

*Master Thesis*  
*Computer Science*  
*Thesis no: MCS-2006:14*  
*January, 2007*

---



# Blocking Privacy-Invasive Software Using a Specialized Reputation System

Tobias Larsson

Niklas Lindén

Department of  
Interaction and System Design  
School of Engineering  
Blekinge Institute of Technology  
Box 520  
SE - 372 25 Ronneby  
Sweden

This thesis is submitted to the Department of Interaction and System Design,  
School of Engineering at Blekinge Institute of Technology in partial fulfillment  
of the requirements for the degree of Master of Science in Computer Science.  
The thesis is equivalent to 20 weeks of full time studies.

---

**Contact Information:**

Authors:

Tobias Larsson

E-mail: tola01@student.bth.se

Niklas Lindén

E-mail: nili02@student.bth.se

University advisers:

Martin Boldt

Bengt Carlsson

Department of Systems And Software Engineering

Department of  
Interaction and System Design  
Blekinge Institute of Technology  
Box 520  
SE - 372 25 Ronneby  
Sweden

Internet : [www.bth.se/tek](http://www.bth.se/tek)  
Phone : +46 457 38 50 00  
Fax : + 46 457 102 45

## **Abstract**

Privacy-invasive software is an increasingly common problem for today's computer users, one to which there is no absolute cure. Most of the privacy-invasive software are positioned in a legal gray zone, as the user accepts the malicious behaviour when agreeing to the End User License Agreement. This thesis proposes the use of a specialized reputation system to gather and share information regarding software behaviour between community users. A client application helps guide the user at the point of executing software on the local computer, displaying other users' feedback about the expected behaviour of the software. We discuss important aspects to consider when constructing such a system, and propose possible solutions. Based on the observations made, we implemented a client/server based proof-of-concept tool, which helped us discover other issues such as the effect on system stability. We also compare this solution to other, more conventional, protection methods such as anti-virus and anti-spyware software.

**Keywords:** Reputation system, spyware, protection

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Background and Related Work . . . . .	4
1.2	Thesis Outline . . . . .	5
1.3	Definitions . . . . .	5
<b>2</b>	<b>Research</b>	<b>7</b>
2.1	Research Motivation . . . . .	7
2.2	Research Question . . . . .	7
2.3	Research Method . . . . .	8
2.4	Proposed Solution . . . . .	8
2.5	Research Contributions . . . . .	8
<b>3</b>	<b>Important Considerations</b>	<b>9</b>
3.1	Information gathering . . . . .	9
3.1.1	Unintentional Incorrect Information . . . . .	9
3.1.2	Intentional Incorrect Information . . . . .	11
3.1.3	Protecting Users' Privacy . . . . .	12
3.1.4	Important Questions . . . . .	13
3.2	Vote Balancing . . . . .	15
3.3	Software Database Handling Issues . . . . .	18
3.3.1	Software Information Gathering . . . . .	18
3.3.2	Software Search and Information Representation . . . . .	21
3.4	Conclusion . . . . .	22
<b>4</b>	<b>Results</b>	<b>23</b>
4.1	Overall System Design . . . . .	23
4.2	Design Choices . . . . .	24
4.2.1	Client Design . . . . .	24
4.2.2	Server Design . . . . .	25
<b>5</b>	<b>Discussion</b>	<b>29</b>
5.1	Client-sided System Stability . . . . .	29
5.2	Internet Connection Requirement . . . . .	30
5.3	Server-sided Stability and Protocol . . . . .	31

5.4	Protection Systems Comparison . . . . .	32
5.4.1	Anti-virus Software With Anti-Spyware Functionality . .	32
5.4.2	Anti-spyware Software . . . . .	32
5.4.3	Specialized Reputation System Against Privacy Invasive Software . . . . .	33
5.4.4	Comparison . . . . .	33
5.4.5	Conclusion . . . . .	34
5.5	Research Questions . . . . .	35
5.5.1	Design Questions . . . . .	35
5.5.2	Problem Identification . . . . .	36
5.5.3	Pros and Cons . . . . .	36
<b>6</b>	<b>Conclusion</b>	<b>38</b>
<b>7</b>	<b>Future Work</b>	<b>39</b>
<b>A</b>	<b>Client GUI</b>	<b>42</b>

# Chapter 1

## Introduction

Today, some sources show that about 90% of all consumer PCs connected to the Internet are infected by privacy-invasive software (PIS) [17, 20]. Many of these users are not aware of the fact that they are running malicious software, relying entirely on anti-virus software and firewalls. However, anti-virus software focuses on pure viruses, worms and Trojan horses. Although they might be privacy invasive a larger group is legitimate software that is gathering information about its users, showing targeted ads, sending user behaviour patterns, visited websites and similar, storing them for an unknown period of time in a central database. These software are often in a legal gray zone, since they normally inform the users of their actions, but often in such a format that it is unrealistic to believe that a normal computer user will read and understand the provided information. The End User License Agreements (EULA) that the user has to agree on before using or installing the software are often written in a legal format, sometimes spanning over 5000 words [19], and many users choose to proceed without actually studying it, giving his or hers consent to whatever might be stated in the EULA, which can be anything the software developer wants.

There are numerous ongoing projects and attempts to produce effective software for removing PIS [10, 12, 18]. However, this requires a classification of some software as “harmful to the user” which might lead to law suits, since the information regarding system behaviour is stated in the license agreement which the user has accepted, and this could be costly for the anti-spyware software company [15]. As a result of this, they may be forced to remove certain software from the list of PIS to avoid future legal actions, and hence deliver an incomplete product to their customers, being unable to correctly classify some software as privacy invasive.

As the problem of PIS is widely spread, and no complete protection is available, there is a need for ways to better inform the user about the software he or she uses, while still not classifying it as “harmful to the user” and hence risking law suits.

There are numerous well-known and popular websites based on the concept

of letting users grade different services, applications, shops, and similar, such as Pricerunner and Flixster [5, 13]. The main concept is to help guide other consumers to, for example, find the best store to shop at and to avoid pitfalls and unethical sellers.

We are planning to combine this concept together with a client software that helps guide the user whenever a program is about to execute, by showing other users rating and comments of the particular software.

The user will be able to rate and comment the software, including answering a number of questions regarding software behaviour, possible malicious functionality, and so on.

## 1.1 Background and Related Work

As mentioned in the introduction, there are already existing software for detecting and removing PIS. These often work the same way as common anti-virus software, using a signature database of privacy-invasive software, and comparing these signatures to files on the local computer [2]. There are also software scanning the Windows registry for known registry keys and modified values used by PIS. These approaches renders the software unable to locate and identify privacy-invasive software that are using obfuscation techniques to hide themselves, much like polymorphic viruses. The development in the spyware field can be compared to that of the virus field, often described as an arms race between virus creators and anti-virus software vendors in discovering new techniques to prevent each other.

The usage of the term spyware has become increasingly popular, both by users, media and software vendors. It has been defined as software that “may track users’ activities online and offline, provide targeted advertising, and/or engage in other types of activities that users describe as invasive or undesirable” [6]. This means that it has come to include practically all kinds of malicious software, ranging from software that displays advertisements based on user behaviour (adware) to trojan key loggers, as well as actual spying software (spyware). A better term to use instead of spyware, would be privacy-invasive software (PIS). In an attempt to clarify the usage of this term, Boldt and Carlsson based their classification of privacy-invasive software on user’s informed consent and negative user consequence, as shown in table 1.1 [1].

There have been research conducted to develop techniques for detecting privacy-invasive software that tries to hide behind obfuscation techniques and other polymorphic behaviour. This enables the anti-spyware software to identify the privacy-invasive software through behavioural characterization rather than file signature [9].

As described by Resnick et al. a reputation system “collects, distributes, and aggregates feedback about participants’ past behaviour” [14]. This can either be part of a larger system, to give the users incentives to behave well in the future knowing that other users will be able to review past transactions e.g. on an auction site, or as a system itself used for rating e.g. resellers of home

Table 1.1: Classification of spyware based on user’s informed consent and negative user consequence

	<i>Negligible Negative Consequences</i>	<i>Moderate Negative Consequences</i>	<i>Severe Negative Consequences</i>
<i>High Consent</i>	1) Legitimate Software	2) Adverse Software	3) Double Agents
<i>Medium Consent</i>	4) Semi-transparent Software	5) Unsolicited Software	6) Semi-parasites
<i>Low Consent</i>	7) Covert Software	8) Trojans	9) Parasites

appliances, Hollywood blockbusters, or basically any kind of product or service.

This helps new users build a trust for a particular reseller or company based on other users’ past opinions about the other party, without any personal contact with the reseller or company in question. This is increasingly important considering the present development rate for e-commerce and online services where customers seldom if ever meet the business representatives they are dealing with.

## 1.2 Thesis Outline

The intended audience for this thesis is anyone interested in computer science, with an emphasis on security and privacy. We will start in chapter 2 with outlining our research motivation, research questions as well as our contribution to the body of knowledge. In chapter 3 we discuss important aspects that need to be taken into consideration when constructing a specialized reputation system to prevent privacy-invasive software. Chapter 4 presents our results including the proof-of-concept tool and the different design choices we made. In chapter 5, we discuss discovered issues, answer our research questions and also make a brief comparison with traditional anti-virus and anti-spyware solutions. Chapter 6 concludes our thesis, and in chapter 7 we discuss future work.

## 1.3 Definitions

Throughout this thesis we will use words and terms that may be ambiguous or unfamiliar to certain readers. Therefore, we include the following definitions:

**EULA** End User License Agreement, an agreement a user has to accept to be able to install and/or use a specific software.

**PIS** Privacy Invasive Software, ranging from legitimate software to trojan horses. More described in section 1.1.

**Spyware** Software that “may track users’ activities online and offline, provide targeted advertising, and/or engage in other types of activities that users describe as invasive or undesirable” [6].

# Chapter 2

# Research

## 2.1 Research Motivation

As seen, spyware is a major and growing problem for today's computer users. Spyware outbreaks are escalating from a productivity problem to a security issue affecting more people than the single computer user [16]. There is a need for a method of informing the users about malicious software behaviour, while not risking law suits from software vendors by classifying them incorrectly. A specific software might be privacy-invasive, and hence classified as malicious by a removal tool, but if the End User License Agreement of the specific software states what the software does and which information it handles, the software vendor can argue that this is legitimate since the user agreed to it upon installing the software. This could in turn lead to the producer of the removal tool being forced to remove this specific software from their signature database, and hence delivering an incomplete product to end users.

## 2.2 Research Question

We believe that informing the user about malicious software behaviour without risking legal actions can be done through a reputation system, where users have the ability to rate and comment on different software. Hence, we intend to address the following research questions:

- How can a specialized reputation system be designed to prevent privacy-invasive software by helping the user make an informed decision regarding the execution of each specific software, based on the expected behaviour of the executable, in terms of privacy invasive, disruptive and malicious outcome?
  - What possible problems have to be addressed in order for such a system to function properly?
  - What are the potential pros and cons running such a system?

## 2.3 Research Method

The research will be conducted through a literature study addressing the potential problems that need to be investigated before actually constructing a reputation system for preventing privacy-invasive software. We will also develop a proof of concept tool, showing the system in action in a small scale, controlled environment.

## 2.4 Proposed Solution

As we mentioned in the introduction we will construct a client/server based reputation system as a proof of concept tool. The server will handle a database of ratings and comments about different software given by users of the system. Once the user tries to execute a program on the local computer, the client will pause the execution process and gather information from the server about the program in question, presenting this to the user. He or she then has the opportunity to make an informed decision about continuing the execution or not, based on the experiences of other users.

The most important aspect of the system is that a user running a client can provide feedback and comments on any software running on his or her local computer. The votes and comments are then stored on the server in order to provide the other users with more detailed feedback about the software they are running. The client will also give the user the ability to either blacklist or whitelist a specific program, and hence blocking or allowing it to run without any further user interaction.

## 2.5 Research Contributions

Our main contribution will be the discussions made throughout this thesis, covering a novel approach to blocking privacy-invasive software as well as extensive discussions regarding possible problems one may come across during the design of such as system and during its execution. This, along with multiple suggested solutions to discovered possible problems and examinations of the drawbacks and benefits of the mentioned solutions will help highlight important issues related to blocking privacy-invasive software. We also intend to implement the proposed solution mentioned above as a proof-of-concept tool. We would like to make a user study on the proof-of-concept tool we implement to verify the usefulness of the suggested system and that the design choices suggested throughout this thesis are valid and relevant. However, as this is outside the scope of this thesis we will not present the results of the user study in this thesis, but instead include it in the future work section.

## Chapter 3

# Important Considerations

There are a number of issues that need to be addressed when considering the design and implementation of the proposed system. Especially, the reputation system server as such is vulnerable to a number of possible ways of abuse. We will go through each consideration individually, explaining the problem at hand, as well as proposing one or more possible solutions that may help fully prevent the problem, or at least reduce the impact.

### 3.1 Information gathering

There are a number of aspects to take into consideration when building a system that is to gather, store and present information from multiple, unknown users. Although the system has been set up for a clear purpose, individual users, or groups of users, may find it more interesting to – for instance – intentionally enter misleading information to discredit a software vendor they dislike, use multiple computers in a distributed attack against the system to fill the database with bogus votes, enter irrelevant or indecent comments, and so on. When it comes to inventing new ways of disturbing peace, the stream of ideas seems to be never-ending.

#### 3.1.1 Unintentional Incorrect Information

Even though it may be done without malice, even in good faith, ignorant users voting and leaving feedback on programs they know nothing or little about may be a rather big problem for a reputation system such as this, especially at a budding phase. If the number of users is low, compared to the number of software to be rated, there is a big risk that many software will be without any, or with just a few, votes. Even worse, if these few votes and comments have been given by users with little actual knowledge about the software they are rating, they may – for example – give the installer of a program bundled with many different PIS a high rating, commenting that it is a great free program, highly recommended.

In a normal environment, this would not be a problem, as a number of more experienced users would already have added negative feedbacks, warning other users of the potential dangers with installing this software package. However, in the cases where there are few users and votes available at any point of time, this may be a big problem, causing the following:

- The main purpose of the reputation system, to help raise user awareness and guide users into making well informed decisions about which software to run, is completely undermined if more than 50% of the votes are false positives, recommending the user to run harmful software.
- As a result of the above, trust is lost for the system; experienced users recognizing the dangerous file that is requesting the execute, in combination with seeing the system actually recommending him/her to allow the execution, will soon realize that he has little to gain from using the system, where-as inexperienced users will probably follow the advice, get over-run by PIS, and in a later stage, blame the system for not properly warning him of the danger.
- Unless the situation is remedied, for instance by more experienced users countering the original votes and comments with more relevant information, negative rumours will start to circulate about the system as a whole, leading to fewer users, which in turn further damages the reliability of the system.

There are a number of possible solutions to the problem, none of which are complete. One that may help prevent it would be to use bootstrapping of the program database at an early stage, preferably before the system is put to use, copying the information from an existing, more or less reliable, software rating database of programs and their individual ratings into the database of the reputation system. That way, it would be possible to ensure that no common program has few or zero votes, and in the event of novice users giving the software unfair positive or negative ratings and comments, the number of existing votes would make their votes one out of many, rather than the one and only. This, in combination with that more votes gives the user running the client more information to support him in his decision, would help reduce the risk of lost trust mentioned above. Rather than solving the early-phase problems, the early stage period of the system is skipped by filling the database before it is actually run.

Also, if users receive good recommendations from the start, they might be more willing to supply good and trustworthy data themselves. This will result in a positive spiral, which in turn could lead to more users starting to use the system, giving the system an even greater user base where the majority of the community are working to help maintain the high quality of the ratings and comments.

Another solution would be to have one or more administrators keeping track of all ratings and comments going into the system, verifying the validity and

quality of the comments prior to allowing other users to view them, as well as working on keeping the program database updated, giving expert advice on certain programs, such as well-known whitelisted applications, etc. However, once the number of users has reached a certain level, this would require a lot of manual work, which could become expensive for maintaining a free program, as well as seriously decrease the frequency of vote updates.

Yet another approach involves allowing the users to rate the comments of other users in terms of helpfulness, trustworthiness and correctness, creating a reliability profile for each user. This profile would be used to help weight the ratings of different users, making the votes and comments of well-known, reliable users more visible and important than those of new users. It does not directly handle the problem of inexperienced users giving incorrect information and ratings, if they are the only ones commenting and voting, but as soon as more experienced users give contradicting votes, their opinions will carry a higher weight, tipping the balance in a – hopefully – more correct direction.

### 3.1.2 Intentional Incorrect Information

Sometimes individuals, or groups of people, decide to abuse existing systems. The motive behind the abuse may be just about anything, ranging from determining whether the system is protected from the attack in question, to seeing how much damage can be done to the system and be able to brag about the success. In the preventive anti-PIS reputation system, one such attack would be to intentionally try to enter a massive amount of incorrect data into the database. The underlying motive in this case may be generalized into two categories:

- Attempts to bloat and/or invalidate the database by filling it with irrelevant information of any kind, either to slow the system down, or even crash it, or just to make it less useful for the users. The motive behind this kind of abuse would be hard to determine clearly, as it is a rather meaningless and some-what random attack.
- Intentional insertion of votes and comments that target specific applications, trying to subject them to positive or negative discrimination. For instance, this could be carried out by software vendors trying to influence users' opinion about their application, such as tricking users into believing it is completely safe to run. Another scenario would be abuse by anti-groups that wish to target a specific company or program, to try to ruin the reputation of the company, claim the software contains malicious components, and so on.

The main question when it comes to vote flooding is how to allow normal users to be able to vote smoothly, and yet be able to prevent abusive users from attacking the system. The client application has to be able to send a request to the server, asking it to register a new vote / comment for the software in question, without making it readily possible for a malicious user to create automated scripts that send a stream of (possibly invalid) votes to the server.

An aspect to take into consideration is that the server must ensure that each user only votes for a program exactly once.

A common solution to this kind of problem would be to let the user register a user account at the server before being able to activate the client software. For each user account, only one vote and comment can be registered for a specific software. Using image verification and requiring a valid email address during the registration of a new user account would help prevent the system for users trying to automatically create a number of new accounts to avoid the limit imposed on the number of votes each user can give to each software [4]. One drawback with this approach is the increased registration time, and lost simplicity for the users trying to get the software installed and running smoothly.

Another way to prevent abuse through vote flooding would be to register the IP of every client voting in the system. That way, it would be possible to discover and block users trying to attack or abuse the system, and if needed be, block the offending IP address or even IP range, if a set of compromised computers from a net are being used. The greatest drawback with registering the IP address of clients voting is that it could be possible for a hacker breaking into the reputation system database server to use this information to gather a list of software, full information including version and filesize, that are being run at a client host, as well as the IP of the client, making it a likely target for future attacks, as further discussed in section 3.1.3.

### **3.1.3 Protecting Users' Privacy**

As the system is built for protecting peoples' privacy, we need to make sure the system itself does not intrude on it more than necessary, as that could lead to people not using it due to privacy issues. If a user is hesitant to run software that displays advertisement, he or she might be just as hesitant to run a software that collects information about which software that is running on his or hers system. However, if we assume that all users' systems are kept clean from privacy-invasive software through the use of our system, we move the target from these users to the server storing the users' data, which leads to only one place needing protection from attackers.

If the system would store sensitive information about its users, such as IP addresses, e-mail address, etc. and linking these to all software the user has ever cast a vote on, an attack on the reputation system server could have serious consequences for all users. If an attacker would get access to the database, this would give him or her access to a list of hosts and all software running on each host, where some of them could be vulnerable to remote exploits. However, not storing any data about which users have cast votes on a particular software could lead to vote flooding and similar, as the system would have no way of ensuring that a user only votes once.

As we need to make sure no users can vote more than once on each particular software, we cannot get rid of the concept of users and user accounts. However, one approach would be to ensure that all kinds of sensitive information that can be of use for an attacker, such as IP address, e-mail address, name, address,

city, or similar, are excluded from the user information stored in the database of the reputation system server. The only thing necessary to store is some kind of unique identifier for each user, such as a user name. However, this will make it hard for the administrators to ban a host that is abusing the system, as they will not have any IP address to ban.

Another solution would be a modification of the one above – we keep the amount of sensitive information at a minimum, while still storing the IP address to be able to prevent abusive usage of the system. However, to prevent abusive usage we only need to store the IP address for a limited amount of time. This would allow the system administrator to notice any strange behaviour, such as vote flooding, from malicious users, without putting the regular users in a risky situation. An attacker would only be able to see the votes that have been registered from a specific host for a limited period of time in the past, which will significantly decrease the size of the attack vector, provided the user does not have a static IP address as this will let the attacker map the user to an IP address once, and from this gain information about all software that are installed on a particular host. However, this is needed as there are ways of cheating the system that can only be discovered through analyzing the gathered IP information combined with submitted software ratings and comments. This could e.g. be several users on a company subnet trying to give their own software a higher rating through intense voting, or a malicious user manually creating several accounts in order to improve the user rating on one of his or hers users.

As mentioned in section 3.1.2, we need to prevent users from signing up several times in an automatic way, and one way of doing this would be to use their e-mail address as an identification item. However, this requires us to store this in the database, and similar to the IP address, this might not be something that people would like to store in a database that keeps track of which software they are running and their opinions on it. A solution to this would be to only keep a hash value of the e-mail address, as this can be used to discover that two e-mail addresses are equal, while it is impossible to recreate the e-mail address from the hash value.

### 3.1.4 Important Questions

One important question to consider during the information gathering is which questions to ask the user, what is the most important things to find out about the software? We have gathered a number of items that may be of value:

- Total rating – how would the user rate this software in total, what is the final verdict, after considering all the positive and negative aspects? For instance, a user may consider a freeware MP3 audio ripper that converts CDs into MP3s as a great deal, even though it also displays banners in some part of the interface.
- Review comment – how would the user describe the software in question? A short textual description of whatever important aspects the user has

discovered about the software. Users should be advised to write serious, relevant comments and try to focus on aspects such as security, stability, and so on, rather than graphical user interface details, for instance the background color of the application main window.

Apart from the rating and the comment, we find it important to ask the user one or more leading questions, to which the user can choose his answer from the options 'yes', 'no' and 'no idea'. The choice of question(s) could be at random, picking one or more random question(s) for the users to reply to. Below is a list of suggested questions:

- Does the software display advertisements and / or banners?
- Does the software degrade the overall performance of the computer after being installed, more than other programs of this kind that you have previously used?
- Does the software install additional software to the computer? For instance, some programs automatically download and install new software such as toolbars, helpers, 'buddy companions', and so on, in the background, without the user actually requesting it.
- Does the software automatically register itself as a startup program without asking?
- Does the software unexpectedly modify the settings and / or behaviour of other software installed on the computer, in a way that should not be expected from a software of this kind? For instance, this does NOT include expected changes, such as a firewall application that configures windows to deactivate the internal windows firewall in order to use itself instead.
- Does the software try to connect to the internet and / or local network without stating so to the user, or explaining what kind of information it is trying to send / receive? For instance, this does not include expected behaviour, such as an anti-virus software updating the virus database, or a software informing the user that it will check for updates on start-up.
- Does the software gather personal and / or impersonal information about the user of the computer, the usage of the computer, webpages visited, programs run, keystroke logging, and so on, without clearly stating the purpose and / or intended usage of the information gathering?
- Can the software be uninstalled without problems? Are all the components of the software removed when uninstalling?
- Given all the positive and negative aspects you have experienced with this software, would you recommend it to other users?

As can be seen from the questions proposed, most of them are matters about program functionality and how it affects the operating system, and are readily answerable with a positive or negative reply (yes or no). Thus, an alternative approach to finding correct answers to these questions would be to use some kind of automated process that installs software, analyzes the traffic it creates and its behaviour over time, and so on, as well as compares the state of the operating system after the software has been uninstalled to that previous to installation, to determine if the software has truly been properly uninstalled. Such an automated system, on a large enough scale, could help gather the answers to these questions for a large number of software executables and keep it up to date with new versions and programs automatically, providing users with relevant, recent and extensive information. One example of such a system, used – among other – to crawl the web for executables to automatically install and analyze on a virtual machine can be found in [11].

## 3.2 Vote Balancing

As mentioned in sections 3.1.1 and 3.1.2, incorrect ratings, either intentional or unintentional, are a problem as that could lead to users making an incorrect decision. Because of this we need to ensure only “valid” votes are taken into consideration when calculating the score for a specific software. The problem is to decide which votes are valid, and which are not.

An easy solution for this would be to simply use the majority of the ratings and discard the minorities as “extreme values”. This could be used with success when the rating distribution looks similar to those shown in figure 3.1, 3.2 and 3.3, except in those cases where the majority is wrong and e.g. a software with low rating should actually have a higher rating.

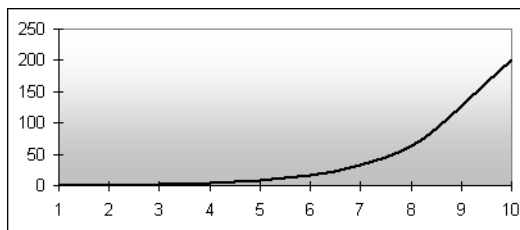


Figure 3.1: Large amount of high ratings

However, if the distribution would be similar to that shown in figure 3.4 it is not obvious whether this is a malicious tool or not. It could very well be intentional, or unintentional, incorrect information from a large user base that causes this rating distribution.

This can be solved by introducing a rating system for the users as well, although a much simpler one. A user that leaves a comment about a software

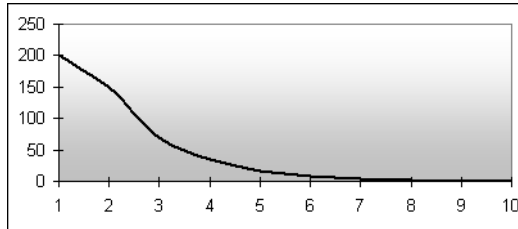


Figure 3.2: Large amount of low ratings

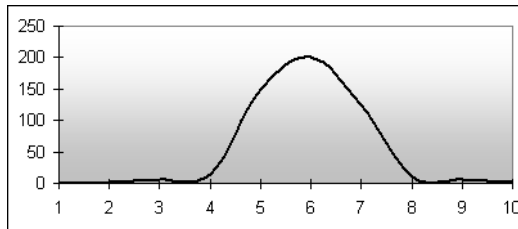


Figure 3.3: Large amount of middle ratings

gets this comment rated as good or bad by other users. From this score, it is decided how trusted this person can be, and from there how much worth his or hers votes are (e.g. assigned a value between 1 and 100). This leads to users leaving clueless comments to degrade in importance, where as the rating from other users which are helpful and correct counts as more important.

We should not make it possible for the user rating to fall below the rating of a new user at any point in time, as a person could easily create a new account instead of trying to improve his or hers rating [21]. One way to solve this problem would be by letting new users have a minimal rating at the beginning, which can later be improved as time passes by and other users rate their comments and feedback as useful.

The above solution will not discard any vote, as mentioned in the introduction, it will only assign a higher value to certain votes which comes from trusted, and valuable, users. This will also help the situation where the majority of the ratings are wrong, while the minority (the experts) are correct, as the experts' votes counts for more than the average user. Who is an expert is, as stated above, decided through the ratings that other users of the reputation system provide for his or hers comments.

As we want to avoid badmouthing of certain users, the comments will be completely anonymous when shown to the users, as this will make it hard for users to consequently bad-mouth another specific user [3]. To avoid abuse of the system from low rated users, such as them rating each and every comment with the lowest score possible, the importance of the user will be taken into

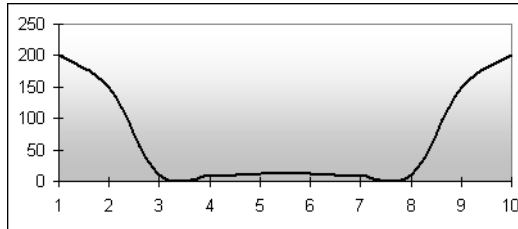


Figure 3.4: Large amount of low and high ratings

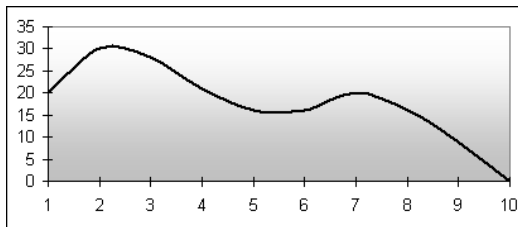


Figure 3.5: Large amount of low ratings, with user rating taken into consideration

consideration also when rating comments.

The average score for a specific software will then be calculated as shown in equation 3.1.

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

Equation 3.1: Calculating the average score for a software based on the submitted ratings from the users, as well as their ratings.  $x_i$  being the vote from a specific user (e.g. 1-10), and  $r_i$  being the casting user's rating (e.g. 1-100) which determines how much weight this vote has.

Using this, and assuming the low rating voters from figure 3.2 have a low rating themselves, while the higher rating voters have a higher rating, will give us a rating distribution similar to the one shown in figure 3.5. As we can see, the large amount of low ratings do not have a very large impact on the total score anymore. As an example, the average score increased by 2.27 points between figure 3.2 and figure 3.5.

One aspect to consider is whether or not to make the user rating visible for the particular user, and all other users on the system. Showing the user his or hers rating could give him or her incentives to write useful comments and give proper ratings, to get as high user rating as possible and e.g. to compete with friends using the same software. However, if the users are clearly aware of their own rating and how the system to raise it works, that would also mean

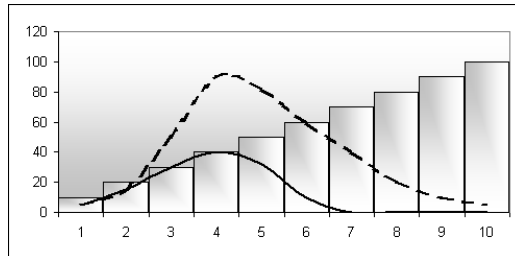


Figure 3.6: The evolution of a user’s rating with (solid line) and without (dashed line) rate growth limitations (vertical bars)

that malicious users could work up a good rating and once they have reached a certain level, turn and start behaving badly. This would allow for explicitly planned abuse, especially if the number of users of the reputation system is low, as a user could be able to create a main account that adds one or more comments on a few specific, rare, software. After that, the user could create a number of support accounts with the function to help raise the user rating of the main account, run the uncommon software while being logged in using the support accounts, rate the comments of the main account, and that way create a powerful masters account without any actual work.

A way to prevent this, or at least to slow the process down and requiring more work from the attackers, would be to introduce a maximum grow rate of the user rating. This means that the user rating can only grow to a certain extent during a specific period of time, e.g. during the first week a user has joined, the maximum rating he or she can get is limited to 10. The next week after that, the limit is 20, and so forth until the maximum rating value of 100 is reached. In figure 3.6 a possible user rating evolution scenario is visualized over a time period of 10 weeks, with the dashed line representing the user rate evolution without limitation, and the solid line representing the rate evolution with limitation. The growth limitation is shown as the vertical bars.

### 3.3 Software Database Handling Issues

When it comes to the questions regarding how to represent different software in the database of the reputation system server, there are a number of considerations that need to be taken into account.

#### 3.3.1 Software Information Gathering

First of all, we need to decide what information should be gathered about each software. Common sense dictates that the following information is the very minimum that needs to be gathered whenever a user decides to submit a rating

and comment about a new software that is currently not represented in the reputation server database:

- File name of the software executable.
- File size of the software executable.
- Fingerprint of software executable, for instance a generated MD5 sum.

Apart from the items mentioned above, the following information would be useful to help guide the user to finding the correct file if he or she is doing a search for a program on the reputation system server webpage:

- Company name of the software company that produced the software executable.
- File version.
- File location.

Some of the information mentioned above, such as company name, file version, etc, can sometimes be found inside executable files, stored as file properties. File location, fingerprint, and so on, can easily be found or generated by the reputation system client software.

When it comes to the details of storing the file location information along with the rating and comment, there are both positive and negative aspects to take into consideration and balance against each other. Storing the file location would be useful to be able to guide the user running the client software with information such as 'According to 98% of our users, this software executable should be located in the folder C:\Windows\System32\. If the file trying to execute at your local system is not stored in the above folder, we suggest you to take special care to verify the origins of the file.' This would support the user in making the decision about whether or not to allow the file execution. However, consider the following scenario:

A user has received a file from a friend over the popular instant messenger program ICQ. The file is stored in a path such as

C:\...\ICQLite\UserUIN\Received Files\*filename*

with UserUIN being the specific user id number used by the ICQ server to keep track of each user, allow users to find and add friends, and so on. The user then runs the software executable he has received, chooses to register a vote and comment in the reputation system, and thinks nothing more of it. A few days later, a malicious user decides to search the reputation system server for a number of known backdoors, including the software executable that the previously mentioned ICQ user has commented on. If the ICQ user happened to be the only, or one of few, user(s) who rated and commented on the software in question, there is a big risk his file location will be clearly visible. From the

file path, the attacker will be able to get the ICQ user UIN, from which the ICQ user's IP can be tracked every time he or she goes online and connects to the ICQ, and an attack can easily be launched.

The issue to keep in mind here is that we can never actually be sure where the file being executed is located on the reputation system user's hard drive. As a consequence of this, we cannot ensure that no sensitive information can be found in – or derived from – the file location, and to store this information as well as possibly displaying it to other users of the reputation system would be irresponsible, and may well be something the user in question has not expected or anticipated. Such a let down of the users integrity may cause negative publicity and dissatisfied users, not to mention the failure of the system goals: to help guide, inform and protect users to a safer computer environment.

However, even though the matter of which information to store in the server database is important, a more pressing issue is determining how to handle the insertion of new software into the database. Special care must be taken to create policies regarding how to handle each of the following scenarios:

- The user request to add a software executable to the database, and there is already a software executable in the database with the same name but another fingerprint and file size.
  - File version may or may not be known.
  - Software developer may or may not be known.
- The user requests to add a software executable to the database, and the file name is the same as an existing software executable in the database.
  - File fingerprint may or may not be equal.
  - Software developer may or may not be known and may or may not be the same between the two files.
- The user requests to add a software executable to the database, and the fingerprint of the new executable is the same as an existing software executable in the database.
  - File name may or may not be the same.
  - File location may or may not be the same.
  - Software developer may or may not be known.

Important considerations to make are – for instance – the level of involvement required by the user adding the software. Are there any of the cases mentioned above where it would be feasible to query the user regarding how to handle the insertion of the file, or is it better to always use a set of rules and policies on the server itself? If so, can a set of rules alone properly handle the addition of new software to the database, or is the surveillance of one or more administrators needed before the addition of new software is actually approved, in order to ensure database consistency as well as the quality of the database contents?

### 3.3.2 Software Search and Information Representation

When it comes to presenting information regarding different software to the user through the reputation system client, there are a few considerations to be made. Foremost is the question of how to provide the best possible information to help guide the user, given the information gathered about the software executable the user is about to execute and the data available in the reputation system database. This could be divided into two main tasks:

- Finding the correct, matching software in reputation system database.
- Gathering and presenting data from the reputation system database to the user in an optimally useful, yet feasible, manner, based on which data is currently available about the software executable in question.

When trying to match the software executable about to be run on the client computer with one in the reputation system database, the main items to look at would be the file fingerprint, file size, file name and developer company. Since the fingerprint is generated through an algorithm such as MD5, the risk of two different files having identical fingerprints is very small. However, since that also means that the fingerprints are different even between files with small modifications, in effect, two different versions of the same program will end up having different fingerprints. This also means they will be considered as separate software executables by the reputation system server, and as such their votes and ratings will be separated from each other. For instance, Nero Burning Rom 5.0.0.1 and Nero Burning Rom 5.0.0.2 will be handled as two totally independent applications. Although a drawback with this approach would be that there will be many different database entries for slightly different versions of the same program, this may in fact be beneficial to the user. For example, one version of an application may be well known to cause degraded performance, display banners, and so on, while in the next version, the developers have fixed the performance issues and decided to use other means to finance their work, and thus the contents of the reputation system will correctly present this to the user.

On the negative side, the concept of matching software executables using fingerprints may lead to great numbers of database entries for different versions of the same program, which in turn may lead to a low number of votes and comments for certain software and versions. To lessen, or even solve, this problem, it would be useful for the client to get information regarding possible related executables, based on any of, or any combinations of, file name, file size, company name, application name, etc. For instance, a user that executes the latest version of an application may receive no direct matches based on the fingerprint of the file, but a useful system should handle this by providing additional information such as informing the user that “No fingerprint match was found. However, 5 related results were found based on the file name, [click here to view more information](#)”.

Furthermore, it would be possible for the system to provide the user with valuable information regarding the calculated mean value of all the ratings the

software company in question has received, giving the user an indication of how well the software developed by this company has previously fared in the reputation system. That way, the user may choose to base his decision on ratings and comments given not only on the current software executable, but also on the derived total rating of the software developing company.

Yet another service that could be provided to the user would be to create an anti-virus portal on the reputation system server. If a user feels unsure about the reliability of the file that is about to be executed, he would be able to upload the file to the server through a webpage. The page, in turn, would forward the file to numerous free online virus scanners, from anti-virus software producers such as Kaspersky, Trend Micro Housecall, Avast, NOD32, Symantec, and present the results as a summary to the user, in a manner similar to Jotti Online Malware Scan [8]. Although this is not part of the reputation system as such, it would still be a useful service for the user.

### **3.4 Conclusion**

As we have illustrated in the previous sections, there are numerous aspects to take into consideration when designing the concepts for a reputation system such as this. Information has to be gathered from the users, the right questions must be asked to find out the most valuable information to help guide the other users. Different ways of abuse, such as vote flooding, positive and negative targeted discrimination, and so on, must be prevented without interfering with normal usage and / or protection of the privacy of the reputation system users. When considering votes and comments, the system has to be able to handle possible abuse, as well as to properly balance the weight of different users ratings and allow users to grade each others, thus improving the credibility of the more expert users and degrading that of users not taking voting and commenting in the system seriously.

# Chapter 4

## Results

### 4.1 Overall System Design

The system will be comprised of three major parts, a client with a graphical user interface (GUI) running on each users' workstation, a server running on one or more machines handling all requests and commits from the clients, as well as a database storing all data. The system will also offer a web based interface, which gives the users more possibilities in searching the information stored in the database. This will be used as an extension to the GUI client, where users e.g. can read more information about some particular software along with all the comments that have been submitted. The overall design can be seen in figure 4.1.

The client will require three major components:

- A hooking device that captures the execution calls from the Windows API, in order to allow the user to choose whether or not he or she really wants to proceed with the execution.
- A GUI that displays information to the user about which software is about to execute, the grades and comments other users have given, etc., examples shown in figures A.1 and A.2.
- A back-end component that handles the connections to the server, asks for ratings and comments about the software to be run, and so on.

The server will require the following:

- A database containing registered user information, ratings and comment for different software that users have previously voted on.
- A web-server that handles the requests sent by the client software, as well as displaying web pages for showing more detailed information about the software and comments in the database.
- An XML interface for the client/server communication.

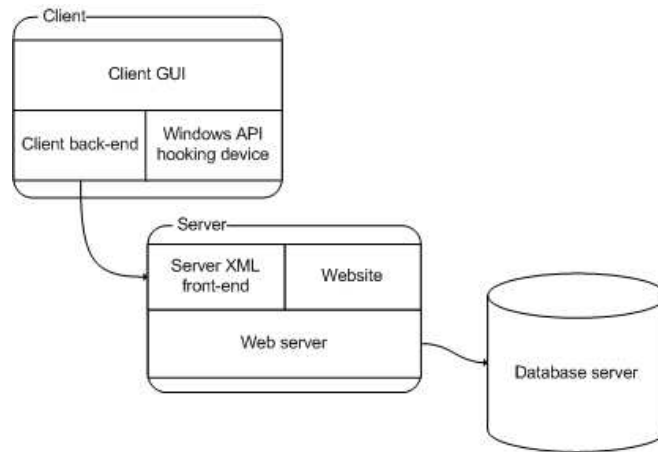


Figure 4.1: Overall system design

## 4.2 Design Choices

We decided to follow the general design outlined in the section 4.1. We created a server/client proof of concept application that uses file fingerprints (MD5) to distinguish between the different software, in the server database as well as in the client. Whenever a software is trying to execute, the hooking device informs the client about the pending execution, which in turn asks the user for confirmation before actually running the software requesting to execute. The client uses different lists to keep track of which software have been marked as safe (the whitelist) and which have been marked as unsafe (the blacklist) in order to reduce the need for user interaction. The proof of concept tool also allows the user to submit ratings and comments, as described in the previous sections, as well as to view compiled information from other users and run statistics about the software about to execute. The server side uses similar implementations to let the user search the database for software, vendors, file names, and display useful information about the afore mentioned through a web-based interface.

### 4.2.1 Client Design

#### API Hooking

The API hooking is used to capture the execution call that goes to the Windows kernel when the operating system tries to execute a program, either acting on input from the user or from another process. It is based on Anton Bassov's Soviet Protector code [7], with slight modifications added. It consists of a system driver that replaces the API call to `NtCreateSection()` with its own version, and a software component that communicates with the driver through a shared section of the memory.

In the original driver, if the user blocked the software executing, the driver would not continue the call to the original `NtCreateSection()` but rather let the operating system believe that it did not have access to execute the software. This led to an error message being displayed each time a software was blocked, stating that the software was not a valid Win32 application. This was modified into making the operating system execute a “dummy executable” provided by us, which at executions does nothing at all, instead of merely denying the access to the original executable. However, as a drawback this resulted in an adverse effect on the operating system stability, as further discussed in section 5.1.

### **White- and Blacklist**

The whitelist and blacklist are used for automatically allowing or denying a software to run, without asking for the user’s permission every time. They can both be populated in a few different ways:

- The user instructs the client to remember the decision made about a particular software at the point of execution. The software is then added to either the whitelist or the blacklist, depending on whether the user allowed or denied the software to execute.
- The user tells the client to recursively scan a directory on the hard drive, and add the located executables to either the whitelist or the blacklist.
- The whitelist can also be populated through an initial scan of the hard drive at the first execution of the client, or at a later stage if the client is started and the whitelist is discovered to be empty.

When the driver discovers an execution and informs the client about it, the client traverses the whitelist and blacklist for an occurrence of the executing software based on a MD5 checksum calculated from the exe-file’s content. If the software is found in either of the two lists, the appropriate response is automatically sent to the driver without the need for user interaction, otherwise the client queries the server and starts downloading information about the executing software to show the user and take action based on the user’s decision.

Should the software be located in both lists at the same time, which could happen through use of the hard drive scanning, the blacklist will take precedence over the whitelist.

## **4.2.2 Server Design**

### **Client Server Communication**

As mentioned earlier, the client and the server communicate over the regular HTTP protocol where the client sends an HTTP GET request and receives an appropriate response from the server. The server feeds the client with XML data, which is parsed and shown to the user. Each XML tag represents a specific row in the database, and by constructing an XML tree structure from the data

we can easily send several rows from the database to the user – as well as adding new data – without much trouble.

A normal session for fetching software information, including comments and answers to questions, looks like this:

1. Client: Requests software information based on an MD5 hash value calculated from the executable file content.
2. Server: If the software exists in the database, the server replies with a **software** XML tag, containing, amongst other, the rating for the software as well as the vendor and its rating and statistics over how many users have allowed or denied the software to execute. If the software does not exist in the database, an error message is sent to the client, and hence the client is unable to display any rating.
3. Client: Requests a top comment for the particular software received earlier, sending along the username and hashed password for the current user, to be notified whether the user already has rated the comment or not.
4. Server: Replies with the “best” comment in a **comment** XML tag. Each comment’s usefulness value is calculated such as  $P^2/T$ , where  $P$  is the number of positive remarks on a comment, and  $T$  is the number of total remarks and a higher value implies a better, more useful comment.
5. Client: Requests answers provided by other users to all the questions available to the users, such as those listed in section 3.1.4, in order to help the user make a more informed choice and have a better idea about the behaviour to expect from the software about to be executed.
6. Server: Replies with an **answercollection** XML tag, listing the number of answers on each alternative (yes, no, unknown) for each question.
7. Client: Requests the actual questions to be able to dynamically display them in the information dialog, making the client more flexible to a change to the underlying structure of the reputation system database and which questions are found important enough to merit inquiring the user about.
8. Server: Sends the question listing as a list of **question** tags.

If the user decides to submit a rating to the software database, the communication is a single request coming from the client, containing the comment, rating, answers to the questions, as well as the software the rating is submitted for. If the database does not contain the software information, it has to be added before any rating can be inserted, thus – as we include the software information among the data transmitted – the server can ensure that the software exists before adding the rating to the database.

The last event that takes place is that the client submits information to the server, informing it of whether the user allowed or denied the software to run. Apart from this data, the request also includes an entire **software** tag,

as the server once again needs to make sure the software exists in the database before adding statistics for it. This data is only sent to allow statistics to be constructed over the amount of allows and denies over a certain period of time, and is completely anonymous in the database to protect the users' privacy. There is no record whatsoever kept over the user account, username, user id, IP address, or any other identifiable information, in connection to the run statistic entry.

### **Keeping the User Anonymous**

The only data stored about the user is a username, hashed password and a hashed e-mail address, as well as timestamps of when the user signed up, and was last logged in. The e-mail address is only there to make it more difficult for a person to create several different accounts, as it is possible to sign up only once per e-mail address, and each address used to sign up must be a valid e-mail address, since it is used for the confirmation and activation of the newly created account.

From this data, it is impossible for us or anyone else getting in hold of the database, to identify a specific user, as long as the username (over the contents of which we have little control) does not reveal too much detailed information. And as our implementation does not store any IP addresses associated with the users, it is also impossible to determine which hosts are running which software, and from there try to launch an attack against a specific host.

The statistics collected, mentioned in section 4.2.2, are only counters containing the number of times each software has been allowed or denied to execute. This data is not associated with the users in any way, so no one can trace the execution history of a particular user.

What can be traced however, is every user's submitted rating, comment and answers for each software he or she has ever rated, as well as each user's submitted remark (positive for a good, clear and useful comment or negative for a coloured, non-sense or meaningless comment) for every comment he or she has ever rated. But as mentioned previously, it is impossible to directly or indirectly associate this data with a particular host, but only to a username, hashed password, hashed e-mail address and two timestamps, which does not put the user at any actual risk from using this software.

### **Rating Calculation**

The software rating is calculated just as described in section 3.2, taking each user's trust factor into consideration when calculating the final score for a particular software. To avoid the need of calculating the software rating each time a request is made to see it, it is updated once a day through a scheduled task and stored along with the software information in the database.

Included in the scheduled task is calculating the vendor rating. This is done after the calculation of the software rating as this is needed to get a correct vendor rating. The rating is calculated simply as an average score of the ratings

of all software belonging to the particular vendor. As with the software rating, this is also stored in the database to avoid dynamic calculation, and hence heavy load on the server when requesting information.

The user rating, used as the trust factor when calculating software ratings, is updated whenever another user submits a comment remark, but similar to the software rating and vendor rating, it is stored in the database to avoid dynamic calculation each time a request is made. Each time a user submits a comment remark, the rating of the user that wrote the comment will be updated with  $0.1 * r_s$  either added to or subtracted from the user's rating depending on whether the comment remark was positive or negative,  $r_s$  being the rating of the user that submits the comment remark. We also implemented the growth limitation, setting the maximum growth per week to 5 units. Hence, you can reach a maximum rating of 5 the first week you are a member, 10 the second week, etc. The second limitation is a minimum rating of 1 (which is also the rating for new users), and a maximum rating of 100.

## Chapter 5

# Discussion

### 5.1 Client-sided System Stability

One issue that we soon discovered during the implementation of the proof of concept tool was the question of system stability. As we give the users the ability to deny the execution of important system components, and the Windows operating system is not prone to graceful degradation, we also handed them the ability to crash the entire system in a single mouse click. This further enhances the need for a white list system to ensure proper operating system functionality in order to avoid inadvertently bringing the operating system down when running the software client. However, given that the user has the free choice to block any program, there is no way to guarantee that the operating system will not be crashed, especially at an initial phase where the user is learning how to use the software client.

As a measure to lessen this problem, during client startup, if the whitelist is empty, the user is prompted to scan one or more folders on his local hard drive and suggested to do so for at least the Windows installation folder. By whitelisting all the executables inside the operating system folder, the risk of accidentally crashing it is reduced.

Another approach would be to keep an internal list inside the client, containing the fingerprints of well-known and vital system components. When a user decides to block the execution of a software that is found inside the internal list, the software client could display an extra warning message, informing the user of the risk of blocking this program. A more radical approach would be to always permit these well-known system components without informing or even asking the user, in order to protect the stability of the operating system. However, this kind of automatic permission system would limit the client usability as part of the client functionality is hidden and unreachable to even the more experienced users. Furthermore, the construction of such a list, for all the different versions of the critical system components for different operating systems would be a difficult task – considering that even a single, minor difference inside

the file will generate a new file fingerprint – and as the list grows, the client system performance may decrease. Such a list would also need to be updated continuously as new updates, patches and versions of existing operating systems are released which normally happens on a regular basis and since not all users actually run the updates, the internal list would have to cover all previously known versions of all critical system components up to today. A possible approach to the problem of whitelisting system components would be examine the file about to execute, to determine if it has been digitally signed by Microsoft, and in case the certificate is present and valid, the file is automatically allowed to proceed with the execution. In theory, it would be possible to implement a signature handling interface in the reputation system client that allows the user to whitelist and blacklist different companies through their digital signatures, which – in turn – could considerably lower the need for user interaction and, for instance, solve the problem of the user being forced to repeatedly add a software application or component to the whitelist whenever an update has been installed.

Another stability problem that we encountered during the implementation and testing of the proof of concept tool was that of an executable running a second executable, for instance during the initialization of a software, or running a downloaded file from a web browser. If the user chooses to deny the execution, the dummy executable is executed in place of the original file, which often seems to lead to unpredictable and undesirable results. In the case of software initialization, it is natural that blocking the second executable would crash the main application, since the expected operations and settings have not been made even after the execution is expected to have been completed, and if the users chooses to do so, there is little else to be expected. However, in our case the main problem seems to be tied to the implementation of the driver and how it handles the files that are about to execute, so although the problem cannot be completely avoided, with the help of a refactoring of the driver system, a more graceful error could be achieved. The error when no return values or initializations are expected, such as when a file is being run from a web browser, should be possible to eradicate completely. The easiest way to do so would be to alter the driver implementation from the usage of a dummy executable back to allowing the Windows operating system to act upon a returned error message. The drawback with this would be that whenever an application is blocked, the user is forced to acknowledge it by closing a message box opened by the Windows operating system, displaying different error messages depending on the return error codes, which would soon become tedious to the user.

## 5.2 Internet Connection Requirement

As mentioned previously, the reputation system is server and client based: the client needs to be able to access the server in order to download the information about the executable file to be run, as well as to be able to submit ratings and comments. Although part of the functionality of the client, such as block-

ing and allowing executions based on the contents of the black and white list, would still be able to function correctly without an internet connection, most of the functionality is still expecting the link to the server to exist. Running the client without the internet connection could lead to delays while waiting for the attempted connections to time out, as well as possible client instability. It would be possible to make the client into a more strictly logged in server client application, letting the client occasionally check for a working Internet connection, and if no connection is found, or the connection is not working properly, the client would return to a log in screen and cease all normal functionality until the user is once again logged in. However, as some useful functionality can still be provided by the client alone, it would feel more relevant to allow it to keep running in a downgraded state. Possibly, one could use the connection checking to determine if there is a working connection to the server and if not, reduce the waiting times on the client software by avoiding to even try to download or upload any information, ratings or comments. Blacklisted and whitelisted applications would still be treated as normal and a minimalistic graphical interface could allow the user to decide whether or not to allow execution of new, previously unlisted software. Such an implementation would increase the usefulness of the reputation system application client even in environments where the underlying connection to the Internet is unreliable, without unduly lowering the performance and functionality.

### 5.3 Server-sided Stability and Protocol

As previously covered, the server was designed and implemented to be based on a standard server platform and language as well as to communicate with the client over a standard protocol (HTTP) and using XML. Worth mentioning would be that this choice resulted in lower implementation time and the easy creation of a reliable server application that can easily be deployed on any server running the Tomcat web server, or any other Java and JSP compatible server. As most web servers are well tested and maintained, and proven over time to be reliable and able to withstand most attacks, one can rest assured that running the server application will not put the system hosting it at any risk. On the other hand, one possible draw-back with this choice would be that it is possible for a user to easily examine the traffic generated by the client software and use a web browser or automated script to try to send enough requests to the server to affect the overall performance of the reputation system or even create a denial of service attack. However, similar attacks are common hand problems for almost all web-based systems where there is a client side where the communication can be observed and that knowledge used to device an attack upon a server. The benefit of knowing that the underlying webserver itself is reliable and durable lessens the concern.

## 5.4 Protection Systems Comparison

There are many existing protection systems available, ranging from anti-virus and anti-spyware applications to intrusion detection systems and firewalls, each with its own set of benefits and drawbacks. We have made a brief comparison between anti-virus software with anti-spyware functionality, pure anti-spyware software and a specialized reputation system designed to help protect users against executing privacy invasive software. We begin doing so by outlining the major characteristics of each system in turn.

### 5.4.1 Anti-virus Software With Anti-Spyware Functionality

Most anti-virus applications can be described as having the following characteristics: rather large, subscription based client system applications, often having a negative impact on system speed, using extensive virus databases, supplying frequent, reliable and large updates, heuristic functionality for finding new viruses based on application behaviour, quarantine or deletion options for discovered infected files, web-based virus removal help and instructions for more difficult infections, real-time scanning, on-demand scanning, and so on. The anti-spyware functionality put into such systems are – naturally – often an extension of the existing anti-virus engine. As mentioned earlier, there is an extensive gray zone between legitimate software and spyware, and there are cases where anti-virus vendors have been facing legal actions for branding a specific software as spyware and been forced to remove them from their protection.

### 5.4.2 Anti-spyware Software

The difference between pure anti-spyware programs and anti-virus applications with anti-spyware functionality depends on the actual implementation, but in many cases they are based on the same kind of functionality but with different specialization areas. Most anti-spyware applications support on-demand scanning of local hard drives and real-time protection of the system, as well as regular updates, heuristic functionality, quarantine options, and so on. One rather big difference would be that although spyware is considered to have a serious impact upon privacy, system performance, stability, and more, people are more incented to running an anti-virus application as can be observed from the pricing policy differences. Most anti-spyware applications are free for personal use, while anti-viruses often use the subscription based approach where the computer owner must pay a monthly or annual fee to be able to update the protection database and software. As with anti-virus software with anti-spyware functionality, anti-spyware software vendors also suffer from the risk of legal actions.

### 5.4.3 Specialized Reputation System Against Privacy Invasive Software

Although many details would be implementation specific, the following general characteristics would still hold true: a smaller client application, server sided program database rather than client based as in the case of anti-spyware and anti-virus applications, no or small updates, real-time manual protection (since no files are blocked without user permission), fully or partially internet connection dependent, with user-submitted information in the database. The user-submitted information provides both benefits and drawbacks: as all the information gathered and displayed is based on user feedback, the vendors of the software are not legally liable. Also, unlike anti-spyware and anti-virus technology, there is no need for experts and developers to work full time to add new information to the protection database, and thus, for instance, the stressful situations when a new virus has hit the Internet and a proper protection must be found as quickly as possible can be avoided. On the other hand, the quality of the information gathered is strongly dependent on the level of commitment of the users, and may need to be monitored, reviewed and weeded. As mentioned earlier, this process can also be time consuming, and failing to do so could result in a loss of trust in the reputation system as the information provided may be inaccurate or confusing. One major difference between more common anti-spyware software and the reputation system based solution we propose is also that in the latter, we are able to gather more complete and useful information regarding the behaviour of software. Instead of a black and white world where an executable is branded as either a virus or not, we are able to touch the previously mentioned gray zone in between. We gather and present information about software that is important and useful to the users, and hard to find. For instance, although an application may not be classified as a virus or spyware, users may think twice about running it if they are informed that it displays banners, connects to the internet for unknown reasons, registers itself as a startup program and does not even provide an uninstall option. This kind of discouraging information will not be provided by the vendor of the application and can only be received from users who have experienced it first-hand and are willing to share their experiences to help others.

### 5.4.4 Comparison

As mentioned above, anti-spyware and anti-virus applications have the benefit of specialized, up to date and reliable information databases that are updated on a regular basis. The drawback is a large database that must be downloaded and updated locally on the client, as well as traversed whenever a file is analyzed. Computational effort may be saved for the client when a centralized database approach is used, and even though more connections need to be made, less data is transferred, compared to downloading large updates to keep the local information up to date. The relevance and reliability of the information provided by the anti-spyware and anti-virus software may be more reliable than that of

users of a reputation system. However, the reputation system is able to cover more details that may be useful to the user, such as if the software displays banners, alter system settings, and so on, and with a sufficiently large user base, the sheer amount of data gathered helps compensate for the afore mentioned reliability issue. Also, by using a more flexible classification, where the user is provided the information about the software about to execute and is allowed to make a decision about allowing it to run or not, one is able to avoid the high contrast environment of anti-virus software and similar, where an executable is either strictly malicious or it is totally safe. Much due to this strict classification system, although no anti-virus or anti-spyware database can ever be complete, users tend to put their faith in the security of the software installed and do not pause to consider that even though the anti-virus software does not react to a file, it may still be infected. Services such as Jotti Online Malware Scan [8] demonstrate the varying performance of the different anti-virus software available today: unless the virus file is infected with a very common virus, it is highly likely that one or more system will be unable to detect it, and that the ones that are able to correctly classify it as a virus will have different names and specifications for it. Many viruses and spyware are never named but instead categorized, such as "Trojan.Dropper.XX" or "Irc.Backdoor", with little detailed information to be found. With so little information available, it is sometimes hard to trust that the antivirus can properly repair all damages dealt to the computer by the virus. Naturally, the database of a specialized reputation system would not be able to contain all programs either, but as the purpose is to protect the user by providing information through community-based means and allowing the user to make the final decision, people are less likely to follow it heedlessly.

#### 5.4.5 Conclusion

In conclusion, the different systems are built on different approaches, and the technology as well as pricing varies. In truth, it would be foolish to believe that either one approach would be a perfect solution to the problem at hand, and the view of the problem itself may differ. However, when looking at the development of the computer world, the Internet, and the on-going arms race in virus and spyware development, it is obvious that more than just one kind of protection is needed, and that there is no silver bullet. At the same time, we firmly believe that a specialized reputation system such as the one we propose would be a useful way to be able to penetrate the gray zone of half-legitimate software and to better inform users of what to expect from the software they are about to execute. It can be seen as trying to share and transfer knowledge between users, improving their level of expertise, instead of creating an expert system that handles all the decisions for the users, being ultimately responsible for the failure when the protection fails.

## 5.5 Research Questions

### 5.5.1 Design Questions

“How can a specialized reputation system be designed to prevent privacy-invasive software by helping the user make an informed decision regarding the execution of each specific software, based on the expected behaviour of the executable, in terms of privacy invasive, disruptive and malicious outcome?”

As clearly outlined throughout this paper, there are many different aspects that must be taken into consideration during the design phase of the creation of a specialized reputation system for the prevention of privacy-invasive software. A brief recapitulation of the most important issues follows below:

- Information gathering – the process of designing the system in order to avoid intentional and/or unintentional incorrect information submitted by the users of the system.
- Protecting the privacy of the users – making certain that a system constructed to help users prevent privacy-invasive software does not, in turn, threaten their privacy by needlessly storing personal and/or sensitive data about the users.
- Data usefulness – ensuring that the data gathered from a user regarding a software is of value to other users: not only ratings and comments, but answers to leading and important questions as well, giving a better overall view of the behavioural expectations of the software in question.
- Vote balancing – the procedure of raising over-all rating quality by determining which users are reliable and which users are not. By allowing feedback on comments and ratings, and take such history into consideration, as well as seeing to the time aspect, users can be presented with more accurate information.
- Data storage – determining which information to gather about new software, as well as how to gather it and to consider how to handle weeding of database entries. Information gathered should be relevant to the identification of different software, as well as useful to present to users browsing the information database through the homepage and similar. The gathering process must be smooth for the user, yet maintainable in a long term perspective.
- Information processing and representation – the information gathered from the users, such as software information, ratings, comments, and so on, must be gathered, processed and presented in an effective manner in order to optimize the usefulness of the reputation system.
- Client system stability – working at a kernel level of the operating system may lead to undesirable stability issues that need to be considered

when deciding how to handle the execution calls. Also, users must be guided to prevent instability caused by needlessly blocking critical system components, out of ignorance or mistake.

- Server considerations – apart from the actual implementation comes the issues of choosing security by obscurity using homemade/non-standard servers and protocols or open, well-tried solutions that may or may not be harder to protect due to that openness and familiarity to many users. It may be harder to understand, attack or abuse non-standard components but it may also result in more severe impacts if such an attack is successful.
- Client connection dependency – deciding during the design how dependable the client software is on an internet connection being present, ranging from fully dependable, partially dependable with graceful degradation or completely independent, where the base functionality of allowing and blocking software would be considered the core, and the possibility of viewing software information from the reputation system server considered additional, non-essential functionality.

### 5.5.2 Problem Identification

“What possible problems have to be addressed in order for such a system to function properly?”

Numerous possible problems have been described and discussed throughout this paper. They include the issue of cheating and the prevention of it, vote balancing, how to properly calculate the software rating, how to handle incorrect information submitted by users, and so on. Please review the list above for a brief overview, or return to chapter 3 and on for more detailed information.

### 5.5.3 Pros and Cons

“What are the potential pros and cons running such a system?” On the positive side, one major benefit with a specialized reputation system would be the previously noted ability to be able to share information about software behaviour without any risk of resulting legal actions, thus being able to reach the hard-to-touch grey zone of half-legitimate software. With a sufficient user base, information about most common and new software can quickly be gathered and shared among the users of the community. Vote balancing and mechanisms allowing the users to determine which comments are useful helps improve the relevance of the information. Such a community encourages the users to understand more about the software being executed on their computers, as well as avoid programs that exhibit undesirable behaviour, and inspires users to contribute with their own knowledge to help others. Reward systems, such as top lists of the most active users on the reputation system homepage, etc, can also be used to further encourage members to participate more actively in keeping the information up to date and of a high quality.

When it comes to the negative aspects, the issue of system stability is definitely worth mentioning, the need for a good, stable and custom written driver being the most apparent consideration. Also, as we have mentioned, the Internet connection dependability should be addressed, either to solve the problem completely by using a more common solution with a local database that is downloaded to the client computer using live update functionality, or partially solved by ensuring the graceful degradation of the client software in case the connection is unreliable or missing.

## Chapter 6

# Conclusion

The goal of this thesis was to explore how to construct a specialized reputation system to be used for blocking privacy-invasive software. To do this, we stated a number of issues to be considered when constructing such a system, such as what to do about incorrect information, be it intentional or unintentional, how to protect the users' privacy, how to calculate the software ratings, what data to store on the server, etc.

To further explore the possibilities, we designed and implemented a client and server-based proof of concept tool. Each time a user is about to execute a program, the client pauses the execution, downloads information and rating about the particular software from the server, and asks the user whether he or she would like to allow or deny the software to run. The tool also included the ability to add software to either a whitelist and blacklist, allowing or denying the software to run without user intervention. On top of this, a web-based interface was constructed to provide users with an easy way of searching the database for software, filenames and vendors, as well as to see statistics of their previous usage of the tool such as which software they have rated and the comments they have submitted.

During the usage and testing of the proof of concept tool, we discovered several side effects that stems from the usage of the system in a real life setting, such as system stability and, with this particular proof of concept tool, the need for an internet connection. Despite this, there is no existing solution that is bullet-proof, and considering the current development in the area, it is unlikely one will be found. We believe our proposed solution to be able to help fill an existing gap in the personal computer protection armour, yet we do not suggest it to be used as the only security software, but rather see it as a part of a larger security software suite solution, including more conventional programs such as anti-virus software, firewall, and so on.

## Chapter 7

# Future Work

We will let a group of users test the existing, proof of concept version of the specialized reputation system in a controlled lab environment, in order to collect comments and feedback about the system from regular users. It will help us further improve the design and to discover possible drawbacks or suggestions for new features that we have previously overlooked.

We also intend to implement a larger scale, more complete version of the specialized reputation system, with a custom developed driver and consideration taken to many of the issues discussed throughout this paper, both during design and actual implementation. Also, we intend to try to determine which approach would be more suitable to the Internet connection dependability problem, and test the resulting client application in combination with more conventional anti-virus and anti-spyware software. Furthermore, it would be interesting to examine different solutions for automated information gathering of program information that can be determined without user interaction, such as if the application registers itself as an autostart program, installs and uninstalls fully without complications, and so on. As long as the core functionality is reliable, that is, the answers it provides are correct, such a system would be useful if allowed to run – even on a smaller scale – since it would continuously be gathering and processing information about new software executables. This, in combination with the answers provided by the users in the client application, could also be used to help determine the reliability of a user, based on whether the answers provided by the user coincide with the answers gathered by the automated process. Repeated incorrect answers provided by a specific user may indicate a low credibility and the rating of the user in question could be adjusted accordingly, under the condition that the correctness of the automated process can be verified.

# Bibliography

- [1] M. Boldt and B. Carlsson, “Privacy-Invasive Software and Preventive Mechanisms”, in the proceedings of the IEEE International Conference on Systems and Networks Communications (ICSNC06), Papeete Tahiti, 2006.
- [2] M. Christodorescu and S. Jha, “Testing Malware Detectors”, in the proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, 2004.
- [3] C. Dellarocas, “Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior”, in the proceedings of the 2nd ACM Conference on Electronic Commerce, Oct 2000.
- [4] J. Douceur, “The Sybil Attack”, in the proceedings for the 1st International Workshop on Peer-to-Peer Systems, March 2002.
- [5] Flixster, <http://www.flixster.com>, 2006-09-13.
- [6] N. Good et al., “Stopping Spyware at the Gate: A User Study of Privacy, Notice and Spyware”, in the Proceedings of the Symposium on Usable Privacy and Security, Pittsburgh USA, 2005.
- [7] Hooking the native API and controlling process creation on a system-wide basis, [http://www.codeproject.com/system/soviet\\_protector.asp](http://www.codeproject.com/system/soviet_protector.asp), 2006-11-23.
- [8] Jotti Online Malware Scan, <http://virusscan.jotti.org>, 2006-10-11.
- [9] E. Kirda, C. Kruegel, G. Banks, G. Vigna, R. A. Kemmerer, “Behavior-based Spyware Detection”, in the proceedings of the 15th USENIX Security Symposium, pp. 273-288, 2006.
- [10] LavaSoft Ad-Aware, <http://www.lavasoftusa.com/software/adaware>, 2006-09-19.
- [11] A. Moshchuk, T. Bragin, S. D. Gribble, H. M. Levy, “A Crawler-based Study of Spyware on the Web”, in the Network and Distributed System Security Symposium Conference Proceedings, Virginia USA, 2006.

- [12] Norton Internet Security, [http://www.symantec.se/region/se/product/nis\\_index.html](http://www.symantec.se/region/se/product/nis_index.html), 2006-09-19.
- [13] Pricerunner, <http://www.pricerunner.com>, 2006-09-13.
- [14] P. Resnick, K. Kuwabara, R. Zeckhauser, E. Friedman, “Reputation Systems”, Communications of the ACM, 2000.
- [15] See you later, anti-Gators? CNET News.com, [http://news.com.com/2100-1032\\_3-5095051.html](http://news.com.com/2100-1032_3-5095051.html), 2006-09-19.
- [16] K. Schultz, “Sticking It to Spyware”, InfoWorld, Vol. 27, No. 38, 2005.
- [17] Spyaudit, <http://www.earthlink.net/spyaudit/press/>, 2006-09-12.
- [18] Spybot - Search & Destroy, <http://www.safer-networking.org>, 2006-09-19.
- [19] “Spyware”: Research, Testing, Legislation, and Suits, <http://www.benedelman.org/spyware/>, 2006-10-09.
- [20] Webroot Software — Internet Spyware and statistics about infection rates, <http://www.webroot.com/resources/stateofspyware/excerpt.html>, 2006-09-12.
- [21] G. Zacharia, A. Moukas, P. Maes, “Collaborative Reputation Mechanisms in Electronic Marketplaces”, in the proceedings of the 32nd Hawaii International Conference on System Sciences, 1999.

# Appendix A

## Client GUI

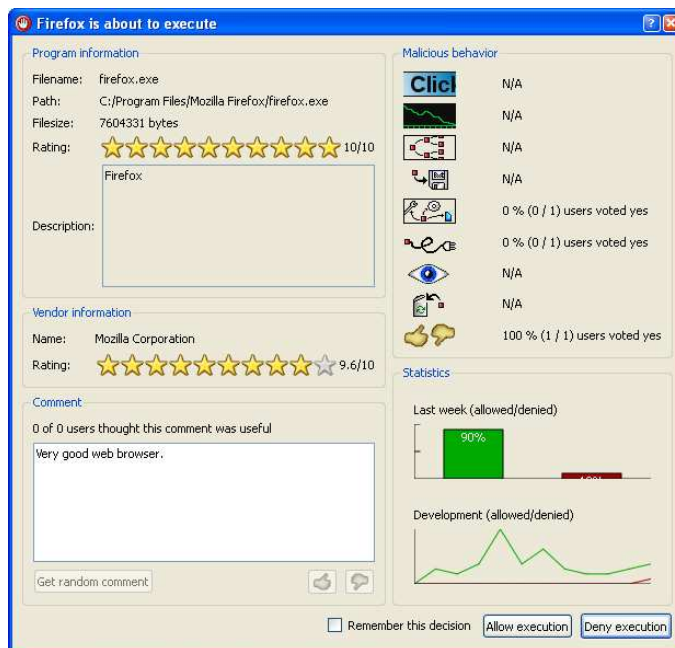


Figure A.1: The information dialog that shows up each time a software is about to execute, showing the software and vendor rating, as well as comments and information about malicious behaviour to the user.

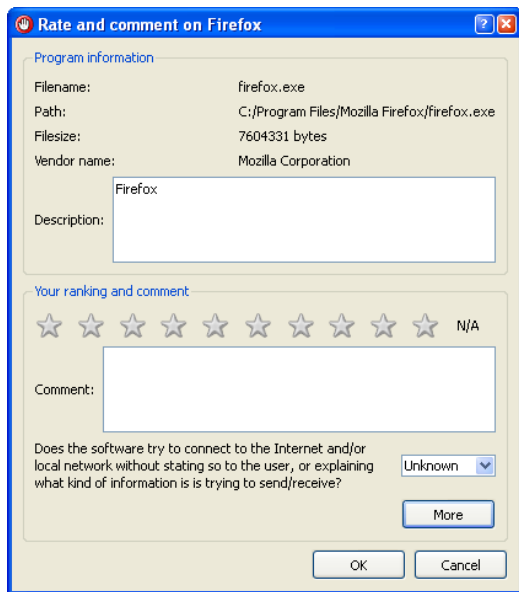


Figure A.2: The dialog used by the user to submit software ratings, comments and answers to the server.