced and does not contain a lot of anti-analysis techniques. However, malware created by Advanced Persistent Threats (APT), often do. And these kind of malware has to be manually analyzed more often than not.

2.5 Obfuscation

Obfuscation is the practice of hiding the meaning of a message or information. This can be done in multiple ways, but the most used in malware is performing XOR calculations on a set of data based on a hidden key.
Just as there are multiple ways of obfuscating information, there is also ways to break the obfuscation.[18]

Using a Debugger is one of the most important tools for any malware analyst. They give the user full insight on everything that is being executed during run time, which allows a very low level analysis.

Re-implementing Decoding Routines can be done by analyzing the malware in order to understand how it hides the information, then one can write the same executing, just in reverse.

Recovering Stack strings by monitoring the system stack during run time. This works because the given string or character has to be loaded in to memory atleast once in clear text in order to perform the xor, or other operations.

2.5.1 Anti virus companies

Anti virus companies gather their samples from multiple sources. Some being reports from online services where the solution failed to detect a sample, but for the most part they download every suspicious sample that triggers on their custom rules on every client computer with their anti virus solution installed.
Once a sample is gathered, they run dynamic analysis on it to extract IoC's for then to upload that information to all other clients.
Most of these solutions work as a preventative measure, and has to perform analysis at a very large scale. Which means that they do not have the resources to manually analyze samples to potentially find completely new, very advanced malware from an APT group.

2.6 scalability

Scalability research has gained a lot of traction the last couple of years, due to the exponential growth of the internet. This growth has also introduced a growth in volume of malware created and distributed, and the sophistication of this malware. Conventional techniques of automatically analyzing samples takes a lot of time and resources. Too much for current computers to handle when the volume increases.
Software is very complex, which means that there are a vast amount of possible features to use when analyzing and trying to classify the sample as malicious or benign.
When choosing which features to use, the time it takes to gather and extract the necessary information also plays a huge role in the scalability.
Roberto Perdisci et al.[11] developed a classifier called McBoost specialized in detecting if the sample was packed or not. They used this technique together with conventional signature based detection, which resulted in only 13,4% of the time spent classifying.
There has been multiple similar research conducted earlier as found in [12, 13, 14]. These kinds of improvement techniques could be interesting for this project to handle the scalability issue. Even if the techniques might not work in this exact case, the idea behind them could be adopted.
Another way to tackle the problem of scalability is to introduce pre-filtering with

dynamical analysis, as Ulrich Bayer et al. did with the open source sandbox ANUBIS [15].
This research resultet in a staggering 75.000 samples classified in less than three hours.
These research papers do cover one of the most important factors in this project. Even though their motivation and end goal is different, this project could be based upon some of the techniques used.

3 Methodology

The following chapter describes an idealistic approach on how to solve our research questions without any limitations.

3.1 Defining interesting

First and foremost the meaning of "interesting samples to analyze" must be defined, as it is the cornerstone of the whole thesis and one of the main research questions. In order to to that, interviews of experts in the malware analysis industry will be conducted. These interviews will map out what is generally perceived as worthwhile, and what a malware analyst looks after when deciding what to spend resources on.
Based on these results together with the literature study in chapter2, a fitting definition of the term "interesting sample" will be defined in chapter5.

3.2 Main Concept

A brief description of the proposed approach can summarized as follows.

- One starts off with a huge amount of unknown samples that could be interesting to further analyze
- Based on the amount of samples, some initial filtration has to be done
- Extract characteristics or features of the remaining samples
- Determine what is considered interesting and relevant for the current situation.
- Based on these features together with the predetermined prioritization of importance and interest, give the sample a score

The end results is a list of samples sorted by their probability of giving good and relevant CTI for the user.
This can help focus analysis resources where they are needed and increase efficiency.
The following chapters will describe each stage in greater detail, and discuss important points to consider.

3.3 Data

Gathering data, in the form of files or samples, is the initial step. This might be the data itself, or sources of continuous data.
The main problem areas in this thesis is due to too large amount of data or samples that are available to analyze.
Some of the ways to obtain samples for analysis is explained in the related works chapter.

3.3.1 Sources

Sources can be quite different and it is important to know what kind of data the source provides.
The data from a service such as VirusTotal will contain a huge amount lot of obviously benign samples, some uncertain samples, and a lot of malicious samples. Where data from an incident response team will most likely be quite a lot fewer samples, and contain a higher percentage of malicious samples.
Knowing that files from VirusTotal has already gone through a rough analysis, one can utilize that and discard samples that is not currently of interest, which will lighten the load of the system and reduce the required resources.

3.3.2 Storage

No matter how or where the data is gathered, it should be aggregated in a structural way.
This way the data can be easier to manually verify later.
All data files should be stored in the same location, but separated by type.
This will help later in the process due to different file types requiring different feature extractors and rules.
In some cases a complete aggregation might not be the best solution, if for example one would like to prioritize certain sources or weigh them differently. If this is the case, any general storage structure should be enough.

3.4 Filtration

Depending on the amount of samples to analyze, some initial filtration is highly probable to be needed.
This filtration must to based on information that require very little resources to acquire, because it has to be gathered from every single sample.
The type of filtration is heavily dependent on available resources and current interests.
Filtration on file sizes might help to both reduce storage space and discard large

programs, if the current interest is office documents. However, such a filtration can also discard files that would be considered interesting to look at, which is why it is important to know the sources and define your current interest before starting to discard data.

3.5 Features

After the initial filtration, only the main set of data remain.
From these remaining samples, features or attributes will have to be extracted and prioritized.
The filtration and rating of the samples is done by comparing and analyzing different features or attributes found in the files.
These features can be pretty much anything the user finds suspicious.

3.5.1 Extraction

There are two main categories of features that can be extracted. These are features gathered by statically analyzing, and by dynamically analyzing the samples.
There is both pros and cons with both kinds of feature extraction, so the user has to decide what works best for their application.

Static features is extracted by analyzing the data and structure of the sample.
Looking at sector sizes, imports and exports, locale information, various time stamps, or information entropy.
These features can be quite sparse, but require minimal resources to extract.
Which is very useful when extracting features from thousands of samples a minute.
Even though the features are limited, they might be good enough for the purpose of triage by interest.

Dynamic features is extracted by executing the file, and analyzing its behavior.
These features may be such as external communication or memory usage.
Dynamically executing every sample will most likely require extreme amounts of resources, and therefor not feasible in most scenarios. However, if the amounts of samples needed to be triaged and scored is small enough, it should be considered. As dynamic features together with static features would lead to the best result.

3.5.2 Selection

Some form of selection often has to be done before extracting said features. This is because there are such a large amount of different file types out there, and different features requires different resources to extract, which is important due to the amount of samples that has to be analyzed.

The selected features should also be as generic as possible , in such a way that all samples will get some values that can be compared with the other samples of the same type.
An example of a generic feature is the creation time stamp. Most files will have this, and if not, that could be interesting in itself.
A more specific feature could be a predetermined string in the file. This would most likely not be common in multiple files, and would not result in an increased accuracy, due to not being able to compare to the other samples.

3.5.3 Values

Since feature can be almost anything, they also come in many variations that can be quite hard to compare.
They can be relations such as the relative size between the data and text sections of a PE file, numeric such as the number of imports, dates such as creation time stamps, boolean values such as does the sample contain any obfuscated data, or other specific values such as detected language.
In order to handle these all these value types, we need to normalize them.
This is done by predetermining their interest values between 0 and 100, before scoring of the sample begins.
Each feature has to be determined one by one. The following is some examples of different types of values.

Language	Interest
RU	80
EN	20
NB	50

Table 1: Language values

A feature where the strings in the file is analyzed and returns a dominant language code is quite simplistic. Here[1] one only has to map a code to a value of codes that are of interest. Rest is set to zero.

DataSector	Interest
<=1	10
>1 && <2	40
>2	90

Table 2: Data sector values

A feature looking at relations between multiple features can be a little more tricky. In the table[2] above is some values based on the relation between the data

sector and the text sector of the sample. Which might indicate some kind of packing or hidden information.
A relation where the file has two times the data than text, is in this example quite interesting.

Compilation zone	Interest
UTC +1	10
UTC +2	10
UTC +3	90

Table 3: Compilation Timezone values

Compilation time of the sample can utilize the same kind of mapping as language codes[1].

Obfuscation	Interest
0	0
1	90

Table 4: Obfuscation values

And lastly, a feature which only returns a boolean value such as if obfuscation is detected or not. As seen in the table[4] above, we also gain the ability to reduce the impact of a true bool. It does not have to be 0 or 100 as some boolean features is less interesting than others.

3.6 Weighting

When as many features as possible is extracted, one would have to select and prioritize which features to focus on.
This is done by setting weights on the extracted features corresponding to how interesting the user finds them.
To completely discard a feature, simply set its weight value to zero.
This concept of weighting interesting or suspicious features is probably the most important part of this thesis.
These weights are applied by the user based on a subjective and dynamically changing definition of what the user currently finds interesting.
The weights are represented with a number from zero to one hundred, where zero means not interesting at all and basically disables that feature, and one hundred is the most interesting.
All features can be active at a time, and all features have a weight assigned to them. The following table[5] shows a mock example with a very few selected features. By converting these features to their predefined values as seen earlier, we get a

Feature	Compilation time	Language	ObfuscationUsed	DataToTextRatio
Value	UTC+03:00	Russian = 1	1	>=2
Weight	80	70	60	80

Table 5: Example weights

fully numeric table[6] of values, that can easily be compared and have some math applied.

Feature	Compilation time	Language	ObfuscationUsed	DataToTextRatio
Value	90	80	90	90
Weight	80	70	60	80

Table 6: Fully numeric

3.7 Rules

For all this to be applicable to any real world application, a way of keeping track of weights and values is needed.
We call the combination of a feature and weight a rule.
A rule is an object which contains information of which feature it applies to, the weight of that specific feature, and various meta data, such as an ID, the author of the rule and a short description.
Rules can be standalone, or dependent.
Dependent rules are, as the name implies, dependent on other rules. They can either not be worth anything until another rule is triggered in some way either by a feature existing, or a feature value over a certain threshold.
These dependent rules can also have different weights depending on which other features are present.
An example of this can be that locale information is more interesting if the detected language of text is the same as the locale.
The rules that does not have a weight of zero is used to determine which feature to extract during analysis. This also provides some consistency as the selection of features is not determined multiple places. It also safes resources by making sure that no features without a weight value is being extracted.

3.8 Scenarios

Due to the threat landscape constantly shifting and what the user finds interesting might change rapidly, a collection of rules can be created for specific situations.
A situation is called a scenario, and it remember the state of which all the features

and weights are set.
This allows the user to easily change the focus of the filtration in order to follow current threat events.
An example of this is that the user might have a set scenario which focuses on Russian threat actors, and one that focuses on Chinese threat actors.
Depending on which actor is most active, the user can change to the relevant scenario and start getting scores on samples that might get good CTI about a specific or similar threat actors.

3.9 Confidence score

Now it is time for the main part of the process, namely calculating the confidence score to see if the sample might be worthy of further analysis and can possibly provide some good CTI, either by a manual analyst or an automated system. In order for the weights to have a consistent impact, and for the resulting scores to be comparable, there is a couple of things to consider.
For the score calculation, we utilize weighted arithmetic mean of all features with the following formula.

$$\bar{x} = \frac{\sum\limits_{i=1}^{n} w_i x_i}{\sum\limits_{i=1}^{n} w_i}$$

This basically means that we sum up the feature values multiplied with the weights, and divide the sum of weights.

$$\bar{x} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}{w_1 + w_2 + \cdots + w_n}$$

However, this only works where sum of weights $==$ 1.

$$\sum_{i=1}^{n} w_i' = 1$$

And since we opted for weight values between 0 and 100 in order to prevent any mistakes where the sum of weights did not result in 100. We now first need to normalize the weights as follows.

$$w_i' = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

This results in a weight score between 0 and 1, where every weight is worth their share.

Combining these steps, results in this quite simple formula.

$$\bar{x} = \sum_{i=1}^{n} w_i x_i$$

3.9.1 Example calculation

Based on the previous example values[6], the calculation would be as follows.
We calculate the normalized weighted scores.
weight of 80 $= \frac{80}{80+70+60+80} = 0.27586206896551724138$
weight of 70 $= \frac{70}{80+70+60+80} = 0.24137931034482758621$
weight of 60 $= \frac{60}{80+70+60+80} = 0.20689655172413793103$
Then we apply these together with the features in the weighted mean function.

$$\frac{((90 \times 0.2758...) + (80 \times 0.2413...) + (90 \times 0.2068...) + (90 \times 0.2758...))}{1} \approx 87.56$$

3.10 End result

This whole process of gathering samples, extracting features, settings weights, and calculating a score, finally results in a sorted list of samples.
This list indicated how interesting the sample currently is for the user, and help with prioritizing where to spend resources for analysis.

3.11 Testing

Testing is integral to this whole system, as with any other automated system.
As we can not blindly trust that either parts of, or the end result is what we actually want.
This section will describe how to test and validate different parts of the process.

3.11.1 Data set

The proposed method is mainly made to work on large sets of data or samples that are continuous and unknown.
However, such data can be very hard to validate due to the necessity of manually analyzing and labeling every sample to see if it is scored correctly and that the score represents the users interests.
In order to fully validate this, we need a synthetic set of samples where every feature is known, and has already gone through human triage by experts.
This makes sure that the outcome is predictable.
If the outcome is not as predicted, then the user knows that the system can not be trusted on unknown samples.
However, the fault may also lay with the creation of the synthetic data set, and the

person who triaged and manually scored the samples.
The synthetic data set should also include some benign samples with similar features, to see if the program manages to mostly ignore them.

Target

This methodology is not meant to be targeted and find specific malware samples. It is not even meant to only find malware. It is meant to find interesting and suspicious files no matter if they would generally be classified as malicious or benign. However, more often than not, the most interesting ones is the malicious ones.
One can test the methodology on the real data set, by injecting known malicious samples of interest and see how they get compared to the other files.
It may not end up at the top, but it should be placed quite high if the right rules is set.

3.11.2 Scalability

Scalability is one of the main issues with automated analysis systems, at least if dynamic features is selected.
In order for the user to test its systems, the data set should be limited at first, then gradually increased.
After every feature extraction is timed and completed, we compare the amount of samples and the time and resources required per iteration of increased data.
If the result shows an exponential growth in resource requirement, then the solution is not scalable and some other features must be selected in the future.
The required time for analysis might not be a big concern in situations where the system has downtime between data input. However, with a source such as the VirusTotal file feed, time requirement is essential to not creating a bottleneck problem.

3.11.3 Accuracy

In order to determine the accuracy of ratings given by the system, multiple experts in the field of malware analysis should test the system.
First, the program will run for an hour on the with as much data as possible. A large stream of samples such as the VirusTotal file feed is preferable.
This will result in a list of rated samples with a confidence score for each sample.
Random rated samples should then be extracted, be shuffled, and having any of the data attached, such as extracted feature or rules, removed.
This will equal the playing field for both the expert and the automated system.
It is preferable with as many samples as possible, but a minimum of 10 should be required to give good indications of accuracy.
These samples is then given to the experts, who has to apply their usual triage and

rate the samples on a scale from 0 to 100 on how interesting the sample seems based on a given situation narrative.
The initial triage done by the experts should be quite swift and shallow, and should represent real world decision making.
The ratings given by the experts will then be correlated with the rating from the program, by calculating the distance between the scores.
Which should give an indicator on the accuracy of the program and how well it can replicate manual human triage.

3.11.4 Consistency

When taking actions away from humans and placing that responsibility in automated systems, it becomes very important to be able to trust the system.
This is tested by generating a predetermined data set of samples, running the rating process multiple times, and shuffle the order on each run.
We can then compare each and every sample rating, and verify that every sample is analyzed separately such that they are always the same no matter which order the samples are analyzed.

3.11.5 Over fitting

As mentioned, this proof of concept consists of quite a small number of rules, which not only means that the confidence is not optimal, but also that over fitting is an issue.
Over fitting describes testing where the test scenario, in this case the rules, are too specific for the given set of data.
This results in a very good performance measure while testing, but that might not be representative of the real world.
To avail over fitting and try to avoid it, the experiment must be conducted on a realistic random set of data.

3.11.6 Real world application

Due to the possibility of overfitting, the system should also be tested on a large real time stream of data from the online service VirusTotal or similar large streams of data.
This data will not be controllable, but the test will give an indicator of how the system handles a large variety of samples. With all the previous tests in mind.

4 Proof of concept

A proof of concept(PoC) was developed based on the methodology described in the previous chapter[3].
This program was created in order to validate the proposed methods, and to further strengthen it.
While the methodology described a idealistic solution, the PoC is more realistic and is constrained by multiple limitations mentioned in the introduction of this thesis.
This chapter will explain how the PoC was created, and it will serve as a description on how the thesis results were gathered.
It will not go too in depth on the technical solutions, such as coding documentation.
The structure of this chapter will be mostly similar to the methodology chapter.

4.1 environment

The PoC was developed with Python3.6. This was mainly because of the simplicity it provides, due to a huge amount of already existing libraries and their ease of access through PIP.
Multiple already well tested and robust packages or libraries was utilized in order to analyze and extract features from the samples.
Python is generally not seen as the most performance friendly language, and since performance is quite a bit deal with this method, another environment such as C would be preferable.

4.2 Data

To get an accurate representation of a real life scenario, the selection of data was quite important.
However, most malware analysts have multiple unique sources of samples through their employer company. There are multiple free malware sharing communities on the internet such as malwr or malwaresb, but these only contain known malware which in many cases may not be as interesting because intrusion detection rules are already written and published about them.

4.2.1 Sources

We managed to get access to VirusTotal's live file feed through an undisclosed third party collaborator.
This is a real time continuous feed of every sample uploaded to the VirusTotal

service.
This feed averaged about 900.000 samples a day during the testing period. Where only about 250.000 samples where detected as malicious and already known by anti virus companies from all over the world.
The data also consisted of a huge variety of different file types, where the plurality was Win32 EXE files.
It is known that malware analysts uses VirusTotal and its YARA toolkit in order to hunt for malware, so this source was perfect.
The VirusTotal feed provided a huge amount of samples for performance testing, and had a file distributing which reflected the real world of malware hunting.

4.2.2 Storage

Due to the huge amount of samples being downloaded at a continuous rate, we could not store every sample indefinitely.
To solve the storage problem, a temporary storage was introduced. This was the first location every sample was stored. After all the feature were extracted, every sample under a certain score were deleted.
Another storage challenge was the features themselves.
As the end result requires a sorted list of samples based on their score, it helpful to store the features in a database instead of a file as comma separated values.
The database of choice ended as an SQLite database.
SQLite is not considered the best performing database. However it utilized the very well known SQL syntax which provides ease of use. Its theoretical maximum size is approximately 1.4e+14 bytes(140 terabytes)[16]. This is considered to be more than enough for most use cases.
The performance may slow down the more entries there is in the database, but at about 5 gigabytes of entries, no performance issues was notices.

4.3 Filtration

The initial filtration is where most of the samples was discarded.
One of the research questions mentions finding interesting samples, and as we have defined interesting as a very subjective and dynamically changing term, we based our proof of concept on the interview with industry experts. During this interview they mentioned that finding malicious samples that was not detected would be the most interesting for them.
Therefore the initial filtration was to discard every file that had a detection rate of more than 5 out of over 50 anti virus solutions.
The samples was also limited to only WIN32 EXE files, and the rest was dropped. This is due to the challenges of extracting features from many different file types

and their structures.

4.4 Features

Selecting and extracting generic enough features from WIN32 EXE samples that were generic enough to be found in most cases, proved to be a bit challenging with limited knowledge about their structures. However, 10 quite generic and simple features which had the possibility of giving results was extracted.

4.4.1 Extraction

The following is a short description of how our selected features was extracted.

PEFILE is a python library which parses PE files and loads the parsed information into a json object.
With the help of PEFILE, mulitple features where extracted including the samples compilation time stamp, the time zone of the machine it was compiled on, the PE time stamp from the header, the different sector sizes, locale information, and general header information which should follow a strict structure.[17]

FLOSS or FireEye Labs Obfuscated String Solver, is a tool made by FireEye for automatically extracting obfuscated strings from Malware.
This tool automatically detects, extracts, and decodes obfuscated strings in Windows PE files. It is also very easy to use. However, as with any automated tool, we can not guarantee detection, but it does give a good indication of obfuscation techniques is present in the sample or not.[18]

langdetect[19], guess-language[20], and ngramj-python[21] were all used in order to detect language of written text, be it comments or variable names, in the samples.
This was done by running the popular Linux tool Strings[22] on each sample to extract all text that could represent words, and then pass those strings to the python language detection libraries.
Multiple language detectors was used due to the relatively high inaccuracy of each one. An aggregation of the results proved to be more reliant.
It was also necessary to pass custom word lists to the libraries, which might have had some impact on performance.

Fuzzy-entropy technique was used in order to detect if sections of the sample had been encrypted with any moderately good crypto algorithms. This was done by looping through 1024 bytes at a time, and checking the entropy of each blob. If enough of the sample had relatively high entropy, then the crypto feature returned True.[23, 24]

4.4.2 Selection

The selected features was based on interview conducted of experts in the malware analysis industry, and was some of the most important things they looked for when deciding which samples to focus their time on.
Given that automation of that exact process of human triage is the main goal of this thesis, it seemed like a good choice to choose these features.
Due to the low amount of extracted features, no further selection was needed after the extraction process. Every extracted feature ended up being utilized in the testing.

4.4.3 Values

As mentioned in the general methodology, every feature needs a normalized value in order to be used.
The definition of these values is based on what the user finds relevant and interesting at a given time, which makes this very subjective.
As the goal of this PoC was to provide a simple and human understandable representation of the automated process, the following values where set for two different scenarios.

Scenario	**Russian**			
Feature	compilation time	time zone	PE timestamp	obfuscation
Condition	if >08:00 <16:00	UTC +3 = 100	!= comptime	1
&&	in UTC +3			
Interest	60	100	40	70

Feature	dataRatio	inconsitentHeader	stringLang	dataLang	locale	crypto
Condition	>2	1	Ru	Ru	Ru	1
&&						
Interest	80	50	100	100	100	30

Table 7: Russian feature values

Table 8: Chinese feature values

Scenario	**Chinese**			
Feature	compilation time	time zone	PE timestamp	obfuscation
Condition	if >08:00 <16:00	UTC +8 = 100	!= comptime	1
&&	in UTC +8			
Interest	60	100	40	80

Feature	dataRatio	inconsitentHeader	stringLang	dataLang	locale	crypto
Condition	>2	1	Ch	Ch	Ch	1
&&						
Interest	40	30	100	100	100	80

4.5 Weighting

Based on the same intervju and the experts definition of interesting, the following weights were used.
The same weight is used for both the Russian and the Chinese scenarios, due to the feature interest values shifting.

	CompTime	**TimeZone**	**PE Time**	**Obfuscation**	**dataRatio**	**inconsistantHeader**	**StringLang**	**dataLang**	**locale**	**crypto**
Input Weight	70	80	40	50	60	40	80	80	90	20
Normalized Weight	0.114	0.131	0.065	0.081	0.098	0.065	0.131	0.131	0.147	0.0327

Table 9: Russian And Chinese weights

4.6 Rules

The technical aspects of a rule is based on how YARA rules are structured.
Every rule with a weight greater than 0 gets its defined extractor function executed.
Following is an example structure of a rule that extracts the samples time zone.

```
rule timeZone
{
        extractor:
                featureExtractor.getTimeZone()
        condition:
                timezone == UTC+3 OR UTC+8
        value:
                UTC+3 = 100
                UTC+8 = 80
        weight:
                60
}
```

4.7 Scenarios

For the purpose of testing and validating the methodology with respect to the weighting system and for the human testers to have a collective narrative to focus on to simulate the real world, two scenarios has been created.

First scenario is a huge increase of Russian APT campaigns, where a relatively large portion of the malware used is hidden within other seemingly benign software.

Second scenario is a huge increase of Chinese APT campaign, where sophisticated and well developed software is prominent.

The technical aspects of these scenarios is the combination of the already defined features, values, and weights.
These scenarios is quite simple and generic, which fits well with the limited amount of features extracted.
The feature values for the Russian scenario has a bigger emphasis on obfuscation and the data sector size, due to the nature of packed software, while the interest values for the Chinese scenario focuses on good use of cryptography and file structure integrity.

4.8 Confidence score

The resulting confidence score is calculated exactly how it is described in 3.9.

4.9 End result

The end result is a large database, with all the feature values and the confidence score.
There has not been developed any user interface to make the representation of the result more accessible.
However, the database is based on SQL syntax which is functional when querying the highest scored samples.

4.10 Testing

This section will describe which tests were performed, why they were chosen, and how the tests was conducted.

4.10.1 Data set

The data which the PoC is tested on comes purely from the VirusTotal file feed, which means that with limited resources validation of every sample was impossible.
However, it provided a huge amount of samples with a lot of variations and a consistent stream of said samples. Which gave a pretty good realistic large scale testing environment.

4.10.2 Stability

Stability is important for any system, and especially when downtown can result in malware being missed and a threat actors going unnoticed.
In order to verify that the system was stable, it was started with the Russian sce-

nario, and left for 24 hours.
During that time, multiple statistics were recorded such as distribution of confidence scores, the number of samples, and what file types was received.

4.10.3 Accuracy

When testing the accuracy of the system, the method described in the methodology chapter was performed.
The system ran for one hour on the samples from the VirusTotal file feed.
Due to the amount of time required to conduct human triage of the samples, only the minimum of 10 samples was extracted from the result, at random.
These samples were made sure to be in the same state as they were when they arrived into the automated system, and then their order was shuffled.
These samples where given to one willing expert from the malware analyzing industry, and two fellow students.
The expert has long experience with this kind of triage and these kind of samples, while the students have only very basic knowledge of reverse engineering software.
The participants where told not to use more than 5 minutes on each samples. They were also told the Russian scenario narrative to give them an indication on what might be considered interesting, and they were tasked with rating the samples on a scale from 0 to 100 based on how interesting they thought they were in regards to the narrative and if they would analyze them further.
The resulting scores where then compared by calculating the distance between the participants score and the score given by the automated system. These scores where then added together to give a final accuracy score of the system.
The distance was calculated by subtracting the participant's score from the automated score.

4.10.4 Consistency

Consistency was checked by feeding the system the same 10.000 files that it had already scored.
Both the these files and their rules where fed and loaded in a different order than the original analysis.
This was repeated three times, without a single sample being affected.

5 Results

This chapter will explain and describe the results gathered from the proof of concept, and the thesis project as a whole.

5.1 Defining interesting

What is interesting to analyze is dynamically changing based on subjective view of current trends, the situation, and desired goals. To define it to one set of things would not be applicable.
However, it is possible to define interesting at one specific point in time for one specific user.
In order to create an example of a definition of interesting samples, one needs a situation and a goal.

Situation: There is a huge increase of Russian APT campaigns, where a relatively large portion of the malware used is hidden within other seemingly benign software.

Goal: Gather samples that might be created by said Russian APTs in order to find CTI to detect and prevent further attacks.

Interesting: Samples with attribution features that indicated Russian origin, and sector sizes that indicates excessive data which is intended to be hidden.

An interesting sample does not seem to be bound to being malicious, as a lot of benign software can contain valuable information at the right time.

5.2 Statistics

Some interesting statistics started to emerge after running the proof of concept for about 5 days.
One of which is the separation between the total number of samples uploaded to VirusTotal every day, and the number of samples that is detected as malicious by one of the currently 56 anti virus (AV) solutions that the samples are being run through.
In the figure 2 below, we can see that the majority of samples uploaded is not detected by any AV.

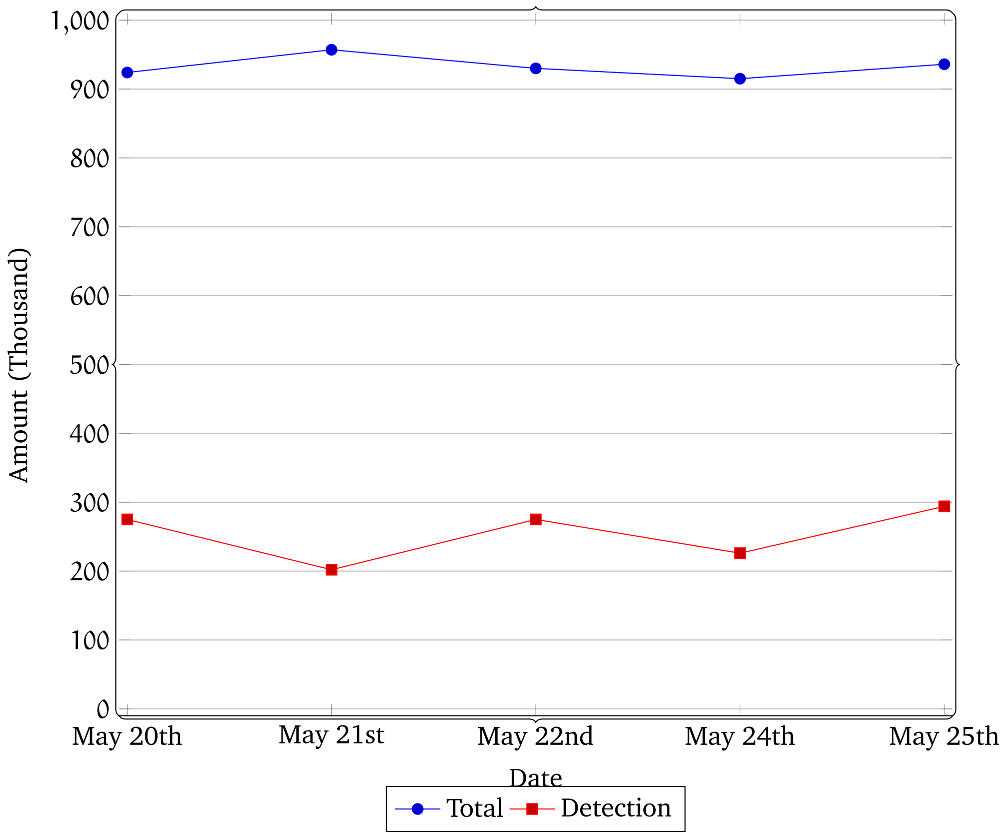

Figure 2: Nr. of samples uploaded

Another very interesting statistic is the distribution of file types that are uploaded to Virustotal.
In the figure 3 below we can see a clear plurality of Windows EXE files.
Followed by Windows DDL files and PDF files.

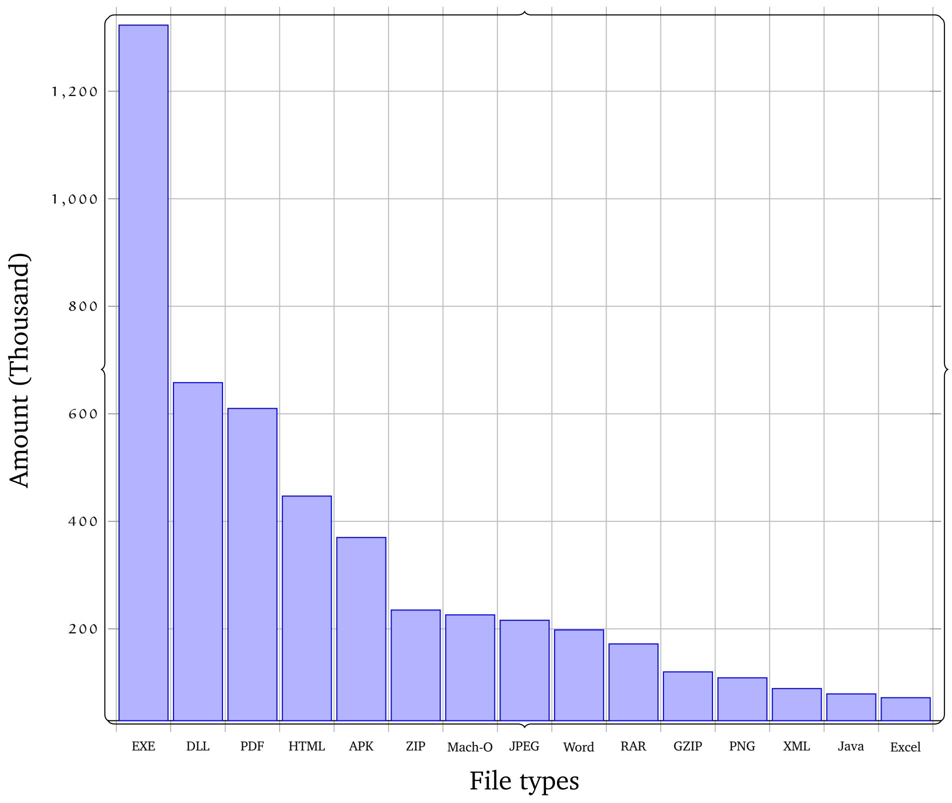

Figure 3: Distribution of file types

5.3 System output

After roughly 1 hour of run time, the system scores about 1600 samples.
The 10 randomly selected samples used for the testing of accuracy of the system, can be seen In the following table 10.
Here we see how quite minor changed can have a drastic impact on the result.
The table shows each feature that was extracted and its given value based on the rules described in the proof of concept chapter.

Sample	**CompTime**	**TimeZone**	**PE Time**	**Obfuscation**	**dataRatio**	**inconsistantHeader**	**StringLang**	**dataLang**	**locale**	**crypto**	**Score**
1	60	100	40	70	80	50	100	100	100	30	81.47
2	60	100	40	70	40	50	100	100	100	30	77.54
3	60	100	0	0	0	50	100	100	100	30	65.24
4	60	0	0	70	80	50	100	100	100	0	64.75
5	60	0	40	70	80	0	0	0	0	30	24.09
6	60	0	40	70	80	0	0	0	0	0	23.11
7	60	0	40	0	80	0	0	0	0	30	18.36
8	0	0	40	0	80	50	0	0	0	30	14.75
9	60	0	0	0	0	0	0	0	0	30	7.86
10	0	0	0	0	0	0	0	0	0	30	0.98
Weight	0.11	0.13	0.06	0.081	0.098	0.06	0.13	0.13	0.14	0.03	
Input Weight	70	80	40	50	60	40	80	80	90	20	

Table 10: Sample output

5.4 Accuracy

The accuracy testing resulted in quite some contradicting data between the precipitants as seen in figure 4 below.

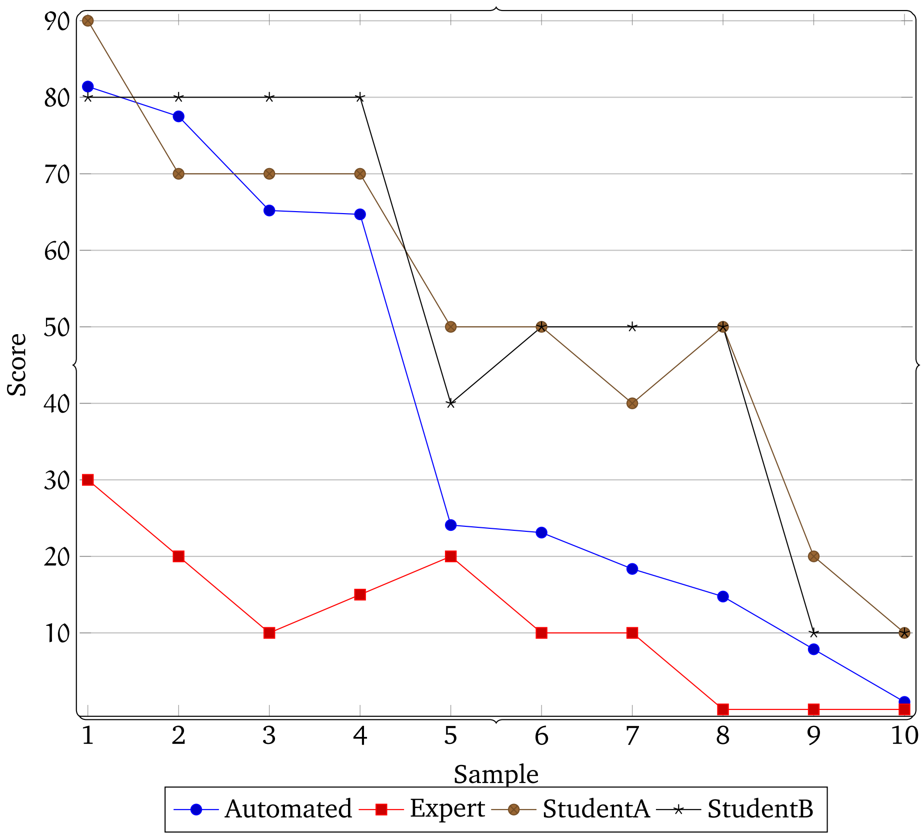

Figure 4: Comparrison of human and automated triage

We can see that the distance vastly varies, probably based on experience of the participant. As the two participants with similar skill level is quite close to each other.
Obviously further testing should be conducted to determine the what the reason behind this spread is, as it might also prove to be the students who have the most accurate triage, and the expert is being too dismissive.
This spread can also be seen by calculating the distance between the participant's score and the automated score as seen in table 5.4.

Sample	1=81.4	2=77.5	3=65.2	4=64.7	5=24.09	6=23.11	7=18.36	8=14.75	9=7.86	10=0.98	Total
Expert	-30=51.4	-20=57.5	-15=50.2	-20=44.7	-10=14.09	-10=13.11	-0=18.36	-0=14.75	-0=7.86	-0=0.98	272.95
Student A	-90=-8.6	-70=7.5	-70=-4.8	-70=-5.3	-50=-25.91	-50=-26.89	-40=-21.64	-50=-35.25	-20=-12.14	-10=-9.02	157.05
Student B	-80=1.4	-80=-2.5	-80=-14.8	-80=-15.3	-40=-15.91	-50=-26.89	-50=-31.64	-50=-35.25	-10=-2.14	-10=-9.02	154.85

Table 11: Distance calculation

Once again we see that the students have almost identical distance values at around 155, while the expert has almost the double at 272.
Higher number is in this case considered worse.
Unfortunately no more participants where able to complete the task at this time.
We can not really base any conclusions on these results due there not being a scientifically significant amount of data.
However, such data can give some indicators on what could be focus of future work.
The selection of participants was also not optimal, with respect to their expertise and knowledge. As seen from the resulting data, experience can have quite a big impact on how one regards data.
It also seems that some accuracy might be lost due to the participants not utilizing the entire score, and rather mostly sticks to the multiple of 10.
This is quite a human thing to do for simplicity, but should be quite easily avoided by pointing it out to the participants before the test.

5.5 Weighting

One of the most integral parts of the weighting method is that each weight actually has an impact on the result, and that is is possible to shift around on the weights to get a more desirable outcome.
Two different scenarios were created during the testing of the PoC, which slightly changed the interest measures of some of the features.
In table 5 we can see the distribution of confident scores based on the given scenarios.
This tells us a lot about how many samples could potentially be discarded, and the reduction of required resources to analyze the rest that would follow.

It also gives an indication on how many samples have some of the selected features.
Note that based on the chosen combination of feature values and weights does not allow any scores to be greater or equal to 90%.
This was not intentional, but it should not matter much as the score itself is only used as a relative comparison between the samples, and it is not designed to say too much as a single entry.

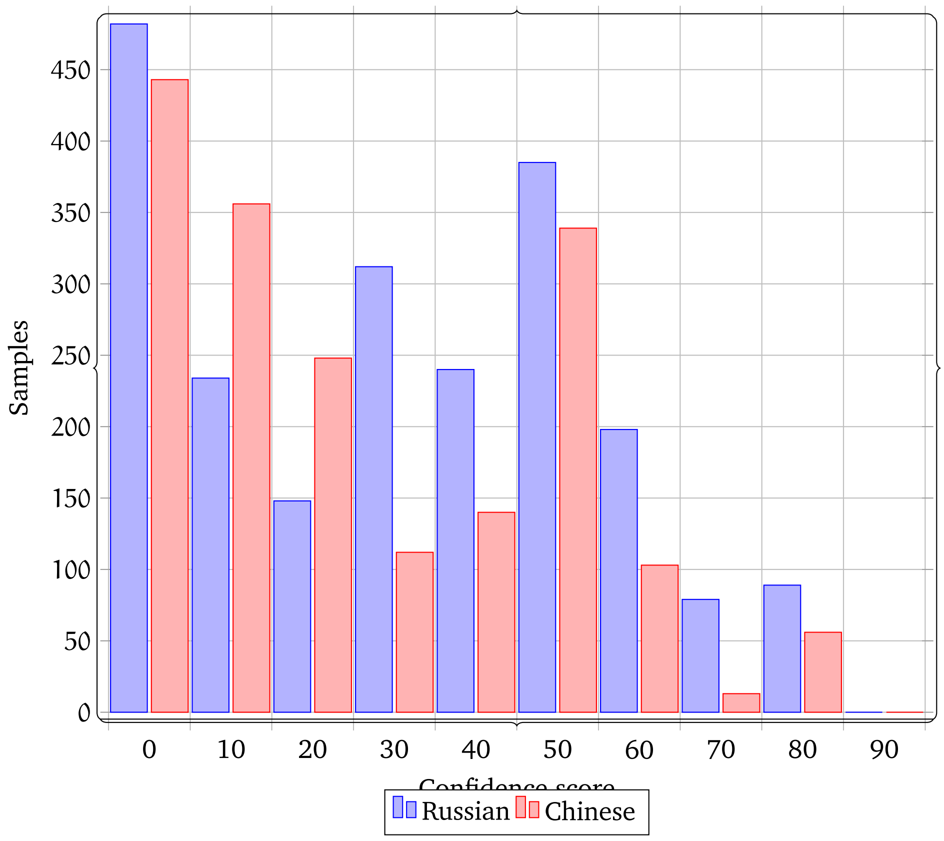

Figure 5: Distribution of scores

This distribution is based on a system run time of 1 hour, which resulted in 2207 samples during the Russian scenario, and 1830 samples during the Chinese scenario.

6 Discussion

Throughout this thesis we have defined an idealistic methodology on how to improve on the aspects brought forth in the problem description, we have created a discussed the creation of program to prove the concept described in said methodology, and we have gathered information and data from resulting tests of the program.
This chapter will tie it all together by discussing and reflecting over choices and results.

6.1 Defining interesting samples

After conducting interviews with multiple security experts and litterateur research, a definition of what determines if a sample is interesting enough to spend analysis resources on, were nowhere in sight. It the became clear that what is interesting is dynamically evolving and changing dependent on recent events and the current situation. But also the specific persons subjective views.
After interviewing 4 malware analysts from the same company, we sat with 4 different views of what would indicate if a sample was interesting.
This resulted in the methodology of defining the currently interesting as seen in chapter 5. By creating a structure for defining what is interesting in a specific point of time, help when communication results and solutions. This was an important research question to answer because everything else in this thesis builds upon it.

6.2 Proof of concept

The proof of concept ended up covering the bare minimum of what would be required to provide data to prove any of our theories, due to all the limitations that had to be considered.
However, it was enough to get some results and data for analysis and answering the research questions, and it ended up working quite well.
There is still much work that can be done in order to improve the program, which might be necessary in order to perform more in depth testing.
Developing a complete system that follows the whole described methodology will require immense amount of work and resources, but based on our findings, it is very possible to provide efficiency and reduce resource requirements in the future.

6.3 Features

The possibilities when it comes to features is almost endless, and it depends on the expertise and knowledge of the analyst whether good features are managed to be extracted.
The proof of concept that was developed extracted an extremely small set of features in order to prove a point.
The set of selected features in the PoC contains multiple quite similar features namely the language specific ones, which makes language very important as they often have the same values. However, these were chosen because attribution is a big part of malware hunting and what users find as interesting. These features also proved easy to find during testing.
One addition we could have made while we already had these features was to make a separate rule which looked for inconsistency of language information by for example detecting different values in every features.
We have a theory that the more features that is included in the rating process, the more refined the result will be, but this still needs to be tested and verified. However, a large amount of feature will probably not be of great help if the user does not understand the features function and how it may impact the scoring in a positive way. As the interest value definition has proven to have a very big impact on the final score of the sample.
Before defining these scores, it is essential that the user knows what the desired output is, and has already defined what he redeems interesting.
Another thing to consider is the possibility of automating the feature selection with the help of machine learning techniques. This will of course require quite a large set of already defined interesting samples for any given scenario, but this might be gathered over time by following the described methodology.

6.3.1 Dynamic features

The proof of concept only utilized features gathered from static analysis. This was due to the immense resources required to automate dynamic analysis of these amounts of files.
However, if such resources is available, it is very likely that a combination of both static and dynamic features will provide more precise results as there are many things hidden in a sample until it is executed.
On thing to consider regarding resource requirements, is to do an initial scoring based on static features, and then chose the top rated samples and extract their dynamic features. This will act as a kind of secondary filtration and might reduce the required resources by quite a bit. As desired features change over time, it is important that the implementation is modular enough to handle introduction and

removing of different kinds of features.

6.4 Weights

By looking at the results in 5.5, we can see that even with very basic features, different samples can be found by adjusting the weights or feature values.
The distribution of samples based on their given score gives a good indication on how many features that can possibly be filtered out or deprioritized during analysis. However, the distribution also showed that a large amount of samples had atleast some of the selected features. This might indicate that the selected features is a little too generic and does not provide useful information to the scoring algorithm. By looking at such a distribution we might be able to further analyze and improve feature selection.
The indication of a possible amount of discarded samples is the closest we come a performance measure, and is probably one of the most important results gathered. If it was proved that everything under the percent of 80 could be discarded, and we were left with a small amount of samples with high probability of baring good CTI, then that could be used to calculate a huge increase in efficiency. As before the analysts would have to pull random samples and triage from a multitude bigger pool of samples.
Even though we did not fully answer the third research question regarding to which extent automation would help improve efficiency of malware analysis, we atleast found a way to get those answers from future work.

6.5 Accuracy

When analyzing the results in 5.4 from the accuracy test, we see quite a distinction between the participants.
This has the obvious reasoning of different expertise and experience between the expert and students, which might have resulted in the expert looking for features which gave a better indication of interest, while the students looked for the easy to find ones that we also had selected. Resulting in comparable rating as the automated system.
By looking at the distance calculation, we see that the expert is quite far from the automated scoring.
The students seem to be very close to each other, which might indicate a similar skill level, but they are also quite far from the automated score.
However, even though the distance metric is quite high, we can still see some correlation between the scores, as even though the expert scored sample#1 quite a bit lower, it is still his highest scored sample for both the automated system and the expert. If arranging the samples by most to least popular, the lists would look quite

similar for all three participants and the automated system.
The confidence scores does not provide much information in of themselves, but is rather an indicator of how interesting the given sample is relative to the other samples.
Because of this, it may not matter that the distance score is quite high, as long as the most interesting stays the most interesting of the given samples.
But due to the low amount of participants, we can't really conclude with anything too strong. Further testing is recommended to get a better picture of the situation and the accuracy.

6.6 Output

Exactly how much of the decision process that should be displayed to the user has not been considered.
As seen in 5.4, the currently displayed information contains the detected feautre's value, the user input weight, the normalized weight, and the final score.
This is quite a verbose output, which could counteractive to the overall efficiency gain we strive after. However, some insight might provide the analyst with information that allows for detection of obvious mistakes. One thing to consider is that the example showed in 5.4 is only 10 samples, whereas there would be thousands upon thousands of such entries where the whole output to be shown.
In a less limited program, there would also be entries with different filet types and values all together.
Having default setting to only show the given score, with the option to show verbose mode would probably be the best solution.

6.7 filtration

When handling such large data amounts as the VirusTotal file feed provides, it will almost always be essential to have some kind of initial filtration.
For the proof of concept we chose to only look at samples that has not been detected by any anti virus solution. This was based on the theory that with such large portion of samples without detection, there is a very high chance of being advanced and interesting malware among them.
However this might not always be the goal, as we have defined interesting as a very subjective term.
And by not testing other forms of filtration we limited our selves to that single output.
By filtration only files with a high detection rate, we might have been able to find advanced malware that has only been briefly analyzed by an automated system, which gathered just a fraction of available good cyber threat intelligence in the

sample.
There is multiple other use cases for such a rating methodology as described in this thesis, and they might require other forms of initial filtration based on the desired output and the type of data sources is available.
An example of a different use case could be automated improvement of network based intrusion detection. In this scenario, a filtration that only kept files which had external communication over the network. This would give a good corpus of files of various types for analysis of how they handle their communication.
It is also important to keep the initial filtration modular, as the desired set of data might change at any time.

6.8 Justification

The core idea and methodology was developed based on discussions and interviews with a third party collaborator which is very highly esteemed in the Norwegian malware community. Because of multiple reasons, it was decided nearing the end of the project, that the best course of action was for them to be anatomized. Unfortunately, this meant that quite a lot of data needed second sources to be found in order to verify various claims.
However, it was decided that after doing the litterateur review, that there is a common agreement within the industry about the problems described in chapter1 As the statistics show in the result chapter 5.2, there are huge amounts of samples that people have found suspicious enough to upload to VirusTotal, but is not detected by any anti virus solutions. The likelihood of there being extremely interesting samples in that mass of data seems quite high. However, analyzing such amounts of samples would require way too much resources. Both automatically and especially manually.
We therefore theorize that by helping filtering out and reducing the amount of samples to look trough can have a large positive impact on how many new samples of malware is found and thoroughly analyzed every day. Leading to better coverage and security for everybody.
The distribution of file types that is being uploaded also gives an indication of which files is most often interesting. This can be used to prioritize which kind of features to focus on and what to include in the initial filters.
We can see that EXE files are by far in plurality, which gives an indication that our focus on EXE files in the proof of concept was a good choice when such a limitation had to be made.
One could also argue that focusing on DLL files or PDF files would be more beneficiary due to still being a major part of the threat landscape, but would require less resources to analyze to to the reduces size of samples.

And lastly, the amount of samples available to analyze proved to be quite large with around 1 million samples every day.
This gives an idea of how much resources one would have to spend to analyze everything, which again leads to the realization and understanding of why there are so many samples that goes undetected.

6.9 Testing process

The method of the testing process was fine, however there should have been more participants in order to get any significant data from it.
Currently we are left with indicators and suggestions of results.
The scenarios used was very target towards Russian and Chinese attribution, not much else other than a couple of features of malware type such as packed files and use of encryption on small sections of the data.
The consistency test however, proved quite successful.
And the same goes for the stability test.
Ideally all the tests would be performed as described in the methodology.

6.10 Evaluation of study

The following section is a brief evaluation on how the study was conducted, and some reflection of challenges along the way.
The scope of the study started out quite small and simplistic, or so we thought atleast.
It became clear, after some extensive, research that the scope was bigger than expected. At one point the described methodology was supposed to be developed and tested.
The scope also shifted a lot during the study based on what kind of resources that became available or went away.
This was a downside with relying on others for information.
However, it worked out in the end.
Another downside of being dependent on others is the increased difficulty of time management and estimating sub goals, as one rarely knew what could be done when.
Finding reputable sources of research aslo proved a challenge, as major parts of research in the field of malware analysis is closed source done for commercial reasons by companies providing anti virus solutions.
These companies do share some insight, but mostly through internet blogs and similar.
And lastly, the required time to gather resources, such as human participants for trials or machine power for analysis, was greatly underestimated.

7 Conclusion

This concluding chapter will give a brief summary of the thesis, followed by a conclusion as a summation of our findings. Then the importance of the conducted study will be discussed, and and a brief overview of possible future work and its possible benefits.
This thesis is then ended with a couple of closing remarks.

7.1 Summary

Analysis of malicious software is a popular method of gathering cyber threat intelligence.
This can be done both manually and automatically, with pros and cons on both sides.
One thing they have in common is that they both require quite a lot of resources, either it is time or computational power.
A problem then arises when the analyst has more software samples to analyze than available resources.
The analyst then has to triage and prioritize which samples to focus resources on.
This triage itself requires quite a lot of resources, which results in a vicious circle.
This thesis proposes a methodology for automating such sample triage, resulting in a list of samples sorted by a score which indicates the probability of containing good cyber threat intelligence.
In order to prove and validate the methodology, a program was developed which followed most of the methods. However, a few limitation had to be considered.
Various tests was conducted on the program output in order to validate it. One of which included comparison between computer and human triage.
The results were not absolute conclusive, due to the small test scale.
Multiple other results where also gathered, and reflected upon before reaching a conclusion.

7.2 Conclusion

Even though the results is not very strongly conclusive, there is still enough to base some conclusions on.
These can be briefly summarized as follows.

- It is shown that aggregation of multiple rules can help filtering down avail-

able samples, and reduce the workload of manual and automated analysis
- Generic rules can be useful if combined with feature values and weights.
- With these weights and feature values, it is possible to dynamically change resulting output depending on situation.
- The proposed methodology of automating human triage of samples, can increase efficiency by give indications on where to best focus resources during analysis.

The field of malware analysis is still quite young, and it is rapidly expanding due to the increasing threat landscape.
There is a huge amount of research to be done in order to increase both effectiveness and efficiency.

7.3 Importance

This research has the possibility of reducing resources required to analyze larger amounts of samples, and have a higher probability that those samples bare good cyber threat intelligence.
Which can result in cheaper CTI, leading to greater acceptance for openly sharing the gathered information.
Which ultimately increases the overall security of our whole digitally dependent society.

7.4 Future work

Throughout this thesis, a lot has been mentioned that could be better or something that should be researched more. The following is a summary of research possibilities for the future related based on this thesis.

Dynamic features

The working theory is that more features will increase the effectiveness of the proposed methodology. This also include dynamically extracted features, even though only static features was used in the proof of concept.
Extracting dynamic features automatically will require a large amount of computational resources, but i would be interesting to see if one could define some really interesting generic dynamic features.

Performance measures

Our results showed indicators of the possibility of increasing efficiency of malware analysis. But no research were committed to testing performance and determine to which extent the efficiency could improve, and what that might mean for the field of malware analysis as a whole.
One of the main contenders to provide such results is to analyze the time saved by

discarding everything with a score beneath a certain threshold.
The figure5 gives an indication of how much one could end up reducing the pool of samples why following this method.

Other applications

The proposed methodology of dynamically weighting features has the possibility to be applied to a host of other applications. Examples is network intrusion detection systems and their rules, and of course other types of intrusion or prevention systems. It could also be interesting to see it applied to very specific trigger systems such as VirusTotal's own YARA tool.

Increased testing

Both the proof of concept and the testing was quite simplistic of nature. However, it would be interesting to see a mare fetched out system with large scale testing of various use cases.
Just an extension of the testing described in this thesis will probably give some good and interesting results.

Automating feature selection

This thesis has described the quality of feature extraction and selection to be very dependent on the users knowledge and ability to understand the importance and function of each feature.
Exploration into automating the process of feature selecting and setting correct weights could further increase the gained efficiency of the described methodology.
One idea to accomplish this is to implement machine learning algorithms that learn what interesting looks like based on already analyzed and manually scored samples.

User experience

The proposed methodology is only going to help if the users can utilize the system. It is therefor important to research and improve the user experience. Maybe by creating a graphical interface.
This could also help increase efficiency even further.

Scalability

One of the elephants in the room is the scalability concerns.
Because of the resource requirements of static and dynamical analysis, together with the fact that more and more suspicious samples are available and in need of analysis, can result in there not being enough resources available to follow the described methodology in the future.
It is therefor important to research the growth of samples, and how the systems react to this growth.

7.5 Closing remarks

This project started with quite a simple idea, namely "Automate a way to find some interesting samples to analyze". The scope rapidly grew and a multitude of challenges arose.
At the end, the project stands more complex than it started, but quite a bit less than it was mid development.
However, the end results turned out quite alright, with the added bonus of more work for the future.

Bibliography

[1] Eilam, E. 2011. *Reversing: Secrets of Reverse Engineering*. John Wiley & Sons.

[2] Eric M. Hutchins, Michael J. Cloppert, R. M. A. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Technical report, Lockheed Martin Corporation, 2000.

[3] Malwr. 2017. Malwr about. https://malwr.com/about/.

[4] Virusshare. 2017. Virusshare about. https://virusshare.com/about.4n6.

[5] VirusTotal. 2016. About VirusTotal. https://virustotal.com/en/about/. "[Online; accessed 02-Dec-2016]".

[6] VirusTotal. 2017. Yara about. http://yara.readthedocs.io/en/v3.6.0/. Accessed: 2017-05-20.

[7] Moser, A., Kruegel, C., & Kirda, E. Dec 2007. Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, 421–430. doi:10.1109/ACSAC.2007.21.

[8] Ligh, M., Adair, S., Hartstein, B., & Richard, M. 2010. *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. Wiley Publishing.

[9] Egele, M., Scholte, T., Kirda, E., & Kruegel, C. March 2008. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.*, 44(2), 6:1–6:42. URL: http://doi.acm.org/10.1145/2089125.2089126, doi:10.1145/2089125.2089126.

[10] Sergio Caltagirone, Andrew Pendergast, C. B. The diamond model of intrusion analysis. Technical report, cciatr, 2000.

[11] Perdisci, R., Lanzi, A., , & Lee, W. 2008. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. *IEEE Transactions on dependable and secure computing*.

[12] Kolter, J. & Maloof, M. A. 2006. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*.

[13] Royal, P., Halpin, M., Dagon, D., Edmonds, R., & Lee.Polyunpack, W. 2006. Automating the hidden-code extraction of unpack-executing malware. *Annual Computer Security Applications Conference (ACSAC)*.

[14] Schultz, M. G., Eskin, E., Zadok, E., & Stolfo, S. J. 2001. Data mining methods for detection of new malicious executables. *IEEE Symposium on Security and Privacy*.

[15] Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., & Kirda, E. 2009. Scalable, behaviour-bsed maware clustering. *ISOC*.

[16] sqlite. 2017. Sqlite limits. http://sqlite.org/limits.html. Accessed: 2017-05-20.

[17] Carrera, E. 2017. Pefile. https://github.com/erocarrera/pefile.

[18] Moritz Raabe, W. B. 2017. Automatically extracting obfuscated strings from malware using the fireeye labs obfuscated string solver (floss). https://www.fireeye.com/blog/threat-research/2016/06/automatically-extracting-obfuscated-strings.html. Accessed: 2017-05-20.

[19] Nakatani, S. 2010. Language detection library for java. https://github.com/shuyo/language-detection.

[20] Phi-Long. 2017. Guess language. https://bitbucket.org/spirit/guess_language. Accessed: 2017-05-20.

[21] ngramj. 2017. Ngramj. http://ngramj.sourceforge.net/. Accessed: 2017-05-20.

[22] linux. 2017. strings manual. https://linux.die.net/man/1/strings. Accessed: 2017-05-20.

[23] Al-sharhan, S., Karray, F., Gueaieb, W., & Basir, O. 2001. Fuzzy entropy: a brief survey. In *10th IEEE International Conference on Fuzzy Systems. (Cat. No.01CH37297)*, volume 3, 1135–1139. doi:10.1109/FUZZ.2001.1008855.

[24] Shannon, C. E. January 2001. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1), 3–55. URL: http://doi.acm.org/10.1145/584091.584093, doi:10.1145/584091.584093.