Remote Assessment of Client Trustworthiness

Yngve Kristiansen

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2005

Institutt for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

ABSTRACT

When two parties communicate over the Internet the server side has little knowledge about the client environment. Despite this situation the server has to trust the data it receives for further processing and that the data it sends to the client is handled in a proper way.

One way to possible reduce this insecurity is to analyze the client environment prior to communicating sensitive data. The client's trustworthiness can be determined in advance and the server can take this into consideration at further communications.

How can we do such a security analysis? There exist applications today that scan systems and give back vulnerability reports. The validity of this report is high when everybody plays by the rules. What happens if an adversary or a Trojan tries to alter the scan results?

In this report we will analyze the different aspects when doing a security analysis of the remote client. Our main focus is how to protect the data and the application performing the security analysis. We define a framework that we apply to our proposed systems in a case study. We believe such a system is feasible, but practical issues will be the biggest challenges.

SAMMENDRAG

Når to parter kommuniserer over Internett har server siden liten kjennskap til klientmiljøet. Til tross for dette må serversiden ha tillit til de data den mottar for videre prosessering og at de data den sender fra seg blir behandlet på en korrekt måte.

En måte som mulig reduserer denne usikkerheten er å analysere klientmiljøet før sensitive data blir kommunisert. Klientens pålitelighet kan fastslås på forhånd og serveren kan ta dette med i betraktning ved videre kommunikasjon.

Hvordan kan vi utføre en slik sikkerhetsanalyse? Det finnes applikasjoner i dag som skanner systemer og gir tilbake sårbarhetsrapporter. Gyldigheten til disse rapportene er høy så lenge alle følger spillereglene. Hva skjer dersom brukeren eller en Trojansk hest forsøker å endre resultatene?

I denne rapporten vill vi analysere de forskjellige aspektene ved gjennomføring av en sikkerhetsanalyse av den eksterne klienten. Vårt hovedfokus er hvordan beskytte data og applikasjonen som utfører sikkerhetsanalysen. Vi definerer et rammeverk som anvendes på våre foreslåtte system i en case studie. Vi mener et slikt system er gjennomførbart, men praktiske hensyn vil være den største utfordringen.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The following chapters will state the problem description, contributions, motivation and justification, and research questions.

1.1 Problem description

Most of the systems today where the general public is the user, web applications are the most common. We see this in online bookstores, internet banking, online newspapers etc. More and more use online services, and at the same time there is an increase in malicious applications and other threats.

To illustrate the problem we cite two examples. In the first example, spyware monitors the communication between the customer and the internet banking operator [39]. To detect if a client is infected and the spyware is present is often impossible. In this case the spyware is visible because it also interferes in the communication when downloading. The bank blocks the user if the spyware is detected. The second example is when malicious software on the client machine hijacks the communication and redirects to another website as described in the article [40]. A business has been forced to close because hijacking has resulted in a lack of customers.

Figure 1 shows an overview of the system we are studying. Here, the server is the trusted home and the client is the unknown executing environment.

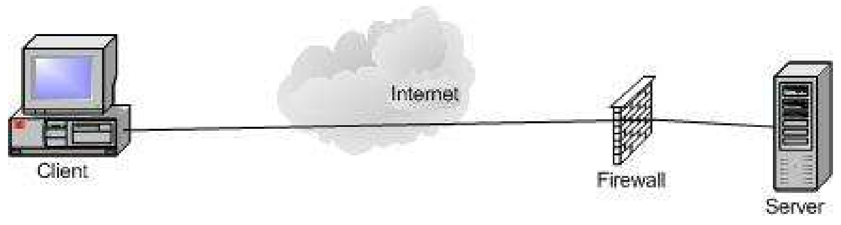

Figure 1: Overview of the system being studied (Client/server model)

When communicating over the Internet as described above, the server side has little knowledge about the client environment. Despite this situation the server has to trust the data it receives for further processing and that the data it sends to the client is handled in a proper way. It is difficult to know how trustworthy the information is when we do not know how the information was generated. If the client belongs to another party than the server, this insecurity increases even further. The server has to assume that the client in basis is trustless.

One way to possible reduce this insecurity is to analyze the client environment prior to communicating sensitive data. The client's trustworthiness can be determined in advance and the server can take this into consideration at further communications. How can we do such a security analysis? There exist applications today that scan systems and give back vulnerability reports. The validity of this report is high when everybody plays by the rules, what happens if the user or a Trojan tries to alter the scan results?

Figure 2 shows in more detail the situation on the client side. The red arrows symbol the client environment attacking the application.

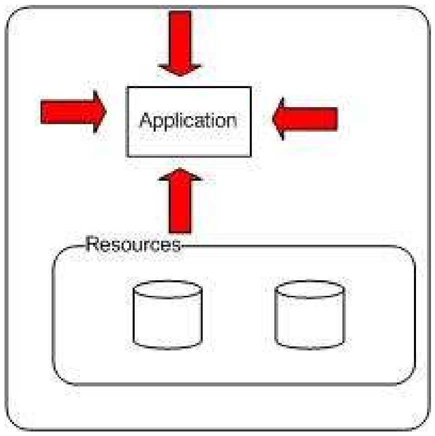

Figure 2: Client side in more detail of the system being studied

In this study we will go into how a security analysis of the remote client environment can be done and the different challenges.

1.2 Claimed contributions

We will illustrate and add knowledge to the different aspects and challenges when analyzing a remote client environment. It is difficult to achieve a one hundred percent security analysis. Based on challenges found in related theory we define a framework and compare two systems and seek what characterizes a better system.

1.3 Motivation and justification

Service providers and end users wish for confidentiality when it comes to business related and sensitive data. Security is a complex area and many users have not the knowledge or the interest. The service providers do an effort in sufficient authentication of the user. When the *PC* becomes a threat, similar to man-in-the-middle attacks, we need methods to ensure the trustworthiness of the whole client side.

1.4 Research questions

In the context of our proposed system:

1. How can we analyze the robustness of a client environment to malicious activities?

2. Given a hostile environment; how can we prevent the gathered data from being modified by an attacking party?

2 CHOICE OF METHODS

In this study we analyze systems that do remote assessment of the client trustworthiness. This means checking for protective measures active on the client prior to communicating business related and sensitive data.

We choose a qualitative approach as described in [41]. We seek to describe the remote assessment process more in depth. In quantitative studies, often experiments and surveys, one can generalize the results. Based on measuring different variables and the outcome, one can accept or reject proposed hypotheses. In our study we seek to find characteristics in systems by following the strategy shown in Figure 3.

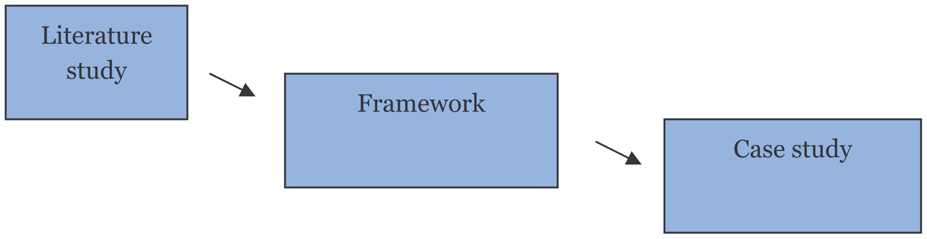

Figure 3: Choice of methods

A literature study is necessary to get an overview of protective measures available for a client environment and how to detect if they are present. We need to understand the threats an application running in an unknown environment is exposed to and how to mitigate these threats with the use of different protection techniques. Our sources for the literature study have been mainly the Internet, online article databases and the library at Gjøvik University College. Based on the literature study we define a framework for comparing different systems. The defined framework is then applied to two constructed systems in the case study. Implementing a demo application/prototype of the system we found to be too time-consuming - so we have decided to only construct two possible models.

3 RELATED WORK

In this chapter we look at related work and results that are relevant for this study. We begin by giving an overview and how the main subject areas relate to the listed research questions in chapter 1.4.

3.1 Introduction

We have identified three main subject areas relevant for this study:

- *Protective applications and checklists* – This will give us knowledge about what and how to search for protective measures running on the client. This is related to research question no. 1 and is covered in chapter 3.2.

- *Mobile Agent Systems* – From this area we will find the threats related to the application that runs on the client and performs the analysis of the client environment. This is related to research question no. 2 and is covered in chapter 3.3.

- *Software protection* – This is the contrast to the previous area. Techniques for protecting our software will help us say something about what level of trust we can have to the data received from the analysis. This subject is also related to research question no. 2 and is covered in chapters 3.4 and 3.5.

3.2 Environment analysis

In the following chapters we describe, and how to search for, protective measures present in the client environment.

3.2.1 Data categories

In order to analyze and say anything about an environment, we need data describing it. There are many items one can check when doing an environment analysis and we have to in advance know and decide what to look for.

In CERT's guide [35], we find items to analyze when determining how robust our own system is to withstand attacks and to detect malicious acts. When attempting to analyze a remote environment many of the same items are relevant. These are listed in Table 1.

Data Category	Types of data to collect
Process performance	• system and user processes • services executing at any given time
Other process data	• user executing the process • process start-up time, arguments, file names • process exit status, time, duration, resources consumed • the means by which each process is normally initiated (administrator, other users, other programs or processes), with what authorization and privileges • devices used by specific processes • files currently open by specific processes
Files and directories	• list of files, directories, attributes • cryptographic checksums for all files and directories • accesses (open, create, modify, execute, delete), time, date • changes to sizes, contents, protections, types, locations • changes to access control lists on system tools • additions and deletions of files and directories • results of virus scanning
Log files	• results of scanning, filtering, and reducing log file contents • checks for log file consistency (increasing file size over time, use of consecutive, increasing time stamps with no gaps)
Vulnerabilities	• results of vulnerability scanners (presence of known vulnerabilities) • vulnerability patch logging

Table 1: Data Categories and Types of Data to collect (adapted list from CERT [35])

3.2.2 Protective applications and tools

Several tools exist for doing a security analysis of an environment. Looking at what data and how these tools scan the environment will give us further knowledge of what is relevant in checking. Below are some examples of tools that analyze client environments for insecure configurations, lack of service packs and hot fixes etc. We summarize what items the tools look for, and what methods they use.

- *Microsoft Baseline Security Analyzer 2.0 (MBSA) [28]* - This tool determine the security state in accordance with Microsoft security recommendations. MBSA detect common security misconfigurations and missing security updates.

- *Windows OneCare Live (Beta) [29]* - This is an “all-in-one” protection and maintenance tool. It contains anti-virus, firewall and anti-spyware. In addition the tool does a monthly tune-up of the computer.

- *Windows AntiSpyware (Beta) [30]* - Helps protect users from spyware and other potentially unwanted software. It is guarding more than 50 ways spyware can enter the system.

- *WinTasks Professional [31]* - WinTasks provides a detailed description of each process that is running.

- *Lavasoft Ad-Aware [32]* - Provides protection from spyware and other threats; known data-mining, aggressive advertising, parasites, scumware, Trojans, dialers, malware, browser hijackers, and tracking components.

- *Spybot – Search and Destroy [33]* - Detects and remove spyware of different kinds from the system.

- *Trend Micro PC-cillin Internet Security [34]* - Detects and removes viruses, worms, Trojans, and spyware. In additions it includes a firewall, spam filtering, and vulnerability assessment among other things.

The different kinds of services that the applications and tools mentioned above perform can be categorized in:

- Malware (anti-virus, anti-spyware etc.)
- Scan current patch-level
- Security configuration settings
- Analyze running applications/processes

There are different ways to search for and detect vulnerabilities. By looking at the tools listed above, we get the following list:

- Analyze running processes and process modules
- Search for and analyze registry keys
- Search for files and folders
- Analyze data streams
- Analyze memory

3.2.3 Windows Security Center

Service Pack 2 for Microsoft Windows XP introduces the Windows Security Center [42]. Here, status about firewall, virus protection and Automatic Updates are collected and monitored in one place. The interesting part is how the Windows Security Center gathers data about products. The *manual* approach is to search for registry keys and files that the different products install. The *automatic* is by using the Windows Management Instrumentation (WMI) [38]. WMI is a component in the Windows operating system that provides access to information about different objects. WMI can be used to query and set information on applications, networks, and other components. In this way third-part applications can report their status to the Windows Security Center through WMI. The Windows Security Center tries to determine the following:

- Whether an antivirus program is present

- If the antivirus signatures are up-to-date
- Real-time scanning or on-access scanning is turned on for antivirus programs
- Whether a firewall is installed and whether the firewall is turned on or not
- Status of Automatic Updates

In our system we want a similar way of collecting data about protective measures and report the status back to the server.

3.3 Executing environment

In the following chapters we describe the potential threat that the executing environment is to the running application.

3.3.1 Malicious host attacks

One area where protection of the application is crucial is *Mobile Agent Systems* [1, 2, 3]. The main components of Mobile Agent Systems are agents and hosts. Hosts offer the runtime environment and agents are entities that consists of code, data and control information. In these systems agents move between hosts autonomously. A lot of research has been done in this area; threats against mobile agents and mobile code. When looking at threats against our system, we base it on work from this area.

Mobile Agent Systems are exposed to the *problem of malicious* hosts. Hohl [1] explains malicious host as:

> "A party that is able to execute an agent that belongs to another party and that tries to attack that agent in some way."

We have to assume that the agents, or more generally any application, will be exposed to *white-box attacks*. The application is subject to attacks originating from the operating system, other software, the hardware etc. This pessimistic threat model is discussed in both [14] and [15].

To illustrate the problem of malicious hosts we include an example with a Mobile Travel Agent [2]:

> "A Mobile Travel Agent is sent out by a user to visit several airlines, find the best offer and book and pay the best flight [...]. A malicious host might spy out the price limits set by the user and the offers by competitors. It might tamper the agent to make the agent falsely believe that the host has the best offer. It might steal the mobile agent's electronic money, credit card number or cryptographic keys."

Security issues for the agents in Mobile Agent Systems described in [1, 4, 5] will be a good basis for the studying of our problem. Table 2 lists possible attacks in Mobil Agent Systems as identified by Hohl [1].

Attack	**Description**
Spying out code	The code of the application has to be readable by the host in order to execute it. Knowing the code leads to knowledge about the execution strategy of the application.
Spying out data	This is reading the private data of the application.
Spying out control flow	By knowing the entire code of the application and its data, the host can determine the next executing step at any time.
Spying out interaction with other agents	The host watches the interaction between two agents (applications).
Manipulation of code	A host can normally modify the program of an application since it has access to the code memory.
Manipulation of data	The host modifies data in memory.
Manipulation of control flow	The host can conduct the behavior of the application by manipulating the control flow.
Incorrect execution of code	A host alters the way it executes code resulting in the same effects as above.
Returning wrong results of system calls	For example returning wrong IP address from a system call that returns the host's current IP address.
Manipulation of interaction with other agents	The host manipulates the interaction between two applications.
Masquerade	The host spoofs its identity.
Denial of execution	The host does not execute the application.

Table 2: Possible attacks on running applications [1]

This shows that in worst-case, we have no control with the host that runs our application. The platform can be maintained by adversaries and built for tampering and analyzing running applications.

3.3.2 Threat model categories

Aucsmith [17] defines three threat model categories.

- *Category 1* - This is the normal "hacker attack". The adversary is on the outside of the PC, and has to attack via the communication protocols and access controls in use.

- *Category 2* - Malicious code running on the platform, e.g. virus and Trojan horse attacks, belongs in this category.

- *Category* 3 - In this category the adversary has complete control of the system. Software and hardware may be substituted at wish. Technical expertise and financial resources are the only limitations for the adversary.

Category 3 is further divided in three subcategories.

a) No special analysis tools required
b) Specialized software analysis tools required
c) Specialized hardware analysis tools required

Available resources are a main factor when securing and attacking a system. These categories help us state our threat model. We will use these categories in connection with the framework.

3.4 Software protection

In the following chapters we look at software protection techniques and methods.

3.4.1 Code obfuscation

Reverse engineering [6] is the process of decompiling the executable back to readable source code. Given enough time an resources, a professional attacker will always be able to reverse engineer and analyze any application. With the introduction if intermediate languages like the Java byte code [7] and the Microsoft .NET Intermediate Language (MSIL) [8], more information about the application is distributed with the executable or compiled files. This way the reverse engineering process has become easier for an attacker. Reverse engineering is typically the first step in an attack where knowledge about the inner workings of an application is helpful in performing further attacks.

Collberg et al says in [9] that *code obfuscation* is the most viable method for preventing reverse engineering. That was written in 1997, but today code obfuscations are still the best way of raising the bar for reverse engineering.

"**A definition is given as [9]:** *Let P → P' be a transformation of a source program P into a target program P'. P → P' is an obfuscating transformation, if P and P' have the same observable behavior. More precisely, in order for P → P' to be a legal obfuscating transformation the following conditions must hold:*

- *If P fails to terminate or terminates with an error condition, then P' may or may not terminate*
- *Otherwise, P' must terminate and produce the same output as P*"

In [10] we find an explanation of code obfuscations:

"Semantics-preserving code transformations used to protect a program from reverse engineering."

There are different ways of performing code transformations. In [9, 11] we find a four-way classification:

- Layout Obfuscations
- Data Obfuscations
- Control Obfuscations
- Preventive Transformations.

This classification is based on what kind of information the code transformation is targeting and how it affects the target.

Layout obfuscations alter information that is not needed for the application to execute correctly. This includes

- *Scramble identifiers*. Names on e.g. variables are useful for an attacker to understand the program. By changing '_toatalAmount' to 'abc', the bar is raised.
- *Change formatting*. This is removing source code formatting information.
- *Remove comments*. Comments and debugging information are useful for an attacker, but not for the application to execute properly.

Data obfuscation focuses on data and data structures in the application.

- *Storage and encoding*.
    - *Split variables*. Split a variable x into two other variables y and z.
    - *Promote scalars to objects*. For example promote an integer value to an integer object.
    - *Convert static data to procedure*. For example a static string can be converted into a program that produces that string.
    - *Change encoding*. An integer variable x can be replaced by x' = y * x + z, where y and z are constants.
    - *Change variable lifetime*. For example a local variable can be changed to be a global variable.
- *Aggregation*.
    - *Merge scalar variables*. Two 32-bit integers could be merged into on 64-bit integer.
    - *Modify inheritance relations*. For example create a parent class for two other classes that has some methods and instance variables from both.
    - *Split, fold or merge arrays*. For example change the number of dimensions in an array.
- *Ordering*. Programmers structure their code for maximizing the readability. By reordering items, two items that logically belongs to each other is now scattered around.
    - *Reorder instance variables*.
    - *Reorder methods*.
    - *Reorder arrays*.

With *control obfuscation* the goal is to disguise the flow of the program. This can be done in several ways:

- *Aggregation*. Related code is often put together when programming. The objective here is to break this logic.
    - *Inline method*. Replace method calls with the actual code.
    - *Outline statements*. Make parts of code a separate method.
    - *Clone methods*. Clone methods, but each time with small differences and let the calling method arbitrarily choose one of the copies.
    - *Loop transformations*. Loop blocking, unrolling and fission.

- *Ordering*. When programming, related code is often placed together to simplify reading and maintaining the code. This can be obfuscated by:
    - *Reorder statements*.
    - *Reorder loops*.
    - *Reorder expressions*.
- *Computations*. Here we modify the real control flow of the program. We can do this by:
    - Insert *dead or irrelevant code*.
    - *Reducible to non-reducible flow graphs*. Utilize that e.g. the assembly language is more extended than the higher-level language. In this way there is no high-level code mapping directly to the native instruction.
    - *Extend loop condition*. Make the termination condition more complex.
    - *Table interpretation*. Build a simplified virtual machine and let the real code be interpreted by this.

Preventive transformation exploits weaknesses in de-compilers and de-obfuscators, not the reverse engineering itself.

- *Targeted*. Utilize weaknesses in current tools.
- *Inherent*. Explore inherent problems with known de-obfuscation techniques.

An example of code obfuscation and Microsoft .NET Framework can be found in [12]. A more theoretical publication is Wroblewski's [13].

3.4.2 Software tamper resistance

The goal with *software tamper resistance* is to detect integrity violations of original software [14]. Different approaches have been proposed [17-20]. We look at some key factors from these later.

To prevent modification of our software, we can add tamper-proofing code to our application. This code should be stealth, redundant, and [16]:

a) Detect if the application is modified, and
b) Cause the application to fail and stop executing if modifications are detected

There are three principal ways to detect tampering [16]:

- Examine the executable program itself to see if it is identical to the original one.
- *Result (program) checking*, i.e. examine the validity of intermediate results produced by the application.
- Generate the executable on the fly. Minor changes to the generating program will hopefully produce code that cannot be executed.

Aucsmith [17] presents architecture for tamper resistance by introducing *Integrity Verification Kernels (IVK)*. The IVKs communicate with each other to create an *Interlocking Trust Model*. The architecture assumes a *System Integrity Program* running on the computer that is available to all programs. The IVKs are responsible for the integrity of the application which it is embedded. Since all IVKs are dependent on each other, tampering with one program means having to compromise all programs. This is illustrated in Figure 4.

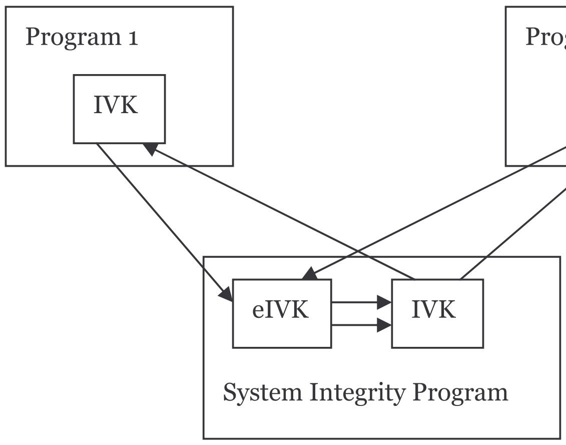

Figure 4: Interlocking Trust Overview [17]

The System Integrity Program has a special IVK called the Entry Integrity Verification Kernel (eIVK). The eIVK has a published, external interface that can be called by any other IVK using the *Integrity Verification Protocol*. The eIVK has a known entry point and a known public key.

A similar approach, with the use of *guards,* is proposed in [18]. The insertion of the protective code is made automated which is important when it comes to extended use of tamper resistance techniques.

In [19] the execution trace of the code is hashed and compared to a fingerprint. The technique is denoted *Oblivious Hashing*. The execution trace contains information about machine instructions and memory references. This technique can also be used for remote code authentication.

3.4.3 Software diversity

The idea with *software diversity* can be compared to genetic diversity [14]. Because of genetic diversity, no single virus or disease wipes out an entire species. Small differences make an attack on one item not possible or with reduced effect on another.

Today, the trend within software is towards homogeneity. There is a very small number in different browsers, operating systems, office applications etc, so the value of software diversity reduces. By introducing software (code) obfuscation, it is possible to a higher degree to utilize the diversity. Instead of making one new application which is equivalent to the original, make several copies instead. Diversity is discussed in [37].

This doesn't raise the degree of difficulty for tampering with one instance, but an automated tool is more difficult to make because of the small differences between all instances.

3.5 Hardware solutions

Because of issues such as the hostile environment, white-box attacks and so on, software-only security solutions have its limitations. The proposed solutions are based on trusted hardware.

3.5.1 Integrity chain

Arbaugh [21] describes the integrity "chain". Systems are organized as layers. Higher layers assume the integrity of lower layers. Since hardware is the lowest layer, the integrity "chain" initially is based on trusted hardware. Transitions to a higher layer occur only after integrity checks are complete and checked on the lower layer.

3.5.2 Execute-only memory

Another possibility for hardware-supported security against tamper-resistance is *execute-only memory* (XOM) as studied by Lie et al [22]. XOM allows instructions stored in memory to be executed, but not otherwise manipulated.

3.5.3 Generic trusted hardware platform (TCG and NGSCB)

Trusted Computing Group (TCG) [23] and Next-Generation Secure Computing Base (NGSCB) from Microsoft [24] are two initiatives in enhancing system security based on a generic trusted hardware platform. Even though the two initiatives are separate, the functionality is similar.

The Trusted Platform Module (TPM) is the trusted root of the system. Figure 5 shows how applications interact with the TPM [36].

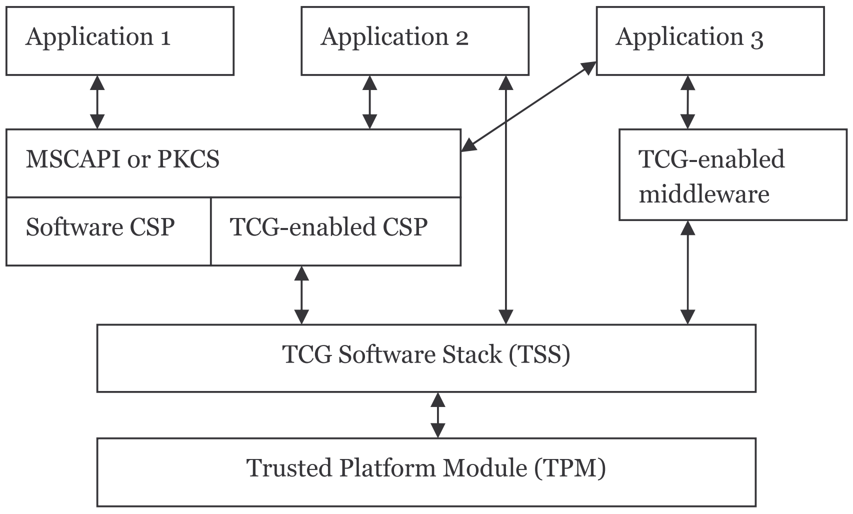

Figure 5: TCG Application architecture [36]

TCG Software Stack (TSS) is the software interface to the TPM. There are different ways that applications can utilize the TPM functionality. Applications can call the TCG-enabled Cryptographic Service Provider (CSP) through an interface confirming to the Microsoft Crypto Application Library (MSCAPI) or the Public Key Cryptography Standard (PKCS). Other options are to interact with the TSS directly, or via TCG-enabled middleware. Such middleware may be software development kits provided by third-parties. Different services are available with the different options.

The most important services of the TCG specifications are [25]:

- Hardware storage for cryptographic keys
- Secure booting
- (Remote) Platform Attestation
- Sealing

In our case the remote attestation is the most interesting feature.

> “Attestation aims to allow 'unauthorized' changes to software to be detected [25].”

This allows for mutual authentication of platforms or applications and the possibility of convincing a third party that you are running some specific configuration, and nothing else. This can even be done remotely, called ‘remote attestation’.

4 CLIENT ANALYZER

In this chapter we give an overview of the system. Our intentions are not to develop a new type of analysis tool. We will base our analysis on already existing tools and other protective applications. Our client analyzer will gather data from the client environment, and based on these data say something about what protective measures are active on the client. These results will be used to assess the level of robustness.

4.1 Objectives

The purpose with the system is to analyze the client environment prior to communicating business related and sensitive data. The objective with the analysis is to say something about how secure or trustworthy the client environment is. It is not intended to solve the problem with insecure computers, meaning installing any software or do any reconfiguration of the client system. This must be handled in other ways. The service operator has to define a set of requirements that the client environment has to fulfill before the business process begins. The system will collect the data and based on these accept or reject further communications.

4.2 Architecture and data flow

The overview of the system is shown in Figure 6. The grey box indicates the client analyzer and the red arrows symbol the host environment attacking the application.

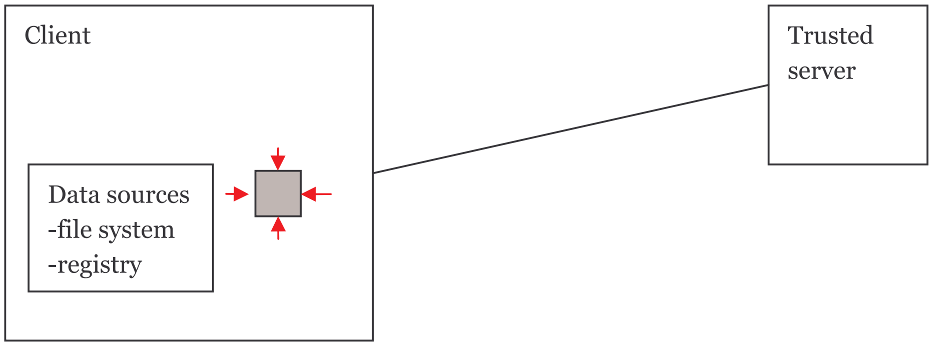

Figure 6: Overview of client analyzer in context

Figure 7 shows a flowchart for the relevant data in the system. The client analyzer collects data from the client resources, and then the data collected are sent back to the trusted server.

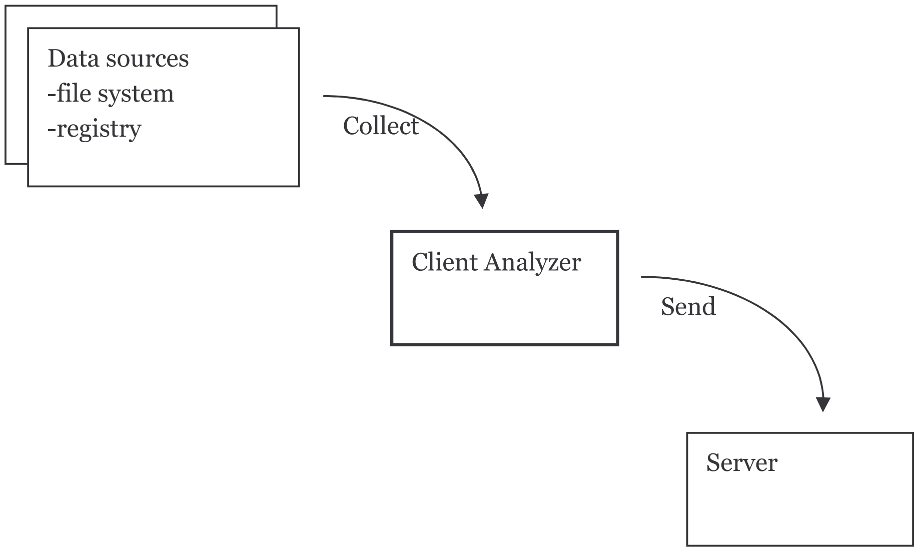

Figure 7: Flowchart of data about client environment

The ultimate goal for the attacking party is to alter the data in some way, so that the server assumes a robust and friendly environment when in reality the client environment is, for instance, infected by malicious applications. The attack can be directed to any of the components and communication channels of the system.

4.3 Data sources

Here we will look at the different resources on the client environment that we will use for gathering data in the environment analysis. We have limited our data sources to:

- File system
- Registry
- Running processes

File system - Different versions of products have different files installed. The files have different name, timestamp and version number. By knowing which files we can search the file system and check if they exist or not. This will indicate if the products are installed, but less whether they are running and configured correctly. We may also look for known signature-files used by e.g. anti-virus applications or log files that keep information about status of the tests that the applications perform.

Registry - As with the file system, we can search the registry for known registry keys added by the different products.

Running processes - Some protective applications run constantly, e.g. anti-virus applications. By searching the process list we can search for running protective applications.

Table 3 shows a summary of the data sources.

Data source	**Method**	**Information**
File system	Search for known files.	• Indicate if product is installed. • If latest files are in use.
Registry	Search for known registry keys.	• Indicate if product is installed.
Running processes	System call to get list of running processes	• Which applications are running

Table 3: Summary data sources on client environment

4.4 Usage scenario

Here we describe the steps when a user logs on and wish to use the services provided.

1. The business application sends on behalf of the user an initiating request to the server. This can be the web-browser requesting the login page of the service.

2. The server checks if the client has recently been analyzed and the results.

3. If the test pass, the communication will proceed as normal and the execution will jump to step 12.

4. If the test fails, the server will return to the business application that a validation is required.

5. The business application will launch the client analyzer.

6. Based on the demands and requirements from the server, the client analyzer gathers data from the client environment.

7. The client analyzer wraps up the gathered data, encrypts and signs the message, and sends the message back to the server.

8. The client analyzer notifies the business application that the analysis is done.

9. The business application makes a second request.

10. If the test passes, the execution will jump to step 12.

11. If the test fails, the server will reject further communication within a fixed timeframe. The user is displayed with a message indicating what is wrong and how to correct it.

12. The start page for the service is displayed to the user.

Figure 8 shows a sequence chart of the steps when validation is required and the test passes.

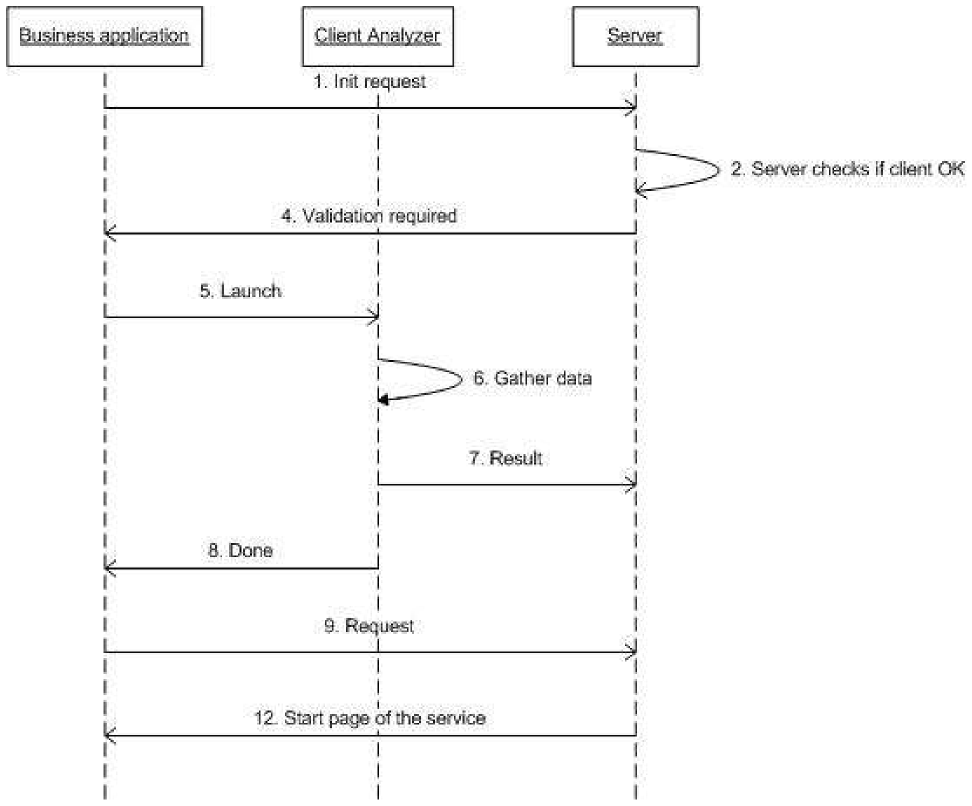

Figure 8: Sequence chart of client analyzer in use

4.5 The Stakeholders

Here we describe the stakeholders of the system – the provider and the end user. These two are known from the business model.

4.5.1 The End User

From the end users perspective, ease of use and low cost are the most important when rating if a service is practical or bothersome. If no one uses your services, you will go out of business. When designing a system one should strive for making it as user friendly as possible. Technical requirements should be kept to a minimum. If you deliver services to the business market, you have the chance of setting more requirements. For instance, you can limit your banking application to run on one specific operating system. If the public are your customers, the system must be based on what is *normal* of both hardware and software. If the system requires any special modules, you as the provider have to make them easily available.

Privacy is just as important as requirements on software, hardware and others. There are two ways of doing the analyses as prepared for in this system. We can gather all data needed, do the analyses, and send the result back to the server. In this case little data about the client environment leaves the client. In the second case we can gather and send all the data back to the server. Then it is up to the server to do the analyses and get the result. In the latter the server side gets a lot of information about the software and hardware configuration on the client side. This might not be widely accepted by end users.

4.5.2 The Service Provider

The service provider's focus is, as any other business, on profits. In the long run you need trust and a positive reputation in the market to achieve good results. For instance, incidents such as information disclosures are unwanted. For security critical systems, the provider sets requirements for the client environment. In our case, where we want the general public as our end users, such requirements can not be too strict.

In the online newspaper article [40] we read how malicious software on the client machine hijacks the communication and redirects to another site than first intended. Such acts might be done by competitors. If end users are unsure of the state of their system or how to perform the necessary tasks to get full control, examples like in the article might lead to acceptance by end users of an analysis performed by the service provider. This way the end user helps the service provider by accepting the analysis.

4.6 Practical considerations

We have identified three practical issues:

- Response time
- Size
- Access rights

Response time is probably the most important issue when it comes to user friendliness. If the analysis process takes *ages* to perform, the system will not be accepted by the users. The size of the application might also play a role regarding response time. On a slow connection, e.g. 9.6kbps through a cellular phone, even checking for new mail takes *to long*. Introducing an application that is slow to download, but necessary for getting access to the internet banking services, will not be very welcome.

When doing an environment analysis full access to the system is probably needed. Because of viruses and other malicious code, runtimes limits to a minimum what an outside application can do. In addition, users are requested to use a normal user account, and not full administrator privileges when not needed. A system like the client analyzer might be seen as a risk to the client environment.

4.7 Challenges

There are several challenges when analyzing a client environment where we do not have full control. The first, and most obvious challenge, is that the only place data about the environment is available, is on the client itself. How can we trust the data collected from the environment? We need methods for assessing the level of trustworthiness of our collected data. Below we list other challenges when analyzing a client environment. Some of the challenges go a bit wider than just the system we are studying. We do this to take into account a real life implementation.

Platform - The client environment can be of different platforms. Windows, UNIX, the different Linux alternatives are just a few examples. How we analyze an environment will vary with which platform the client is running. Different levels of security are built into different platforms, and different analyzer applications must be developed for the different platforms. In addition, there is a variation in the malicious activity against different devices and platforms. Another thing is; with stationary clients on internal networks we may assume more control with the environment. Security policies are defined and applied, physical security is increased, among other factors. With home computers and laptops not necessarily all these things are in place.

Ownership - It's impossible to say if a client environment is secure if we haven't defined secure in advance. If we own the client environment, i.e. the machine is on our internal network or a corporate laptop; we know how the client environment is configured etc. This way we know what to look for in a security analysis. If the client is unknown, the complexity of this task increases. All we know is that somewhere on the internet, someone has downloaded our client software and now wants to communicate with our server. This can be e.g. in a shopping scenario or Internet bank. We have no knowledge about the user or the machine.

Environment - When doing a security analysis of a client environment, we can in reality only do this on a technical level. Since security is not only technical, but also non-technical procedures, the users' attitude and behavior etc., we only get one part of the truth. A client environment analysis says nothing about the surroundings of the environment. The biggest threat to systems today, is not platforms or applications, but its users. When doing a technical security analysis, we don't get any data or information on the users controlling the mouse and keyboard.

Up-to-date scanning - Anti-virus, anti-spy ware etc. all uses input or definition files so that the scanning engine has the most recent data about the threats it's trying to detect. This is a necessity for getting the most correct security analysis. New security breaches are discovered continuously and one successful attack is all it takes to do the worst damage to the system. This leads to frequent changes in protective applications and measures running on the client. When scanning a client environment for protective measures in use, the scanning application has to be aware of all these changes so that the results reflect the situation on the client as accurate as possible.

Configuration - Does the anti-virus software scan the environment with the latest definition files? Does the anti-virus run at all? By installing an anti-virus application, the possibility for virus infections decreases, but the behavior of the users are just as important. We can find protective applications installed, but they could be de-activated, run with old configuration and/or signature files etc. Most applications can only detect known threats to a system. If a malicious application rapidly changes it will be harder to detect. We need a way to communicate with applications similar to the way described in chapter 3.2.3 with Windows Management Instrumentation and Windows Security Center. This makes it more complex and involving a collaboration of lots of vendors.

"The other way around" – There are two ways of performing a remote security analysis. The first is to let the client do a security analysis of its own environment and then send the results back to the server. The second way is to let the server do the analysis on the basis of raw data the client sends back. In this way the server gets information about installed software, how software is configured, what security measures are in place etc. Now the roles are turned, and the security scan from the server has become a threat to the client. This is a possible way for rogue servers to collect information about client environments in connection with a *bigger* attack.

Runtime - When designing a new runtime environment or new application platform, a lot of resources are spent on implementing security measures that will protect the host running the application. Two examples of such platforms are the before mentioned .NET from Microsoft and Java from Sun. These platforms include security measures like the sandboxing-technique, code access security, code evidence or software signing etc. These help the user on the host to verify the authenticity of the application and have more control with what the application can do. These techniques make it more difficult to do a remote analysis, since this requires wide access to the system. The thought of having an application from a service provider you partially trust scanning the system may not be widely accepted.

Host vs. network – A firewall can be placed both on the host and as a network firewall. If we cannot find a firewall on the host, the host might be on a corporate network with a network firewall installed. This and similar scenarios will cause a challenge for the security analysis. The outcome of the analysis of the client environment will be negative, even if the protective measure is present to some extent.

5 FRAMEWORK

In this chapter we describe the framework.

5.1 Introduction

The purpose of the framework is to be able to compare two systems in regards to the threats against the client analyzer and the process of analyzing the client environment. To document the threats we use threat modeling. The threat modeling process will be based on the technique presented by Howard and LeBlanc [27].

5.2 Documenting the threats

The main goal for an attacker is to be able to alter the result to an "All security checks passed" when this is not true. The attacker could be either a user with physical access to the client system, or it could be a malicious application running alongside. The malicious application's interests could be the business or other sensitive data, attempting a denial-of-service attack and so on. If the malicious application were detected; we can choose to reject further communication, or the client analyzer could warn the user, among other things.

We begin by decomposing our abstract system from chapter 4. Figure 9 shows a data flow diagram of the system. The dotted lines indicate the boundary for the client analyzer application.

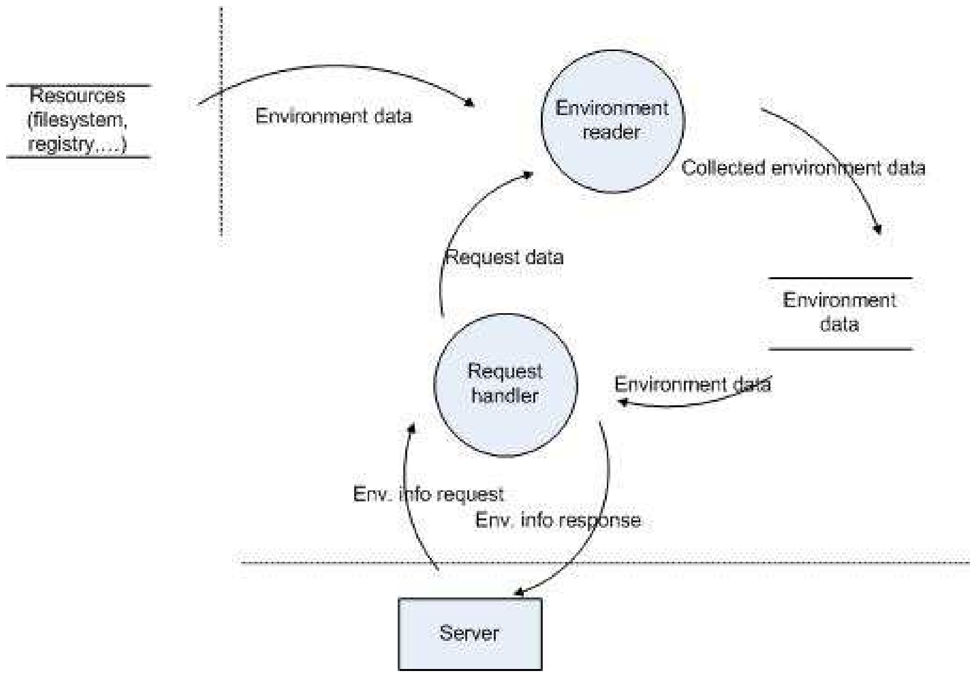

Figure 9: Data flow diagram of the client analyzer

We have separated the client analyzer application in two processes; the *environment reader* and the *request handler*. We do this to better make visible the storage of environment data in memory.

An asset is a resource of value. This is typical data or a process. The decomposition gives us the following assets listed in Table 4.

Asset	**Description**
Environment data	This is data collected from the client environment, e.g. file system information, data from the registry etc.
Request handler	This process handles the communication with the server and other parts of the client analyzer.
Environment reader	This is the part of the client analyzer that collects data from the client environment.
Environment info response	This is the communication back to the server.

Table 4: Assets

Our main asset in this system is “data about the client environment”. This includes its correctness and quality. By correctness we mean whether the data is modified to affect the analysis results, i.e. its integrity. Quality can be compared to validity; “How well does the collected data describe reality?” Our main challenge with this data is that it is not under our control, but still we have to base our analysis on it. We described this challenge in chapter 4.7. Protecting data in process requires typically protecting the software processing it.

A threat is a potential occurrence that might damage or compromise the assets. There are two things interesting for an adversary to attack:

1. The results from the analysis, and
2. How the application did the analysis.

With knowledge of the analysis results and how these are stored, the adversary has a greater chance of modifying them if desirable. The results are stored within the trust boundary of the client analyzer application. Since the client analyzer application is analyzing the surrounding environment, it has to communicate with the environment in some way. By knowing what external modules or components the application is communicating with, it will give the adversary knowledge about the entry points for the application.

We use the list of malicious host attacks presented in 3.3.1 as attack techniques when identifying threats to the client analyzer. The threats are shown in Table 5 and Table 6.

Title	Modify data about client environment
Description	The adversary or malicious application can modify the data about the environment and in this way affect the result of the analysis.
Asset	Environment data
Attack tree	1. Modify environment data 1.1. Modify data outside application boundary 1.1.1. Access for modifying resources on client environment (AND) 1.1.2. Knowledge about what data to modify 1.2. Modify data inside application boundary 1.2.1. Access for modifying storage area in memory (AND) 1.2.2. Knowledge about where in memory data is stored 1.3. Wrong results from system calls 1.3.1. Modify runtime/execution environment

Table 5: Threat #1 Modify data about client environment

Title	Bypass environment analysis
Description	The adversary can patch around the procedure that does the environment analysis.
Asset	Environment reader/request handler
Attack tree	1. Patch around environment analysis 1.1. Modify executable 1.1.1. Knowledge about where to modify (AND) 1.1.2. Knowledge about how to modify 1.2. Modify code in-memory 1.2.1. Knowledge about where in memory to modify (AND) 1.2.2. Knowledge about how to modify 1.3. Incorrect execution 1.3.1. Modify runtime/execution environment

Table 6: Threat #2 Bypass environment analysis

We have chosen to leave out the communication between the client and the server in this version of the framework. The communication part is in no way less important, but because of time limitations and security in communication being a big research area, we have made this choice. The fourth asset *Environment info response* will therefore not be included.

5.3 Results

In Table 7 we arrange the described threats from the previous chapter in a tabular form making the basis of comparison for our case study.

Attack	**Assessing level of robustness**
Modify data outside application boundary	
Access for modifying resources on client environment	Degree of difficulty for accessing and modifying the resources where the client analyzer gathers data.
Knowledge about what data to modify	Degree of difficulty for obtaining information about what to modify of client resources to fool the tests.
Modify data inside application boundary	
Access for modifying storage area in memory	Degree of difficulty for accessing and modifying the memory where the client analyzer stores data.
Knowledge about where in memory data is stored	Degree of difficulty to find out where the client analyzer stores data in memory
Modify executable	
Knowledge about where to modify	Degree of difficulty to find out where in the executable to modify code to bypass the environment analysis.
Knowledge about how to modify	Degree of difficulty for modifying the executable to bypass the environment analysis.
Modify code in-memory	
Knowledge about where in memory to modify	Degree of difficulty to find out where the code instructions for the client analyzer are stored in memory.
Knowledge about how to modify	Degree of difficulty for accessing and modifying the code instructions stored in memory for the client analyzer.
Incorrect execution/wrong results system calls	
Modify runtime/execution environment	Degree of difficulty for modifying the runtime for the client analyzer.

Table 7: Framework - Basis of comparison

When assessing the level of robustness for each attack we propose the rating scale shown in Table 8.

Rating	**Degree of difficulty**
Low	The attack can be performed by a novice adversary or a tool may be developed to automate the attack.
Medium	The attack requires a skilled adversary, but no special tools.
High	To perform the attack requires in-depth knowledge and specialized tools.

Table 8: Rating scale

6 CASE STUDY

In this chapter we describe the two system models and apply the framework from the previous chapter.

6.1 System descriptions

In the following chapters we specify the two system models. We denote the two systems *Client Analyzer A* and *Client Analyzer B*. The models differ in architecture, but solve the same task. The greatest difference is the protection technique in use. The first system model will have software only protection techniques implemented, the second will be based on trusted hardware. Software only protection is inexpensive and easy to use. Many development environments have software protection tools incorporated. Hardware is relative to this costly and deployment of the system gets a bit more extensive. In addition to the software, a hardware device must be distributed to every user, or all users require having a general purpose trusted hardware installed.

6.1.1 Client Analyzer A

We begin with the system model of Client Analyzer A. This system is meant to reflect a standard hardware and software configuration available today. The architecture on the client side of the system is shown in Figure 10.

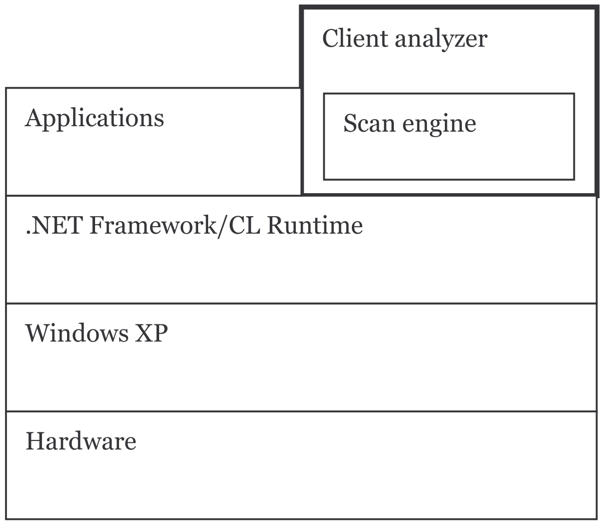

Figure 10: Client Analyzer A in context

The system has a standard hardware configuration with no special requirements. The operating system in client analyzer A is Microsoft Windows XP. The runtime for our application is the .NET Framework. Our client analyzer is build as a .NET application. The client analyzer behaves and interacts as described in chapter 4.4 “Usage scenario”.

We assume the use of code obfuscation as software protection technique for the client analyzer. Code obfuscation is available as standard in the .NET Framework development environment [26].

6.1.2 Client Analyzer B

In the system model for Client Analyzer B we have in addition to the standard hardware configuration a trusted hardware device. The architecture of system B is shown in Figure 11. Client Analyzer B is based on the description and references in chapter 3.5.3.

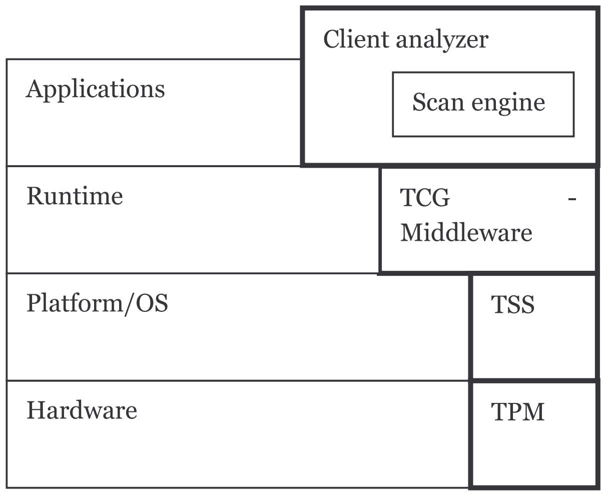

Figure 11: Client Analyzer B in context

The platform of system B is a TPM enabled architecture. We assume that algorithms implemented in the hardware are correct. Otherwise, this would be a vulnerability introduced to the system. The client analyzer communicates with the trusted platform module through the TCG-enabled middleware and the TCG Software Stack (TSS).

6.1.3 Technical comparison

Here we give a technical comparison of the two Client Analyzer systems on hardware, platform and the Client Analyzer application.

	Client Analyzer A	**Client Analyzer B**
Hardware	• Standard hardware configuration • No special requirements	• TPM enabled architecture
Platform/ operating system	• Microsoft Windows XP • .NET Framework	• TPM enabled operating system • TCG-enabled middleware
Client Analyzer application	• Client analyzer is build as a .NET application • Code obfuscation	• TPM-enabled application

Table 9: Technical comparison - Client Analyzer A & B

6.2 Applying the framework

In this chapter we will compare the two systems based on the defined framework. The result is shown in Table 10.

	Client Analyzer A	**Client Analyzer B**[1]
Modify data outside application boundary		
Access for modifying resources on client environment	-Low- Client resources will always be accessible for the administrator of the client system.	NN
Knowledge about what data to modify	-Low- By auditing the environment while the client analyzer is run, a list of data to modify can be easily obtained. This can be posted on the Internet and we have to assume easy available.	NN
Modify data inside application boundary		
Access for modifying storage area in memory	-High- Here we assume encrypted data. The timeframe to attack between plain text and encrypted text should be too small for being an easy attack point.	NN

[1] See chapter 6.3 for why not completed.

Knowledge about where in memory data is stored	-Medium- This requires searching the memory and appropriate tools.	NN
Modify executable		
Knowledge about where to modify	-Medium- We rely on code obfuscation to prevent reverse engineering. The level depends on the complexity of the code obfuscation process.	NN
Knowledge about how to modify	-Low- With some effort a novice programmer can patch around the test if the source code is available in a readable and understandable format.	NN
Modify code in-memory		
Knowledge about where in memory to modify	-High- This requires in-depth knowledge and tools.	NN
Knowledge about how to modify	-High- This requires in-depth knowledge and tools.	NN
Incorrect execution/wrong results system calls		
Modify runtime/execution environment	-High- This requires in-depth knowledge of the whole runtime and the interaction between runtime and the platform.	NN

Table 10: Framework applied to Client Analyzer A & B

6.3 Validating the framework

When applying the framework to our two system models, we came across different issues and challenges. In this chapter we review our approach.

The framework was developed based on an abstract design of a traditional application running in an unknown environment, very similar to Client Analyzer A. With the low-level details in the framework, this makes it difficult to generalize and use on two fundamental different systems as in our case study. Systems with security enforced in hardware needs its own framework for comparison or the framework must be at a more general level. Therefore we find this framework not suitable for Client Analyzer B. In a real life implementation the business application itself will be TPM-enabled and utilizing the advantages of a trusted hardware module.

If we assume a threat model with no specialized software or hardware tools for analyzing and modifying the application code or data, we get the impression that the system is ‘waterproof’, i.e. it can not be tampered with. Only the environment data outside the application boundary is subject for attack. We think this is a weakness with the framework. A reason for this we believe is the vagueness and subjectivity when assessing the robustness and level for each attack. The framework needs a more detailed scale for each entry where there is no doubt about where on the scale a certain system with specified characteristics belongs.

We see that the framework lacks much of the assessment of the client’s robustness against outside threats, e.g. virus, spyware etc., that one would anticipate would bee included. Our focus has mainly been on the process of gathering data and securing the software running in the unknown client environment.

We will suggest these changes in future work.

7 CONCLUSIONS

Implementing a client analyzer that does the environment analysis transparent to the user will be difficult. This mainly because it will be a third party application to the environment and will be large and complex to handle every type of client side configuration. In addition, the application will reuse results from other protective applications running in the client environment without standardized way of communicating. A *remote* and adapted version of the Windows Security Center would be our suggested solution in a homogeneous environment. This would not work in an internet banking scenario where the clients can be of different platforms.

In this study our focus has been on protection techniques. A malicious application will have difficulties in modifying the client analyzer on-the-fly. Protection techniques are more directed towards adversaries with physical access to the client environment that wants to tamper with the client analyzer. We think that it is feasible to implement a version of the client analyzer with sufficient security, but that the practical issues will the big challenge.

Because of the challenges when applying the framework to the two system models we did not get the desired results from the case study, i.e. what characterizes a more robust client analyzer. This is because our framework was too detailed and specific. If attempting to apply the framework to Client Analyzer B and assess the level of robustness, the subjectivity in the process would lead to the level *High* on every item.

The theory related to TPM-enabled architectures suggests that the business application itself is implemented as TPM-enabled. This way the application directly utilizes the security available and enforced in the TPM-enabled architecture.

8 FUTURE WORK

We propose the following for future work.

- Implement a prototype of the client analyzer and analyze what we have suggested in theory.

- A method for assessing the level of trust we can have to data gathered from the client environment.

- Extend the framework to include the client's robustness to outside attacks, e.g. virus, spyware etc.

- Incorporate into the framework the communication between the client and the server. Spoofing, man-in-the middle attacks and replay-attacks are key factors.

- Make the framework more general and objective. Define a more detailed scale for assessing the level of robustness for each attack against the running application.

REFERENCES

1. Hohl, F. 1998. Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. In *Lecture Notes in Computer Science*, 1419

2. Saeb, M., Hamza, M. and Soliman, A. 2003. Protecting Mobile Agents against Malicious Host Attacks Using Attacks Threat Diagnostic AND/OR Tree. In *Proceedings of Smart Objects Conference*.

3. Bierman, E. and Cloete, E. 2002. Classification of Malicious Host Threats in Mobil Agent Computing. *Proceedings of SAICSIT,* 141-148.

4. Hohl. F. 1998. An Approach to Solve the Problem of Malicious Hosts. Presented at the *4th ECOOP Workshop on Mobility: Secure Internet Mobile Computations*.

5. Farmer, W. M., Guttman, J. D. and Swarup, V. 1996. Security for Mobile Agents: Issues and Requirements. In *Proceedings of the 19th National Information Systems Security Conference, Baltimore*, 591-597.

6. Sommerville, I. 2001. *Software Engineering 6th Ed*. FORLAG.

7. Java Technology. Sun's Website.
[URL: http://java.sun.com/]

8. .NET Framework Developer Center. Microsoft's Website.
[URL: http://msdn.microsoft.com/netframework/]

9. Collberg, C., Thomborson, C., Low, D. 1997. A Taxonomy of Obfuscating Transformations. *Technical Report 148, Department of Computer Science, University of Auckland*.

10. Heffner, K., Collberg, C. 2004. The Obfuscation Executive. In *Lectures Notes in Computer Science*, 3225, 428-440.

11. Low, D. Protecting Java Code via Code Obfuscation.
[URL:
http://www.cs.arizona.edu/~collberg/Research/Students/DouglasLow/obfuscation.htm]

12. Masood, A. 2004. Intellectual Property Protection and Code Obfuscation.
[URL: http://www.15seconds.com/issue/040310.htm]

13. Wroblewski, G. 2002. *General Method of Program Code Obfuscation*. PhD thesis, Wroclaw University.

14. Oorschot, P. C. van. 2003. Revisiting Software Protection. In *Lecture Notes in Computer Science*, 2851, 1-13.

15. Main, A. and Oorschot P. C. van. 2003. Software Protection and Application Security: Understanding the Battleground. In *Lecture Notes in Computer Science*, (to appear).

16. Collberg, C. and Thomorson, C. 2002. Watermarking, Tamper-Proofing, and Obfuscation – Tools for Software Protection. *(IEEE) Transactions on Software Engineering*, vol. 28, no. 8, 735-746.

17. Aucsmith, D. 1997. Tamper Resistant Software: An Implementation, *Proc. 1st International Information Hiding Workshop (IHW)*, Cambridge, U.K. 1996, Springer LNCS 1174, 317-333

18. Chang, H. and Atallah, M. 2002. Protecting Software Code by Guards, *Proc. 1st ACM Workshop on Digital Rights Management (DRM 2001)*, Springer LNCS 2320, 160–175.

19. Chen, Y. et al. 2002. Oblivious Hashing: A Stealthy Software Integrity Verification Primitive, *Proc. 5th Information Hiding Workshop (IHW)*, Netherlands (October 2002), Springer LNCS 2578, 400–414.

20. Horne, B., Matheson, L., Sheehan, C. and Tarjan, R. 2002. Dynamic Self-Checking Techniques for Improved Tamper Resistance, *Proc. 1st ACM Workshop on Digital Rights Management (DRM 2001)*, Springer LNCS 2320, 141–159.

21. Arbaugh, W. A., Farber, D. J. and Smith, J. M. 1997. A Secure and Reliable Bootstrap Architecture, *Proc. 1997 IEEE Symp. Security and Privacy*, 65–71.

22. Lie, D. et al. 2000. Architectural Support for Copy and Tamper Resistant Software, *Proc. 9th International Conf. Architectural Support for Programming Languages and Operating Systems (Nov. 2000).*

23. Trusted Computing Group. [URL: http://www.trustedcomputinggroup.org/home]

24. Next-Generation Secure Computing Base (formerly Palladium), Microsoft web site.
[URL: http://www.microsoft.com/resources/ngscb/default.mspx].

25. Weiss, R. et al. 2004. TCG 1.2 – fair play with the 'Fritz' chip. Presented at the *4th International System Administration and Network Engineering Conference* (2004)

26. PreEmptive Solutions. Dotfuscator.
[URL: http://www.preemptive.com/products/dotfuscator/index.html]

27. Howard, M and LeBlanc, D. 2003. *Writing Secure Code 2nd Ed.* Microsoft Press.

28. Microsoft Baseline Security Analyzer.
[URL:
http://www.microsoft.com/technet/Security/tools/mbsahome.mspx]

29. Windows OneCare Live (Beta).
[URL: http://beta.windowsonecare.com]

30. Windows AntiSpyware (Beta).
[URL:
http://www.microsoft.com/athome/security/spyware/software/defaul
t.mspx]

31. WinTasks Professional
[URL: http://www.liutilities.com/products/wintaskspro/]

32. Lavasoft Ad-Aware
[URL: http://www.lavasoftusa.com/software/adaware/]

33. Spybot – Search and Destroy
[URL: http://www.safer-networking.org/en/index.html]

34. Trend Micro PC-cillin Internet Security
[URL: http://www.trendmicro.com/en/products/desktop/pc-cillin/evaluate/overview.htm]

35. Allen, J. 2001. The CERT Guide to System and Network Security Practices. Addison-Wesley.

36. Werhan, C. 2005. Steps for writing trusted applications. Computerworld. [URL: http://www.computerworld.com/printthis/2005/0,4814,100216,00.html]

37. Forrest, S., Somayaji, A., Ackley, D. H. 1997. Building Diverse Computer Systems. Proc. 6th Workshop on Hot Topics in Operating Systems, IEEE Computer, Society Press, pp. 67–72

38. Microsoft. Windows Management Instrumentation. [URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/anch_wmi.asp]

39. digi.no. 18.3.2005. *Nettbanken sier nei til kunder med spyware*. [URL: http://www.digi.no/php/art.php?id=210977]

40. digi.no. 23.9.2005. *Kriminell adware utrydder e-butikker*. [URL: http://www.digi.no/php/art.php?id=274179]

41. Creswell, J.W. 2003. *Research Design 2nd Ed.* Sage.

42. Microsoft. Knowledge Base article 883792. [URL: http://support.microsoft.com/kb/883792/en-us]