JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Stefan Bachmair

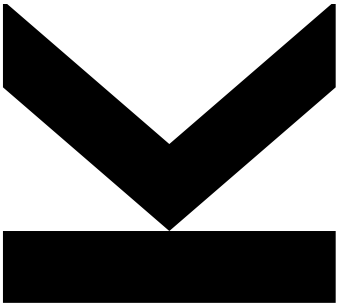
Submitted at
Institute of Networks and Security

Supervisor
Univ.-Prof. Priv.-Doz. DI Dr. René Mayrhofer

Co-Supervisor
(External)
Stefan Proksch
Christian Praher

Month Year
11 2016

EXTERNAL SECURITY ANALYSIS OF EXISTING WINDOWS SOFTWARE BASED ON A CASE STUDY



Master Thesis
to obtain the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

JOHANNES KEPLER UNIVERSITY

Abstract

Faculty of Engineering and Natural Sciences
Institute of Networks and Security

Diplom-Ingenieur

EXTERNAL SECURITY ANALYSIS OF EXISTING WINDOWS SOFTWARE BASED ON A CASE STUDY

by Stefan BACHMAIR

Large software packages are often the result of a development cycle, which dated over several years back. This leads to that some vulnerabilities can rise, which the developers were not aware of during their current testing cycle.

This thesis answer the question if it is possible for an external person, with only limited access to the classified internal information like the source code, to help the company to try to find such additional vulnerabilities.

This was done with a greybox approach, where only the needed information is given to the tester by the developers. It is possible for the company, to use this concept in addition to their currently used threat mitigation or as a basis for further investigations to find potential exploits in their software.

To check, how this approach performs in a real life example, a case study with an enterprise level Windows based software was done. The feedback from the developers backed the usefulness of this method, as some new insight in their software was gained, as it shows the security aspects from another point of view compared to internal tests and documentation.

JOHANNES KEPLER UNIVERSITY

Abstract

Faculty of Engineering and Natural Sciences
Institute of Networks and Security

Diplom-Ingenieur

EXTERNAL SECURITY ANALYSIS OF EXISTING WINDOWS SOFTWARE BASED ON A CASE STUDY

by Stefan BACHMAIR

Komplexe Softwarepakete sind oft das Resultat eines jahrelangen Entwickungsprozesses. Dadurch konnen Angriffspunkte entstehen, über die Entwickler während des Testens der Software noch nicht berücksichtigen konnten.

Diese Arbeit beantwortet die Frage, ob es für eine externe Person möglich ist, mit nur sehr eingeschränkten Zugriff auf firmeninterne Informationen wie den Sourcecode, den Entwicklern zu helfen solche zusätzliche Angriffspunkte zu finden.

Die Umsetzung erfolgte mit einem Greybox Ansatz, wo nur die benötigte Information von den Entwicklern zur Verfügung gestellt wurde. Das Softwareunternehmen soll in der Lage sein, dieses Konzept zusätzlich zu der bereits vorhandenen Behandlung von Softwareangriffen oder als Basis für eine weitere Untersuchung auf Schwachstellen in der Software zu verwenden.

Um zu überprüfen, wie dieser Ansatz an einer aktuellen Softwarelösung runktioniert, wurde eine Fallstudie mit einer Windows basierenden Unternehmenssoftware durchgeführt. Anhand der Rückmeldung der Entwickler konnte die Nützlichkeit dieser Arbeit bestötigt werden, da diese eine zusätzliche Betrachtung der Software aus einem anderen Blickpunkt ermöglichte.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Univ.-Prof. Priv.-Doz. DI Dr. René Mayrhofer for the continuous support of my Master study, for his patience, motivation, and knowledge.

My special thanks are extended to the staff of Sophos Linz for giving me the opportunity to create my thesis in using their software product. I am particularly grateful for the assistance given by Mr. Stefan Proksch and Mr. Christian Praher. Without their dedicated involvement, this thesis would have never been accomplished.

Last but not the least, I would like to thank my family and friends for supporting me throughout writing this thesis and my life in general.

Contents

1 Introduction

Software development is a continuing process, so many large software packages, which are currently available on the market, started as small tools several years ago and changed over the time into the large products, they are now. So, it is not surprising, that over these years several factors come into play, which increase the complexity of the software:

- Addition of features
- Support of other platforms
- Support or migration to a newer OS version
- Change of the development environment
- Change of team members
- And more...

These factors alone are quite challenging during further development, but there is an additional and crucial aspect, which must be taken care of. Because the exploitation of vulnerabilities in computer software is an actual threat nowadays, it must be made sure that the developers do everything possible to avoid such vulnerabilities.

This is where threat mitigation techniques are used, some of the most established ones are Static Code Analysis, STRIDE, Attack Trees, and Threat Modeling tools. These allow the developers to find potential vulnerabilities in their software during the development cycle, where they can still be fixed without causing damage in a productive environment.

1.1 Motivation

These well proven techniques mentioned above are normally applied during the planning stage of the software and grow with the further development. So, it is quite complex and time consuming to introduce these tools on an already existing large scale software product. This leads to the problem that the different teams, which work on the software, do their own vulnerability mitigation (for example Static Code Analysis), but only on smaller scale, because it is not centrally organized and reviewed like it is possible with Threat Modeling for example.

To solve this issue, it is helpful to have a recommended strategy to add a well proven threat mitigation technique to an already existing software product.

So, the motivation of this thesis is to introduce an organized threat finding technique to an already existing large scale software product.

1.2 Goals

The goals of this thesis can be interpreted as a check list, which the different sections of the thesis are based on:

- Gather information of the software in a blackbox approach
- Build a threat model of the software based on the found information
- Identify potential vulnerabilities
- Select some of the found vulnerabilities and check if they can be exploited
- Verify, if this approach was useful for the developers to further improve the security of their software
- This thesis is adapted to match the requirements to analyze the case study, but the methology has to be as general as possible to allow the usage with other software packages.

1.3 Restrictions

To gain acceptance from the developers and managers, this technique must be simple to use and the impact on the current work flow must be as small as possible.

Also, it must be possible that an external contributor can do this task, so only limited access to the proprietary source code must be necessary to keep the confidentiality of the company property intact.

There are many different platforms to run software on, so only software for Microsoft Windows will be part of this work. However, the principle must be applicable to software for other platforms. Also, the network communication is out of scope for this thesis as well.

Since the analyzation of every aspect from a large scale software is time and resource consuming, this thesis only concentrates on the most crucial parts of it, which had been given from the software company.

These considerations give the following list of restrictions of this thesis:

- Tools must be easy to use
- The tester must not require any source code, relevant information must be provided by the developers

- Windows based
- Network communication was out of scope
- Focus on security critical aspects rather than full product

1.4 Approach

The main strategy to achieve this task consists of the following points:

- Evaluate the currently available threat finding techniques and tools
- Enumerate the most common threats targets of the software
- Make an approach for finding applying a threat mitigation tactic
- Make a case study by applying the approach to an already existing software
- Evaluate what worked well and what not

The evaluation of currently available tools consists of finding out how such a test can be classified, and what well known techniques are available and what their advantages and disadvantages are. Not all of them are suitable for this thesis, and tradeoffs have to be made.

To make finding potential exploits easier, an enumeration of targets for exploitation is done. To list all potential targets is too excessive, so only the most promising ones are taken into consideration.

Using the knowledge from this already available works, which are needed in the last 2 points, an approach on how to analyze a large scale software is discussed. This approach can be applicable to a large variety of different software, so it have to be as generic as possible.

Then, the approach is applied to an existing enterprise scale software. In this case, the Windows based client was chosen, because this thesis was done in cooperation with Sophos Linz.

Finally, a discussion is made on how the approach performed and what steps must be taken to further improve the security of the software.

1.5 Threat Model

The attacker, which the threatmodeling is based on, is limited to the following properties:

- Uses a standard user account
- Only has local access to the computer
- No physical modification of the computer hardware is possible

The analysis was done from the point of a normal user, because an administrator has access to restricted components of the system.

Since the network part is out of scope for this thesis, only the local access is taken into account.

Physical modifications of the hardware is out of scope, because the software cannot prevent such attacks.

1.6 Case Study

The software used for the case study was the Windows client of Sophos SafeGuard Enterprise. This is an enterprise level software for encryption and a detailed description of it is made in section 4 in this thesis.

1.7 Background

When a software is tested, one of the first questions include what type of software test is being done. This helps the involved persons to get a first idea what principles and techniques are used in the test.

1.7.1 Black-, White- And Greybox Testing

The most common classification concepts for security testing of software are [1]:

- Blackbox Testing – Examine fundamental aspects of the system without having any knowledge of internal logic and structure of the code
- Whitebox Testing – Investigation of internal logic and structure of the code. Full knowledge of the source code is necessary.
- Greybox Testing – Test the software with limited knowledge of the internal working

So, the greybox testing technique fits the requirements for this thesis be best, since the source code is only partially available (via the developers) and there is only limited information of the internal functionality of the software product.

Compared to whitebox and blackbox testing, the advantages and disadvantages of grey box testing are described in 'A Comparative Study of White Box, Black Box and Grey Box Testing Techniques' as follows[1]:

Advantages:

- *Grey box testing provides combined benefits of white box and black box testing techniques.*
- *In grey box testing, the tester relies on interface definition and functional specification rather than source code.*
- *In grey box testing, the tester can design excellent test scenarios.*
- *The test is done from the user's point of view rather than designer's point of view.*
- *Create an intelligent test authoring.*
- *Unbiased testing.*

Disadvantages:

- *Test coverage is limited as the access to source code is not available.*
- *It is difficult to associate defect identification in distributed applications.*
- *Many program paths remain untested.*
- *If the software designer has already run a test case, the tests can be redundant.*

Some of the disadvantages, like the test of the source code or the untested program paths, can be mitigated by using additional software testing techniques, which are described later.

1.7.2 Static And Dynamic Analysis

Another well known classification, which is often mentioned when software is tested, is the distinction between static and dynamic analysis.

The basic principle about these analysis techniques is that static ones are done by analyzing the software without running it, while the dynamic one is inspecting the software while it is executed.

In an online article by William Jackson, he lists the advantages and disadvantages of static and dynamic software tests, as they were listed by Maj. Michael Kleffman of the Air Force's Application Software Assurance Center of Excellence [2]:

Static analysis advantages:

- *It can find weaknesses in the code at the exact location.*
- *It can be conducted by trained software assurance developers who fully understand the code.*
- *It allows a quicker turn around for fixes.*
- *It is relatively fast if automated tools are used.*
- *Automated tools can scan the entire code base.*
- *Automated tools can provide mitigation recommendations, reducing the research time.*

- *It permits weaknesses to be found earlier in the development life cycle, reducing the cost to fix.*

Static analysis limitations:

- *It is time consuming if conducted manually.*
- *Automated tools do not support all programming languages.*
- *Automated tools produce false positives and false negatives.*
- *There are not enough trained personnel to thoroughly conduct static code analysis.*
- *Automated tools can provide a false sense of security that everything is being addressed.*
- *Automated tools only as good as the rules they are using to scan with.*
- *It does not find vulnerabilities introduced in the runtime environment.*

Dynamic analysis advantages:

- *It identifies vulnerabilities in a runtime environment.*
- *Automated tools provide flexibility on what to scan for.*
- *It allows for analysis of applications in which you do not have access to the actual code.*
- *It identifies vulnerabilities that might have been false negatives in the static code analysis.*
- *It permits you to validate static code analysis findings.*
- *It can be conducted against any application.*

Dynamic analysis limitations:

- *Automated tools provide a false sense of security that everything is being addressed.*
- *Automated tools produce false positives and false negatives.*
- *Automated tools are only as good as the rules they are using to scan with.*
- *There are not enough trained personnel to thoroughly conduct dynamic code analysis [as with static analysis].*
- *It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.*

This comparison shows that is better to use an approach, where static and dynamic analysis techniques are used in combination, and not just rely on one method alone. This mitigates some of the limitations and gives a larger insight into the software to test.

2 Related Work

There are several different works regarding the analysis of potential security flaws of software products, so there must be a selection made regarding what techniques and tools will be the most helpful ones during the inspection of the software product to test.

The first step is a comparison of common known techniques for finding potential exploits. Not all of them are suitable for this work. Also, a selection of the most promising points of attacks on Windows software are discussed.

At last, other techniques to further improve the security of the software are mentioned. These are out of scope of this thesis, but are recommended to include in the development process.

2.1 Finding Vulnerabilities

While it is possible to find potential vulnerabilities for small software projects by writing simple test cases and 'playing around', this is not practical for larger projects at all. So a structured tactic to analyze the product for potential vulnerabilities is needed.

There exist several different approaches, each of them has its advantages and disadvantages, depending on the product and the environment, where it is used. Adam Shostack, who is the developer of Microsoft's SDL Threat Modeling Tool has listed the following major methods for finding potential threats [3]:

- STRIDE
- Attack Trees
- Attack Libraries

To find out, which of these methods will suite the requirements of this thesis the best, a comparison of the advantages and disadvantages is made. However, there are other tools and strategies as well, but listing all of them or even make a detailed comparison between them is out of scope.

2.1.1 STRIDE

STRIDE, which is an acronym that stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege, is a method of finding potential threats in technical systems and was introduced by Loren Kohnfelder and Praerit Gark in 1999. Their article for internal use at Microsoft is called 'The Threats to Our Products' but is made

available via a blog post by Adam Shostack [4].

The principle of STRIDE is to try to enumerate threats by their categories, so it's basically trying to find out what can possibly go wrong in a program.

The different threats and their corresponding violated security properties are as follows [3]:

TABLE 2.1: STRIDE

Threat	**Property Violated**	**Threat Definition**
Spoofing	Authentication	Pretending to be something or someone other than yourself
Tampering	Integrity	Modifying something on disk, on a network, or in memory
Repudiation	Nonrepudiation	Claiming that you didn't do something, or were not responsible. Repudiation can be honest or false, and the key question for system designers is, what evidence do you have?
Information Disclosure	Confidentiality	Providing information to someone not authorized to see it
Denial of Service	Availability	Absorbing resources needed to provide service
Elevation of Privilege	Authorization	Allowing someone to do something they're not authorized to do

It can be tiresome to try to find all potential threats, so there are variants of STRIDE, which make it more comfortable to work with:

- STRIDE per Element - simplified version of STRIDE, only check threats against the most likely affected elements (External Entity, Process, Data Flow, Data Store)
- STRIDE per Interaction – another simplification of STRIDE, with improvements over STRIDE per Element. Uses threat enumeration over tuples (origin, destination, interaction)
- DESIST - (Dispute, Elevation of privilege, Spoofing, Information disclosure, Service denial, and Tampering), a variant of STRIDE

STRIDE is a powerful and flexible system to find threats in all different kind of technical systems. It is difficult for unexperienced persons to find potential threats, but modern tools, which support STRIDE, are already shipped with libraries of all different kind of threats. This makes the usage

accessible for non security experts.

The tool, which was selected in this thesis for the threat modeling (see section 'Threat Modeling Tools') is based upon STRIDE.

2.1.2 Attack Trees

Attack trees are another approach to find threats in technical systems. It is basically a tree based approach to describe and find potential vulnerabilities in a system, based on varying attacks. The root node is usually the goal of the attack, and the leaves the different ways to achieve it [5].

There are differences in variants for visual representation, but also there are logical differences between different attack tree systems. For example, there are OR Attack Trees, where only 1 node is necessary, while on AND Attack Trees all of them must occur (there are mixed variants possible as well).

An example attack tree looks like this:

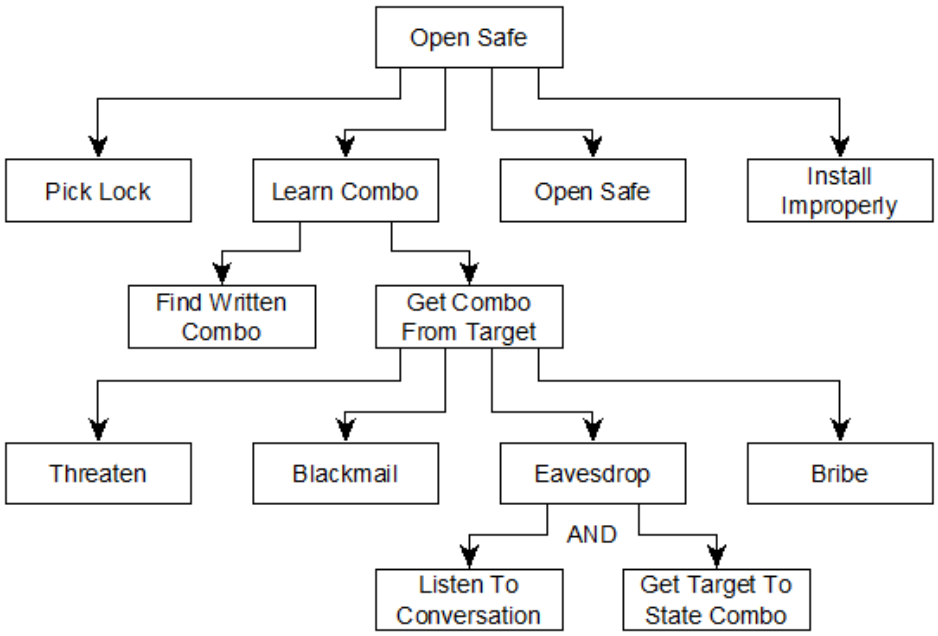


FIGURE 2.1: Attack Tree per Bruce Schneier[5]

Attack trees require deeper understanding about the technical system to analyze. There are sample attack trees available, but for this thesis they are not applicable. Since it takes too much time from the developers to build such specified attack trees for the product, attack trees are not used here.

2.1.3 Attack Libraries

Compared to STRIDE, attack libraries try to find potential threats on a more low level approach. Such libraries can be built upon the security expert itself, or already available libraries can be used.

Examples for preexisting attack libraries are:

CAPEC (Common Attack Pattern Enumeration and Classification)

CAPEC is described as a comprehensive dictionary and classification taxonomy of known attacks that can be used by analysis, developers, testers, and educators[6].

The total number of attack patters are 504 (August 2016), sorted in different categories (mechanisms of attack or domains of attacks)

OWASP Top Ten Project

The latest version is from 2013, but they are currently (August 2016) collecting data to update a new version most likely in 2017. The Top Ten List 2013 is[7]:

- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Cross Site Scripting (XSS)
- A4 Insecure Direct Object References
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Missing Function Level Access Control
- A8 Cross Site Request Forgery (CSRF)
- A9 Using Components with Known Vulnerabilities
- A10 Unvalidated Redirects and Forwards

Attack libraries are better suited in an internal security test environment, because the developers have better insight in the different parts of the software to test it against the predefined libraries or can built an individual attack library themselves. However, for this thesis it is too much overhead to go through all the attacks listed in these libraries, but they can be useful to get ideas for additional vulnerabilities, which are not handled by this thesis.

2.1.4 Threat Modeling Tools

Since the modeling of a threat model all by yourself is uncomfortable, there are several tools available, which will make this process easy to handle. In the Threat Modeling book, there are following tools listed[3]:

- TRIKE
- SeaMonster
- Elevation of Privilege
- ThreatModeler
- Corporate Threat Modeler (not available anymore)
- SecurITree
- Little-JIL
- Microsoft's SDL Threat Modeling Tool

To select a suitable tool for this thesis, a comparison between these different tools is made:

Name	**TRIKE**		
Licence	Open Source	**Free**	Yes
Description			
Trike is an open source threat modeling methodology and tool and there are 3 different versions of it (1, 1.5 and 2). Depending on the version, the documentation is not yet fully complete, but threat modeling is still possible with it. To use Trike, either a standalone tool or an Excel spreadsheet can be downloaded from the homepage. However, the usability was not intuitive, especially when using the spread sheet.			

Name	**SeaMonster**		
Licence	Open Source	**Free**	Yes
Description			
SeaMonster is a graphical tool for building threat models, which supports attack trees and misuse case modeling. This makes it better suitable for the developers itself than for an external tester.			

Name	**Elevation of Privilege**		
Licence	CC BY 3.0 US	**Free**	Yes
Description			
Elevation of Privilege is basically a card game which helps clarify the details of threat modeling and examines possible threats to software and computer systems. This can be a fun way for the developers to find potential vulnerabilities, but it is not suited for this thesis.			

Name	**ThreatModeler**		
Licence	Proprietary	**Free**	No
Description			
ThreatModeler is a commercial software for threat modeling, attack surface analysis, and other useful help to find potential vulnerabilities. It supports a centralized threat library, (automatically generated) attack trees, threat model templates and the chaining of threat models. The tool was not tested, because the access for a demo must be scheduled and is only available for companies.			

Name	**SecurITree**		
Licence	Proprietary	**Free**	No
Description			
SecurITree is, as the name suggests, an attack tree based threat analysis tool. Similar to ThreatModeler, SecurITree was not tested because of the commercial nature of the software.			

Name	**Little-JIL**		
Licence	Proprietary	**Free**	Yes
Description			
Little-JIL is a graphical language for defining processes, but not specifically build for threat modelling. So, threat finding features are not included like in other tools. This makes Little-JIL not as suitable as them.			

Name	**Microsoft's SDL Threat Modeling Tool**		
Licence	Proprietary	**Free**	Yes
Description			
This is a graphical threat modeling tool, which is based on the STRIDE concept. It is easy to use and understand, but larger and complexer models need to be split into several different smaller ones or else it is too confusing.			

The selected tool for this thesis is Microsoft's SDL Threat Modeling Tool. The reasons for this decision are:

- The software is Windows based, and therefore the tool can be used out of the box
- It's free
- Easy to understand, even for non developers
- Some of the developers already have used it for smaller parts of the software
- It's updated regularly, and because of that it includes newer threats

2.2 Selecting Potential Exploit Targets

As already mentioned during the explanation of Attack Libraries, there is a wide range of potential vulnerabilities available. Testing all of them is a

huge amount of work, and since the focus of this thesis is about to introduce a way for a security analysis of a Windows based software to find potential vulnerabilities and then try to exploit a selection of them as a proof of concept, it is out of scope to test the software for all possible exploits.

The following potential exploit targets were selected because they represent a wide range of different points of attacks. They were the most promising ones to find vulnerabilities during the case study.

But it must be noted, that these exploit targets are only a recommendation, and as already mentioned they are by far not all available targets. Also, due to the limit of resources, only a small selection can be discussed during the case study, so further work is still needed to test them in depth.

The major parts of the software, which were selected for finding potential security flaws are:

- Binaries
- Access Rights / Permissions
- Communication Paths
- Memory

Tampering with the binaries can be done using several different ways. Some of them require the manipulation of the binaries itself, while other abuse the absence of security enhancements. Also, when the permissions to the files and services are set too weak, an attacker can use this to its advantage.

Another way to manipulate the software or to get classified information is to tamper with the data, which is used by the software, itself. This can be with the data files of the software or with the content of the memory itself. So, when there is unprotected data stored in there, these is a promising target for attackers. Another place, where information can be stored, is the Windows Registry.

The communication over the network to the server is not part of this thesis, so there is only the IPC (Inter Process Communication) checked. The relevant technologies used by the case study software are named pipes and WMI (Windows Management Instrumentation)

2.2.1 Binaries

The attacks and therefore also the protection of binaries in Microsoft Windows has evolved over several years. Some of these protections can be done by Windows alone, while others will require the binaries to be rebuild with an actual version of a compiler, which supports it.

The most important defenses against attacks are as follows[8][9]:

- /GS Stack buffer overrun detection.

- /SafeSEH exception handling protection.
- Structured Exception Handler Overwrite Protection (SEHOP).
- Data Execution Prevention (DEP) / No eXecute (NX).
- Address space layout randomization (ASLR).
- Pointer Encoding.
- Heap corruption detection.
- Migration of buffer overrun prone functions to safer versions.

Since some of these defenses are only available during compilation (for example the Stack Buffer Overrun Detection), it can not be verified if the final product was using them when only the binaries are available for inspection.

However, the binaries can be checked if they support the following defenses by the inspection of the following flags in the binary headers[9]:

- DEP/NX
  Data Execution Prevention (DEP) / No eXecute (NX) is used, which prevents code from being executing in data segments. The CPU needs to support this technology to work (Intel – DEP, AMD – NX)
- ASLR
  ASLR moves executable images into random locations when a system boots, making it harder for exploit code to operate predictably. For a component to support ASLR, all components that it loads must support ASLR.
- SafeSEH
  An exception handler is a unit of code executed when an exceptional condition, such as a divide by zero, occurs. The address of the handler is held on the stack frame of the function and is therefore subject to corruption and hijacking if a buffer overflow allows an attacker to overwrite the stack. An advanced method is the use of SEHOP, as this provides a better defense but it requires Windows Vista SP1 and later.

2.2.2 Digital Signatures

When acquiring software from a potential unsafe source, it is problematic to verify that the files were not tampered with (violation of integrity) or where it came from (violation of authenticity). This can allow attackers to change or add additional components to software packages with the intend that unwary users will install it and therefore compromise their system using the tampered code. A possibility to avoid these problems is to provide hashes for the software packages, so that the user can verify the integrity of the data. However, users with non technical background can be overwhelmed by this task.

On modern Windows systems, there is a more user friendly mitigation of these problems using code signing, which ensures the integrity and authenticity of software. A complete description about the Windows implementation of code signing can be found on Microsoft's 'Introduction to Code Signing' at the MSDN[10].

The principle of code signing is built upon digital certificates and digital signatures.

Digital certificates are a set of data, including the public encryption key, that completely identifies an entity. These certificates are issued by a certification authority only after it the identity has been verified.

Digital signatures are created using hashing and a public key signature algorithm. The basic principle of the process for digitally signing and verifying a file is as follows 2.2:

1. A one way hash of the file is produced.
2. The hash is encrypted with the private key, thereby signing the file.
3. The file and the signed hash are transmitted.
4. The recipient produces a one way hash of the file.
5. Using the digital signature algorithm, the recipient decrypts the signed hash with the sender's public key.
6. If the signed hash matches the recipient's hash, the signature is valid and the file is intact.

However, like every technology, the signing of executables has vulnerabilities. These include[11]:

- Copying Certificate information from clean files
- Selfsigned certs with fake name
- MD5 forgery
- Get certified and be evil
- Get certificate with misleading name
- Find someone to sign your stuff for you
- Steal a certificate
- Infect developers' system and get signed with software release

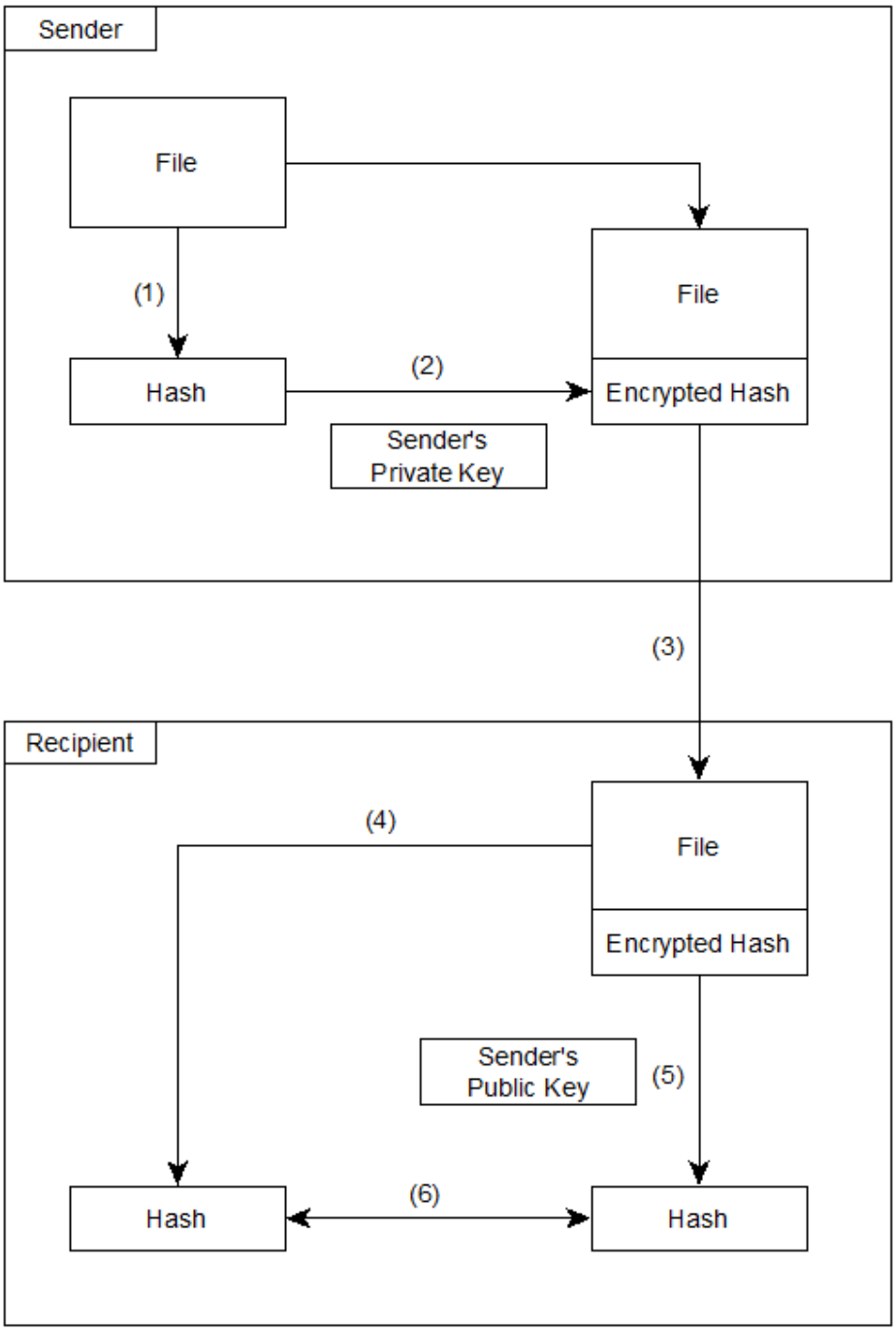


FIGURE 2.2: Digital Signatures per Stallings[12]

Most of these problems are out of scope for this thesis, because planting malicious code and stealing the private key means that parts of the company security itself is compromised. Vulnerabilities of the algorithms is not part of the software itself, so that cannot be checked either.

The part, which can be verified, is that the right company certificate is used to sign the executables for this software.

Another potential exploit, which can be checked, is if the software itself cares about the correct digital signature at all. Because the usage of digital signatures is meaningless if they are not checked in the first place.

2.2.3 Access Rights

As already mentioned in the introduction, the attacker is a normal user without administrative access rights, who has local access to the computer. Therefore, when too weak directory permissions are set for normal users, it allows the attacker to tamper with the files in these directories, which result in the corruption of the data or stops the software from working entirely. So, when there is write access needed (cache files, user specific configuration files), the software must include hardening features to automatically handle corrupt data and configuration files.

There is another critical problem regarding the 'hijacking' of DLLs, which occurs because due to the loading algorithm of DLLs by Windows[13]. The search order for DLL in Windows are as follows:

1. The directory from which the application loaded
2. 32bit System directory (C:\Windows\System32)
3. 16bit System directory (C:\Windows\System)
4. Windows directory (C:\Windows)
5. The current working directory (CWD)
6. Directories in the PATH environment variable (system then user)

So, when unprivileged users have write access to the directories where the executables reside, it is possible to place a forged DLL in there, which acts as a wrapper for the original one, but executes malicious code as the user which starts it. This is problematic for services which are started as the 'SYSTEM' user, because it allows the system to become compromised.

To avoid DLL hijacking, libraries must be loaded with an absolute path and the permissions of the directories must be set restrictive, especially when there are services started automatically.

2.2.4 Memory

The memory is a crucial part of the system, because it is the storage for all different kinds of data, which is used by the operating system and the software. Due to the complexity of the Windows operating system, it is a quite complicated topic to deal with, as not only the RAM is used for this, but also space on the disk (Swap file). The data, which is stored there can be interesting for inspection, because many confidential information can be available there in plain text.

There are technologies, which support the encryption of the memory, for example AMD Memory Encryption[14]. This deals with the problem

when sensible data on the disk is encrypted, but it lacks encryption while it is loaded into memory. This makes snooping this data possible, especially on newer technologies, where nonvolatile memory is used, as these chips can be removed while the data stored there is still intact.

2.2.5 Files

Files are still the primary way for storing information on a computer system. Because of that, this information needs protection to prevent malicious entities from accessing to it.

A special kind of file is the registry, which is an important part of the Windows operating system. It is basically a database in which applications and the operating system can store and retrieve data[15].

With improper access permissions set, an attacker can modify the data of the software to corrupt its settings and its working behavior. Another problem rises, when the data stored in the registry gives the attacker information about potential security flaws.

2.2.6 Named Pipes

Larger modern software packages consist normally of more than one running process. To allow these processes to communicate to each other methods of Inter Process Communication (IPC) are needed.

One such method on Windows based systems are named pipes. They operate in a server client model and support local and remote connections. Basically, the handling of these names pipes is like files on a disk, with the exception that they are deleted when there is no handle left to the pipe[15].

Named pipe unique names have the following notation:

```
\\<machine_address>\pipe\<pipe_name>
```

So, for example, a named pipe on a local machine named 'TestPipe' can be opened with the unique pipe name `'\\.\pipe\TestPipe'`.

Portcullis describes security problems regarding named pipes and possible solutions to them[16]. These problems can occur because of racing conditions when creating the server pipe and when clients have higher privileges as the server pipe, and thus make an escalation of privilege possible.

For a safe implementation of named pipes following security considerations on server and client side are advised:
Server side

- When creating the server instance, it must be made sure, that this is the first instance
- Use pseudo random names, which are hard to guess for the pipe

- When the number of client is known, limit the number of connections to the pipe

Client side

- Use only the minimum level of privileges
- Use the right security flags to prevent impersonation by the server
- When impersonation is needed, use additional protection methods (handshake, encryption, ...)

2.2.7 WMI

Windows Management Instructions (WMI) allows to manage almost all the Windows based computers resources, either on a local or on a remote computer. It is the implementation of the Web Based Enterprise Management (WBEM) standard. This allows the collection and management of data on local and remote machines[15].

The four main parts of WMI, listed in there, are:

- Management applications - Windows applications that access and display or process data about managed objects.
- WMI infrastructure – This is where the core of WMI, the Common Information Model (CIM) Object Manager (CIMOM) provides the connection between the management applications and providers.
- Providers, which need to define and export the representation of the objects that management applications are interested in.
- Managed objects might represent one component, such as a network adapter device, or a collection of components, such as a computer.

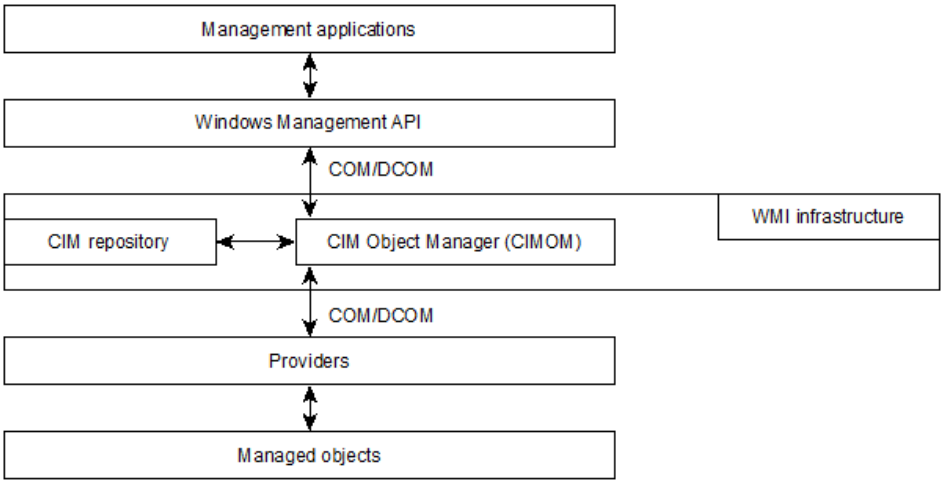


FIGURE 2.3: WMI per System Internals[15]

The security of WMI is based on a namespace level implementation. An administrator can control, which users can access specific namespaces, and therefore allow a management application to connect to it. When the connection succeeds, the application has full access to the object in this namespace.

To avoid problems regarding escalation of privilege, it must be made sure, that the users only have access to the absolute minimum of needed namespaces on a computer.

2.3 Other Improvements

Using the method mentioned before, it is possible to find a wide variety of possible vulnerabilities in the software. Another way to find problematic behavior of software is the use of fuzzers, which is basically a brute force attempt to find problems, which will be caused by unexpected or faulty input. A basic overview of different fuzzing methods, and if they are useful in this thesis is discussed in its own section later.

There are other ways to further improve the security of an application, but require full access to the source code to work properly:

- Code Analysis
- Code Proofing

Because one of the goals is to get as little information on the code as possible, Code Analysis and Code Proofing are out of scope for this work, however it is highly advised to think about the usage of these technologies.

2.3.1 Fuzzing

As mentioned before, fuzzing is a widely used technique used to find potential problems in software, but there are limitations coming with it. In the article "SAGE: Whitebox Fuzzing for Security Testing" the basic principle of fuzzing is explained (which is called blackbox fuzzing) and how to mitigate problems coming with it[17].

Blackbox fuzzing is a blackbox testing technique where data is randomly modified and provided as input to a program to see how it behaves. To make the testing more efficient, rules can be used to generate the data.

This is an effective way to test interfaces, but since there is a random component involved, not all possibilities are tested or at least require a large amount of time (brute forcing all input).

The article gives the following example in C code to demonstrate this limitation:

```
int foo(int x) { // x is an input
 int y = x + 3;
 if (y == 13) abort(); // error
 return 0;
}
```

So, for a 32bit integer as input, there is only a 1 in $2\hat{3}2$ chance to trigger the error. To mitigate this limitation, the fuzzer must know what is going on in the program itself. This is basically what happens using whitebox fuzzing.

While blackbox fuzzing basically relies on random generated input data, whitebox testing analysis the program dynamically to find constrains on inputs from conditional branches encountered during execution. This constrains are then used to generate new input data, so that these execution paths are tested as well.

However, this has not to be confused with code verification, because even if all possible input can be checked theoretically, this is too complex and time consuming to do.

Whitebox fuzzing was not possible due to the limitations to the source code of the software given in this thesis but blackbox fuzzing can be an option. Unfortunately, the data on the communication interfaces was encrypted, so this was not done because this is too complex for this thesis to crack this encryption.

2.3.2 Static Code Analysis Tools

While working on smaller software projects, the developers might still have an overview of most of the components they use and what is going on there on source code level. When the project grows larger, especially when there are several teams working on different subprojects, it is not possible to keep this overview. A deep inspection per hand is not possible, because it is too time consuming and complex to do.

A solution to this problem is the usage of static code analysis with tools, specifically designed for this purpose. These tools can analyze the source code for different problems that might occur.

Examples of such problems are:

- Usage of obsolete functions
- Usage of unsecure functions
- Possible Deadlocks
- Buffer overflows
- SQL injections

Modern compilers might find some of these problems, but it is highly recommended to include such analysis tools in the development cycle. These tools can be effective to find problems in normally hard to reach positions in the program. As the static code analysis does not need a complete software, it can test parts of it. This is useful during the early stages of the development process.

Like whitebox fuzzing, Code Analysis is not be used in this thesis, because of the restrictions about the limited access to the source code. Since the usage of such tools have to be obligatory, especially for security based software companies, it must be checked if these tools are updated on regular basis.

2.4 Summary

While this thesis uses STRIDE with the use of Microsoft SDL Threat Modeling Tool, the other approaches for finding vulnerabilities can be used in addition to it or to get ideas about other attack vectors.
The listed potential exploit targets were chosen because these were the most promising ones for the case study. They are by far not all attacks which are possible for the thesis, but due to the limited resources a selection had to be made.

3 Approach

As already defined by the goals in section 1, this thesis is about to find potential vulnerabilities in a software and then tries to exploit them. Some vulnerabilites can be found by the inspection of the installation of the software, but a much better overview is given when a threat model is used in addition. This allows the identification of other threats, which were not taken into account when no such threat model was used.

With the available technologies and possible attacks discussed in section 2, the question is now how to make use of this knowledge to make a security analysis of the provided software package possible.

The following restrictions must be taken care of in this approach:

- A proprietary software package is provided by the developers
- The result must be reproducible for further investigation
- There are manual and guides from the homepage available, but in general there is only little insight to the software.
- When additional information is needed, an appointment must be made with the developers to get this information
- Only specific parts of the software can be tested

The first step to make security tests from the software is to get used to the functionality of the software itself. This requires the creation of a reproducible test environment.

The next measure, which must be done is to identify the different components of the software. This can be done without the help of the developers. With this information, it is now possible to start the creation of a threat model of the software. While it is possible to create a basic model in this manner, it might be necessary to get input from developers on the more nuanced aspects of the system.

When the threat models are created, potential threats have to be identified and then are tried to be exploited.

In summary, to create a threat analysis from scratch, the following steps were taken:

- Creating the Test Environment
- Information Gathering
- Threat Modeling

- Exploiting Vulnerabilities

A detailed explanation of these steps in done in the following sections.

3.1 Creating the Test Environment

As mentioned before, the crucial point for creating a useful test environment is the creation of reproducible results. On a simple standalone computer, this is unpractical to do (disk cloning, etc...), so it is a much better approach to use virtual machine for this task.

The usage of virtual machines allows the creation of snapshots, which comes handy when the system is damaged between repair. It provides a common starting point for the different exploits, which are tested.

Examples for software, which allows the creation of virtual machines are:

- Oracle VM VirtualBox[18]
- Several different tools from VMware[19]
- Microsoft HyperV[20]

All the virtualizers mentioned above allow the creation of virtual networks, which are handy for complexer test environments.

For the selection on the installed software on the system, it is recommended to ask the developers if there have any preferences. This and the preselection on supported software and operating systems reduce the overhead on the configurations to test. For example, it is useless to try to test x64 software on a x86 system. Also, the inclusion of old software, whose support will be dropped anyway is unnecessary extra work and only have to be done if the developers especially wants it.

For the installed software, only the most recent version is used, except the developers from the software to test have other preferences. But it must be made sure, that no updates are installed during the tests, because these changes in the system can influence the software to test.

3.2 Information Gathering

The information gathering is the first part of the analysis, where the provided software package is inspected. This is done with a blackbox approach and the goal is to get as much useful information of the software as possible.

This starts with the inspection of the installation files. To ensure, that other testers can reproduce the results, it had to be made sure that they use the same version of the installation files. Therefore, a hash was created of these files. In addition it is inspected if the files are protected.

The next step is to document what changes in the system were made during the installation. There are several tools available to compare different system states, so an appropriate one have to be chosen. The most interesting changes include:

- Added directories/files
- Services
- Registry

To track these changes in the system, there are several tools available. One of them is the 'Windows System State Analyzer', which is available from Microsoft and allows to create and compare snapshots of different machine states[21].

Another example for a tool to compare different system states is the 'System Explorer' by Mister Group[22]. This tool gives insight into the system itself (modules, processes, ...), which can become helpful during the inspection.

It is good to know how the different binaries are dependent to another. This means what binaries are listed in the import table of a PE binary. Although, this is only a static dependency, it can be useful when there are security flaws in a binary. This gives the attacker, who was described in section 1.5, the opportunity to target otherwise secure binaries, which depend on the faulty one.

Simple, but powerful attacks rely on the usage of weak access rights to parts of the system (for example files/directories and services). This allows attackers to exploit these permissions to make an escalation of privilege attack or to manipulate the software in a harmful way.

Rather traditional, but still powerful attacks include security flaws in binaries. There are techniques, which mitigate such problems, but they must be used properly. So, it have to be checked if these techniques are implemented.

While small software only consists of one running thread, larger software has many different processes and threads, which need to communicate with each other. This communication lines are a vulnerable point of the system, if they are not protected properly. So, these IPC communications has to be identified as well.

In summary, the following parts of the software were chosen in this thesis:

- Installation files
- Changes in the system
- Dependencies
- Access rights (directory, services)

- Security enhancements of the binaries
- Communication between the components of the software

It must be noted, that it is by far not possible go find all possible information about the software during this stage. However, it can be enough to start basic threat modeling and perhaps identify first potential vulnerabilities.

3.3 Threat Modeling

As mentioned in section 2, the most suitable tool for threat modeling is the Microsoft SDL tool. While it is not possible to set all the properties in the models, and therefore create an automatically generated threat list, it still helps with the identification of potential threats. This is due to the restrictions of the thesis (only limited information available). But it is possible, that the developers use the created threat model and complete the missing information, so they can use the SDL tool as intended.

Since the properties of the different components are not included in the graph, an explanation of the relevant ones is given is given after the graph using tables.

To keep the naming of the components consistent, they must be enumeratated in a consistent way. This must be logically plausible, so that by looking at the number, it must be easy to identify to which part of the software the component belongs to.

Since the SDL tool does not support such an enumeration list, it must be made by hand.

An example for such a list is:

(1) User
(2) Client Software
    (2.1) Process 1
        (2.1.1) Thread 1
        (2.1.2) Thread 2
    (2.2) Process 2
(3) Server Software

3.4 Exploiting Vulnerabilities

When talking about vulnerabilities, it must be kept in mind the special case, when the attacker already has administrative access to the operating system. Therefore, the system is already compromised, so this cannot be mitigated by the software itself. However, it is still useful to test for vulnerabilities as root, as it can be possible that the attacker uses another exploit to

gain specific permissions to parts of the system. An example is the write access to binaries.

During the information gathering and the threat modeling, several possible vulnerabilities are found. In this section, it is now tested, if it is possible to exploit those vulnerabilities to harm or even compromise the system. It must be mentioned during the write up if the potential exploit can be used in conjunction with another attack to have further impact.

The approaches are divided into several categories, and for each approach, a detailed explanation is given on how to reproduce it.

To get a better overview about the exploits which are tried, a table is used to give a description about it. This gives a summary about the attack, so that it is not needed to read through the whole technical explanation of it.

TABLE 3.1: Sample Exploit

Name	Name of the exploit
Impact	Classification of the impact
Attack succeeded	Did the attack or parts of it succeed?
Prerequirements	Are there prerequirements for this attack?
Description	
A short summary of the attack.	
Further use by other exploits	
Can this attack be used in conjunction with another attack to amplify the vulnerability?	

To provide a good starting point for possible exploits, a selection of attacks is discussed in the following points. These are by far not all attacks which attackers are able to make, but it is impossible to check every exploit.

These selection was made because they seemed to be the most promising ones for an attack to succeed. To let them be as representative as possible, different aspects of the software was chosen (binaries, storage, communication). Unfortunately, is turned out that testing all of these selected was not possible, this was either because of some additional protection, which made the attack too complex to execute, or due to the available information was not enough and to make an successful attack attempt more classified information was needed.

Other potential exploits were also possible, but as already mentioned, only the most promising ones in the eye of the tester were chosen.

If possible, a recommendation of tools and further information about the attack is provided. But as already mentioned, these are by far not all available attacks, so it is highly advised to research what specific attacks can succeed on the software to test.

3.4.1 Tampering with binaries

There are several ways to compromise binaries, especially when there are security improvements missing (Digital Signature, ASLR, DEP). During the Information Gathering, the tester creates a list of all binaries and if they are proper protected.

But even if they are properly protected, it is still possible to manipulate them. So it is of interest, if the software recognizes if the binaries are changed. Examples of tests include:

- Can the certificate be removed or changed? - violation of integrity
- Is it possible to add payload to the binary? - violation of integrity

These points are discussed now, and examples are given (if possible).

Certificates

The removing or changing of a digital certificate from binaries in a Windows based software is a easy task to do. There are already tools available to do this, so two specific ones were selected as an example.

The certificate can be removed with a small tool called 'delcert' by deepred, which was posted on the xnadevelopers forum[23]. The syntax to remove the digital signature of a binary is as follows:

```
delcert.exe MyFile.exe
```

which gives the output when succeeding:

```
Target file(s): MyFile.exe

Stripping file: [path]\MyFile.exe.
Succeeded.
```

To sign a binary with another certificate, the command line tool 'SignTool.exe', which is available as a part of the Windows SDK can be used [24]. The syntax used to replace the digital signature is as follows:

```
Signtool.exe sign /a MyFile.exe
```

If there is no suitable certificate file available, a new one can be created by this command:

```
makecert.exe -r -pe -ss MY -sky exchange -n
             CN=MyPrivateCert CodeSign.cer
```

When the binaries are correctly signed, the output look like this:

```
signtool.exe sign /a *
Done Adding Additional Store
Successfully signed: feshellx64.dll
Successfully signed: SGDrvHlp.exe
Successfully signed: sgmbasen.dll
Successfully signed: SGNCredProvn.dll
Successfully signed: WMIListener.exe

Number of files successfully Signed: 5
Number of errors: 0
```

Adding payload to binaries

It is possible to append data to a binary, but this is a trivial method and it can be detected easily with digital signatures. But there is a way to still add additional content to a binary, without the signature taking note of it. This is described in the article 'Changing a Signed Executable Without Altering Windows Digital Signature' by Aymeric Barthe[25].

This is possible, because data is excluded in the hash computation during the signing process. This is because these sections cannot include themself during the generation. The sections are:

- the Checksum in the optional Windows specific header. 4 bytes.
- the Certificate Table entry in the optional Windows specific header. 8 bytes.
- the Digital Certificate section at the end of the file. Variable length.

This allows to add payload in the binary, which will not be detected during the signature check. The attack, which still worked during the creation of this thesis, is described in the article as follows:

1. Locate beginning of PE header (PE)
2. Skip COFF header (+=28 bytes)
3. Go to Certification Table Entry in the Windows specific optional PE header (+=120 bytes after COFF; total +=148 bytes)
4. Change size of Certificate Table as defined in IMAGE_DATA_DIRECTORY.Size to add the size of the payload.
5. Go to location defined IMAGE_DATA_DIRECTORY.VirtualAddress. This is the absolute location of the Certificate Table within the file.
6. Change again the size of the header, inside the PKCS1_MODULE_SIGN.dwLength
7. This should normally be the last section in the executable; so, go to the end and add payload
8. Possibly calculate the new checksum of the file

There is the source code for a small command line tool available to download. This tool currently only supports x86 binaries, so it is not possible to test x64 based software.

The usage of the tool to add a payload is:

```
AppendPayLoad.exe OriginalFile.exe Payload ModifiedFile.exe
```

3.4.2 Windows Registry

For basic attacks of the registry, no special tools are needed, as the Registry Editor is already included in Windows. There, it can be checked, what access rights the different users have to the registry entries of the software. Potential vulnerabilities here are too weak permissions or the violation of confidentiality, when there is classified information stored there.

Other vulnerabilities include a wrong or faulty configuration, which allows the attacker to find weak spots in there. An example are binaries without the full path names, so that the attacker can spoof them.

3.4.3 Services

In Microsoft Windows, services are an integral part of the system. They often run with elevated user privileges (Local System), so when the services are not configured properly, it allows the attacker to successfully attempt an escalation of privilege exploit. So, it must be made sure, that a manipulation or tampering with this services by an attacker is not possible, or at least it is detected and the changes are reverted by the SafeGuard client.

There are several different vulnerabilities, which the attacker can use. Some of them can be easily found with just one line of instruction in the commandline, while others need deeper analysis of the binaries.

The following 3 possible vulnerabilities are discussed:

- Stopping, disabling, or modifying services
- Unquoted service paths
- DLL hijacking

Stopping, disabling, or modifying services due to weak permissions

This is a rather straightforward attack, so it must be checked, if the access rights are properly set.

Unquoted service paths

A simple misconfiguration in the service properties can be used to start other executables than the service. A blog post on commonexploits.com[26] describes this misconfiguration as 'Unquoted service paths', because the binary path was entered without quotes. This is problematic when the path contains whitespaces. An example about this problem is as follows:

This is the correct way to start the program:

```
"c:\program files\sub dir\program name"
```

And this is the problematic one:

```
c:\program files\sub dir\program name
```

While the problematic one still starts the service, Windows also tries to start the following executables, if they are available:

```
c:\program
c:\program files\sub
c:\program files\sub dir\program
```

This allows an attacker to place malware at these locations, so it will be started as a service with its access rights. A detailed explanation about this vulnerability, can be found at the commonexploits website[26].

To find such misconfigured services, only a single command can be used:

```
wmic service get name,displayname,pathname,startmode |findstr /i "
    auto" |findstr /i /v "c:\windows\\" |findstr /i /v """
```

DLL hijacking

The basic principle of this attack is to intercept the DLL loading of the service with a DLL, which is provided by the attacker. A full description about this attack is in the article 'Elevating privileges by exploiting weak folder permissions' on GreyHatHacker.NET[27].

To allow the exploit to be successful, the following conditions must be met:

- One of the paths of the DLL search order has write access to the attacker
- The service tries to load a DLL, which is either not available or is in a lower ranking of the DLL search order, so that the attacker can put its own DLL in a higher ranking one

The write access to a DLL search path is due to weak folder permissions. This can be the result when program adds its own path to the search path and still has write access for more users than necessary to it. This is out of scope for the SafeGuard client, because the folder permissions are set properly and they are not added the search path.

To see which DLL are loaded and what their dependencies are is too complex to do without the proper tools (see Information Gathering – Dependencies). It is better be done with static source code analysis from the developers itself.

So, this potential exploit was not researched any further because it is too time consuming for this thesis, but it is highly recommended to do this in further research. In a proper configured system, the folder permissions prevent an attacker to use this exploit, but it is better to make it as hard as possible to use this exploit.

3.4.4 Drive Encryption / BitLocker

Actual versions of Microsoft Windows have a build in drive encryption called BitLocker, which is used by SafeGuard.

When the user has administrative rights, it is trivial to get the recovery key with the following command:

```
manage-bde -protectors c: -get
```

The same result can be achieved when the WMI class Win32_EncryptableVolume from the namespace Root\cimv2\Security\MicrosoftVolumeEncryption is used.

Another attemptis to search the memory if the key is stored in there in plain text (see next point). But if Windows itself is keeping the recovery key in memory, it is out of scope for the SafeGuard client to manage.

3.4.5 Memory

To check if there is any valuable information in the memory available, the attacker must get access to it. On Windows based systems, there are several possible ways to get access to the memory, or at least to parts of it. Examples are:

- Direct access
- Hibernation file
- Page file
- Crash dump
- Saved state of a virtual console
- And so on...

Some of these methods are only possible with elevated permissions (for example, administrator), but it can be possible to access the memory content, which is saved on the disk like the page or hibernation file, when the harddrive is connected to another computer.

Since the analysis of the memory is a complicated task, it is highly advised to use tools to make this much easier. One such tool is called Volatility Framework, which is an open source tool made by the Volatility Foundation[28]. Some of the developers of the Volatility Framework have published the book 'The Art of Memory Forensics'[29], which gives an detailed explanation of how to get information out of the memory with the usage of the Volatility Framework.

There are other special tools, which automatically search the memory for encryption keys. One such tool is 'Elcomsoft Forensic Disk Decryptor', which allows the forensic analysis of encrypted disks and volumes protected with BitLocker, PGP and TrueCrypt. It can automatically mount these volumes with the keys found during the analysis[30].

The search of encryption keys is by far not a trivial task, because it's like the search of a pin in a haystack. However, the attacker can use hints to search for the keys. These hints include the structures where the keys are stored in.

An example for such an attack is the BitLocker plugin for the Volatility Framework. This plugin uses metadata stored on the disk to get the Full Volume Encryption Key (FVEK) of a volume[31].

This tactic is nice to find keys, when it is known where to look for them. Unfortunately, this is not so easily possible for proprietary software, especially when there is no access to the source code. An option is to inspect the software with (kernel) debugger and see what data structures it uses. With this information, it can be possible to craft a Volatility plugin to scan for the encryption keys.

3.4.6 Named Pipes

Since named pipes can be accessed as files, it is quite simple to read the information, which is sent over them. But it is much more convenient to use tools for this purpose.

One such a tool is IO Ninja, which is supports different modules to read and write to a wide array of IO communication standards. Beside the 'Pipe Monitor' module, which allows the sniffing of a named pipe, and the 'Pipe Listener', which listens to incoming connections to a named pipe server, it supports several other communication protocols (for example: TCP, UDP, Serial)[32]

Another approach is the usage of WireShark. There is no native support for sniffing named pipes, but it is possible to make a workaround to support it. This is possible, because WireShark supports doesn't care what data is displayed, if it is delivered in a supported format. This is either the Libpcap file format or the pcap-ng format. To send the prepared data to WireShark, there are several ways, which are described in the CaptureSetup documentation[33].

3.4.7 RPC

Remote Procedure Calls (RPC) are a common type of communication between Windows programs. As there is information sent over RPC, it is interesting to sniff this communication and see if it is possible to get classified data out of there.

During the research about how to get information out of the RPC communication, these 2 tools were found to be the most promising:

- RPCSniffer, which allows WINDOWS RPC message sniffing in a given RPC server process[34].
- RpcView for exploring and decompiling all RPC functionalities present on a Microsoft system[35].

3.4.8 IOCTL

Getting information of IOCTL communication is no trivial task, but there is a discussion on StackOverflow on how to log DeviceIOControls on a Windows based system[36]. There are several possibilities listed, for example the usage of Tools like Dr. Memory, IRPTracker or API Monitor. While it is interesting to sniff the communication of the target software with these tools, they require administrative privileges, so this was out of scope for this thesis, because of the attacker profile in section 1.5.

3.5 How to add additional Content

Since this thesis consists of different parts, adding content can be divided into the following 2 main parts: 'Improving the Threat Model' and 'Finding further exploits'

Improving the Threat Model

Since there are only parts tested, the generated threat models are by far not complete. But fortunately, they can be used as a basis to add additional components and to compete the already included one.

The first rule is always to keep it simple and only add relevant information. This prevents the threat model from being overloaded with information.

The recommended development cycle is:

1. get info from the developers
2. add the info to the threat model
3. find potential exploits
4. report back to the developers
5. get information about the improved software (restart at point 1)

Finding further exploits

As already mentioned, the listed sample exploits are only a small part of a wide range of attacks. To find further vulnerabilities, it is needed to verify what technology the software uses at all. And by software, also the 3rd party code, which is used by the software, must be considered. For example, it seems to be useless to check for RPC problems, when there is no RPC communication in the software itself, but when 3rd party code relies on RPC communication, it must be checked as well. So, skipping parts for inspection can only be done if it is made sure that they are not included in the software and its dependencies at all.

At the best case, potential targets are already found during the information gathering or the threat modeling. But it is helpful to speak with the developers if they have any hints or recommendations for exploiting.

4 Case Study Sophos SafeGuard Enterprise 7 Windows Client

To verify how the approach is performing on a real life example, it is applied to the Sophos SafeGuard 7 Window client software. This thesis was written in cooperation with Sophos Linz, and the needed software packages for a working client and server environment was provided by them.

4.1 Sophos SafeGuard

SafeGuard Enterprise is a modular security suite that enforces security for endpoints on a cross platform basis, using administrator defined policies. The main protection functions of SafeGuard Enterprise on an endpoint are data encryption and protection against unauthorized access through external media. The System administration is carried out centrally in the SafeGuard Management Center[37].

There is a client support for the most common platforms (Windows, Mac, iOS, Android, and cloud based file sharing), but this thesis only handles the Windows based one.

The modules used by SafeGuard Enterprise are as follows:

- SafeGuard full disk encryption
  - SafeGuard Power-on Authentication
  - Volume based encryption
- BitLocker with pre boot authentication managed by SafeGuard Enterprise
- SafeGuard Data Exchange
- SafeGuard File Encryption
- SafeGuard Cloud Storage

Due to the complexity of the software, only small parts can be inspected about vulnerabilities. These parts to analyze were specified by Sophos Linz and are:

- Key handling
- BitLocker module

4.2 Threat Ranking

To get an overview what is considered as a threat, and their importance, Sophos Linz provided a list of threats and their corresponding ranking. Due to the classified information contained in this list, only the general description is provided in this thesis.

Ranking: Critical

- Access to credentials of other users or SGN
- Access to BitLocker PIN/Password
- Stealth modification of critical parts of the software
- Authentication bypass
- Privilege escalation

Ranking: High

- Access to credentials of current user
- Access to plain content of files encrypted with key not assigned to current user
- Modification of policies applied to the system

Ranking: Medium

- Denial of Service - Blue Screen

Ranking: n/a

- Denial of Service – Corruption of SGN settings

4.3 Creating the Test Environment

The first step of the testing process is to build a suitable testing environment. This was done with 2 physical computers, one was used for the server software, the second one was running a Windows system with VirtualBox 5.0.20 r106931 for the virtualization of the client.

The SafeGuard binaries, which were provided by Sophos, are:

During the analysis of the client, several changes in the operating system, the SafeGuard software and its configuration are needed, so it is recommended to use virtualization techniques. This enables the usage of different snapshots of the machine, which is comfortable when using different setups (standalone, managed) or when attacks put the system in a non recoverable state.

TABLE 4.1: SafeGuard installation binaries

Filename	Version	Size (Bytes)	SHA1
SGNServer.msi	7.00.0.109	19.423.232	54BB798C7CD371 E94AC3023E2169 382D1E9D8542
SGNManagement Center.msi	7.00.0.109	40.755.200	AB6572F64DA592 6974790A32428F 16D12437007C
SGxClientPreinstall.msi	7.00.0.109	9.003.008	8E39CEAAC8070E 12F4944838B71C FEB61049AF0B
SGNClient_x64.msi	7.00.0.109	125.177.856	C903E40EE911E0 5A99D7C3700EE3 309E77147B2E

There are 2 different basic configurations for the SafeGuard client (Managed and Standalone), so a Virtual Machine for each on these configurations was created.

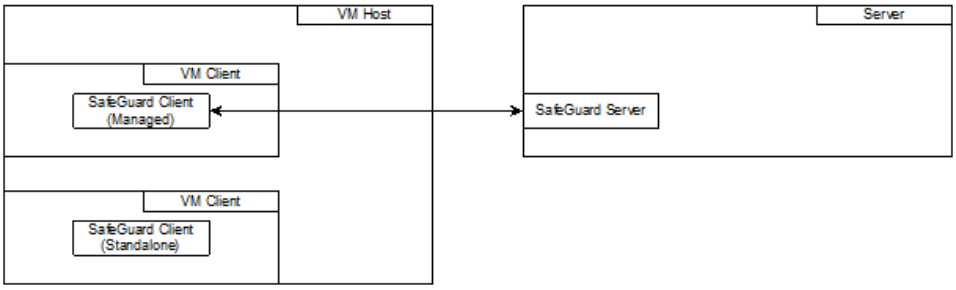


FIGURE 4.1: Test Environment

4.3.1 Client

The client itself runs on a virtual machine using VirtualBox 5.0.20 r106931 using the following settings:

TABLE 4.2: ClientVM

CPU	2 Cores
RAM	4 GB
HDD	45 GB (dynamically allocated)
Network	Intel PRO/1000 MT Desktop Bridge to host network interface (see below)
Enabled Hardware Features	IO-APIC
	PAE/NX
	VT-x/AMD-V
	Nested Paging
Operating System	Windows 10 Pro 64bit (1607 – 10.0.14393) (Without updates)

The host machine of VirtualBox has the following specifications:

TABLE 4.3: ClientHost

CPU	Intel Core i7-5820K (6 Cores, 12 Logical Processors)
RAM	16 GB
Network Interface (to Server)	Intel I218-V
Operating System	Windows 10 Pro 64bit (1607 – 10.0.14393)

To get the managed client of SafeGuard working, additional configurations must be made to the virtual machines:

- The DNS must be configured to correctly resolve the server name. This must be done, because the server runs in an isolated network and therefore has no official entry.
- The SSL certificate for the IIS server running the SafeGuard Enterprise server must be installed as a trusted root certification authority. This must be done by hand, because it was a self signed certificate and was not distributed with group policies.

4.3.2 Server

To use the managed client, a computer running Sophos SafeGuard Server is needed.

The specifications of the Server are:

TABLE 4.4: Server

CPU	Intel Xeon E3-1220v3
RAM	16 GB
Network Interface	Intel I210
Operating System	Windows Server 2012 R2 Standard
Installed Software	IIS Version 8.5.9600.16384
	MS SQL Server 2014 (64-bit)

The installation and configuration of the server was done using the 'SafeGuard Enterprise - Installation Best Practice' guide from Sophos[38].

4.4 Information Gathering

With the test environment set up, the next step is to get as many information of the SafeGuard client as possible. This is done by inspecting the installation files itself, and then by identifying what changes were made on the system by the client installation.

To determine what changes did occur during the installation of the software packages, the following basic procedure must be done:

1. Save System State
2. Install Software
3. Save new System State
4. Compare these 2 States

The question now is how to save the System State and what information is relevant to for investigation.

The most obvious information are the changes in the filesystem. But only these changes are not enough, because additional metadata is needed to check changes about drivers and services. This can be done with a comparison of the current registry with the old one.

A software to achieve this is 'Windows System State Analyzer', but unfortunately it was unstable and crashed unpredictable. So, the software 'System Explorer'[22] was used instead.

Since there are rather many changes done, only the relevant ones are listed:

- Root of the added directories
- Changes in the registry
- Installed services

However, it can be problematic that the naming scheme is not always consistent. Also, there are names still referring to 'Utimaco', which is confusing, but is necessary due to compatibility with older versions.

4.4.1 Installation Files

The most common way to install new software on a Windows based computer is with the use of executable or MSI (MicroSoft Installer) files. There is already the possibility to use specialized programs to install only trusted software (e.g. Windows Store), but unfortunately, this is not always possible or practical. So, the installation binaries, which are distributed by the software company or 3rd party sources still must be trusted.

It is possible to view the content of MSI files with the use of archiving software that support that format (e.g. 7zip). The problem here is now that the content of these files is not self explaining and can contain additional setup programs or archives itself. A solution is to check it with a malware scanner, but the problem remains, it is still not known, if the installer has been tampered with or not.

TABLE 4.5: Installer content

File	**Content**
SGxClientPreinstall.msi	Visual C Runtimes
SGNClient_x64.msi	Client Software (details see later)
Standalone Client (Standard).msi	Configuration Files
Managed Client (Standard).msi	Configuration Files Company Certificate

To verify that the software is the untampered one distributed by the software company, it can be checked, if it is signed with the right signature. This can be easily done in Windows by viewing the properties of the binary.

TABLE 4.6: Installer certificate

File	**Signed by**	**Algorithm**
SGxClientPreinstall.msi	Sophos Limited	sha1
SGNClient_x64.msi	Sophos Limited	sha1
Standalone Client (Standard).msi	n/a	n/a
Managed Client (Standard).msi	n/a	n/a

While the preinstall and the client itself are signed by Sophos Limited, the customer generated configuration files are not. This is by design, as those are dynamically generated on the client's installation, and to be signed by Sophos the product have to include the code signing certificate. This causes a problem in bigger environments, because someone can tamper with the installers on the distribution ways (for example include another

quiet installation of a backdoor). However, an Administrator can sign the configuration installers itself.

Also, the sha1 algorithm is proven to be problematic [39], so a change to sha2 or sha3 is recommended.

4.4.2 Dependencies

Since there are a rather large number of binaries in this software, it is interesting to see how they are connected to each other. A possibility to do this is to check the static linker information of them and generate a graph from this information.

This can be done by hand using a program like Dependency[40], which displays the dependencies in a tree view, but it is too time consuming to generate a graph with this information, because much work must be done manually. So instead, a small self developed tool was used to do this task to automatically generate a graph.

However, since there are over 200 binaries included in the software, the generated graph is complex and it is not be possible to include it in this document in readable form. But while it is too complicated to get an overview of the program using the graph it is still handy when searching for potential vulnerabilities using a graph viewer, because single connections between binaries can be filtered.

4.4.3 Directories

After inspecting the changes in the system, the following installation and working directories of the software were identified (see listing below). Additionally, it is now needed to find out who has access to these directories.

To display the discretionary access control lists (DACLs) on files or directories, the builtin Windows tool icacls.exe can be used. The basic usage for displaying these DACLs is:

```
icacls <FileName>
```

or

```
icacls <Directory>
```

To keep this list readable, the following users and groups, which are predefined by Windows, are left out intentionally:

```
NT SERVICE\TrustedInstaller
NT AUTHORITY\SYSTEM
BUILTIN\Administrators
APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES
APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APP PACKAGES
```

C:\Program Files\Sophos\SafeGuard Enterprise						
	Full Control	**Modify**	**Read & Execute**	**List folder contents**	**Read**	**Write**
Users			X	X	X	

C:\Program Files (x86)\Sophos\SafeGuard Enterprise						
	Full Control	**Modify**	**Read & Execute**	**List folder contents**	**Read**	**Write**
Users			X	X	X	

C:\Program Files (x86)\Sophos\SafeGuard Shared						
	Full Control	**Modify**	**Read & Execute**	**List folder contents**	**Read**	**Write**
Users			X	X	X	

C:\ProgramData\Utimaco\SafeGuard Enterprise						
	Full Control	**Modify**	**Read & Execute**	**List folder contents**	**Read**	**Write**
Users	X	X	X	X	X	X

C:\Users\All Users\Utimaco\SafeGuard Enterprise						
	Full Control	**Modify**	**Read & Execute**	**List folder contents**	**Read**	**Write**
Users	X	X	X	X	X	X

The full access rights to everyone to the last 2 directories listed is problematic on a shared computer system, because every user (even non SafeGuard ones) can mess with the data in there. Since these directories contain the cache, attackers can manipulate it to prevent SafeGuard from running. Also, the backup of the local cache is in the same directory with the same access rights.

A recommendation to prevent tampering with these files is the use of restrictive access rights and the backup in a save location, so that only SafeGuard itself can access it.

4.4.4 Files

To list all the installed files is not useful, but as already mentioned in the inspection of the installation files, the installers add the following files to the system:

- Visual C runtimes
- Client software
- Configuration files

The Visual C runtimes are out of scope, because they are not a part of the client itself. Also, the client software was installed in directories, which

had sufficient access rights set, so they are properly protected.

The interesting files for finding potential vulnerabilities because of weak permissions are the ones in the following directories:

```
C:\ProgramData\Utimaco\SafeGuard Enterprise
C:\Users\All Users\Utimaco\SafeGuard Enterprise
```

This is the LocalCache and its backup, which consists basically of the configuration and the actual status of the client software.

4.4.5 Registry

The relevant changes in the registry are:

- Set BitLockerPendingActions
- (Managed client only) Set Remote Management System
- Addition of the Keys listed below

TABLE 4.7: Registry changes

Type	Changes
Registry key added	HKLM\SOFTWARE\Policies\Utimaco
Registry key added	HKLM\SOFTWARE\Utimaco
Registry key added	HKLM\SOFTWARE\WOW6432Node\Policies\Utimaco
Registry key added	HKLM\SOFTWARE\WOW6432Node\Utimaco
Registry key added	HKLM\SYSTEM\ControlSet001\Control\Utimaco

The permissions to these keys are properly set, as only administrators and the SYSTEM has write access to it.

4.4.6 Services

Windows services are a crucial part of the operating system, so it is necessary to find out is who has access rights to the SafeGuard services and their dependencies.

There are two ways how the services of the client are started. The first one is when the service has its own executable, which is started. The second

way is by loading a DLL with the help of the SGN_MasterServicen.exe. The services, which are loaded by SGN_MasterServicen.exe are listed at the following registry location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Utimaco\SafeGuard
    Enterprise\MasterService\
```

The acquisition of the service dependencies is trivial, because it can be viewed in the properties of the service.

To get the access rights of the service, additional must be done. To view the access rights, the following command is used:

```
sc sdhow <service name>
```

This displays a service's security descriptor, but this is not easy to read, because it used a syntax called 'Security Descriptor Definition Language (SDDL)'. The detailed description of SDDL can be found online at the MSDN network[41].

All of the services returned the following SDDL string:

```
D:(A;;CCLCSWRPWPDTLOCRRC;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
    (A;;CCLCSWLOCRRC;;;IU)(A;;CCLCSWLOCRRC;;;SU)
```

A full explanation about this string is not given here, because it is too complex, but the important part of this string is that only the 'Local system' (SY) and the 'Built-in administrators' (BA) have the rights to start (RP), stop (WP) or pause/continue (DT) the service.

A more convenient way to get a list of users and their permissions is to use the accesschk.exe tool from the sysinternals suite. The usage to list the permissions of a service is:

```
accesschk.exe -ucqv <service-name>
```

With this knowledge, the following information was found about the services:

Service	**SafeGuard(R) MasterApplication WatchDog**
Description	Service monitors the SafeGuard MasterApplication
Executable	`C:\Windows\SysWOW64\SGN_MasterServicen.exe (C:\Windows\SysWOW64\SGMasterWatchDogN.dll)`

Permissions	`Medium Mandatory Level (Default) [No-Write-Up]`
	`RW NT AUTHORITY\SYSTEM`
	`SERVICE_ALL_ACCESS`
	`RW BUILTIN\Administrators`
	`SERVICE_ALL_ACCESS`
	`R NT AUTHORITY\INTERACTIVE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`
	`R NT AUTHORITY\SERVICE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`

Service	**SafeGuard(R) Authentication Service**
Description	SafeGuard Enterprise Authentication Service
Executable	`C:\Program Files (x86)\Sophos\SafeGuard Enterprise\Client\SGNAuthServicen.exe`
Dependencies	RPC

Permissions	`Medium Mandatory Level (Default) [No-Write-Up]`
	`RW NT AUTHORITY\SYSTEM`
	`SERVICE_ALL_ACCESS`
	`RW BUILTIN\Administrators`
	`SERVICE_ALL_ACCESS`
	`R NT AUTHORITY\INTERACTIVE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`
	`R NT AUTHORITY\SERVICE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`

Service	**Sophos Encryption For Cloud Storage**
Description	SGN Cloud Storage Encryption service
Executable	`C:\Windows\SysWOW64\SGN_MasterServicen.exe (C:\Windows\SysWOW64\SGNCloudEncServicen.dll)`

Permissions	`Medium Mandatory Level (Default) [No-Write-Up]`
	`RW NT AUTHORITY\SYSTEM`
	`SERVICE_ALL_ACCESS`
	`RW BUILTIN\Administrators`
	`SERVICE_ALL_ACCESS`
	`R NT AUTHORITY\INTERACTIVE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`
	`R NT AUTHORITY\SERVICE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`

Service	**SafeGuard(R) Filebased Encryption Machine Policies**
Description	Manage policies for filebased encryption that apply to all users on the machine.
Executable	`C:\Windows\SysWOW64\SGN_MasterServicen.exe (C:\Windows\SysWOW64\SGN_FEService.dll)`

Permissions	`Medium Mandatory Level (Default) [No-Write-Up]`
	`RW NT AUTHORITY\SYSTEM`
	`SERVICE_ALL_ACCESS`
	`RW BUILTIN\Administrators`
	`SERVICE_ALL_ACCESS`
	`R NT AUTHORITY\INTERACTIVE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`
	`R NT AUTHORITY\SERVICE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`

Service	SGNSafeModeService
Description	SafeGuard Enterprise SafeMode Authentication Service
Executable	C:\Program Files (x86)\Sophos\SafeGuard Enterprise\Client\SGNSafeModeServicen.exe
Permissions	Medium Mandatory Level (Default) [No-Write-Up] RW NT AUTHORITY\SYSTEM SERVICE_ALL_ACCESS RW BUILTIN\Administrators SERVICE_ALL_ACCESS R NT AUTHORITY\INTERACTIVE SERVICE_QUERY_STATUS SERVICE_QUERY_CONFIG SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS SERVICE_USER_DEFINED_CONTROL READ_CONTROL R NT AUTHORITY\SERVICE SERVICE_QUERY_STATUS SERVICE_QUERY_CONFIG SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS SERVICE_USER_DEFINED_CONTROL READ_CONTROL

Service	SafeGuard(R) Filebased Encryption Master
Description	Main service supporting filebased encryption.
Executable	C:\Windows\SysWOW64\SGN_MasterServicen.exe (C:\Windows\SysWOW64\SGNFileEncServicen.dll)
Permissions	Medium Mandatory Level (Default) [No-Write-Up] RW NT AUTHORITY\SYSTEM SERVICE_ALL_ACCESS RW BUILTIN\Administrators SERVICE_ALL_ACCESS R NT AUTHORITY\INTERACTIVE SERVICE_QUERY_STATUS SERVICE_QUERY_CONFIG SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS SERVICE_USER_DEFINED_CONTROL READ_CONTROL R NT AUTHORITY\SERVICE SERVICE_QUERY_STATUS SERVICE_QUERY_CONFIG SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS SERVICE_USER_DEFINED_CONTROL READ_CONTROL

Service	**SafeGuard(R) Log Service**
Description	SafeGuard(R) Log Service is a logging and reporting manager for various logging destinations
Executable	`C:\Windows\SysWOW64\SGN_MasterServicen.exe (C:\Windows\SysWOW64\SGM_LogPlayern.dll)`

Permissions	`Medium Mandatory Level (Default) [No-Write-Up]`
	`RW NT AUTHORITY\SYSTEM`
	`SERVICE_ALL_ACCESS`
	`RW BUILTIN\Administrators`
	`SERVICE_ALL_ACCESS`
	`R NT AUTHORITY\INTERACTIVE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`
	`R NT AUTHORITY\SERVICE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`

Service	**SafeGuard(R) System Event Manager**
Description	System Event Service
Executable	`C:\Windows\SysWOW64\SGN_MasterServicen.exe (C:\Windows\SysWOW64\SGN_Semn.dll)`

Permissions	`Medium Mandatory Level (Default) [No-Write-Up]`
	`RW NT AUTHORITY\SYSTEM`
	`SERVICE_ALL_ACCESS`
	`RW BUILTIN\Administrators`
	`SERVICE_ALL_ACCESS`
	`R NT AUTHORITY\INTERACTIVE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`
	`R NT AUTHORITY\SERVICE`
	`SERVICE_QUERY_STATUS`
	`SERVICE_QUERY_CONFIG`
	`SERVICE_INTERROGATE`
	`SERVICE_ENUMERATE_DEPENDENTS`
	`SERVICE_USER_DEFINED_CONTROL`
	`READ_CONTROL`

Service	SafeGuard(R) Transport Service
Description	SafeGuard(R) Transport Service for client server communication
Executable	C:\Windows\SysWOW64\SGN_MasterServicen.exe (C:\Windows\SysWOW64\SGTransCtrln.dll)
Dependencies	SafeGuard(R) System Event Manager
Permissions	Medium Mandatory Level (Default) [No-Write-Up] RW NT AUTHORITY\SYSTEM SERVICE_ALL_ACCESS RW BUILTIN\Administrators SERVICE_ALL_ACCESS R NT AUTHORITY\INTERACTIVE SERVICE_QUERY_STATUS SERVICE_QUERY_CONFIG SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS SERVICE_USER_DEFINED_CONTROL READ_CONTROL R NT AUTHORITY\SERVICE SERVICE_QUERY_STATUS SERVICE_QUERY_CONFIG SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS SERVICE_USER_DEFINED_CONTROL READ_CONTROL

Service	SafeGuard(R) WMI Listener
Description	Service monitor for BitLocker
Executable	C:\Windows\SysWOW64\WMIListener.exe
Dependencies	BitLocker Drive Encryption Service
Permissions	Medium Mandatory Level (Default) [No-Write-Up] RW NT AUTHORITY\SYSTEM SERVICE_ALL_ACCESS RW BUILTIN\Administrators SERVICE_ALL_ACCESS R NT AUTHORITY\INTERACTIVE SERVICE_QUERY_STATUS SERVICE_QUERY_CONFIG SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS SERVICE_USER_DEFINED_CONTROL READ_CONTROL R NT AUTHORITY\SERVICE SERVICE_QUERY_STATUS SERVICE_QUERY_CONFIG SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS SERVICE_USER_DEFINED_CONTROL READ_CONTROL

This shows, that the security permissions are proper set and that they cannot be exploited by the attacker, who does not have an administrator level account.

4.4.7 Security Flags

Another way to make an attack of a software easier is the lack of security enhancements. As mentioned before, there are several technologies available, and it is possible to check them by inspecting the binary file.

The selected security enhancements are:

- ASLR (Address Space Layout Randomization
- DEP (Data Execution Prevention
- SafeSEH (Safe Exception Handlers)
- Authenticode – the binary was signed with a digital signature

To check for these enhancements, a small console application using the dependency graph tool framework was written. This tool can verify the security enhancements for either a single binary or a list of binaries listed in a text file as input.

The list of all the binaries is too long to include here, but there were several of the security features missing from files.

TABLE 4.8: Security flags

	ASLR	**DEP**	**SafeSEH**	**Authenticode**
Missing	103	103	208	13

To help mitigate these problems it is recommended to use a software, which helps to verify that these security enhancements are included during the compilation of the software. One such tool is Microsoft BinScope 2014, which is available online without of charge[42].

BinScope Binary Analyzer helps to ensure that the binaries are built in compliance to the SDL requirements and recommendations. It identifies coding and building practices which result in potential vulnerabilities of the application.

4.4.8 Communication

Communication interfaces in software are a critical point, because a weak implementation can allow an attacker to interfere with the communication and it can be possible to read crucial information and even send manipulated data to break the software.

Most of the IPC communication in SafeGuard uses Names Pipes. To get a list of all the Names Pipes in Windows, a simple PowerShell command can be used:

```
get-childitem \\.\pipe\
```

The following table shows the Named Pipes used by the managed SafeGuard client during runtime. The instances show how many clients are

connected to the pipe currently, while the max. instances define if there is a limit used (-1 stands for infinite).

Pipe Name	Instances	Max. Instances
BRC_BAK_CHANGEDGdmKeyBackup	1	-1
BRC_BIN_CHANGEDBitLockerNativeWrapper	1	-1
BRC_BitLockerMAppEvtBitLockerNativeWrapper	2	-1
BRC_BitLockerMAppReqBitLockerMAppn	1	-1
BRC_CRL_CHANGEDSGNAuthService_CRL	1	-1
BRC_EVT_GDM_BLBitLockerNativeWrapper	1	-1
BRC_GdmKeyBackupBitLockerNativeWrapper	1	-1
BRC_GdmKeyBackupKeyBackupResultCollector	1	-1
BRC_INFOCOLLECTORUPDATETimestampCollector	1	-1
BRC_KMGLoaderClassSDXKMGLoader	1	-1
BRC_KMGLoaderClassSGNAuthService_KeyStore Reload	1	-1
BRC_KMGLoaderClassSGNCloudEncController	1	-1
BRC_KMGLoaderClassSGNCloudEncService	1	-1
BRC_KMGLoaderClassSGNFileEncController	1	-1
BRC_KMGLoaderClassSGNFileEncService	1	-1
BRC_KMGLoaderClassSGNFileShareController	1	-1
BRC_LSH_CHANGEDSystray_Status	1	-1
BRC_POLCHANGEBitLockerNativeWrapper	1	-1
BRC_POLCHANGEFEDec	1	-1
BRC_POLCHANGESGM_LogCtrl	1	-1
BRC_POLCHANGESGNAuthService_PolChange	1	-1
BRC_POLCHANGESGNCloudEncController	1	-1
BRC_POLCHANGESGNCloudEncService	1	-1
BRC_POLCHANGESGNFileEncController	1	-1
BRC_POLCHANGESGNFileEncService	1	-1
BRC_POLCHANGESGNFileShareController	1	-1

BRC_POLCHANGETrans	1	-1
BRC_SEMSGNCloudEncController	1	-1
BRC_SEMSGNCloudEncService	3	-1
BRC_SEMSGNFileEncController	1	-1
BRC_SEMSGNFileEncService	1	-1
BRC_SEMSGNFileShareController	1	-1
BRC_SYSPOLCHANGETrans	1	-1
BRC_TRANSSNDPRIOTRSSRV	3	-1
BRC_TRANSSNDREQSGM_LogCtrl	1	-1
BRC_TRANSSTATUSFEDec	2	-1
BRC_TRANSSTATUSSGNCloudEncController	2	-1
BRC_TRANSSTATUSSGNCloudEncService	1	-1
BRC_TRANSSTATUSSGNFileEncController	2	-1
BRC_TRANSSTATUSSGNFileEncService	2	-1
BRC_TRANSSTATUSSGNFileShareController	2	-1
BRC_TRANSSTATUSTransStatusCollector	2	-1
BRC_UMA_CHANGEDBitLockerNativeWrapper	1	-1
BRC_UMA_CHANGEDSGNAuthService_LockDown	1	-1
BRC_USER_KEYSTORE_ESTABLISHEDBitLocker NativeWrapper	2	-1
BRC_USER_KEYSTORE_ESTABLISHEDSDXAUTHSRV	1	-1
BRC_USER_KEYSTORE_ESTABLISHEDSGNCloudEnc Controller	1	-1
BRC_USER_KEYSTORE_ESTABLISHEDSGNCloudEnc Service	1	-1
BRC_USER_KEYSTORE_ESTABLISHEDSGNFileEnc Controller	1	-1
BRC_USER_KEYSTORE_ESTABLISHEDSGNFileEnc Service	1	-1
BRC_USER_KEYSTORE_ESTABLISHEDSGNFileShare Controller	1	-1
SGM_LogMessages_NamedPipe	5	-1

TABLE 4.9: Named pipes

None of these pipes has a restriction to maximize the number of connections. It is not always be possible to set a hard limit of the connections, but is advised to limit them to prevent unwanted connections.

Another problem is that the names of the pipes are predetermined. This makes it easy for an attacker to connect to a specific named pipe. Since the communication over the named pipes in encrypted, this attack scenario was not performed.

The predetermined name allows attackers to use race conditions to spoof these pipes. But as already mentioned, this was not tested, because of the encrypted communication over the named pipes.

4.5 Threat Modeling

The modeling was done with a top down approach, beginning on a basic schematic of the software (Context), then selecting parts of the diagram and model them in detail. These detailed diagrams are called 'Levels', followed by a number, describing the detail level of it.

The first Level is obviously the SafeGuard software itself and shows the running processes and their communication parts to the other used components of the system.

Since the software is too large to analyze every part of it, an agreement with Sophos Linz was made to only include parts to focus on. The following parts were chosen:

- BitLocker
- Key Handling

This means, that the diagrams are only modeled as far as information was available. So, many parts of the software are not included here, or are only mentioned as out of scope. To help to include these missing parts in future works a tutorial is included in this thesis.

A challenge was encountered when modeling the different diagrams of SafeGuard. These diagrams can grow quick in size, especially when there are more than a handful of components includes. It can be possible to handle them directly in the program on the computer, but it is difficult to explain them to other people or to show them at presentations. It is problematic to include them in this document, because larger diagrams can span over several pages.

To mitigate these problems, the following simplifications were made:

- Data flow connections are displayed as a single line using a generic description, even if there are more connections between the components. The detailed description of the different connections is made in higher level diagrams (if possible).

- Larger diagrams are split into several smaller parts, where only the relevant objects are included.

The basic layout of the diagram descriptions are as follows:

- Explanation, what the diagram is about
- The diagram itself
- Description of the relevant objects (processes, communication lines, ...) including potential vulnerabilities

The MS SDL Tool uses properties to describe the different objects. But since it is not easily possible to include the properties of the different objects in the diagram itself, the separate explanation had to be made.

It must be noted, that the properties of the threat models itself are by far not complete. The reason of this shortcoming is that only the developers have enough insight in the software to complete the properties of the different objects. This means, that the generation of the potential attacks in the MS SDL Tool generates too much entries, so to use the tool properly, the developers have to add the missing properties. For this thesis, it is not useful to include all the possible attacks generated, so only a few possibilities were chosen. These attacks are explained in the diagram descriptions, and are chosen to be executable using the limited resources and information, which are available for this thesis. So, attacks, which require deeper understanding of the code (either with the actual source code or with the usage of binary analysis) cannot be handled here.

To keep a better overview of the different components and to keep the naming and numbering consistent, it is recommended to keep a list of them

(1) Human User
(2) Machine Trust Boundary
(3) SGN Client
    (3.1) AuthApp
    (3.2) Tray
    (3.3) CredProv
    (3.4) AuthService
    (3.5) MasterService
        (3.5.1) SafeGuard(R) Filebased Encryption Machine Policies
        (3.5.2) SafeGuard(R) MasterApplication WatchDog
        (3.5.3) SafeGuard(R) Transport Service
        (3.5.4) Sophos Encryption For Cloud Storage
        (3.5.5) SafeGuard(R) Filebased Encryption Master
        (3.5.6) SafeGuard(R) Log Service
        (3.5.7) SafeGuard(R) System Event Manager
    (3.6) KeyStoreDrv
    (3.7) FileFilterDrv
    (3.8) File System
    (3.9) Cache

(3.10) Current User
(3.11) SYSTEM
(3.12) SGNMaster (SGNMaster.exe)
(3.13) OS Process
(3.14) WMIListener
(3.14.1) BLDFlag Thread
(3.14.2) Bitlocker Change Thread
(3.14.3) Event Threads
(3.14.4) Bitlocker RPC Service
(3.15) Memory (Properties)
(3.16) Volume Change Thread
(3.17) Registry Hive
(4) Internet Boundary
(5) SGN Server

4.5.1 Context

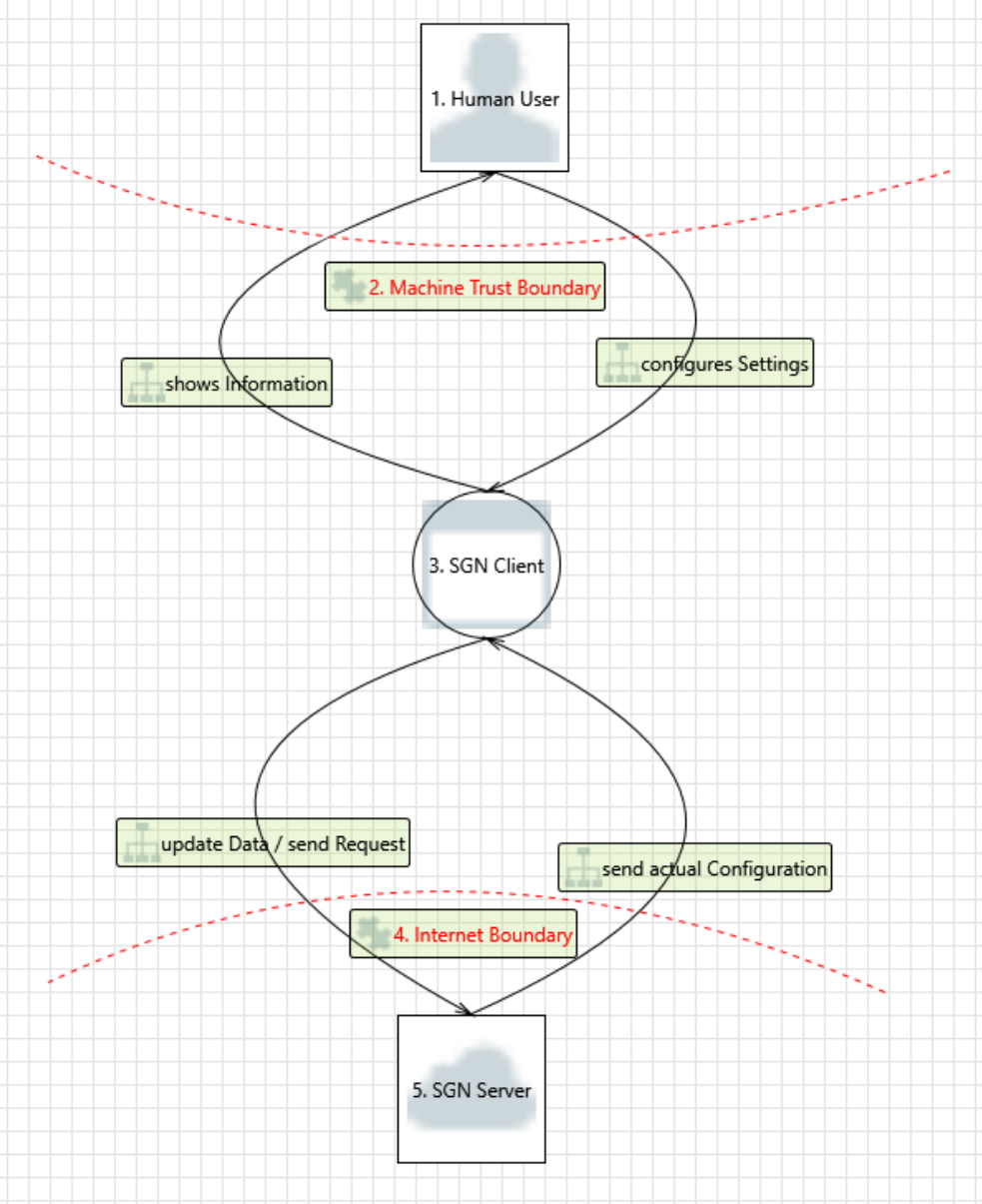


FIGURE 4.2: Threat Model Context

The context diagram shows the basic communication flow of the software. Since this thesis only handles the windows client, the server part is included as just one block.

The user only has basic interaction with the software. This include the input of credentials or the request of the generation of new keys. It can be requested, that the client displays information about the software (last update from server, and so on...)

The communication between the server and the client is encrypted and will not be further analysed, because the network communication is out of

scope in this thesis.

Since this diagram is too high level, there are no actual problems to discuss.

4.5.2 Level 1 – Key Handling

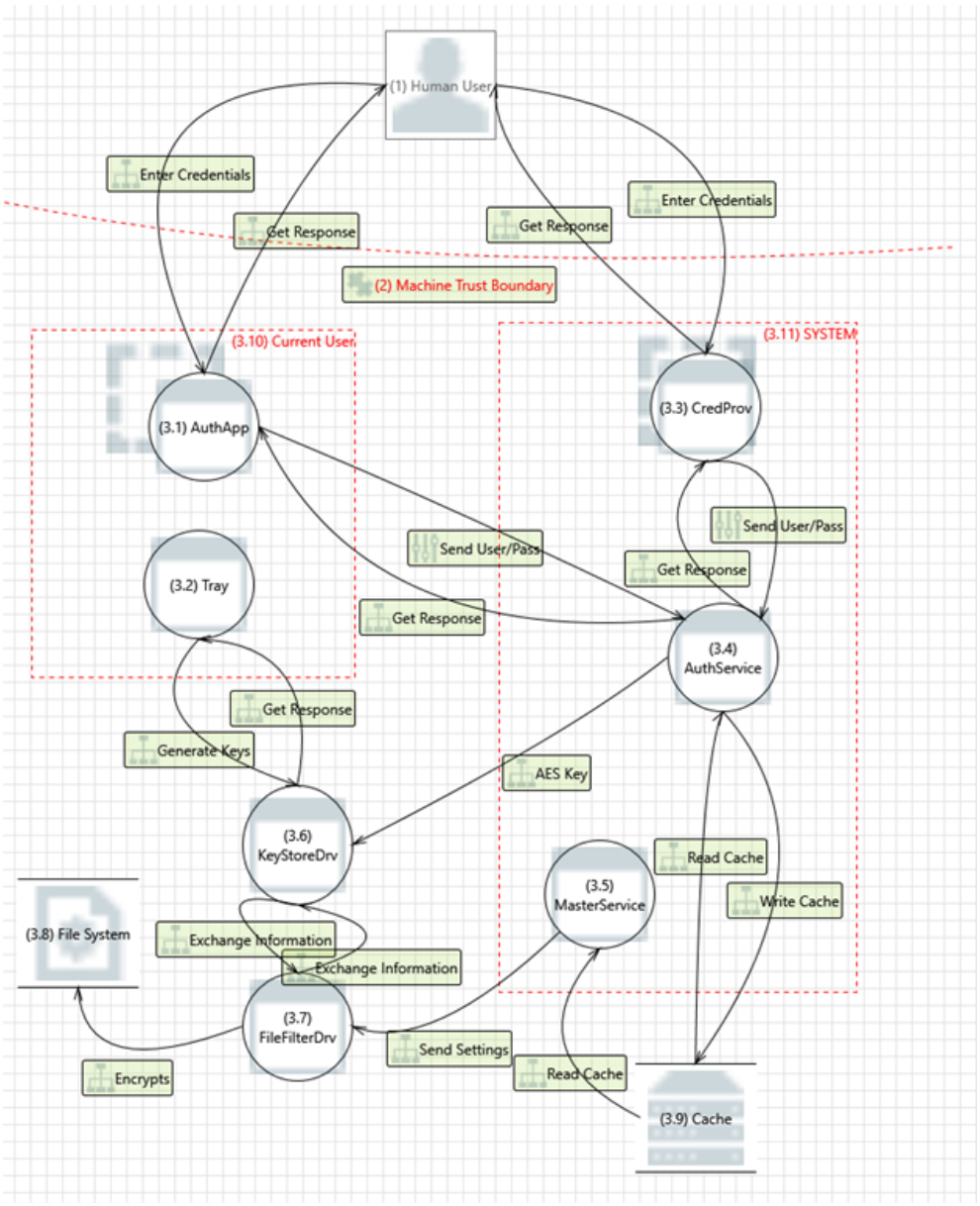


FIGURE 4.3: Threat Model Key Handling

This diagram is a describes a crucial part of the client software. It is about the keys, which are necessary to encrypt or decrypt the files or drives of the system. A break of confidentiality of these keys is highly fatal to the whole system (see the threat ranking, which was provided by Sophos). So, it must be made sure that it is never possible for the actual user, and especially for other users to get these keys.

Name	**Context**
(1) User	-
Communicates with	**Protocol used**
(3.3) Credential Provider	-
(3.1) Authentication App	-
Description	
The user must input its credentials, which are later used by the Authentication Service (3.4). This is possible during the following 2 situations: • During the logon screen of Windows, using the custom Credential Provider (3.3). • With the Authentication App (3.1), this only happens, when the entered user credential for the system does not match the credentials of the user from SafeGuard.	
Potential Vulnerabilities	
The security problems concerning the user itself are not part of this thesis, because the user is not part of the SafeGuard client.	

Name	**Context**
(3.1) AuthApp (SGNAuthAppn.exe)	Current User
Communicates with	**Protocol used**
(3.4) AuthService	RPC (SPT)
Description	
The Authentication App (AuthApp) requests the user (1) to enter its SafeGuard username and password, when the entered credentials during the logon doesn't match the one from SafeGuard.	
Potential Vulnerabilities	
The Authentication App (AuthApp) runs in the user context and requests the SafeGuard credentials of the user. It can be possible for malware to intercept or hijack the communication between the AuthApp and the AuthService (3.4). A Keylogger can be used to get the credentials, but this is out of scope to include here, because this is not SafeGuard specific and it means that the system is already compromised. Also, per the developers, the RPC connection is authenticated and encrypted. So, the sniffer have to decrypt the message.	

Name	**Context**
(3.2) TrayApp	Current User
Communicates with	**Protocol used**
(3.6) KeyStoreDrv	IOCTL
Description	
It is possible for the user (1) to generate new encryption keys using this application.	
Potential Vulnerabilities	
The tray application is not well suited for an attack, because it only allows low risk operations such as the display of information, server sync or the generation of new keys.	

Name	**Context**
(3.3) CredProv (SGNCredProvn.dll)	SYSTEM
Communicates with	**Protocol used**
(3.4) AuthService	RPC (SPT)
Description	
This is a custom credential provider for the windows logon. This allows the SafeGuard client to unlock the user certificate with the same credentials as the windows user account if they match. Otherwise the user is prompted to enter their SafeGuard username and password using the AuthApp (3.1).	
Potential Vulnerabilities	
The Credential Provider (CredProv) is only used during the logon screen, so it can not be easily manipulated by the user itself. However, when the attacker gains the possibility to change the binary (or its dependencies) the credentials can be logged.	

Name	**Context**
(3.4) AuthService (SGNAuthServicen.exe)	SYSTEM
Communicates with	**Protocol used**
(3.9) Cache	File IO
(3.6) KeyStoreDrv	IOCTL

Description

The Authentication Service (AuthService) retrieves the encrypted certificate of the user from the cache (3.9) and tries to decrypt it with the provided credentials.
If the decryption succeeds, the AuthService send the AES Keys, which are in the user certificate to the KeyStore driver (3.6) using IOCTL operations. When it fails, the provided credentials are invalid and the user gets the corresponding feedback.

Potential Vulnerabilities

The AuthService handles classified data (keyring), so the control over it can provide useful information. But since it runs in the SYSTEM context, the access to it from a basic user account is not trivial. The IOCTL calls to the KeyStore driver (3.6) send interesting data, but sniffing them is out of scope, since it requires administrator rights. DLL hijacking is an option, but with properly set access rights this is not possible without administrator rights. The AuthServicen.exe depends directly or indirectly on the following binaries (listed without the ones from Microsoft):

```
C:\Program Files (x86)\Sophos\SafeGuard Enterprise\Client\
    SGNAuthServicen.exe C:\Windows\system32\SGMCachesn.dll
C:\Windows\system32\SGMBASEN.dll
C:\Windows\system32\SGMXMLN.dll
C:\Windows\system32\SGMSBASEN.dll
C:\Windows\system32\SGLocalCacheEnginen.dll
C:\Windows\system32\SGMEntitiesn.dll
C:\Windows\system32\SGEvtMann.dll
C:\Windows\system32\SGMSecUtiN.dll
C:\Windows\system32\SGMCBIN.dll
C:\Windows\system32\sgmtdfN.dll
C:\Windows\system32\SGM_LogStatn.dll
C:\Windows\system32\SGMPipeConnectionn.dll
C:\Windows\system32\SgmGdmn.dll
C:\Windows\system32\SGTransCtrln.dll
C:\Windows\system32\SGMWebClientn.dll
C:\Windows\system32\SGM_ReportAPIn.dll
C:\Windows\system32\KMGLoadern.dll
C:\Program Files (x86)\Sophos\SafeGuard Enterprise\Client\
    SPTBASEN.dll
C:\Windows\system32\sgmpwfn.dll
```

Name	**Context**
(3.5) MasterService (SGN_MasterServicen.exe)	SYSTEM
Communicates with	**Protocol used**
(3.7) FileFilterDrv	IOCTL
(3.9) Cache	FileIO
Description	
The MasterService reads the encryption settings from the cache (3.9) and sends them to the FileFilter driver (3.7) using IOCTL.	
Potential Vulnerabilities	
The Master Service includes many different modules, but the relevant for the key handling is the handover of the settings to the FileFilter driver (3.7). Like the AuthService (3.4), the MasterService runs in SYSTEM context and communicates with the driver using IOCTL. Also, the cache is accessed the same way as in the AuthService (3.4), so the attack possibilities are the same as listed above.	

Name	**Context**
(3.6) KeyStoreDrv	Driver (Kernel)
Communicates with	**Protocol used**
(3.7) FileFilterDrv	IOCTL
Description	
This is where the actual encryption keys are stored. The FileFilterDrv (3.7) uses this driver to encrypt the files (3.8)	
Potential Vulnerabilities	
The encryption keys are the most confidential part of the SafeGuard client. To get access to the keys, it can be tried to access the memory itself and try to find them. However, this is complicated because the lack of access rights and the amount of data to analyze. It can be possible to get access to the communication between the KeyStore driver and the FileFilter driver (3.7)	

Name	**Context**
(3.7) FileFilterDrv	Driver (Kernel)
Communicates with	**Protocol used**
(3.6) FileFilterDrv	IOCTL
(3.8) File System	FileIO
Description	
The FileFilter driver (3.7) uses the KeyStore (3.6) driver to encrypt the files (3.8) based on the rules in the settings. There are no keys stored in this driver itself.	
Potential Vulnerabilities	
A potential point of attack is the communication between this and the KeyStore driver (3.6). Another possibility is the manipulation of the setting, so that nothing gets encrypted at all.	

Name	**Context**
(3.8) File System	-
Communicates with	**Protocol used**
-	-
Description	
The file system of the computer	
Potential Vulnerabilities	
The file system itself is out of scope, because it is not a part of SafeGuard itself.	

Name	**Context**
(3.9) Cache	-
Communicates with	**Protocol used**
-	-
Description	
The cache is where the current state of the SafeGuard client is stored. It includes the settings, certificates, connection details, and so on.	
Potential Vulnerabilities	
The cache is one of the most interesting parts of the key handling, because of the information stored there. The access rights to the Cache are set to full access for everyone, so even a non SafeGuard user can manipulate it. A rather radical option is the corruption or deletion of the Cache. This puts the SafeGuard client in a non operational state, but since a LocalCache corruption is explicitly flagged as not critical in the threat ranking (it is an intended behavior), this is out of scope. A promising attack is to try to get as much information out of the cache as possible and to manipulate the cache, so that no corruption occurs (see section 4.6).	

4.5.3 Level 1 - WMIListener (BitLocker)

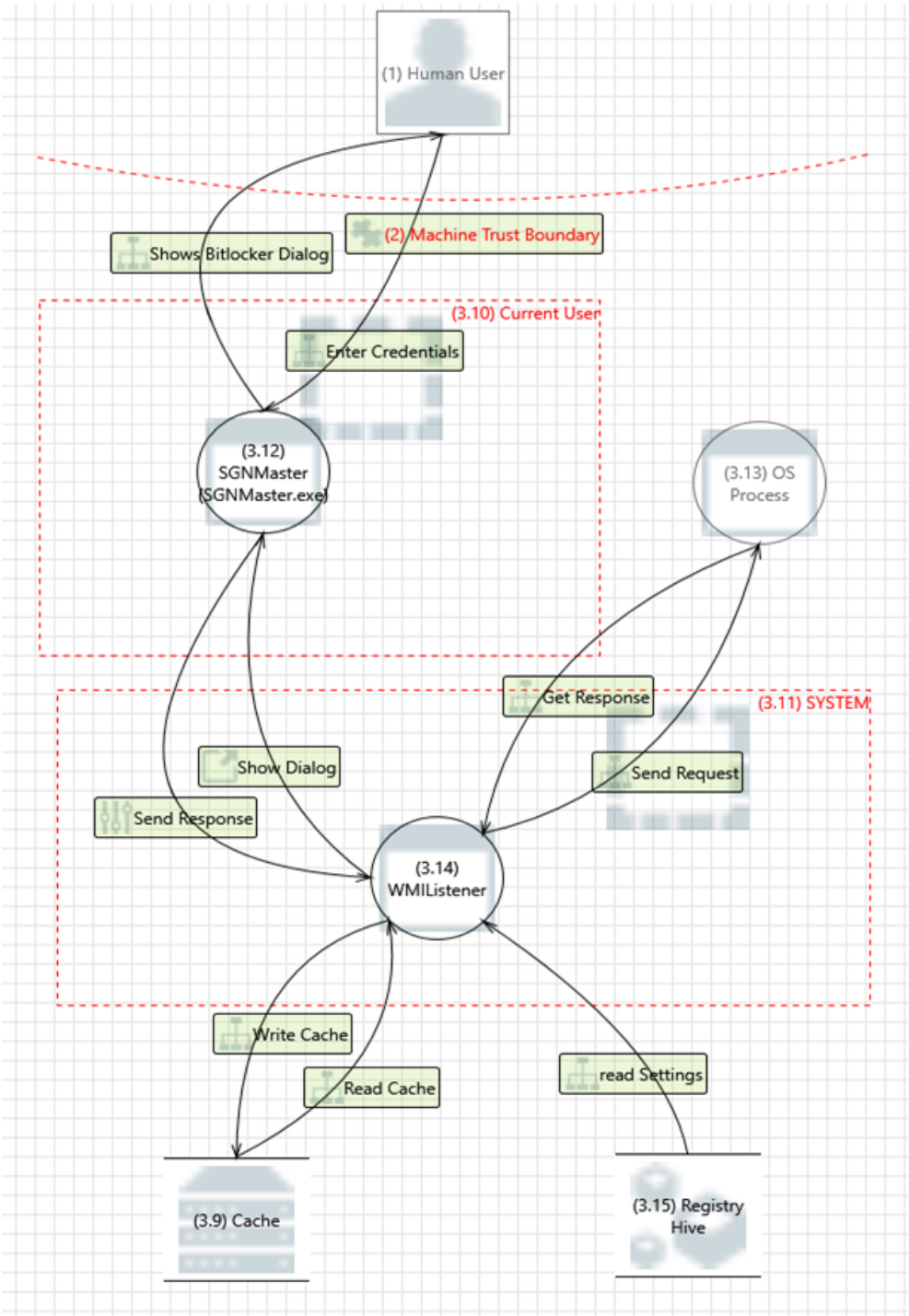


FIGURE 4.4: Threat Model WMIListener

The WMIListener consists of several different parts, so it is too complex to draw them all into one diagram. So, in this part, only the basic functionality is explained and a detailed description of the different objects is made later in the discussion of the subparts of the WMIListener.

The name WMIListener is a little confusing, because this module is not only listening to WMI events, but also for other different events. This includes:

- BitLocker changes
- BLD (BitLocker Dialog) Flag changes
- Volume changes
- Other events

The main component of the WMIListener is a thread, which listens to those events. When such an event occurs, the WMIListener starts a separate event thread to handle the necessary actions. The main thread itself is waiting to the started event thread to finish, so this is a blocking action. This can cause problems, because it is not guaranteed that other events, which are arriving meanwhile, are handled properly.

BitLocker encrypts the whole volume, so the loss of the password is critical. To still have the possibility to access the drive when the password is lost, a recovery password can be crated. Since SafeGuard manages the encryption of the system, it handles the recovery password as well, so that the user has no direct access to it and can therefore create a lack of confidentiality. The BitLocker recovery password is either stored on the server (managed client) or on the client itself (standalone client)

There is broadcast IPC communication used to signal changes in the SafeGuard system (policy change, ...). This communication is using names pipes and can have payload appended. The names of these named pipes give a good hint about what they are used for (see the discussion under 'Named Pipes'). The communication over them is encrypted, but it can still be possible to get information from it.

Name	**Context**
(1) User	-
Communicates with	**Protocol used**
-	-
Description	
When a BitLocker dialog is displayed, the user must provide the needed feedback to it. This can be for example, the PIN or a passphrase.	
Potential Vulnerabilities	
The user is out of scope of the SafeGuard client itself, so this is not discussed here.	

Name	**Context**
(3.12) SGNMaster (SGNMaster.exe)	Current user
Communicates with	**Protocol used**
(1) User	-
(3.14) WMIListener	RPC
Description	
Displays the BitLocker Dialog.	
Potential Vulnerabilities	
The response from the user can be sniffed, but this communication is protected by encryption methods.	

Name	**Context**
(3.13) OS Process	-
Communicates with	**Protocol used**
-	-
Description	
The BitLocker handling is done by the operating system itself, and the SafeGuard client uses WMI to communicate with it.	
Potential Vulnerabilities	
Since this is part of the operating system, it is out of scope of the SafeGuard client.	

Name	**Context**
(3.14) WMIListener	SYSTEM
Communicates with	**Protocol used**
(3) SafeGuard Client (several modules)	Named Pipe
(3.9) Cache	FileIO
(3.13) OS Process	WMI
(3.15) Registry	-
Description	
See description above. The changes, which are signaled by the events and the general settings itself are either stored in the cache (3.9) or in the registry (3.15).	
Potential Vulnerabilities	
As mentioned in the key handling, the cache (3.9) can be accessed by every user on the client system. The registry (3.15) contains no classified information, but it can be still possible to manipulate it to generate unintended behavior. Since there are several modules which send broadcast messages using named pipes, it is possible to get information out of it. The names of these pipes are well known and their function is kind of self explained because of it. So, the communication can be manipulated to stop the SafeGuard client to function properly. But since the communication was encrypted and have error handling, this was not possible. The communication to the BitLocker system in the operating system is done with WMI calls, so when there are improper access permissions set the encryption of the system volumes can be prevented.	

Name	**Context**
(3.9) Cache	-
Communicates with	**Protocol used**
-	-
Description	
The cache has already been explained in the key handling discussion.	
Potential Vulnerabilities	
The same as discussed in the key handling.	

Name	**Context**
(3.15) Registry Hive	-
Communicates with	**Protocol used**
-	-
Description	
Stores information about the BitLocker encryption used by the WMIListener (3.14).	
Potential Vulnerabilities	
The BitLocker settings in the registry is part of the operating system and not of the SafeGuard client, so this is out of scope for the client to handle.	

The following objects are not further included in the analysis of the sub parts below to reduce redundancy. This affects the following objects:

- (3.9) Cache
- (3.13) OS Process
- (3.15) Registry

4.5.4 Level 2 - WMIListener BitLocker Change Thread

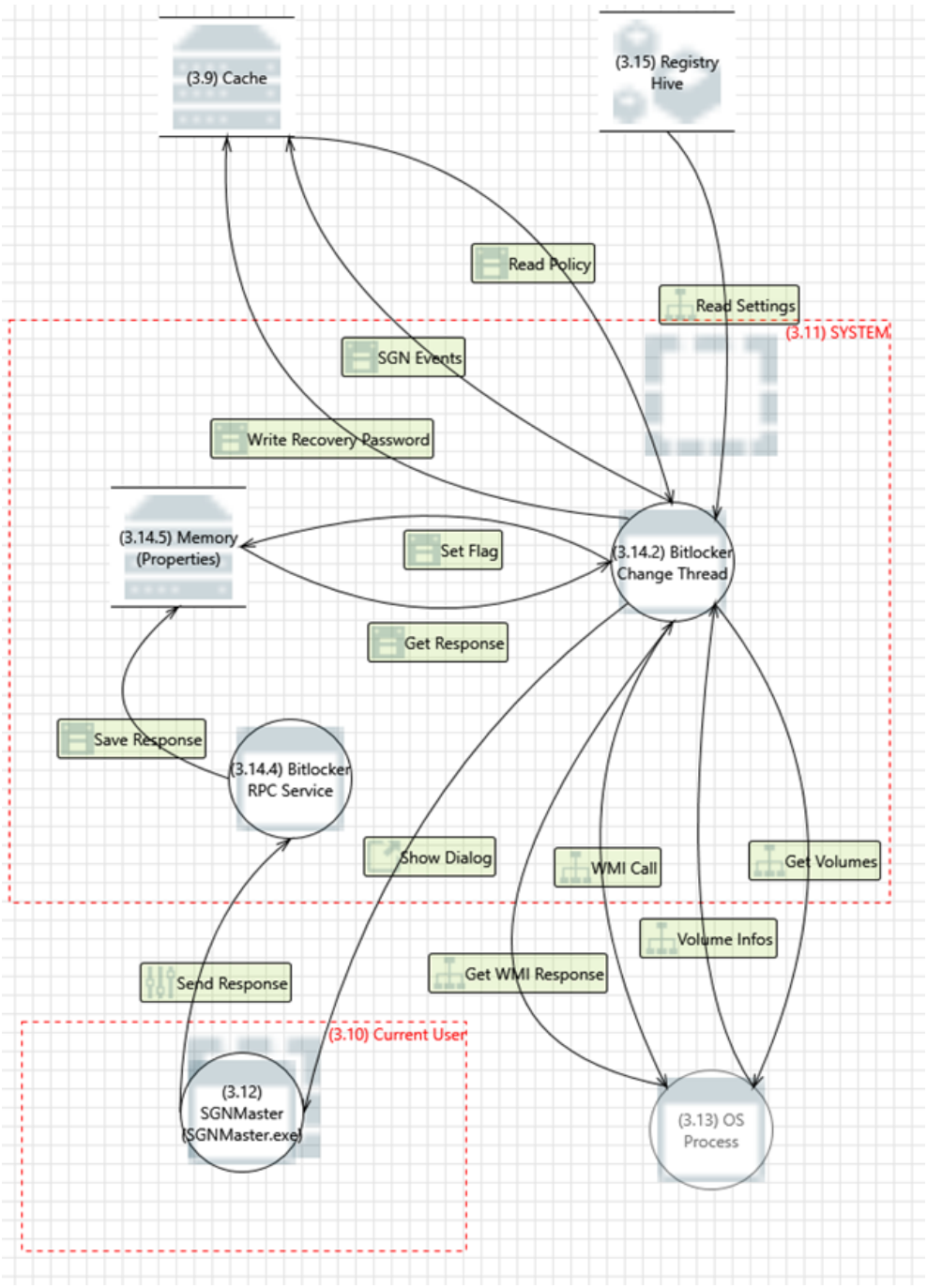


FIGURE 4.5: Threat Model WMIListener BitLocker Change Thread

This thread is started when there where changes in the BitLocker system are needed to be done.

The BitLocker components of the operating system are accessed with the usage of WMI and the needed information about it and the actual policies are received from the registry and the cache respectively.

The SGNMaster (3.12) is used for the SafeGuard specific BitLocker communication with the user. This is invoked with a named pipe, and when the result is ready, it is sent back per RPC.

Name	**Context**
(3.14.2) BitLocker Change Thread	SYSTEM
Communicates with	**Protocol used**
(3.9) Cache	FileIO
(3.12) SGNMaster	Named Pipe
(3.13) OS Process	API Calls
(3.13) OS Process	WMI
(3.14.5) Memory (Properties)	-
(3.15) Registry	API Calls
Description	
This is the core of the subpart of the WMIListener as described above.	
Potential Vulnerabilities	
Potential attacks regarding the Cache (3.9), SGNMaster (3.12), OS Process (3.13) and the Registry (3.15) are already discussed in the Level 1 diagram, so the only additional part here is the communication the memory (3.14.5). With proper access rights it is possible to access the memory to tamper with the flags and stored information to provoke unintended behavior. It can be tried to find the recovery password in there, but with proper protection this is extremely difficult, if not impossible to fulfill.	

Name	**Context**
(3.14.4) BitLocker RPC Service	SYSTEM
Communicates with	**Protocol used**
(3.14.5) Memory (Properties)	-
Description	
This service receives the result of the user communication from the SGNMaster (3.12). This is stored directly in the memory of the process.	
Potential Vulnerabilities	
As mentioned in the analysis of the BitLocker Change Thread (3.14.2), it can be tried to get access to the memory to tamper with the data.	

Name	**Context**
(3.14.5) Memory (Properties)	SYSTEM
Communicates with	**Protocol used**
-	-
Description	
This is the actual memory of the process itself.	
Potential Vulnerabilities	
The process/binary can be lacking security enhancements, so that the attacker can have the possibility to access the information stored in the memory.	

Name	**Context**
(3.12) SGNMaster	Current user
Communicates with	**Protocol used**
(3.14.4) BitLocker RPC Service	Named Pipe
Description	
The SGNMaster sends the user response to the BitLocker RPC Service (3.14.4) over a named pipe back to the WMIListener (3.14), where it gets stored in memory.	
Potential Vulnerabilities	
Since the response is sent over a named pipe, it can be read when it is not protected properly, for example with encryption.	

4.5.5 Level 2 - WMIListener BLD Flag Thread

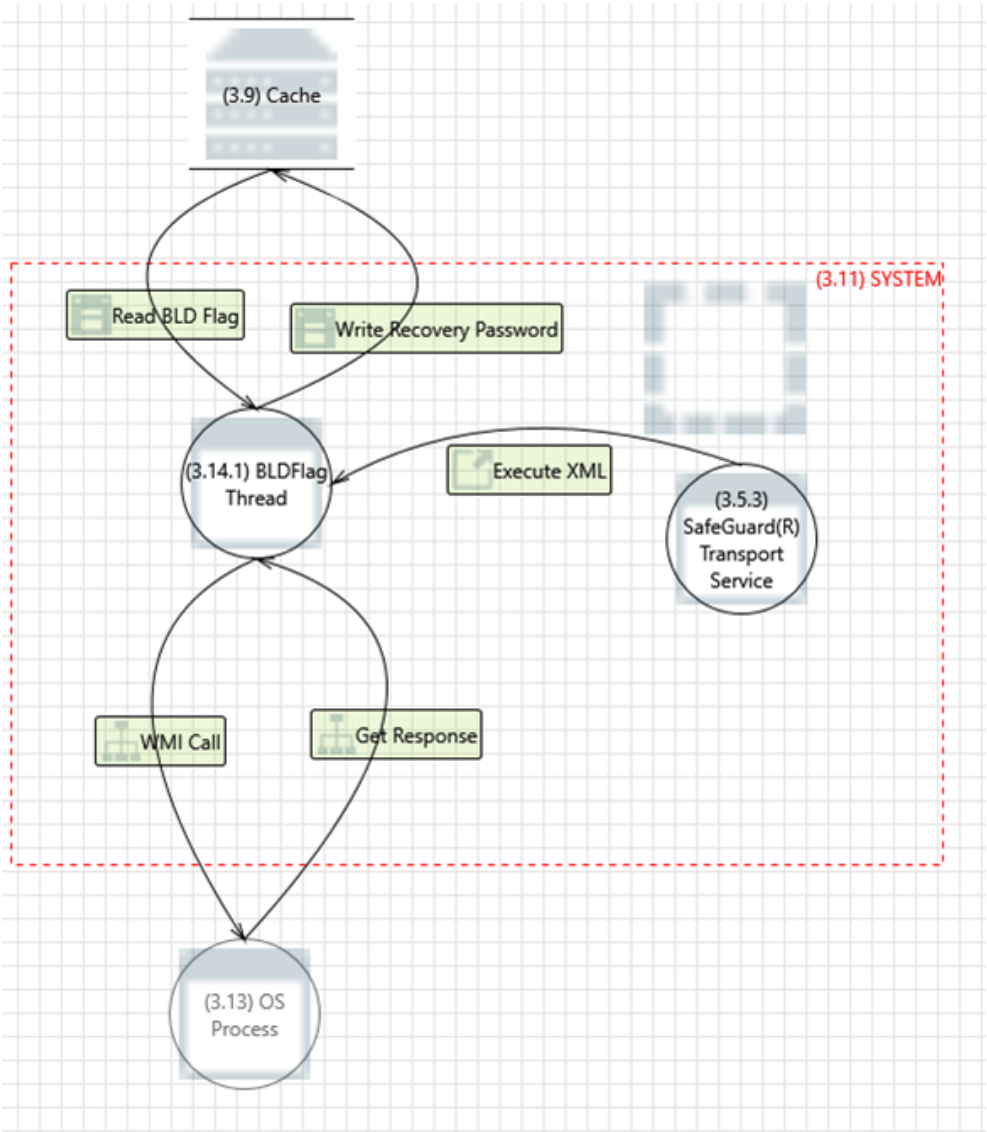


FIGURE 4.6: Threat Model WMIListener BLDFlag Thread

The BLDFlag Thread can receive configuration data for the BitLocker (XML data) from the SafeGuard Transport Service (3.5.3) over a named pipe. These settings are applied via WMI calls to the operating system.

It is possible that the recovery password is saved into the cache.

Name	**Context**
(3.14.1) BLDFlag Thread	SYSTEM
Communicates with	**Protocol used**
(3.9) Cache	FileIO
(3.13) OS Process	WMI
Description	
See description above.	
Potential Vulnerabilities	
Potential attacks regarding the Cache (3.9) and the OS Process (3.13) are already discussed in the Level 1 diagram.	

Name	**Context**
(3.5.3) SafeGuard(R) Transport Service	SYSTEM
Communicates with	**Protocol used**
(3.14.1) BLDFlag Thread	Named Pipe
Description	
The Transport Service is responsible for the communication to the SafeGuard server. This communication is obviously only available on a managed client, as the standalone one has none.	
Potential Vulnerabilities	
It can be possible to tamper with the communication to the BLDFlag thread (3.14.1) to manipulate the new settings.	

4.5.6 Level 2 - WMIListener Volume Change Thread

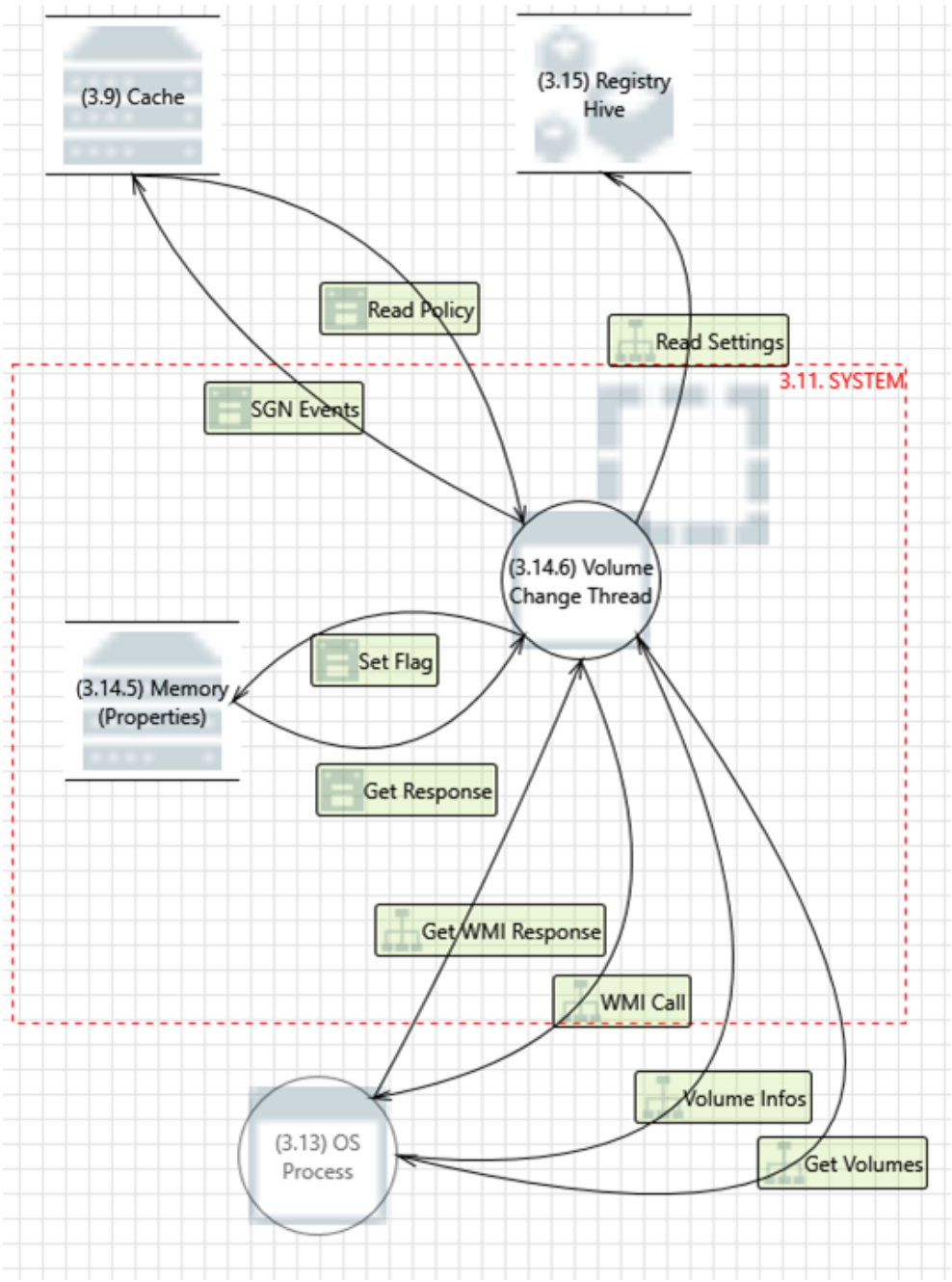


FIGURE 4.7: Threat Model WMIListener Volume Change Thread

This event thread is started, when there was a change regarding the system volumes. This can require the client to adjust the BitLocker settings and therefore needs a response to the SafeGuard state in the cache (3.9).

Name	**Context**
(3.14.6) Volume Change Thread	SYSTEM
Communicates with	**Protocol used**
(3.9) Cache	FileIO
(3.13) OS Process	API Calls
(3.13) OS Process	WMI
(3.14.5) Memory (Properties)	-
(3.15) Registry	API Calls
Description	
See description above.	
Potential Vulnerabilities	
Potential attacks regarding the Cache (3.9), OS Process (3.13) and the Registry (3.15) are already discussed in the Level 1 diagram and the attacks targeting the memory are mentioned in the Level 2 – WMIListener BitLocker Change Thread diagram.	

4.5.7 Level 2 - WMIListener Event Threads

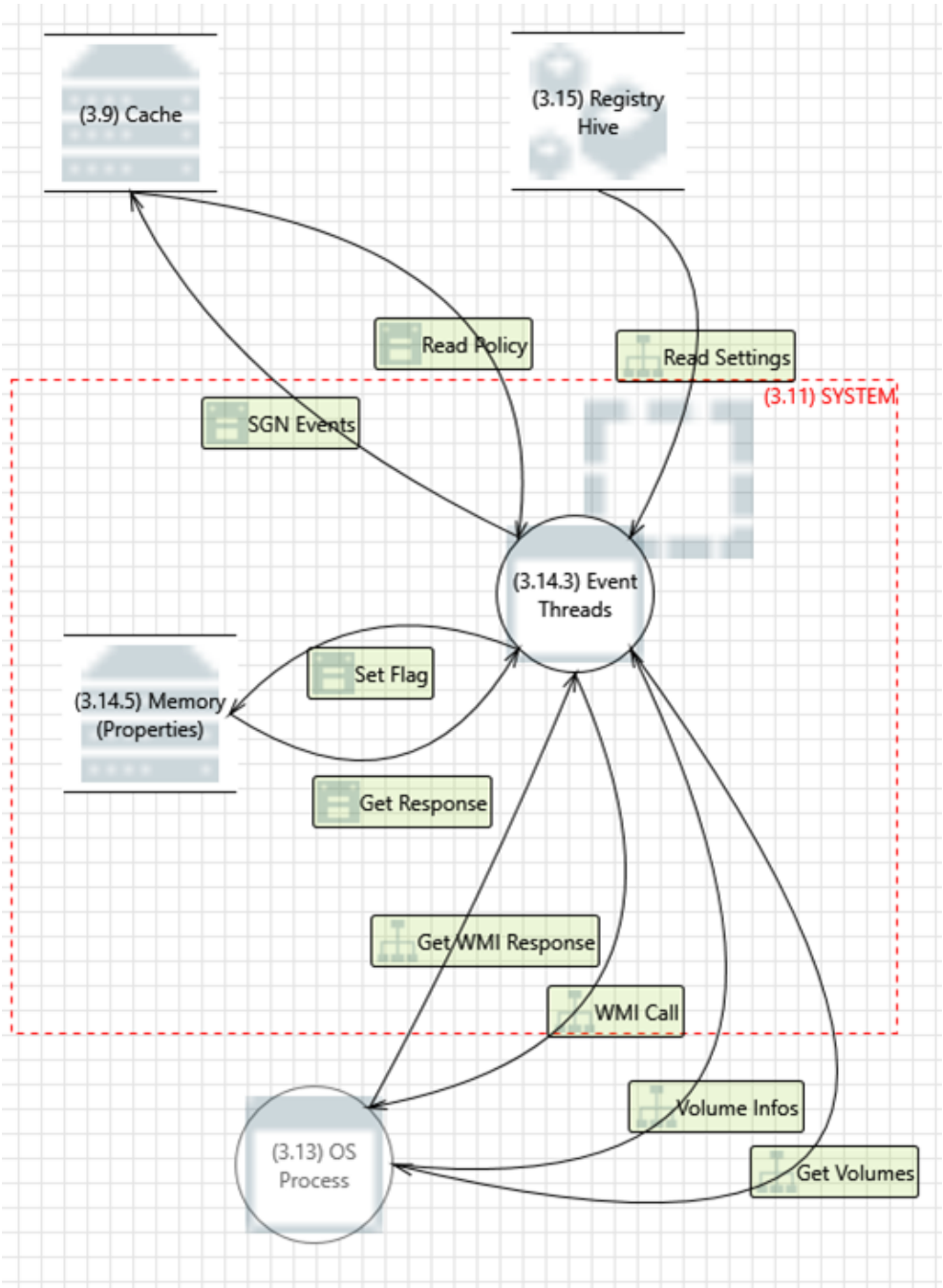


FIGURE 4.8: Threat Model WMIListener Event Threads

This diagram is a collection about the event threads, which were not already mentioned above. They can be modeled and discussed in detail separately in a future work, but this overhead is out of scope for this thesis. This diagram can be used as a template for these additions.

Name	**Context**
(3.14.3) Event Threats	SYSTEM
Communicates with	**Protocol used**
(3.9) Cache	FileIO
(3.13) OS Process	API Calls
(3.13) OS Process	WMI
(3.14.5) Memory (Properties)	-
(3.15) Registry	API Calls
Description	
See description above.	
Potential Vulnerabilities	
Potential attacks regarding the Cache (3.9), OS Process (3.13) and the Registry (3.15) are already discussed in the Level 1 diagram and the attacks targeting the memory are mentioned in the Level 2 – WMIListener BitLocker Change Thread diagram.	

4.5.8 Level 2 - MasterService

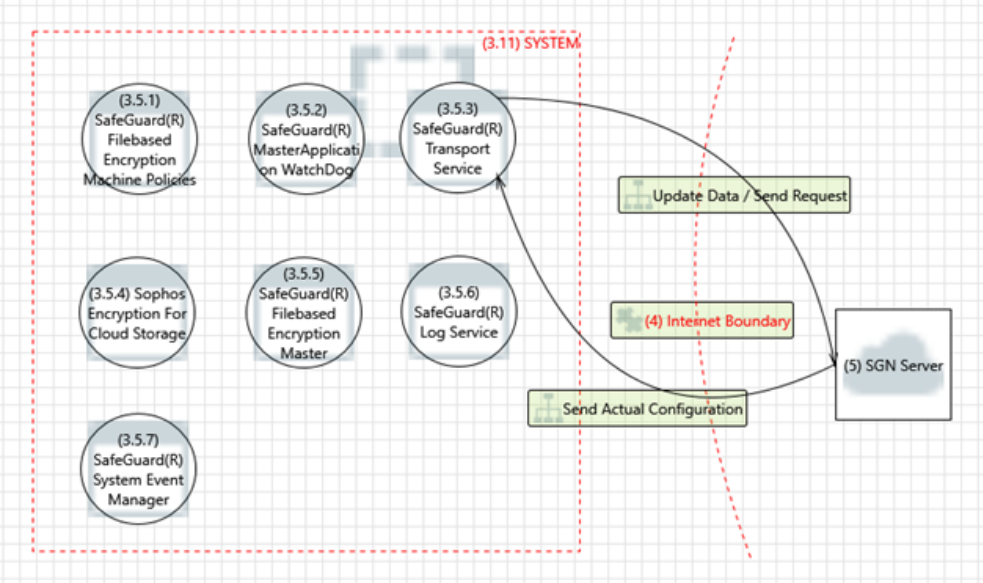


FIGURE 4.9: Threat Model MasterService

The SGN_MasterServicen.exe binary contains the services, which are automatically run at the client system startup. Since this thesis only analyzes parts of the software, only the communication lines of these parts are included in the diagram.

The other services are only added without communication lines, to build a starting point when the corresponding parts get added in the future. This requires the diagram to be split up like the WMIListener because of readability.

Name	**Context**
(3.5.3) SafeGuard(R) Transport Service	SYSTEM
Communicates with	**Protocol used**
(5) SGN Server	SSL / proprietary
Description	
The Transport Service is responsible for the communication between the managed SafeGuard client (3) and the SafeGuard server (5). The protocol used either a proprietary one, which was developed by Sophos, or for actual version SSL. The SSL variant is preferred.	
Potential Vulnerabilities	
The communication between SafeGuard client (3) and the SafeGuard server (5) is not part of this thesis, so this is out of scope.	

4.5.9 Summary

During the analysis of the components from the different threat models, additional vulnerabilities have been found, which were not evident during the inspection of the information gathering. Especially the communication paths between the selected components are now clearer as without the threat models. This is because the graphs are easy to understand. But the modeling required the assistance of the developers, as information about the internal workings of the software was needed.

A modeling, which goes deeper into detail was not useful, as it requires such a high amount of information from the developers, that it is less resource consuming when they are modeling it themself.

4.6 Tested Vulnerabilities

As described in the approach, some of the found potential vulnerabilities were tested if it is possible to exploit them. The types of impact were taken from the threat ranking, which was provided by Sophos.

Since there was no access to the source code itself, attacks, which require deeper knowledge of the software itself, were not possible in a reasonable time efford. But there was a possibility to get hints from the developers for trying a specific exploit. If such a hint was given, it is mentioned explicitly in the description.

4.6.1 Binaries

The access rights to the binaries only allow administrators write access to the binaries. Nevertheless, it is still useful to include testing scenarios to ensure, that these binaries are protected properly.

Potential problems regarding ASLR and DEP are already discussed during the information gathering, and since the exploiting of these missing security enhancements require a deep understanding of the binaries itself, it is not included here.

Things, which are still interesting to test is to further reduce the used security features by tampering with the signature of the binaries or the addition of payload to it and to observe, if and how the SafeGuard client reacts to it.

The test where made using the standalone client, as its behavior is the same as in the managed client version.

Name	Removing / Changing Code Signing Certificate
Impact	None, if used alone. Can vary when used in conjunction with another attack.
Attack succeeded	Yes
Prerequirements	Write access to the binaries
Description	
A change in the digital signature of a binary can induce that its content was changed by malware or another type of attack. It is tested, if the software recognizes if there was a change regarding: • The removal of the signature • Another signature was used	
Further use by other exploits	
It is easier to tamper with binaries, when there is no check made if they are the valid ones from the vendor. When there are further security enhancements missing like ASLR or DEP, they can be used in addition to this to binary exploiting.	

The SafeGuard binaries are installed at the following directories (and their subdirectories):

```
C:\Program Files (x86)\Sophos\SafeGuard Enterprise
C:\Program Files (x86)\Sophos\SafeGuard Shared
C:\Program Files\Sophos\SafeGuard Enterprise
C:\Windows\System32
C:\Windows\SysWOW64
```

There is a total of 241 binaries added by the SafeGuard client installation, so only a few of them are tested. The selected files are:

```
C:\Windows\System32\sgmbasen.dll
C:\Windows\SysWOW64\WMIListener.exe
C:\Program Files\Sophos\SafeGuard Enterprise\FileEncryption\
    feshellx64.dll
C:\Program Files (x86)\Sophos\SafeGuard Shared\SGDrvHlp.exe
C:\Program Files (x86)\Sophos\SafeGuard Enterprise\Client\x64\
    SGNCredProvn.dll
```

To modify these binaries, Windows was booted in safe mode. This can be done using the msconfig.exe tool (administrative permissions needed) or, depending on the Windows version, during startup or via the settings. During the removal of the digital signature, the modification date of the binary has changed.

As mentioned in the approach, the tool used to remove the certificates from the binaries was 'delcert' by deepred[23] and the one to sign a binary with another certificate was 'SignTool.exe', from of the Windows SDK[24].

The result of the system, when the binaries with the modifies binaries are as follows:

FileName	Removed certificate detected	Another certificate detected
sgmbasen.dll	No	No
WMIListener.exe	No	No
feshellx64.dll	No	No
SGDrvHlp.exe	No	No
SGNCredProvn.dll	No	No

This results show, that the modification of the digital certificates of the binaries was not detected at all by the SafeGuard client. To prevent that an attacker uses this vulnerability, it is highly recommended, that the files are checked for the correct digital signatures during startup or that a watchdog is verifying this during runtime.

Name	Adding payload to binaries
Impact	Can vary when used in conjunction with another attack.
Attack succeeded	Yes
Prerequirements	Write access to the binaries
Description	
Additional payload can be added in the binary, so without proper protection it is possible for an attacker to hide additional data in the binary.	
Further use by other exploits	
Like the 'Removing / Changing Certificate' attack, this can be used in addition to other vulnerabilities to give the attacker additional options to exploit the system.	

To add payload to binaries, the tool mention in the article 'Changing a Signed Executable Without Altering Windows Digital Signature' by Aymeric Barthe[25] was used.

Since this tool currently only supports x86 binaries, the attack was only tested on the following 2 binaries:

```
C:\Windows\SysWOW64\WMIListener.exe
C:\Program Files (x86)\Sophos\SafeGuard Shared\SGDrvHlp.exe
```

The usage of the tool to add a payload is:

```
AppendPayLoad.exe OriginalFile.exe Payload ModifiedFile.exe
```

As payload, a text file containing the string 'TEST' multiple times was used.

Unfortunately, during the tests, where the original binaries were replaced by the ones with the payload, the SafeGuard system did not notice these changes.

4.6.2 Registry

The permissions of the SafeGuard client registry entries are set as read only for normal users, so the only thing, which can be tried is to get interesting information out of it, which can help for further exploitation.

Name	Reveal vulnerabilities using the registry
Impact	n/a
Attack succeeded	-
Prerequirements	-
Description	
The information given by the registry entries can give an attacker hints about weaknesses to exploit.	
Further use by other exploits	
The usage of filenames alone (for example: IgnoredApplicationsSGN, AutoDirSizeProcesses) without the full path can allow attackers to adjust the names of their malware to have it treated by the SafeGuard client like these programs. The log file can be deleted or manipulated by attackers to hide their actions. This is easily done, because every user has full access rights to the path, where it is stored in.	

With the usage of the registry editor (or other tools) it is possible to view the entries of the SafeGuard client.

As mentioned in the Information Gathering, the registry entries of the SafeGuard client are at the following 5 locations:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Utimaco\SGMTrace]
[HKEY_LOCAL_MACHINE\SOFTWARE\Utimaco]
[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Utimaco]
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Utimaco]
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Utimaco\SGLCENC]
```

The interesting information in these locations are:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Utimaco\SGMTrace]
[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Policies\Utimaco\
    SGMTrace]
```

Contains the location of the LogFile.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Utimaco]
```

The MachCert is stored here. Information about what DLLs the SafeGuard services are using. Information about the installed config

```
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Utimaco]
```

Contains the following information about the encryption driver:

```
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Utimaco\
    SGLCENC]
"IgnoredApplicationsSGN"="SGPortable.exe;"
```

```
"AutoDirSizeProcesses"="wmplayer.exe"
"IgnoredApplicationsSGNCSE"="~"
"NoPersistentEncryptionApplications"="<SYSTEM>\\MSPAINT.EXE"
"IgnoredDevicesSGN"=""
"DefaultIgnoreRules"="C:\\*.*;C:\\*.*"
"DriverName"="LCENCM"
"ExcludedDevices"=""
"RulesFromSIDApplications"="C:\\Windows\\system32\\WUDFHost.exe
    "
"RecyclePath"=":\\$RECYCLE"
"SystemRoot"="C:\\Windows"
"DirSizeCorrection"="PROFILES"
```

This is interesting, because there are exceptions for the encryption driver listed.

This registry entries are from a standard installation, without further additions by an administrator. It is notable, that binaries there are included with their path, while others are only listed with their name alone. This allows attackers to rename their malware to match the names of the exceptions, so that it can abuse the special privileges given to it.

4.6.3 Services

The following 3 possible vulnerabilities are discussed:

- Stopping, disabling, or modifying services
- Unquoted service paths
- DLL hijacking

Because there were only limited resources available for this thesis, not all of them can be fully analyzed. But it will be mentioned how to further research these vulnerabilities.

Name	Stopping, disabling, or modifying services
Impact	Critical (Configuring/Modifying (adding/removing protectors etc) without administrative credentials)
Attack succeeded	No
Prerequirements	-
Description	
Stopping or preventing services from being started can break the protection by SafeGuard. Additionally, a manipulation of the service can allow the attacker to start another binary instead or in addition to the original one.	
Further use by other exploits	
Since the services run under the 'Local System' context, an escalation of privileges can be possible.	

There are different possibilities to prevent services from starting:

- Disable the service
- Remove/Rename the executable
- Remove access rights to the account, which the service runs
- Disable/Remove another service, which it depends on

So basically, each of these possible attacks requires sufficient access rights to the service.

During the information gathering, it is shown, that only the relevant accounts have the corresponding rights to modify the services. This means, that the services are properly secured and it was therefore not possible to stop them from working.

Name	Unquoted service paths
Impact	Critical (Privilege escalation (e.g. via SYSTEM daemons))
Attack succeeded	No
Prerequirements	-
Description	
A misconfiguration in the binary path of the service allows other binaries to execute as the service.	
Further use by other exploits	
Since the services run under the 'Local System' context, an escalation of privileges can be possible.	

The following command, as mentioned in the approach, was used:

```
wmic service get name,displayname,pathname,startmode |findstr /i "
    auto" |findstr /i /v "c:\windows\\" |findstr /i /v """
```

The SafeGuard services were not vulnerable to this attack.

Name	DLL hijacking
Impact	Critical (Privilege escalation (e.g. via SYSTEM daemons))
Attack succeeded	n/a
Prerequirements	-
Description	
The attacker can abuse the DLL search order of the system to load its own malicious DLL.	
Further use by other exploits	
Since the services run under the 'Local System' context, an escalation of privileges can be possible.	

As mentioned in the approach, the following conditions must be met to allow the exploit to be successful:

- One of the paths of the DLL search order has write access to the attacker

- The service tries to load a DLL, which is either not available or is in a lower ranking of the DLL search order, so that the attacker can put its own DLL in a higher ranking one

The write access to a DLL search path is due to weak folder permissions. This is out of scope for the SafeGuard client, because the folder permissions are set properly and they are not added the search path.

To see which DLL are loaded and what their dependencies are is be too complex to do without the proper tools (see Information Gathering - Dependencies). This can better be done with static source code analysis from the developers itself.

So, this potential exploit was not researched any further, but it is highly recommended to do this in further research. In a proper configured system, the folder permissions prevent an attacker to use this exploit, but it is better to make it as hard as possible to use this exploit.

4.6.4 BitLocker

Microsoft BitLocker is the technology, which is used in the SafeGuard client for the full disk encryption. So, the access to the recovery key is a critical break of security, because it can be used to decrypt the volumes and allow attackers to access it.

Name	Getting the BitLocker recovery key
Impact	Critical (Access to BitLocker PIN/Password)
Attack succeeded	n/a
Prerequirements	-
Description	
Using the recovery keys, anyone can decrypt the BitLocker volumes in the system.	
Further use by other exploits	
-	

As already mentioned, when the user has sufficient access rights to the WMI class Win32_EncryptableVolume from the namespace Root\cimv2\Security\MicrosoftVolumeEncryption, it is possible to get the recovery key with a simple command:

```
manage-bde -protectors c: -get
```

The access rights to this namespace are as follows:

Group or user name	Permissions
LOCAL SERVICE	Execute Methods Provider Write Enable Account
NETWORK SERVICE	Execute Methods Provider Write Enable Account
Administrators	Execute Methods Full Write Partial Write Provider Write Enable Account Remote Enable Read Security Edit Security

So, it is not possible to get the credentials using the WMI functions.

Another attempt is the scanning of the memory to find the BitLocker key. Unfortunately, due to the lack of resources, the software was not acquired, but it can be an option for the developers to do this, because it is possible to recovery other keys as well.

4.6.5 Cache

The cache contains the current state and settings of the SafeGuard client. So, this is an interesting point for attackers to look for potential vulnerabilities. Additionally, it is shown during the information gathering, that all users on the system have full access to the cache, which makes it easy to modify the files.

However, this is not an easy task, because there are most certainly counter measures to deal with. Corrupting the cache will result in a reboot loop, which is an intended behavior as mentioned in the threat ranking. So, a simple deletion or corruption of the cache is not counted as an attack itself.

Most of the files in the cache have the extension .XML, but they are not in the standard text format, so modifying them is not trivial. Although it might be possible to copy the files from different configurations to keep the format intact. It seems, that these XML files have a binary format, but when it was tried to open them with tools like Stylus Studio, which supports many different XML formats, including BXML (Binary XML), it did not work. There are text chunks readable, so basic information is obtainable. But to further investigate the files, the specification is needed.

To find potential vulnerabilities regarding the cache, the following attacks were performed:

- Getting classified information out of the cache
- Modify file/directory permissions to prevent parts of the SafeGuard client from working

- Modify the cache itself to change the SafeGuard settings

Name	Getting classified information out of the cache
Impact	n/a
Attack succeeded	n/a
Prerequirements	-
Description	
The configuration files contain information, which are readable with a simple text editor.	
Further use by other exploits	
It can be possible to find new potential targets and information about the system for further attacks.	

As mentioned before, the configuration files are not in plain text. But with the right tools and modifications, it is possible to view most of the containing XML information. To view the content of the files, the following steps were taken:

- The advanced text editor Notepad++ was used[43].
- The additional Notepad++ plugin 'XML Tools' are installed
- Open the XML file
- Replace '\0' with an empty string
- Reformat the XML information with Menu -> Plugins -> XML Tools -> Pretty Print (XML only – with line breaks)

With this, it was possible to find information regarding the users and machines, which are stored in the configuration. This and the configuration itself can be useful when it is tried to manipulate the cache (which is possible for every user, due to weak permissions)

Name	Modify file/directory permissions to prevent parts of the SafeGuard client from working
Impact	High (Modification of policies applied to the system)
Attack succeeded	Yes
Prerequirements	-
Description	
By modifying the directory permissions of the local cache, the BitLocker client can not further access them and therefore stop working.	
Further use by other exploits	
Since there was no reboot loop started, it can be possible for attackers to get access to the system.	

During the information gathering it was shown, that the permissions to the local cache are set that everyone has full access. So, it is possible to modify these permissions to lock the SafeGuard client from accessing these files.

The following steps were taken to produce this behavior:

1. Disconnect the client from the network, so that the server can not send an update during the attack
2. Set all permissions for the Cache and CacheBackup to 'Deny' for the SYSTEM account. This is possible, because of the weak folder permissions in these directories.
3. Reconnect the network connection to the server
4. Now, the client will shut down automatically, this is because of the problems when accessing the cache
5. Start the client again
6. At the startup, now the following error message is shown 2 times: "Sophos SafeGuard® Authentication Service is not running, no further action possible!"
7. This windows can just be closed when OK is clicked.
8. Now there should be a reboot loop, but this did not happen, instead there was a local logon possible.
9. During the next reboot, the system stays at logon screen in a dead lock

The normal behavior puts the system into a reboot loop or a deadlock, but this was not happening during the first reboot. Another error message and a deadlock or lockdown of the computer is better suited here.

Name	Use a full copy of the cache itself to change the SafeGuard settings
Impact	High (Modification of policies applied to the system)
Attack succeeded	Yes
Prerequirements	-
Description	
Using an older configuration of SafeGuard, the client can have weaker security settings, so that confidential information is not protected properly. An attacker can use this security flaw to get this information.	
Further use by other exploits	
The attacker can put harmful software in those protected folders and change the configuration back to the actual status. So, this malware got into the protected area.	

As mentioned, the actual configuration of the SafeGuard client is stored in the local Cache. So when there was a copy made of the cache, it can be possible to use this copy after the configuration changes to revert it to the previous one.

If this attack succeeds, it can allow the user to revert to an older configuration, where less restrictive permissions are set and therefore can allow an attacker to disable security features.

This attack is possible, because of the weak access rights to the Cache. This allows every user full access to the directories. Also, the backup of the Cache has the same permissions set, so a validation between these two states is not useful.

To reproduce this attack, the following steps must be done:

1. A configuration was applied, where the BitLocker disc encryption was disabled.
2. The client synchronized with the server to get this configuration.
3. A copy of the local Cache and its backup on the client was made.
4. Now the server changes the configuration to enable BitLocker.
5. The client synchronizes again, to get this new configuration. Now the volume got encrypted with BitLocker.
6. When it was now tried to disable BitLocker on the client, it was enabled again immediately.
7. Now the client got disconnected from the network to prevent synchronizing.
8. The actual version of the local Cache was replaced with the previous one.
9. A reboot was done on the client
10. Now the previous configuration was used again, which allows to disable BitLocker on the client.
11. Obviously, when the client was reconnected to the network, a sync with the server was again possible and the actual configuration was applied again.

To prevent this from happening, the directory/file permissions have to be adjusted to prevent the full access from the user. If this is not possible, at least the backup of the Cache must be better protected, so that it can be used for validating if the most current version was used.

Name	Use a copy of the cache itself to change SafeGuard settings
Impact	High (Modification of policies applied to the system)
Attack succeeded	No
Prerequirements	-
Description	
Like the attack, which uses an older version of the configuration, but this attack uses not all but only a selection of the configuration files.	
Further use by other exploits	
Since the configuration can be modified by the attacker, it can used to modify classified information or even put malware in the protected areas. When the configuration was changed back, it is highly possible, that this attacks is undetected by the user.	

This attack is like the previous one, to the steps are:

1. A configuration was applied, where the BitLocker disc encryption was disabled.
2. The client synchronized with the server to get this configuration.
3. A copy of the local Cache and its backup on the client was made.
4. Now the server changes the configuration to enable BitLocker.
5. The client synchronizes again, to get this new configuration. Now the volume got encrypted with BitLocker.
6. When it was now tried to disable BitLocker on the client, it was enabled again immediately.
7. A comparison of the LocalCache with the copy was done to find the changes regarding BitLocker (see notes)
8. Now, another change of the configuration was made on the server and synchronized with the client. This had to be done to make this attack different to the previous one.
9. Now the client got disconnected from the network to prevent synchronizing.
10. The config files regarding BitLocker of the actual Cache were replaced with the previous ones (see notes)
11. Reboot the client
12. The client was now in an undefined state and was locking the computer

Note: To find the relevant configuration files, which are necessary for enabling BitLocker, the 2 configuration directories have been compared. The files, which changed were the following:

Filename	**Folder**	**Comparison result**
`lcdir.bak`	`.\`	Binary files are different
`lcdir.dir`	`.\`	Binary files are different
`ida_drive_id.xml`	`.\auditing\ inventory`	Binary files are different
`ima_last_policy_received.xml`	`.\auditing\ inventory`	Binary files are different
`ima_poa_type.xml`	`.\auditing\ inventory`	Binary files are different
`ima_unencrypted_drives.xml`	`.\auditing\ inventory`	Binary files are different

0x99b634b1923a4d478e7a8 b7c459cf9bd.xml	.\auth_u\alo	Binary files are different
0x99b634b1923a4d478e7a8b 7c459cf9bd.xml	.\kfiles\user	Binary files are different
0x424a297c41e0400aa10ec1 fd1f55218b.xml	.\polassign\ machine	Binary files are different
0x99b634b1923a4d478e7a8b 7c459cf9bd.xml	.\polassign\user	Binary files are different
0x424a297c41e0400aa10ec1 fd1f55218b.xml	.\policy_m\enc	Only with BitLocker enabled
0x424a297c41e0400aa10ec1 fd1f55218b.xml	.\policy_m\enc\. lock	Only with BitLocker enabled
0x424a297c41e0400aa10ec1 fd1f55218b.xml	.\policy_m\ref	Only with BitLocker enabled
0x424a297c41e0400aa10ec1 fd1f55218b.xml	.\policy_m\ref\. lock	Only with BitLocker enabled
0x99b634b1923a4d478e7a8b 7c459cf9bd.xml	.\policy_u\enc	Only with BitLocker enabled
0x99b634b1923a4d478e7a8b 7c459cf9bd.xml	.\policy_u\enc\. lock	Only with BitLocker enabled
0x99b634b1923a4d478e7a8b 7c459cf9bd.xml	.\policy_u\ref	Only with BitLocker enabled
0x99b634b1923a4d478e7a8b 7c459cf9bd.xml	.\policy_u\ref\. lock	Only with BitLocker enabled
global.xml	.\systray\cache	Binary files are different
57e229a71areport.xml	.\transout	Only with no BitLocker Enabled
57e229a71breport.xml	.\transout	Only with no BitLocker Enabled
57e22a811creport.xml	.\transout	Only with no BitLocker Enabled
57e22a811dreport.xml	.\transout	Only with no BitLocker Enabled

57e22a811ereport.xml	.\transout	Only with no BitLocker Enabled
57e229a71areport.xml	.\transout\.lock	Only with no BitLocker Enabled
57e229a71breport.xml	.\transout\.lock	Only with no BitLocker Enabled
57e22a811creport.xml	.\transout\.lock	Only with no BitLocker Enabled
57e22a811dreport.xml	.\transout\.lock	Only with no BitLocker Enabled
57e22a811ereport.xml	.\transout\.lock	Only with no BitLocker Enabled
57e229a71areport.xml	.\transout\ref	Only with no BitLocker Enabled
57e229a71breport.xml	.\transout\ref	Only with no BitLocker Enabled
57e22a811creport.xml	.\transout\ref	Only with no BitLocker Enabled
57e22a811dreport.xml	.\transout\ref	Only with no BitLocker Enabled
57e22a811ereport.xml	.\transout\ref	Only with no BitLocker Enabled
57e229a71areport.xml	.\transout\ref\.lock	Only with no BitLocker Enabled
57e229a71breport.xml	.\transout\ref\.lock	Only with no BitLocker Enabled
57e22a811creport.xml	.\transout\ref\.lock	Only with no BitLocker Enabled
57e22a811dreport.xml	.\transout\ref\.lock	Only with no BitLocker Enabled
57e22a811ereport.xml	.\transout\ref\.lock	Only with no BitLocker Enabled

4.7 Untested Vulnerabilities

There were already untested vulnerabilities already mentioned before, but there are still other interesting areas for potential attacks, which are worth to further investigate. These are mainly the communication parts of the SafeGuard client, so by compromising them, it is likely to at least stop the client from working or at worst case, a critical break of confidentiality can happen.

In the following subsections, these communication types and their usage in the client are short explained and recommended types of possible attacks are listed.

4.7.1 Memory

Due to the limited access to the source, and the deep inspection on how the SafeGuard clients works internally (especially the data structures where the keys are stored in) it was not possible to achieve the inspection of the memory in a reasonable amount of time. But when the internal information about the key storage in provided, it can be possible to write a plugin for Volatility to scan for nonencrypted keys in the memory. This can then be automated and included in the testing process, for example when the machine gets into hibernation or it crashes and creates a memory dump. Also, the page file can be used as source for the scan.

4.7.2 Named Pipes

The Windows SafeGuard client is doing much of the IPC communication over named pipes. During the information gathering, it was found out, that these pipes have fixed names and no maximum limit of connections.

With the help of the header files and a conversation with a developer, the naming scheme of the named pipes was interpreted as follows:

[Type][EVT-Class][Instance]	
Type	BRC_ ... Broadcast
EVT-Class	Name of the EVT (event) class
Instance	Class Name

Examples:

`BRC_BIN_CHANGEDBitLockerNativeWrapper`

Broadcast-event 'BIN_CHANGED' from BitLockerNativeWrapper

`BRC_BitLockerMAppEvtBitLockerNativeWrapper`

Broadcast-event 'BitLockerMAppEvt' from BitLockerNativeWrapper

It was mentioned by the developers, that the information and data sent over the named pipe is encrypted, which makes sniffing these IPC communication harder. Although, it might be possible to find patterns in the communications.

Possible attacks, which involve named pipes are:

- Get information by sniffing named pipes.
- Flood specific named pipes, to prevent modules of the SafeGuard to work properly.
- Using race conditions to spoof named pipes

It is needed to develop a custom built tool for this, as there was no suitable one found online. This tool must be able to simultaneously handle multiple pipes, as there are over 50 different ones used by the SafeGuard client.

4.7.3 RPC

Beside named pipes, Remote Procedure Calls (RPC) are the common used type of communication by the SafeGuard client.

4.7.4 IOCTL

The communication to and between the FileEncryption Driver and the KeyStore Driver is done with IOCTL. So, this is an interesting point to place a sniffer and try to get classified information from.

4.8 Next Steps

The software package provided consisted of the SafeGuard 7 client (and server), however the current version is 8, while the new version is starting to be developed soon. Since there are obviously many changes made during the different version, there are several regarding the components, which are discussed in this thesis. So, these changes had to be included in the already made threat models.

As it was shown during this thesis, the test of the complete client is not realistically achievable by a single person. But with sufficient access to the relevant source code or documentation can allow a detailed test of components, which had to be skipped in this thesis. This include:

- The parts under 'Untested Vulnerabilities': Named Pipes, RPC, IOCTL
- Exploits, which require binary analysis (Buffer overflow, ...)
- Tampering with the Cache

5 Case Study Feedback

To see, how well the security analysis was received, the developers at Sophos were asked for a feedback about it. Their response was the following:

> *Feedback on case-study in "Thorough Analysis of the Security Architecture of a Windows-based Enterprise Encryption"*
>
> *The approach presented in this paper was evaluated in a case study on a rather complex software product. Contrary to our internal threat modelling process, this approach represents an "outside-in" method that could be used to retro-actively create initial threat models for existing software.*
>
> *During the initial stage ("Information Gathering") there was very little guidance required, as the proposed methodology and tooling proved to be a suitable mechanism for getting an overview of the modifications to the system caused by installation and usage of the software. However, due to the complexity of the product and the number of components and their relations, it quickly became apparent that the scope of this analysis had to be reduced from the full system to a set of features deemed "security relevant" in order to be conducted within a reasonable time.*
>
> *Based upon this reduction of scope and a number of abstractions, it was possible to create data-flow diagrams that served as the starting-point for threat-modelling. As the data-flows were rooted in an analysis of an implementation rather than that of a specification, it provided the benefit of verifying the information present in existing threat-models.*
>
> *The approach presented in this paper managed to capture the core entities of the software was able to highlight inaccuracies in existing models. Due to the nature of this "outside-in" method it was however not able to show data-flows that cannot occur in the specific test environment (e.g. different data-flows on other Windows versions).*
>
> *Retro-actively developing a threat-model for existing complex software using the proposed "outside-in" methodology has shown to quickly provide interesting results in the realm of implementation bugs and to identify inconsistencies between specification and implementation upon review. It does suffer from the apparent danger of missing architectural flaws or critical aspects that were not observed in the information gathering phase.*
>
> *We suggest to employ this approach complementary to threat-modelling based on design specifications, as it provides verification of the implementation as-well as promoting a mindset that is closer to that of an attacker rather than that of a developer. On complex software it might be necessary to limit the scope of the analysis, considering only certain entities or features of a software.*

This shows, that the case study was helpful for them to further improve their process in increasing the security of their software.

6 Conclusion

In general, it must be said, that although it was quite challenging for an external person to get an insight to such a complex software, it was interesting to see how much stuff is going on in the background. For the users point of view, the software consists only of a few dialogs, but in the background, there are many different components, which had to work together smoothly for the system to work.

Also, it was an interesting task to select and introduce a threat mitigation technique to such a software. Some developers had already been using the Microsoft SDL tool during the development, but it was only used for small parts and not as a top down approach. Hopefully, this concept can be further used by the developers, because when the threat models are maintained regularly, it will make the detection of potential exploits more comfortable and manageable.

6.1 What worked well/what didn't?

First, as already mentioned in the feedback from the developers, it was possible to make a security analysis of the Windows SafeGuard client as an external tester. Naturally, it was not possible to create the same level of detail as an internal security analysis can produce, but it can create another point of view to find potential problems, which were not thought of before.

The setup of the testing environment and getting used to the usage of the software (client and server) is quite trivial and can be done without much afford.

Also, the selection of the Microsoft SDL Tools was a good choice, because it allowed the building of a threat model using a top down approach. It was comfortable to show the status during the meetings, because the models are quite self explaining. However, the SDL tools have limitations, which are be needed to deal with (see the section 'Next Steps').

The information gathering was easy manageable, because of the wide variety of tools and techniques, which are available for blackbox testing.

However, not everything worked out as planned. During the early stages of the information gathering, it was shown that the software was complexer than expected at the start. So, handling all the components with the limited resources available was not realistically possible. So, there was an agreement made to only inspect parts of the software.

As the threat models got more detailed, additional information from the developers was needed. So, while it was possible to build a threat model with the information got at a meeting, this model can not include all possible information. For example, during the meetings a threat model was made with a named pipe as communication path. But it was not possible to find out all the different properties of these named pipe, because it required the assisting developer to research or talk to other developers to find out this information. So, it is much more effective if the completion of the models is done by an internal employee.

Also, the testing of complex vulnerabilities, which include encryption or proprietary technologies is hard manageable without the full specification of the software. It can be possible with the usage of the source code, but since it was a restriction of this thesis to not use any source code at all this was out of scope.

Worked well:

- Information gathering using black box testing
- High level threat modeling
- Finding potential exploits

Worked not as intended:

- Make a detailed threat model of the parts to test
- Find more detailed exploits

Was not possible:

- Get a full overview of the program
- Make a complete threat model

6.2 Next Steps

This thesis can be viewed as a starting point to introduce threat modeling into the development process, so there are recommendations on how to continue and improve this work.

One such improvement is to see how much work it requires to adapt this thesis, so that it can be applied to software, which runs on Linux, Android, Mac, or other platforms. The vulnerabilities have to be adjusted to the specific platform, but the general approach is still be the same (Information Gathering, Threat Modeling, Exploiting).

The Microsoft SDL Tool worked quite well for this approach, but when more developers want to use this tool, especially when they want to work at the same time on the models, it is better to find an alternative to it. This is required when other environments than Microsoft Windows have to be supported, as the generated threat list only considers Windows specific threats. It is possible to add additional content to the SDL Tool, but this will require

extra work and deep insight to the vulnerabilities to the system.

Other improvements regarding this thesis are to expand the list of the potential vulnerabilities. To find proper ones, it is a good starting point to look at Windows exploitation courses to get ideas how attackers work. A popular course is Advanced Windows Exploitation, which offered by Offensive Security [44]. The syllabus of the course is available online, and several rather interesting topics are handled there (DEP/ASLR/EMET Bypass, Kernel Drivers Exploitation, and so on...).

Additional improvements can include the automatization of the testing process. This is not be possible for every part (for example Threat Modeling), but the Information gathering and parts of the potential attacks can be scripted to reduce the time it takes to fulfill this tasks.

Bibliography

[1] M. E.K. F. Khan, "A comparative study of white box, black box and grey box testing techniques", *International Journal of Advanced Computer Science and Applications, Vol. 3, No.6*, pp. 12–15, 2012.

[2] W. Jackson, *Static vs. dynamic code analysis: Advantages and disadvantages*, https://gcn.com/articles/2009/02/09/static-vs-dynamic-code-analysis.aspx, Accessed: 2016-11-20, 2009.

[3] A. Shostack, *Threat modeling*. John Wiley & Sons, Inc., 2014.

[4] A. Shostack, *The threats to our products*, https://blogs.microsoft.com/microsoftsecure/2009/08/27/the-threats-to-our-products/, Accessed: 2016-11-20, 2009.

[5] B. Schneier, "Attack trees", *Dr. Dobb's journal*, pp. 21–29, 1999.

[6] MITRE, *Common attack pattern enumeration and classification*, https://capec.mitre.org, Accessed: 2016-11-20, 2016.

[7] OWASP, *Owasp top ten project*, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, Accessed: 2016-11-20, 2016.

[8] M. H.M.M.J.L. M. Thomlinson, *Windows isv software security defenses*, https://msdn.microsoft.com/en-us/library/bb430720.aspx, Accessed: 2016-11-20, 2010.

[9] M. Miller, "A brief history of exploitation techniques & mitigations on windows.", Tech. Rep., 2007.

[10] Microsoft, *Introduction to code signing*, https://msdn.microsoft.com/en-us/library/ms537361(v=vs.85).aspx, Accessed: 2016-11-20, 2016.

[11] J. Niemelä, "It's signed, therefore it's clean, right.", in *CARO 2010*, 2010.

[12] W. Stallings, *Cryptography and network security (fourth edition)*. Prentice Hall, 2005.

[13] U. H.V. L. Naidu, "A survey on windows component loading vulnerabilities.", *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) 2.5*, pp. 1780–1783,

[14] D. K.J.P. T. Woller, *Amd memory encryption*, http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, Accessed: 2016-11-20, 2016.

[15] M. R.D.A.S. A. Ionescu, *Windows® internals part 1 - sixth edition*. Microsoft Press, 2012.

[16] Portcullis Labs Research and Development, *Windows named pipes: There and back again*, https://labs.portcullis.co.uk/blog/windows-named-pipes-there-and-back-again/, Accessed: 2016-11-20, 2016.

[17] P. G.M.Y.L. D. Molnar, "Sage: Whitebox fuzzing for security testing", *ACM Queue - Networks, Volume 10 Issue 1*, p. 20, 2010.

[18] Oracle, *Oracle vm virtualbox*, https://www.virtualbox.org/, Accessed: 2016-11-20.

[19] VMware Inc., *Vmware*, http://www.vmware.com/, Accessed: 2016-11-20, 2016.

[20] Microsoft, *Hyper-v overview*, https://technet.microsoft.com/en-us/library/hh831531.aspx, Accessed: 2016-11-20, 2016.

[21] T. Newton, *An introduction to the windows system state analyzer*, https://blogs.technet.microsoft.com/askperf/2010/01/11/an-introduction-to-the-windows-system-state-analyzer/, Accessed: 2016-11-20, 2010.

[22] Mister Group, *System explorer*, http://systemexplorer.net/, Accessed: 2016-11-20, 2016.

[23] deepred, *Delcert - sign strip tool*, http://forum.xda-developers.com/showthread.php?p=2508061#post2508061, Accessed: 2016-11-20, 2008.

[24] Microsoft, *Signtool.exe (sign tool)*, https://msdn.microsoft.com/en-us/library/8s9b9yaz(v=vs.110).aspx, Accessed: 2016-11-20.

[25] A. Barthe, *Changing a signed executable without altering windows digital signature*, https://blog.barthe.ph/2009/02/22/change-signed-executable/, Accessed: 2016-11-20, 2009.

[26] Common Exploits, *Unquoted service paths*, http://www.commonexploits.com/unquoted-service-paths/, Accessed: 2016-11-20, 2012.

[27] Parvez, *Elevating privileges by exploiting weak folder permissions*, http://www.greyhathacker.net/?p=738, Accessed: 2016-11-20, 2013.

[28] The Volatility Foundation, *Volatility foundation*, http://www.volatilityfoundation.org/, Accessed: 2016-11-20, 2014.

[29] M. L.A.C.J.L. A. Walters, *The art of memory forensics*. Wiley, 2014.

[30] V. Katalov, *Breaking bitlocker encryption: Brute forcing the backdoor (part i)*, http://blog.elcomsoft.com/2016/06/breaking-bitlocker-encryption-brute-forcing-the-backdoor-part-i/, Accessed: 2016-11-20, 2016.

[31] Tribal Chicken, *Extracting bitlocker keys with volatility (poc)*, https://tribalchicken.io/extracting-bitlocker-keys-with-volatility-part-1-poc/, Accessed: 2016-11-20, 2015.

[32] Tibbo, *Introducing io ninja*, http://tibbo.com/ninja.html, Accessed: 2016-11-20.

[33] GuyHarris, *How to set up a capture*, https://wiki.wireshark.org/CaptureSetup, Accessed: 2016-11-20, 2013.

[34] AdiKo, *Rpcsniffer*, https://adiko.github.io/RPCSniffer/, Accessed: 2016-11-20.

[35] J.-M.B.J.B.J.B. Y. Girardin, *Rpcview*, http://rpcview.org/, Accessed: 2016-11-20.

[36] user460153, *How to log the deviceiocontrol calls of a program on windows*, http://stackoverflow.com/questions/9947933/how-to-log-the-deviceiocontrol-calls-of-a-program-on-windows, Accessed: 2016-11-20, 2012.

[37] Sophos, *Safeguard enterprise user help*, https://www.sophos.com/en-us/medialibrary/PDFs/documentation/sgn_7_h_eng_user_help.pdf?la=en, Accessed: 2016-11-20, 2014.

[38] Sophos, *Safeguard enterprise installation best practice*, https://www.sophos.com/en-us/medialibrary/PDFs/documentation/sgn_7_bpg_eng_installation_best_practice.pdf, Accessed: 2016-11-20, 2014.

[39] M. S.P.K. T. Peyrin, "Freestart collision for full sha-1", in *IACR - EUROCRYPT*, 2016.

[40] dependencywalker.com, *Dependency walker 2.2*, http://www.dependencywalker.com/, Accessed: 2016-11-20.

[41] Microsoft, *Security descriptor definition language*, https://msdn.microsoft.com/library/aa379567.aspx, Accessed: 2016-11-20.

[42] Microsoft, *Binscope 2014*, https://www.microsoft.com/en-us/download/details.aspx?id=44995, Accessed: 2016-11-20, 2015.

[43] D. Ho, *Notepad++*, https://notepad-plus-plus.org/, Accessed: 2016-11-20, 2016.

[44] Offensive Security, *Advanced windows exploitation*, https://www.offensive-security.com/information-security-training/advanced-windows-exploitation/, Accessed: 2016-11-20.

CV

Stefan Bachmair

Curriculum Vitae

PERSONAL DETAILS

Birth	May 13, 1981
Birthplace	Voecklabruck, Austria
Mail	stefan.bachmair@gmx.net

EDUCATION

BSc. Computer Science 2012-2015
Johannes Kepler University Linz

WORK EXPERIENCE

System Administrator 2006-present
LM Investment GmbH, Hightech Fabrics GmbH, Part Time
Maintaining Windows-based Clients and Linux + Windows Servers

Software developer 2001-2005
TAB Austria, Full Time
Developing touchscreen games in Visual C++

SKILLS

Languages	German (mother tongue) English (fluent)

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.
This printed thesis is identical with the electronic version submitted.

Place, Date

Signature