On the Simulation of a Software Reputation System

Martin Boldt, Anton Borg, Bengt Carlsson
School of Computing
Blekinge Institute of Technology
372 25 Ronneby, Sweden
{mbo,atb,bca}@bth.se

Abstract— Today, there are difficulties finding all malicious programs due to juridical restrictions and deficits concerning the anti-malicious programs. Also, a “grey-zone” of questionable programs exists, hard for different protection programs to handle and almost impossible for a single user to judge. A software reputation system consisting of expert, average and novice users are proposed as a complement to let anti-malware programs or dedicated human experts decide about questionable programs. A simulation of the factors involved is accomplished by varying the user groups involved, modifying each user’s individual trust factor, specifying an upper trust factor limit and accounting for previous rating influence. As a proposed result, a balanced, well-informed rating of judged programs appears, i.e. a balance between quickly reaching a well-informed decision and not giving a single voter too much power.

Keywords—Reputation systems, simulation, computer security, malicious software.

I. INTRODUCTION

Today several hundred thousands of software programs exist, making it almost impossible for a single user to by herself decide what is good and what is bad. Of course tools to prevent and remove viruses and spyware have existed for a long time, but not all malicious programs are found due to juridical restrictions, i.e. the legal status of these applications are questioned, placing them in an grey-zone between good and bad software. This results in a large amount of applications that anti-malware developers are being cautious about removing, due to the potential for legal retribution. So, a “grey-zone” of questionable programs exists, hard for different protection program to handle and almost impossible for a single user to judge. Also, the availability of preventive software has been limited, already installed malicious software are found and removed but then the damage might already be done.

The inability of traditional anti-malware applications to handle, due to restrictions put upon them, the programs that exist in the previously mentioned grey-zone, leaves user unprotected. A complement, to using anti-malware software for deciding about unwanted programs, is to use a reputation system, i.e. ranking of new and previous unfamiliar software as a method for investigating the “true” appearance of a program. Using professional experts for doing this is both expensive and unrealistic due to the huge amount of non-investigated programs. Instead we propose a pool of ordinary users with different skills making necessary decisions about the quality of different software. However, there is still a need for more traditional anti-malware tools for targeting the clear-cut malware types that by no means could be placed inside the “grey-zone” between good and bad software, such as viruses and worms.

The purpose of this work is to investigate how many and what impact expert users need to have on a reputation system making it reliable, i.e. if it is possible to get a stable system by having few experts compensating for a vast majority of users with limited ability to rate an application. We simulate a reputation system with input from different skilled users and investigate a way of mitigating bad user ratings by using trust factors rewarding good users’ good actions and punishing bad actions. The goal of the simulation is to find critical parameters for correctly classifying large number of different programs with a realistic base of different skilled users.

The remaining part of this paper is organized as follows. First we discuss the related work in Section II and introduce the software reputation system in Section III. We continue in section IV by introducing the simulator software and in Section V we present the scenarios. In Section VI we present our results, which then are discussed in Section VII. We conclude by stating our conclusions and suggestions for future work in Section VIII and IX respectively.

II. RELATED WORK

Recommender systems are used to provide the user with an idea of other users’ thoughts about products, i.e. whether they are good or bad. These kind of systems are mostly used in commercial websites suggesting additional products, which the user might consider buying, exemplified in Amazon [2]. Recommender systems are not limited to commercial services, but also exist in other recommendation services such as Internet Movie Database (IMDb) [6]. IMDb uses recommender systems to suggest movies to users based on the opinions of other users that have shown similar tastes. Adomavicius and Tuzhilin provide, in their survey on the subject, a deep introduction of recommender systems, as well as some of the limitations [1].

eBay [5] makes use of a reputation system that allows users to rate buyers and sellers within the system, as a way to establish reputation among users. This reputation system makes it easier for users to distinguish dishonest users from trustworthy users. Experiments conducted by Resnick et al. also show that users with a higher reputation have a higher likelihood to sell items [10]. So, while recommender systems...
deals with the items involved, reputation systems instead deals with the involved users. In this paper we refer to our system as a reputation system due to the importance of the trust factors associated with the users.

Since reputation systems rely on the input of the users to calculate the ratings, it has to be able to establish trust between users and towards the system [7][11]. This is especially important when one considers the fact that the users of a software reputation system will have varying degrees of computer knowledge, and their ability to rate an application will thus be of different quality. There also exists the possibility of a user acting as several agents and actively reporting an erroneous rating in order to give a competitor a bad reputation or increase rating of a chosen object, i.e. a Sybil attack [4]. Even though this can be a potential problem to our proposed system, it is not within the scope of this paper to further analyze such scenarios. Furthermore there exist proposed solutions to this problem, for instance SybilGuard by Yu et al. [15].

The problem of erroneous ratings will, in a system such as IMDb, correct itself over time, but in a system such as the one proposed by Boldt et al. [3], where the intent is to advice on malicious software to users who might not be able to tell the difference, this presents a greater problem. Whitby et al. has put forth an idea of how to solve this problem [14] by filtering the unfair ratings, and their simulations show that the idea has merit. Traupman and Wilensky [13] try to mitigate the effects of false feedback in peer-to-peer applications by using algorithms to determine a users true reputation. However, these ideas might not be ideal under all circumstances, as they add another layer of complexity to the system, as well as another step of work to be done.

Jøsang et al. [7] summarize, among other things, different ways of computing the rating of an object and one of the conclusions is that reputation systems originating from academia have a tendency to be complex compared to industrial implementations. We have opted for a simpler system, where the rating is weighted by trustworthiness of the user.

Among simulations done on the area of reputations systems, Jøsang et al. has conducted a simulation on an e-market, concluding that reputation systems are necessary in order for the e-market to become healthy [8]. They also come to the conclusion that reputation systems should be configured to forget old ratings in order for new ratings to have impact, i.e. the system should be able to change opinion concerning an object.

III. SOFTWARE REPUTATION SYSTEM

As presented in the previous section, ranking of new and previous unfamiliar software is a common method for investigating the “true” appearance of a program before installing it. Using professional experts for doing this is both expensive and unrealistic due to the huge amount of software programs that are developed every year. Instead the opinions are gathered from a pool of ordinary voluntary users that agree to benefit from the common knowledge by providing ratings for the software they are most familiar with. In this way each participant is asked to rate software on a discrete scale (1 to 10) after they have used that software during a certain time-frame, i.e. the user have had time to form an opinion about that particular software program. The ratings given by the system users should be all-embracing, i.e. including different parts such as (but not limited to) the software’s features, behavior, usability, stability, performance and privacy aspects.

A. System Design

We propose a client-server based system where each user has a small client software installed through which it is possible for the user to both send and retrieve information from the central server that handles all software reputation calculations. The client identifies software by calculating hash digests on the actual program file data, e.g. SHA-256. This means that a software reputation is associated with each new version of a program, but the reputation of several subversions can be propagated up to one major version that is then presented to the user. It is also possible to calculate for instance the average rating of a certain software vendor based on the individual reputations of all programs that a particular vendor has developed.

In an attempt to get as accurate ratings as possible from the user, the client software asks the user to rate the software he/she uses most frequently, i.e. the user is familiar with the software and has an opinion about it to base the rating on. Each rating includes one mandatory field that represent an overall rating on some grading scale, in this case [1,10] inclusive. It is also possible for the user to provide additional information, but this is optional. We believe it is of great significance not to ask the users to provide too much information since many users would find this most annoying, and therefore provide random or no feedback at all. However, we believe computer users would accept to rate a few software per month if they in return get access to all previous users’ ratings for software programs that the user is considering installing.

To address the ever-existing problem with participants that provide false information to a collaborative system we incorporate user-individual trust factors (TF). This means that each user is assigned a TF that states the amount of influence each user have in the system. New users are always given the lowest possible TF, but as they use the system and prove to be trustworthy this value increases. Each time a user uses the client program to submit a rating it is forwarded to the reputation server for further processing. On the server-side the rating is compared to the average of all previous ratings on that particular software and if it is close then the user’s TF is increased, otherwise it is decreased. That way the TF of users that provide accurate ratings increases which give them more influence in the system, while it decreases for the rest of the users. Although the effect of this implementation is that the input from some users is amplified to dominate a large portion of the overall system, we believe it is important to include all users’ votes when calculating the resulting software ratings in the system. This way, even non-expert users such as novice and new users can rest assured that their voice is listened to.
It is of significant importance to make sure the users’ privacy is sufficiently protected in a software reputation system, since it handles sensitive information about what software each user have installed on their computer and their associated ratings. A situation where it would be possible to combine IP addresses with the information about what software these computers include could for instance reveal which computers that are vulnerable to certain remotely exploitable vulnerabilities. In addition to this it is also important to protect users’ privacy since one of the main goals of a software reputation system itself is to assist users in protecting against potentially malicious programs that invade privacy. It is therefore important to make sure that the system does not intrude on users’ privacy more than absolutely necessary. However, we still need to store some minimal amount of information about the user to address the problem with vote flooding, i.e. we need to distinguish between unique users’ votes for each software to guarantee that duplicate votes do not occur. A thorough description of the techniques and design choices used for this software reputation system is available in [3].

IV. SOFTWARE REPUTATION SYSTEM SIMULATOR

In this section we start by describing the design and workings of the simulator and then move on to explain how we modeled the users in our experiments.

A. Simulator Design

The simulator itself was implemented in Java and all configuration of the simulator is carried out through configuration files that allow the operator to fine-tune every aspect of the scenario that should be simulated. The simulator is deterministic, meaning that it is possible to rerun a scenario several times and always get the same results, or more interestingly to change a certain variable in the scenario setup and be sure that the changes in the end-result are due to the alteration of that particular variable.

Individual objects represent users and software that are simulated, i.e. one Java object per simulated user and software. These objects are stored in two different linked lists that keep track of all user and software objects. A simulation basically consists of iterating through the linked list of all user objects in sequence, allowing each user to rate a randomly selected software object, until the correct number of votes has been simulated. An important addition to this process is that the linked list of all user objects is shuffled before each iteration proceeds. At certain intervals, for instance every 10% progress, the simulator outputs various degrees of statistics depending on the particular configuration.

Each software object includes variables that store information about its “correct” rating, the number of votes it has received and the sum of all weighted votes, which makes it possible to calculate the software’s weighted average rating as explained in Equation 2 in the next subsection. The “correct” rating mentioned above is used for two purposes in our simulator. First, it is used for evaluating the accuracy of the simulated reputation system. Even though such a correct rating might not exist in the real world due to users’ subjective beliefs, we use them as a way to evaluate the accuracy of the simulated reputation system. Secondly it is also used when constructing the user’s vote as described in the next subsection.

The evaluation of a simulation consists of summarizing the absolute distance between software’s correct rating and weighted average rating, and finally dividing it with the number of software included in the simulation. The resulting value is the evaluation score (ES), i.e. the average distance from all software programs’ correct ratings. This score represent how accurate the simulated software reputation system is when providing software ratings to its users. One should always strive to reach an as low ES as possible, since an ES of 0.0 represent that the software reputation system on average provides its users with ratings that are 0.0 votes from its correct value. In other words bang on target. In the next subsection we present how the users in the simulations are modeled.

B. User Models

We have divided the simulated users into three groups based on their technical knowledge and accuracy in rating software. Each user simulated belongs to exactly one of these groups, which determines the users voting variance. Figure 1. shows each groups’ voting variance (or error rate), which lies within the interval [+5, -5] inclusive. The expert users in Figure 1. rate software correctly 50% of the times, and in the remaining part rate the software either one step below or above its correct rating, i.e. the expert users always manage to rate a software within a 1 step wide window around its correct rating. The second group is the average users that tries to rate software correctly, but with lesser accuracy than the experts, i.e. they rate up to 3 steps above or below the correct rating due to lack of skills. Still an average user is better than a novice user that has an error margin of 5 steps above or below the correct rating. Figure 1 also shows, as lines, the actual outcome of the distribution during our simulation. The discrepancy of these values are due to problems of giving the worst rated grades for certain types of programs that already are close to one of the rating scale borders. In such cases a new vote variance is randomized based on the user’s voting distribution in Figure 1., hence the greater probability for a voting variance close to 0.

![Figure 1.](image)

Figure 1. The voting variance for each of the three simulated user groups, which lies between +5 and -5 grades away from the software’s correct rating. The modeled voting variances are shown as bars, and the actual ones as lines.
In addition to the users’ own ability in rating software it is also possible for the simulator to simulate that they are influenced by previous user’s ratings. This would for instance occur when a user is unsure what rating to assign a certain software and therefore use that software’s current average rating as guidance when making up his/her mind. In our simulations we refer to this as the previous rating influence (PRI), which is represented as a number on the continuous scale [0-1] inclusive. We argue that the PRI effect increases as users become less confident about how to rate software. Experts are not very influenced by the already existing rating and thus have a relatively low PRI, in this case 6.25 %. The group of average users is more likely to be influenced, thus earning them a PRI of 12.5 %. The novice group on the other hand will most likely be very influenced by the already existing rating, and be more inclined to give a rating that is similar to the existing. To simulate this we give the novice group a PRI of 25 %. As shown in Equation 1, below users’ votes are generated by adding the user’s vote variance to the software’s correct rating, which results in that users from the different groups rate software differently.

\[
\text{Vote} = (1 - \text{PRI}) \times (\text{CorrectRating} + \text{VoteVariation}) + \text{PRI} \times \text{AverageRating} \quad (1)
\]

When no PRI is used a user’s vote is calculated by simply adding the software’s correct rating with the user’s randomized vote variance. However, when PRI is used the vote is instead pushed towards the software’s average rating to various degrees, based on the amount of PRI that is simulated.

An important aspect in the simulations is how to adjust the users’ trust factors. The simulator allows its operator to tune four different variables that directly control how the TF is being calculated. First of all the operator has to decide if the TF should increase or decrease in an exponential or linear fashion. Secondly, decide what the change-rate should be, e.g. if a linear value of 2.0 is used then TF would increase or decrease by 2.0 based on whether the user manages to pinpoint the software’s correct rating or not. If, on the other hand, an exponential value of 1.25 is used the user’s TF will either increase or decrease with a factor 1.25 based on the current value. The third variable that is available to the operator is the potential to include a maximum level, or ceiling, which the TF cannot exceed. Finally it is also possible to decrease the TF faster than it increases by enabling the decrease factor (DF), i.e. a DF of 1.5 will result in that a user’s TF decreases with a factor 1.5 more than it increases. The DF could be used as a sanction method against misbehaving or cheating users. However, it has not been further investigated in the experiments presented in this paper.

C. Simulation steps

During the initialization of the simulator each program is randomly assigned its “correct” rating which is used for evaluation purposes. Next, the users are assigned to the simulated groups according the proportions defined in the configuration files. Then the simulation starts and executes according to the following steps:

1. Shuffle the list of users
2. Sequentially select each user from the list
3. Randomly select a software
4. Randomize new vote variance for user
5. Create vote (Equation 1.)
6. Increase vote counter by 1
7. Update software’s weighted average (Equation 2.)
8. Update user’s trust factor
9. Repeat for each user in list
10. Repeat until specified number of votes are reached

The software’s weighted average score is calculated based on both the user’s vote and trust factor, as explained in Equation 2. This renders in that users’ votes are being weighted differently based on their individual trust factors, i.e. amplifying the votes from trustworthy users.

\[
\text{AverageRating} = \frac{\sum_{i} (\text{vote}_i \times \text{TrustFactor}_i)}{\sum_{i} \text{TrustFactor}_i} \quad (2)
\]

After each vote the user’s TF is updated based on how far away from the software’s current weighted average the vote is. If the vote is exactly the same as the weighted average the TF is increased, while it is kept as is if the vote is 1 step above or below. However, if the distance is further than 1 step the TF is decreased.

V. SIMULATED SCENARIOS

As seen in the background section, ranking of new and previously unfamiliar software is a common method for investigating the “true” appearance of a program before installing it. Using professional experts for doing this is both expensive and unrealistic due to the huge amount of non-investigated programs. Instead a pool of ordinary users with different skills is proposed where each participant repeatedly votes between 1 and 10 before new programs are installed. The voting is based on the user’s skill, but also on the previous rating of the program. A skilled user may improve his own reputation by repeatedly giving votes close to the “true” value, i.e. similar to the professional expert. By doing so it is possible to increase the value of the vote either by a linear or an exponential increase.

In a recommendation system different actors may appear. We used the previous described groups of experts, average and novice users in the simulation of our reputation system. All users have one equal valued vote to start with and are supposed to repeatedly rate new software. All simulations in this work include a fixed population of different skilled users with 9.4 % experts, 27.1 % average users and 63.5 % novice users. We decided to use these estimates based on the PEW Internet & American Life Project’s statistics of user demography of information technology users [9].

Within this population all groups give a vote based on actual skills, and in some cases also based on the influences from previous voters, i.e. the weighted average for that particular software. Some of the simulations also measures what effects scaling up or down the proportion of different users have on the system, e.g. how system accuracy is affected when scaling down the number of expert users by half.
The above-mentioned groups were simulated for one million users voting for 100000 different programs, i.e. it is unlikely that one user will vote more than once for a single program. We chose to include 100000 programs based on the number of software application included in Web-based reputation systems, e.g. Softonic [12]. One million users is argued to be a realistic number due to the fact that such a system is globally accessible and therefore benefiting from network effects. The scenarios that we simulate in this paper include 48 and 96 million votes, which represent a two or four years use of a system with one million users, and an average voting frequency of two votes/month.

VI. RESULTS

First we investigate how big an impact we may give a single voter without looking at the result other voters have given for the population of voters described above. Next, we look at the impact of previous rating influence where the different groups of voters are more or less influenced by the judgment already done. Then the proportions of experts, average and novice users are varied. Finally we investigate how system accuracy is when the correct rating for 25% of all simulated software programs are known before the simulation is started, i.e. that they are bootstrapped with the correct rating.

A. Trust Factors and Limits

Different trust factors varying in range from 1.05 to 2.0 were investigated with either linear or exponential increase. Each group of users starts with a TF set to 1.0 with an increase of the chosen trust value for each vote that is placed within one step above or below the software’s current average rating. If the vote is up to one step away from the correct value the TF is left unchanged. Otherwise, i.e. two steps or more, the TF is decreased by the same trust factor. A user can never have a TF lower than 1.0. Each participant voted 48 times each and in all 1 million users voted for 100000 programs in this simulation. Figure 2. shows the outcome for the chosen number of different users and various maximum TF limits.

Figure 2. 9.4% expert, 27.1% average and 63.5% novice users voting with a varying trust factor during 48 votes each.

The linear outcome for different TFs shows a very limited improvement with increasing TFs and thus is not further investigated. The lowest average distance was reached for an exponential TF or 1.25 where both lower and higher TF showed worse performance. For this reason we decided to use a 1.25 exponential TF during the rest of the simulations.

Next we investigated the need for a maximum limit of the user’s TF. Figure 3 shows what happens when the following limits are specified for the TF; 1, 10, 100 and 1000 and unlimited.

An unlimited TF settles around 0.5 from the correct value but may give a single voter an un-proportional big impact on the voting system. Limiting the TF to 1000 is a reasonable compromise where the first 25 votes will behave as unlimited and settles around 0.7 with a slightly increase during prolonged voting time. This has to do with a relative TF increase of less skilled average users due to the expert reaching the 1000 TF limit as shown in the next figure. Figure 4 shows the outcome for each category on a logarithmic scale. The novice user hardly reaches above 1 where the average user has a small but constant increase. However, the TF of the expert users reaches 10, 100 and 1000 respectively when these limits are specified, and reaches 1 million after 48 votes when no limit at all is used. So, in all cases the expert voter has a dominant position, but not a 100% voting accuracy. As even expert users commit errors when voting they should not be given unrealistically high TF. In our setting an expert is predicting the correct value 50% of the time, i.e. the impact from an expert with huge TF giving a wrong value distort the result from the correct decision. The “knee” on the curve associated with the experts in Figure 4 is due to the lack of measurements for TF limits between 10000 and unlimited.

Figure 3. 9.4% expert, 27.1% average and 63.5% novice users voting with a 1.25 exponential trust factor during 96 votes each.

B. Previous Rating Influence

Some commercial recommendation systems make previously given votes available to the users when they
decide on their vote, i.e. the previous rating influence (PRI). This can give both positive and negative consequences that should be considered. The different user groups have different levels of knowledge, and thus are guided by the already existing rating to different degrees. To measure the effects that PRI have on a software reputation system we simulated a scenario where each user-group were influenced by 6.25 %, 12.5 % and 25 % for expert, average and novice users respectively. When for instance a novice user rate a software his vote is to 75 % decided based on his level of knowledge and to 25 % based on that program’s current average rating. Figure 5. shows the results of this simulation.

As seen in Figure 5, there is a slight improvement in the beginning of the simulation when PRI is used. However, performance then stabilizes around the same levels as when no PRI is used. There is for instance no noticeable improvement in the case with a TF limit of 1000 compared to the results in Figure 3. When activating PRI in the simulation both novice and average users will improve their marking ability, which in turn result in that their TF increases. This can be seen in Figure 6 where the trust factor of both the novice and average users has increased several times compared to the results in Figure 4.

The results in Figure 6 show that the TF of all user groups has increased when PRI is enabled. This can be attributed to the fact that for instance the novice group has a higher tendency to follow the already set ratings, i.e. the results from the average and expert users. Due to this, they will rate an application more accurately, which will increase their trust in the system. As already stated, this results in a higher TF for the novice group. Based on the size of the novice group they will have a higher impact on the system. Even though the novice group has a higher tendency to follow the average ratings, they will still introduce votes that are distanced from the correct rating of software, which deteriorates the overall system accuracy. We have also simulated scenarios with higher PRI values for all user groups, but without any significant improvement of the system accuracy. These results therefore show that the system accuracy is not significantly improved when PRI is enabled and that the reputation system is more stable without it.

C. Demography Variations and Bootstrapping

Figure 7 shows how the user demography of the simulated users affects the system accuracy. We varied the size of the different user groups around the survey results presented by PEW Internet & American Life Project. We can see that the size of the expert and average groups clearly make a difference in the beginning of the simulation, i.e. higher number of expert and average users increases system accuracy. However, as the simulation progresses and the TF of each simulated user is fine-tuned the system accuracy increases for all user constellations. An increased number of experts perform better than increasing the number of average users in relation to novice users.

To improve the system accuracy further we also simulated a scenario where already available software reputations were collected from third parties and used to initialize the software reputation database before it was put into use. Figure 8 shows how the accuracy of the system is affected when 25 % of the software inside the reputation database is bootstrapped with trusted data with a total value of 5000, i.e. equal to five votes from full-fledged experts. When compared with Figure 7 it is clear that the system accuracy is positively affected in the beginning of the simulation, but as the simulation continues the difference between whether bootstrapping is used or not decreases.
In this investigation we bootstrapped the reputation for 25% of the software with reputation data from trusted sources.

We also ran a set of experiments without any expert users at all just to investigate whether or not the software reputation system would still function properly in such a scenario. When omitting all expert users, leaving 27% average and 73% novice users, the system shows an accuracy score of approximately 1.8. in the beginning of the simulation which then improves towards 1.2. Finally we also simulated the use of a software reputation system during an extended period of time, in this case 1200 votes per user, without seeing any tendency of degraded system accuracy. Even though the scenario is questionable, since none of the software programs are updated, it doesn’t show any evidence of decreased performance, i.e. the reputation system seems to be stable. In fact the accuracy is continuously improving through out the simulation during this extended simulation, and the overall system accuracy stops at 0.95. During the whole simulation the TF of the novice users are quite low, with an average of 2.8. At the same time the TF of the average users stabilize around 890 while the experts quickly reach the TF limit of 1000.

VII. DISCUSSION

The idea behind the simulated reputation system is to reward users that provide accurate ratings and punish faulty or not properly thought through decisions, which will improve the software reputations within the system. The experiments assume that there exists a “correct” rating of each software for initial settings and evaluation purposes. However, whether such a correct rating exists in a real world setting is not necessarily true. It is therefore hard for a single person to define exactly what the correct rating for the particular software should be, but based on several users’ ratings it is possible to come to a common compromise, which then is used as a baseline when deciding whether or not to increase or decrease users’ TFs.

In this work we have used simulation as the means to identify how the users’ TF should be adjusted within a software reputation system to reach an accurate, stable and sustainable system to mimic the view of a professional expert. In this investigation we primarily look at the behavior of the voter. From a single program’s point of view different skilled voters may end up voting close to the “correct value”, i.e. on average votes below and above this value are compensated. So, the average value for a single program is better than the user’s performed absolute distance value. By introducing a trust factor and/or previous rating influence the overall performance for various groups of users was increased, i.e. the absolute distance value coming closer to the average value. This was true even in groups dominated by unskilled novice and average users making the outcome of the voting procedure more robust.

Unlike professional experts the simulated population consists of different skilled users voting twice a month during two or four years. In the beginning of the simulations each voter has given a very limited amount of votes, i.e. only a small fraction of all available programs are being rated by a single voter. The simulation results show an exponential increase of the TF to be better than a linear increasing, and that a factor of 1.25 was most promising for an accurate system. Through the simulations we also found that an upper TF limitation of 1000 is preferable. If this limitation is being omitted the TF of the expert users quickly increases to levels that make the whole system unstable because of these users’ small, but still existing, rating fluctuations. Therefore the upper TF limit creates a balance between fast reaching and well-informed ratings, without giving a single user too much influence. In a real situation other factors, such as malicious behavior, makes this argument even more important within the system. If a single user can get extremely high TF values it is possible for malicious actors to use this to manipulate the rating of the particular software for their own personal gain.

Intuitively it might seem interesting to allow users to see all previous ratings (PRI) when they make up their mind about how to rate a certain software, since this at first could be thought to improve the decisions of the less skilled users. However, if the system makes use of TFs this is not the case since novice and average users will improve their TFs, i.e. their normal voting variance will exceedingly negative influence the program evaluation. Through the simulations we also investigated how the proportion of experts, average and novice users affects the system accuracy and stability. By increasing and decreasing the number of experts and average users we were able to draw the conclusions that the system accuracy improves as expert users sign up to the system. For all investigated populations the system accuracy improves from initial not-to-well judgments in the beginning of the simulation to closer to the correct ratings, for instance when 85% novice users and less than 5% experts are being simulated.

In our simulations we have also showed that it is possible to improve the initial system accuracy before the system is made publically available by bootstrapping the database with trusted reputation data. Such bootstrapping data could for instance be gathered from web-based services available on the Internet. Furthermore we also show that the system is stable and accurate when users submit ratings with a higher frequency. In this case when each user submits on average
1200 votes over a four years period, i.e. about one votes per day.

When summarizing all simulated scenarios we come to the conclusion that it is possible for computer users to rely on a stable software reputation system that assist them when identifying well-reputed software, as well as when avoiding questionable programs.

VIII. CONCLUSIONS

A software reputation system consisting of expert, average and novice users was simulated as an alternative to let anti-malware programs or dedicated human experts decide about questionable programs. Within the simulated population, the different skilled users voted twice a month during at most four years. Each voter starts with a single vote and by varying the increase of the trust factor (TF), the upper TF limit and previous rating influence (PRI), with a resulting balanced, well-informed rating of judged programs as the proposed outcome.

An exponentially increased TF was better than a linear, and a system that allowed voters to see previous users’ votes performed better, i.e. with PRI. More precisely an exponential increase of 1.25 for the TF with a TF limit of 1000 within an experimental setting of less than 5% experts in a population exceeding 80% novices still performed well. Such a setup allowed a balance between quickly reaching a well-informed decision and not giving a single voter too much power.

In our opinion the reputation systems will become a more commonplace advisory tool in the future with the possibility to provide an advice to the user that most likely will be helpful, i.e. being able to handle erroneous, good and bad ratings, and without losing the integrity of the system.

IX. FUTURE WORK

Our simulated environment lacks some real world parameters that will be further investigated as the work progresses. First, we will simulate various attack schemes that are used by malicious actors to gain control over the reputation system. Secondly, we will investigate how system accuracy is affected when users are handled more dynamically, e.g. when new users join up during the simulation and that established users’ leaves. Finally, this should also include scenarios where new and unknown programs are being added on the fly or when old programs are being overridden.

X. REFERENCES