Voting Systems with Trust Mechanisms in Cyberspace: Vulnerabilities and Defenses

Qinyuan Feng, Yan Lindsay Sun, Ling Liu, Yafei Yang, Yafei Dai

Abstract—With the popularity of voting systems in cyberspace, there is growing evidence that the current voting systems can be manipulated by fake votes. This problem has attracted many researchers working on guarding the voting systems in two areas: relieving the effect of dishonest votes by evaluating the trust of voters, and limiting the resources that can be used by the attackers, such as the number of voters and the number of votes. In this paper, we argue that powering voting systems with trust and limiting attack resources are not enough. We present a novel attack named as Reputation Trap (RepTrap). Our case study and experiments show that this new attack needs much less resources to manipulate the voting systems and has a much higher success rate compared with existing attacks. We further identify the reasons behind this attack and propose two defense schemes accordingly. In the first scheme, we hide the correlation knowledge from attackers to reduce their chance to affect the honest voters. In the second scheme, we introduce the robustness-of-evidence, a new metric, in trust calculation to reduce their effect on the honest voters. We conduct extensive experiments to validate our approach. The results show that our defense schemes not only can reduce the success rate of the attacks but also significantly increase the amount of resources an adversary needs to launch a successful attack.

Index Terms—Voting system, Trust mechanism, Reputation system, Security.

1 Introduction

Research has shown that *online user opinions* are having increasing influence on consumers' decision-making, providing incentive for good behaviors and positive impact on market quality [19], [26], [28]. There are various systems that *collect* and *aggregate* users' opinions. These opinions are referred to as *online voting, rating or review*.

In this paper, we use *online voting systems* to refer to the services that judge the quality of items based on users' opinions[1]. Here, items can be products, transactions, digital contents, search results, and so on. There are many popular operational online voting systems. In Amazon, users give one to five stars to products. Digg is one of the most popular websites for people to discover and share content on the Internet. The cornerstone function of Digg allows users to vote a story either up or down. In P2P file-sharing systems, voting systems are used to identify whether a file is real or fake [16].

However, the *manipulation* of voting systems is also rapidly growing. Examples include buying a joke for a penny to gain a positive feedback in eBay [18], buying fake positive review for 65 cents [37], and winning "Best Producers of Short Content in Twitter" award through purchasing online votes [38]. The attackers first control a lot of fake user IDs [4] and then design their strategies carefully to achieve their goals [20], [23], [27]. Researchers as well as Internet users have already realized this problem. If fake votes are not under control, they will damage the fairness and usability of online voting systems, mislead consumers, and hurt the businesses hosting voting systems in the long run.

Nowadays, *trust mechanism* is used as a popular and yet effective approach to address the fake voting problem. Various trust schemes have been proposed [3], [6], [9], [16], [20], [26], [29], [33]. They estimate whether a user is trustworthy or not, and give low weights to the votes from less trustworthy users when generating the final results. These schemes have shown to increase the robustness of online voting systems.

Another popular approach to defend online voting systems is to limit the resources that can be used by the attackers. The resources can be the number of user IDs [4] under an attacker's control and the maximum number of votes can be inserted by these user IDs. Several research efforts [17], [32], [36], [39] have recently been put forth along this direction.

In this paper, we argue that powering the voting systems with existing trust mechanisms and limiting attack resources are not enough. We present a novel attack that can manipulate the voting systems with limited resources, analyze the inherent vulnerabilities utilized by this attack, and propose two defense schemes. Concretely, this paper makes three unique contributions:

First, we introduce an attack model that inspires the discovery of a new attack, named as Reputation Trap or RepTrap in short. Compared with existing attacks, RepTrap needs less resource and has higher success rate to achieve the attack goal. The basic idea of RepTrap is as follows. The malicious users first identify a set of high-quality unpopular items whose voters also vote for the target item. Then, the malicious users give a large

- *Q. Feng and Y. Dai are with Peking University, Beijing, China. Feng is a visiting student in Georgia Institute of Technology when this work is done.E-mail: {fqy,dyf}@net.pku.edu.cn*
- *Y. Sun and Y. Yang are with University of Rhode Island, Kingston, RI, USA.E-mail: {yafei, yansun}@ele.uri.edu*
- *L. Liu is with the College of Computing, Georgia Institute of Technology, Atlanta, GA, USA. E-mail: lingliu@cc.gatech.edu*

1. Our definition of online voting system is much broader than the definition used in the election system.

number of dishonest votes to these unpopular items such that the system wrongly believes these items have low quality. As a result, the votes from the honest users, which disagree with the item quality from the system's point of view, are considered as dishonest, and the trust of the honest users is reduced. Meanwhile, the trust of the malicious users is increased. Finally, after increasing their own trust and reducing the honest users' trust to certain degree, the malicious users can successfully attack the target item.

Second, we analyze and identify the key factors that facilitate RepTrap. These factors include the lack of ground truth about the quality of the items, availability of correlation knowledge, attackable unpopular items, and information shrinking in voting aggregation.

Third, we propose two defense schemes to increase the difficulty of launching RepTrap successfully. *Knowledge hiding* is used to prevent the attackers to find correlation among items; *robustness-of-evidence* is introduced as a new metric to evaluate the difficulty of undermining the voting system's judgment on the quality of an item.

We conduct extensive experiments to compare RepTrap with two known types of attacks: bad-mouthing (referred to as RepBad in this paper) and self-promoting (referred to as RepSelf in this paper) [8], [20], [27]. We also evaluate the proposed defense schemes under various attack scenarios. The results show that *RepTrap is much stronger than known attacks in terms of attack success rate, even when voting systems are powered with existing trust mechanisms and the attackers can only use limited resources to conduct the attack*. For example, with 5% of the user IDs controlled by the attacker, RepTrap can successfully attack the most popular item with more than 90% success rate, whereas the success rate of RepBad and RepSelf is almost zero. Meanwhile, RepTrap can reduce the total number of votes from the attacker by more than 50%. Furthermore, the *proposed defense schemes can successfully reduce the power of RepTrap*. Concretely, each defense scheme alone can reduce the success rate of RepTrap by up to 30% and increase the number of required malicious votes by up to 50%. The combination of the two defense schemes is significantly more effective. We argue that through the in-depth analysis of RepTrap and the use of the proposed defense schemes, we put the arms race between attacks and defenses for online voting systems one step forward.

The rest of the paper is organized as follows. In Section 2, we present a reference architecture for voting systems powered with trust mechanisms and review the related work. We develop the attack model and describe RepTrap in Section 3. We analyze the reasons behind RepTrap in Section 4 and develop defense schemes in Section 5. We report our experimental results in Section 6 and conclude the paper in Section 7.

2 Background and Related Work

In this section, we first present a reference architecture for voting systems with trust mechanisms. We then build a concrete system of majority voting with beta function trust mechanism. This system will serve as the platform for presenting RepTrap in later sections. At the end of this section, we review the related work.

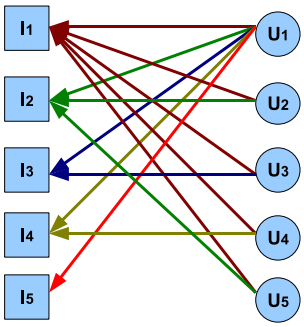


Fig. 1. Item-user graph in a voting system

2.1 Voting systems with trust mechanisms

In a voting system, there are *items* whose quality should be identified by the system and *users* who can vote on the items. The basic function in a voting system is users giving votes to items, which can be described by an item-user graph as illustrated in Figure 1. Each vertex on the left hand side represents an *item*. Each vertex on the right hand side represents a *user*. An edge between an item and a user means that this user votes on this item. To simplify the discussion, we assume that the quality of an item is binary and the users' votes are also binary.

As discussed before, trust mechanisms are used to deal with dishonest votes. Figure 2 shows the architecture of voting systems with trust mechanisms.

- **Raw voting data** is collected from the users.
- **Item quality** is calculated based on the voting values and the user trust. The algorithm that calculates the item quality is referred to as the *IQ Algorithm*.
- **User trust** is calculated based on users' votes and item quality. Briefly speaking, if a user's voting values disagree (or agree) with the item quality identified by the IQ Algorithm, this user's trust value would be reduced (or increased). The algorithm that calculates the user trust is referred to as the *UT Algorithm*.

2.2 Majority voting system with beta function trust mechanism

With the reference architecture, we present a majority voting system with beta function trust mechanism for P2P file-sharing networks. This system will be used to demonstrate the attack and defense ideas as well as the experimental results in later sections. We should note that the application can be easily changed to others.

In a P2P file-sharing network, the items are the files. After a user downloads a file, he/she can vote whether the quality of the file is real ('1') or fake ('0'). The voting system collects the votes and calculates the file quality, describing the system's judgment on whether a file is real or fake.

For voting systems, one of the most popular designs is based on majority rule [1]. That is, the decision made

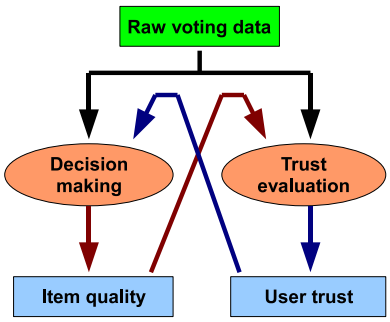


Fig. 2. Architecture of voting systems with trust mechanisms

by the system should agree with the majority's opinion. Specifically, for each item, let n_0 denote the number of users who vote 0 and n_1 denote the number of users who vote 1. The decision is 0 if $n_0 > n_1$ and 1 if $n_0 < n_1$.

For trust mechanisms, one of the most popular design is based on beta function. It first counts the number of honest and dishonest behaviors a user has conducted, and then calculates the trust value with beta function [3]. If we combine them together, the system is as follows.

- **Item quality algorithm**: Let U_k^1 and U_k^0 denote the set of users who vote item I_k with 1 and 0, respectively. The trust value of user u_j is denoted by $T(u_j)$. Then, the quality of item I_k, denoted by $Q(I_k)$, is calculated with majority voting as

$$Q(I_k) = \begin{cases} 1 & \text{if } G(I_k) \geq B(I_k), \\ 0 & \text{if } G(I_k) < B(I_k). \end{cases} \quad (1)$$

where

$$G(I_k) = \sum_{u_j \in U_k^1} T(u_j) \; ; \; B(I_k) = \sum_{u_i \in U_k^0} T(u_i). \quad (2)$$

In this calculation, the system utilizes the trust values as weight factors, and compares the summation of the weights of the users who vote 0 and that of the users who vote 1.

- **User trust algorithm**: When a vote is given to item I_k by user u_j, if the vote value is the same as the item quality $Q(I_k)$ identified by item quality algorithm, this vote is considered as an honest vote. Otherwise, it is considered as a dishonest vote. For a user u_j, the system calculates the number of honest votes given by this user, denoted by $H(u_j)$, and the number of dishonest votes given by this user, denoted by $D(u_j)$. The trust value of user u_j is calculated with beta function [3] as:

$$T(u_j) = \frac{H(u_j) + 1}{H(u_j) + D(u_j) + 2}. \quad (3)$$

The user trust and item quality is calculated in an iterative way as follows.

1) For each user u_j, initialize $T(u_j)$ as 0.5.
2) For each item I_j, update $G(I_j)$ and $B(I_j)$ using (2), and calculate $Q(I_j)$ using (1).
3) For each user u_j, update $H(u_j)$ and $D(u_j)$, and calculate $T(u_j)$ using (3).
4) If $T(u_j)$ for any user u_j changes in step 3, go to step 2; else END.

2.3 Related work

Online voting systems have been adopted in many real-life systems, including Google, Digg, eBay, Amazon, IMDB and so on. Most of them simply present raw data or conduct straightforward aggregation (e.g. the average voting scores). Two types of attacks have been identified in the current online voting systems. In the first type of attacks, the attacker can register many user IDs [4] and then manipulate the quality of the target item by inserting dishonest votes directly. This is often referred to as the bad-mouthing attack [20] [27], and we use **RepBad** to denote this attack.

Trust mechanisms are developed to defeat the bad-mouthing attack by calculating and utilizing the trust values of the voters. Reputation and trust has been well studied in the area of P2P networks. Representative schemes include EigenTrust [6] for reputation management in P2P networks, PET [15] for reputation and risk evaluation in P2P resource sharing, Credence [16] for identifying fake files in P2P file-sharing systems, PeerTrust [12] for supporting reputation-based trust for P2P electronic communities, PowerTrust [29] and GossipTrust [33] for trust aggregation in P2P networks. Ooi et al. [5] examined the issue of managing trust in P2P systems. Vu and Aberer [30] proposed a probabilistic framework in the computation of quality and trust in decentralized systems. Damiani et al. [9] proposed digital reputations that can be seen as the P2P counterparts of client-server digital certificates. Sensoy et al. [22] proposed an ontology-based service representation and selection approach. Khambatti et al. [11] proposed a role-based trust model for P2P communities and dynamic coalitions. Fan et al. [14] proposed a reputation system based on exponential smoothing that can serve as a sustained incentive mechanism for the sellers. A good survey about trust and reputation systems for online service provision can be found in [26] and a comparison of different rating aggregation algorithms can be found in [31]. In addition, a lot of work [13], [21], [24], [25] has discussed trust negotiation as an approach for establishing trust in open systems.

Though trust mechanisms make an adversary's attack more difficult to succeed, these mechanisms do not prevent the attackers to misuse and abuse the trust evaluation schemes. For example, the attackers can strategically give honest votes to the items that they are not interested in for the purpose of boosting their trust value, before giving dishonest votes to the items that they really want to attack. This is the second type of attacks and it is often referred to as self-promoting [27] or reputation recovery [20]. In this paper, we use **RepSelf** to denote it.

To counter RepSelf, the key is an effective identity management scheme to prevent Sybil attack [4]. It means limiting the resources can be used by the attackers. Two kinds of defense schemes are designed accordingly. The first kind is trying to introduce a process to judge whether a user is a human or a machine. In this process, a task which is easy to human but hard to machine

is introduced, such as the reCAPTCHA. The second kind is based on the structure of the trust graph. The assumption is that though the attackers could control a lot of fake IDs, but it is hard for them to build the correlation — which is cut in the trust graph — from the bad clique to the good group. Some latest work includes SybilGuard [17], SybilLimit [32], Sybil-Resilient [36], and Self-Certification [39].

In this paper we argue that simply combining existing trust mechanisms with the techniques of limiting attackers' resources is insufficient. We present a new attack, named as **RepTrap**, which is much stronger than any known attacks such as RepBad and RepSelf. We further study the vulnerabilities of online voting systems under RepTrap and possible defense schemes.

3 Reputation Trap attack

In this section, we formalize our problem and present an attack model. Afterward, we propose RepTrap with case studies and formal attack procedure.

3.1 Problem formulation

The goal of the attackers is to manipulate the system's judgment on the quality of certain items. These items are referred to as the target set, represented as S_T. Without loss of generality, we assume that the true quality of these items is high, and attackers want the system to mark them as low quality.

Notations in this section are as follows.

- The items are denoted by I_1, I_2, ... , I_n, where n is the number of items in the system. Let $Q(I_j)$ denotes the quality of item I_j identified by the system. Its value is either 0 or 1.
- There are N_H honest users, denoted by U_i, $i = 1, \cdots, N_H$. There are N_M malicious users, denoted by X_i, $i = 1, \cdots, N_M$.
- *Vote allocation of honest users*, denoted by $\mathbf{V_H}$, is an N_H by n matrix. Its element is denoted by $V(U_i, I_j)$, the voting value given by user U_i to item I_j. Here, $V(U_i, I_j) = 1$ if U_i votes '1' to I_j; $V(U_i, I_j) = -1$ if U_i votes '0' to I_j; $V(U_i, I_j) = 0$ if U_i does not give vote to I_j. Obviously, if U_i has voted on I_j, $|V(U_i, I_j)| = 1$; otherwise $|V(U_i, I_j)| = 0$.
- *Vote allocation of malicious users*, denoted by $\mathbf{V_M}$, is defined in a similar way. Its element is $V(X_i, I_j)$, takes value 1, 0, or -1, describing the voting value given by X_i to item I_j. The total number of votes from malicious users is $K_M = \sum_{i=1}^{N_M} \sum_{j=1}^{n} |V(X_i, I_j)|$.

The attackers would like to achieve their goal using minimal resources, and the resources are represented by N_M and K_M. In most cases, the number of malicious users is fixed. Thus, we formulate the problem as minimizing the number of malicious votes (i.e. N_M), given K_M, n, S_T, and $\mathbf{V_H}$, under the constraint that the items in the target set (S_T) is marked as low quality. In particular,

$$\begin{aligned} \text{Inputs:} \quad & N_M, n, S_T, \mathbf{V_H} \\ \text{Outputs:} \quad & \mathbf{V_M} \end{aligned}$$

The optimization problem:

$$\min_{\mathbf{V_M}} \sum_{i=1}^{N_M} \sum_{j=1}^{n} |V(X_i, I_j)| \tag{4}$$

under constraint:

$$Q(S_T) = 0 \tag{5}$$

This optimization problem does not have closed form solution. To find the optimal $\mathbf{V_M}$, we need to determine the values of $n \cdot N_M$ elements of $\mathbf{V_M}$, and each element has 3 possible values. So the computation complexity to enumerate all the state space is $O(3^{n \cdot N_M})$. Since practical systems often contains a very large number of items (n) and the number of malicious users (N_M) can range from a few tens to a few hundreds, it is impossible to enumerate all of them. Therefore, a heuristic algorithm will be developed in Section 3.4.

In some other scenarios, the attackers' goal may be minimizing the number of malicious IDs involved given a limited number of malicious votes. In these cases, we formulate another optimization problem as minimizing N_M, given K_M, n, S_T, and $\mathbf{V_H}$, under the constraint that $Q(S_T) = 0$. This optimization problem can be solved through a binary search as long as the problem in (4) is solved. Let N_M^{max} denote the maximum number of malicious users that could be controlled by the attacker. The solution is as follows.

1) $R_{left} = 0$, $R_{right} = N_M^{max}$
2) Solve (4) by setting $N_M = \lfloor \frac{R_{left}+R_{right}}{2} \rfloor$. Assume the solution requires the malicious users to insert K_M^* malicious votes.
3) If $K_M^* > K_M$, $R_{left} = \lfloor \frac{R_{left}+R_{right}}{2} \rfloor$ and Goto step 2.
4) If $R_{left} = R_{right}$, Output N_M and End.
5) $R_{right} = \lfloor \frac{R_{left}+R_{right}}{2} \rfloor$ and Goto step 2.

The computation complexity of conducting the binary search is $O(\log(N_M^{max}) \times$*cost of solving (4)*$)$.

3.2 Attack description model

From the previous analysis, we know that it is impossible to enumerate all the possible attack strategies. This motivates us to propose an attack description model to launch the attack heuristically.

In this model, we classify all the items in the system into 4 categories.

- **Target set (S_T)** contains the items that are the attacker's targets. In other words, *the attacker's goal* is to make the system falsely identify the quality of the items in S_T.
- **Honest set (S_H)** contains the items that receive honest votes from the malicious users.
- **Dishonest set (S_D)** contains the items that do not belong to S_T and receive dishonest votes from the malicious users.

- **Unrelated set (S_U)** contains all the items that do not receive votes from the malicious users.

To launch an attack on S_T, the attacker needs to (1) specify S_H and S_D, and (2) choose the malicious user IDs that will vote on the items in S_T, S_H and S_D. This model can easily describe the two known attacks. When $S_T \neq \phi$, $S_D = \phi$ and $S_H = \phi$, it describes RepBad in which the attackers only give dishonest votes to S_T; when $S_T \neq \phi$, $S_D = \phi$ and $S_H \neq \phi$, it describes RepSelf in which the attackers will first give honest votes to S_H to gain trust and then give dishonest votes to S_T.

Based on this attack description model, we naturally ask a question: is there any benefit for the attacker to vote in S_D? This question leads to the discovery of RepTrap, which has non-empty S_D and is more powerful than the existing attacks.

3.3 Case studies

To illustrate the basic idea of RepTrap and compare the effectiveness of various attacks, we present a detailed example in a simple application scenario.

3.3.1 A simple application scenario

We create a scenario with 5 items: I_1, I_2, I_3, I_4, and I_5, and 5 honest users: u_1, u_2, u_3, u_4, and u_5. All the items have high quality and therefore all the honest users vote '1'. Here, u_1 gives five votes; while u_2, u_3, u_4 and u_5 each gives two votes. The relationship between the items and the users is illustrated in Figure 1.

When there are no malicious users, we calculate the trust values of users as

$$T(u_1) = 0.86;\ T(u_2) = 0.75;\ T(u_3) = 0.75;$$
$$T(u_4) = 0.75;\ T(u_5) = 0.75.$$

The calculation is based on equations in Section 2.2. And the item quality is calculated as

$$Q(I_1) = Q(I_2) = Q(I_3) = Q(I_4) = Q(I_5) = 1.$$

So the system marks all the items as high quality.

3.3.2 Goal of attacker

In this case study, the **attacker's goal** is to make the system mark $S_T = \{I_1\}$ as a low quality item while minimizing the *attack effort*.

The attack effort is described by two values (N_M, K_M), the number of malicious users and the total number of malicious votes respectively. For the attacker, acquiring user IDs is usually more difficult than providing votes, since there are techniques to defend against creating multiple fake IDs [17], [32], [36]. Therefore, reducing the N_M value has higher priority than reducing the K_M value, from the attacker's point of view. Thus, the attacker first tries to minimize the N_M value, and then minimize the K_M value given the minimal N_M value.

From equation (1), we know the item I_k is marked as low quality when $G(I_k) < B(I_k)$. So if the summation of trust values of malicious users, who vote on I_k with '0', is larger than $G(I_k)$, the system will mark item I_k as low quality.

Based on the above discussion, the attack goal is translated to finding a specific way to insert malicious votes such that (1) N_M value is minimized and (2) K_M value is minimize given the minimal N_M value under the constraint that $G(I_1) < B(I_1)$.

3.3.3 Comparison among different attack strategies

In the simple scenario described in Section 3.3.1, all five items have high quality and receive votes with value '1'. When there is no attack, the $G(I_k)$ value, referred to as *hardness value*, is calculated by equation (2) as

$$G(I_1) = 3.86;\ G(I_2) = 2.36;\ G(I_3) = 1.61;$$
$$G(I_4) = 1.61;\ G(I_5) = 0.86.$$

RepBad: In the RepBad attack, the trust for each malicious user is 0.5, and the hardness of I_1 is 3.86. Since $3.86/0.5 = 7.72$, at least 8 malicious users are needed to successfully attack $S_T = \{I_1\}$. In this case, the attack effort is $(N_M = 8, K_M = 8)$.

RepSelf: In the RepSelf attack, the malicious users first provide honest votes to the items that are not the target of the attack. In this example, they can provide honest votes to I_2, I_3, I_4 and I_5, and accumulate their trust value up to $(4+1)/(4+2) = 0.83$. Since $3.86/0.83 = 4.65$, at least 5 malicious users are needed.

Not all 5 malicious users need to provide 4 honest votes. After enumerating all the possible ways to perform self-promoting, we find that the malicious users need to provide at least 18 votes. Specifically,

- Two malicious users provide honest votes to I_2 and I_3, and their trust values become $\frac{2+1}{2+2} = 0.75$.
- Three malicious users provide honest votes to I_2, I_3 and I_4, and their trust values become $\frac{3+1}{3+2} = 0.80$.

Next, the 5 users vote '0' to item I_1, the $B(I_1)$ value is calculated as $B(I_1) = 0.75 \times 2 + 0.80 \times 3 = 3.90 > 3.86 = G(I_1)$. Then, the attack goal is achieved with 5 users, 13 votes to $S_H = \{I_2, I_3, I_4\}$, and 5 votes to $S_T = \{I_1\}$. The attack effort is $(N_M = 5, K_M = 18)$.

RepTrap: In RepTrap, the attacker puts some high quality items in S_D, then provides a sufficient number of dishonest votes to these items such that these items are marked as low quality. If an item in S_D is falsely marked by the system, this item is turned into a **"trap"**, which will hurt the trust values of the users who give honest votes to this trap. In particular, when the system makes wrong judgment about the item quality, the system thinks the attacker's dishonest votes agree with the item quality and the honest votes from honest users disagree with the item quality. As a consequence, the trust values of the malicious users will be increased whereas the trust values of honest users will be reduced.

It is difficult to find the optimal way to conduct RepTrap. Instead, we just show one method to successfully attack I_1 using RepTrap. The attack effort of this method is $(N_M = 3, K_M = 13)$.

1) The initial trust values of the three malicious users are 0.5. Since $0.5 \times 2 = 1$ is larger than $G(I_5) = 0.86$, they can turn I_5 into a trap by providing two

TABLE 1
Trust values and hardness values change in RepTrap

	Initial state	After 1^{st} round	After 2^{nd} round	After 3^{rd} round	After 4^{th} round
$T(u_1)$	0.86	0.71	0.57	0.43	0.29
$T(u_2)$	0.75	0.75	0.75	0.75	0.50
$T(u_3)$	0.75	0.75	0.75	0.50	0.50
$T(u_4)$	0.75	0.75	0.50	0.50	0.50
$T(u_5)$	0.75	0.75	0.75	0.75	0.50
$T(X_1)$	0.50	0.67	0.75	0.80	0.83
$T(X_2)$	0.50	0.67	0.75	0.80	0.83
$T(X_3)$	0.50	0.50	0.67	0.67	0.75
$\sum_{i=1}^{3} T(X_i)$	1.50	1.84	2.17	2.27	2.41
$G(I_1)$	3.86	3.71	3.32	2.93	**2.29**
$G(I_2)$	2.36	2.21	2.07	**1.93**	#
$G(I_3)$	1.61	1.46	**1.32**	#	#
$G(I_4)$	1.61	**1.46**	#	#	#
$G(I_5)$	**0.86**	#	#	#	#

dishonest votes to I_5. This has three consequences. First, the trust of two malicious users increases to $\frac{1+1}{1+2} = 0.67$. Second, the trust of honest user u_1 is reduced to $\frac{4+1}{5+2} = 0.71$. Third, $G(I_1)$, $G(I_2)$, $G(I_3)$, and $G(I_4)$ which depend on the trust of u_1, are reduced. The results are shown in the second column in Table 1, where we use X_1, X_2, and X_3 to denote the three malicious users.

2) After the first step, the malicious users can turn I_4 into a trap. Note that the summation of their trust values is larger than $G(I_4)$, i.e. $0.67 \times 2 + 0.5 > 1.46$. Then, the trust of the malicious users are increased to $\frac{2+1}{2+2} = 0.75$ or $\frac{1+1}{1+2} = 0.67$, and the trust of the honest user u_1 and u_4 is reduced. $G(I_1)$, $G(I_2)$ and $G(I_3)$, which depend on the trust of u_1 or u_4, are also reduced. See the third column in Table 1.
3) Two of the malicious users further turn I_3 into a trap. Note that $0.75 \times 2 = 1.5$ is larger than $G(I_3) = 1.32$. Then, the trust of the malicious users is increased to 0.8 or 0.75, the trust of u_1 and u_3 is reduced, and $G(I_1)$ and $G(I_2)$ continue to drop.
4) Similarly, three malicious users turn I_2 into a trap. Then, the trust of malicious users becomes 0.83 or 0.75, the trust of u_1 becomes 0.29 and the trust value of u_2, u_3, u_4 and u_5 becomes 0.5. $G(I_1)$ is reduced to 2.29.
5) Finally, the summation of the malicious users' trust values becomes $0.83 \times 2 + 0.75 = 2.41$, which is larger than $G(I_1) = 2.29$. This means that the attacker can successfully attack I_1. In total, the malicious users give 13 votes, including 10 votes to $S_D = \{I_2, I_3, I_4, I_5\}$ and 3 votes to $S_T = \{I_1\}$.

In this case study, RepTrap reduces the requirement on the number of malicious users by 63% when compared with RepBad, and by 40% when compared with RepSelf. RepTrap also requires 28% less votes than RepSelf. To achieve the same attack goal, RepTrap needs much less attack resources than the known attacks.

3.4 RepTrap attack procedure

From the case studies, we know that the basic idea of RepTrap is to reduce the trust of honest users, who vote on the target items, by undermining the system's estimation on item quality. We develop a *formal procedure* to conduct RepTrap in this subsection. We first introduce some important concepts.

- To make the presentation clearer, we make the following assumptions: the target set S_T contains only one item, denoted by I_T; all the items are high quality items; the honest users will always give the honest votes, so all their votes will be '1'.
- The *item set* of user u_i, denoted by $S^{item}(u_i)$, contains the items receiving votes from user u_i.
- The *user set* of item I_k, denoted by $S^{user}(I_k)$, contains the users voting on item I_k.
- The *correlation* between item I_{k_1} and item I_{k_2}, denoted by $C(I_{k_1}, I_{k_2})$, is defined as

$$C(I_{k_1}, I_{k_2}) = |S^{user}(I_{k_1}) \bigcap S^{user}(I_{k_2})| \ . \quad (6)$$

  Thus, the correlation between item I_k and the target I_T is $C(I_k, I_T)$.
- The *correlated set*, denoted by S^{cor} contains the items whose correlation with the target set is non-zero, i.e. $S^{cor} = \{I_k, \ \forall k \ s.t. \ C(I_k, I_T) > 0\}$.
- The *uncorrelated set*, denoted by S^{uncor} contains the items whose correlation with the target set is zero, i.e. $S^{uncor} = \{I_k, \ \forall k \ s.t. \ C(I_k, I_T) = 0\}$.
- The *ability of attacker*, denoted by A_X, is defined as $A_X = \sum_{i=1}^{N_M} T(X_i)$, which is the summation of trust values of all malicious users.
- TRAP(I_k) means the process that the malicious users provide dishonest votes to make I_k into a trap.
- HONEST(I_k) means the process that the malicious users provide honest votes to I_k to gain trust.

The RepTrap attack procedure is shown in Procedure 1. This procedure can be roughly divided into four parts. In the first part (step 1-4), the attacker calculates some initial values. In the second part (step 5-7) the attacker checks whether it can successfully attack the target item. In the third part (step 8-17), the attacker finds the item that should be turned into a trap (i.e. construct S_D) by first check the correlated set and then check the uncorrelated set. Obviously, turning an item in the correlated set is more beneficial than turning an item in the uncorrelated set because the former can reduce the trust of honest users who also votes on the target item at the same time. In the forth part (step 18-25), when the ability of attacker (A_X) is not sufficient to create any more traps but the attack goal is not achieved yet, the attacker conducts simple self-promoting. That is, the attacker construct S_H by giving honest votes first to uncorrelated items and then to correlated items. This may allow them to increase A_X. Finally, when they cannot do anything, they will exit with failure.

Next, we examine the computation complexity of Procedure 1. For Step 1, the computation complexity is $O(n \cdot |I_T| \cdot v)$, where n is the number of items associated with the attack, $|I_T|$ is the number of target items, and v is the average number of voters for one item. Note that $|I_T|$ often equals to 1. Step 3 to step 25 is a loop with computation complexity $O(n \cdot r)$, where r is the number

Procedure 1 Procedure of RepTrap Attack

1: Calculate $C(I_k, I_T)$ **for** $k = 1$ **to** n
2: Construct S^{cor} and S^{uncor}
3: Calculate A_X
4: Calculate $G(I_k)$ **for** $k = 1$ **to** n
5: **if** $A_X > G(I_T)$
6: TRAP(I_T)
7: **EXIT** with **success**
8: **for** $I_k \in S^{cor}$
9: **if** $A_X > G(I_k)$
10: TRAP(I_k)
11: Remove I_k from S^{cor}
12: **goto** step 3
13: **for** $I_k \in S^{uncor}$
14: **if** $A_X > G(I_k)$
15: TRAP(I_k)
16: Remove I_k from S^{uncor}
17: **goto** step 3
18: **for** $I_k \in S^{uncor}$
19: HONEST(I_k)
20: Remove I_k from S^{uncor}
21: **goto** step 3
22: **for** $I_k \in S^{cor}$
23: HONEST(I_k)
24: Remove I_k from S^{cor}
25: **goto** step 3
26: **EXIT** with **failure**

of the rounds. Particularly, in each round of the loop, the computation complexity is $O(n)$, as all items are checked to make the decision for the next action. The number of rounds (r) depends on the hardness of the I_T. From the experiments we find that the number of rounds is normally within [10, 30]. Therefore, when the number of items and users increases, the term n and v dominate. The overall complexity of Procedure 1 is $O(n \cdot v)$.

Since the real system may have a large number of items, it is necessary for the attacker to reduce this computation complexity. The most effective way is to reduce n by only examining a small portion of the items for the attack. For example, if the target item is a TV, the attacker may only examine the items in electronics category in Procedure 1.

It is important to point out that this is a simple procedure for illustrating the basic idea of RepTrap. We will formulate an optimization problem that defines the "optimal" way to insert votes for the malicious users in the next subsection. Further, the RepTrap attack procedure need revision before it becomes suitable for other scenarios.

3.5 Details of RepTrap

As discussed earlier, procedure 1 simply illustrates the basic idea of RepTrap. In this section, we propose a heuristic *trap selection + vote allocation* procedure to search for the best way to conduct RepTrap.

Trap Selection

In Procedure 1, the attacker simply selects one item in the correlated set to turn it into a trap. However, we notice that the attacker should select an item that (1) has larger impact on the target set if it is turned into a trap and (2) requests less attack effort to be turned into a trap.

The first condition can be described by a higher correlation value between this item and the target, and the second condition can be described by a lower hardness value. Therefore, we define

$$W(I_j, I_T) = C(I_j, I_T)/G(I_j) \tag{7}$$

and modify step 9 in Procedure 1 as

9: select I_k that maximizes $W(I_j, I_T)$ among all items whose hardness values are smaller than A_X.

Details of $\mathbf{TRAP(I_k)}$

After selecting the item I_k to be turned into a trap, the attacker needs to determine which malicious users provide dishonest votes. The basic method is asking all the malicious users to insert dishonest votes to this item. However we can improve this procedure as follows.

From the calculation of the beta function trust value, we know that the gradient of a user's trust value increase becomes smaller as this user conducts more and more good behaviors. For example, when the system observes a user gives no dishonest votes and t honest votes, the trust value is $2/3$ for $t = 1$, $3/4$ for $t = 2$, and $4/5$ for $t = 3$. Thus, the trust value gained for one additional honest vote is $3/4 - 2/3 = 0.08$ when $t = 2$, and $4/5 - 3/4 = 0.05$ when $t = 3$.

Based on this observation, in each round, we order the malicious users according to their current trust values from low to high as $X_{t_1}, X_{t_2}, \cdots, X_{t_{N_M}}$. Then, we select the malicious users $\{X_{t_1}, X_{t_2}, \cdots, X_{t_s}\}$ to provide dishonest votes to item I_k, such that

$$\begin{cases} \sum_{i=1}^{s} T(X_{t_i}) > G(I_k) \cdot (1+\delta), \\ \sum_{i=1}^{s-1} T(X_{t_i}) \leq G(I_k) \cdot (1+\delta), \end{cases} \tag{8}$$

where δ is called the *hardness margin*. The hardness margin leads to an overestimate of the required dishonest votes, which has two usages. First, even if the attackers cannot infer the trust of users or the hardness of the items accurately, the overestimated value will ensure successful trap creation. Second, with overestimating dishonest votes, the trap will not be overturned (i.e. marked as high quality by the system) even if some honest users insert honest votes during the trap creation process. From our experiments, we find that a choice of $\delta = 0.2$ could maximize the success rate.

Details of $\mathbf{HONEST(I_k)}$

As stated in Procedure 1, when the malicious users cannot create more traps but the attack goal has not been achieved, the malicious users will vote honestly to promote their trust values. This is represent by $HONEST(I_k)$.

In particular, in step 18 and step 22, the attacker randomly select an item I_k in uncorrelated or correlated

set, respectively. Then, when $HONEST(I_k)$ is called, all the malicious users will provide an honest vote to I_k to increase their trust.

4 Vulnerabilities analysis

The simulation results in Section 6 will show the effectiveness of RepTrap. For instance, with 5% of the user IDs controlled by the attacker, RepTrap can successfully attack the most popular item with more than 90% success rate, whereas the success rates of RepBad and RepSelf are almost zero. Meanwhile, RepTrap can reduce the total number of votes from the attacker by more than 50%. Why is RepTrap so effective? Can we defeat it? In this section, we will study the fundamental reasons that facilitate RepTrap. In the next section, we will study the possible defense schemes.

4.1 Factors affecting RepTrap

Compared with RepBad and RepSelf, the advantages of RepTrap come from the usage of S_D to reduce the hardness of the targets. Specifically, if we can reduce the hardness of the target from $G(I_T)$ to $p \cdot G(I_T)$, we can roughly reduce the minimum number of malicious users needed by $1-p$. When we use the same number of malicious users, we can reduce the number of malicious votes needed. So the effect of RepTrap heavily depends on p. Several factors could affect the p value:

1) *No ground truth*: There is no ground truth about the quality of the items, so the system can only judge the behaviors of the users according to the estimated item quality which may be manipulated. This is the basic requirement of RepTrap. If the system know the ground truth of the item quality, the attacker will not be able to make any traps and $p = 1$.
2) *Correlated items*: In order to reduce the hardness of the targets, the attackers should first find the items correlated with the target. The more they can find, the more they may reduce the hardness of the targets. In most practical systems, the attacker can gain a large amount of information about who give what votes to which items. This knowledge enables the attacker to find the correlation between items.
3) *Attackable items*: The correlated items is useless for the attackers to reduce the hardness of the targets if the attackers can not make them into trap. It means that the attackers could only make the items with few voters into trap as they usually only control limited number of malicious voters. However, in almost all practical systems, the popularity of items follows a power-law distribution [2]. That is, there are a large number of items that receive only a few votes. Therefore, even with a smaller number of malicious users, the attacker can turn these unpopular items into traps.

We cannot change the first factor, since if we can get ground truth by other methods, we would not need voting system to identify the quality of an item. We cannot change the third factor either, since the system has no control on the popularity of items. Thus, to defense against RepTrap, or at least to make RepTrap less effective, we have to target the second factor – the correlation of items. In Section 5.1, we will develop the first defense scheme based on this factor.

4.2 Information shrinking

We have identified that the trust evaluation algorithm can be an enabling factor for RepTrap. In this subsection, we reveal the fundamental reason why attackers can take advantage of the trust evaluation algorithms.

Let us examine a simple example. We assume that there is an organization with three divisions. There are 20, 40 and 50 staff members in each division, respectively. When the organization needs to make a decision, each division makes its decision by performing majority voting inside the division, and then the organization performs majority voting based on the decisions from the divisions. That is, if two or more than two divisions' decision is "yes", the final decision is "yes". We may ask the question: what is the minimal number of staff members needed to control the final decision? The answer is 32, with 11 people in the first division and 21 people in the 2nd division to control the decision of the two divisions. It means that $\frac{11+21}{20+40+50} = 29\%$ of the staff members, which is the minority, can control the final decision, even if the organization uses majority voting in each level.

The reason for this kind of phenomena is that there is rich information about the votes of an item. However the voting system will shrink the information into a binary value. In the previous example, the division with 20 staff members is treated equally with the division with 50 staff members. So when the organization makes the final decision, it only considers the binary results from each division and shrinks all the other information, this opens the door for the attackers to manipulate the results with minority even when we use majority voting in each step.

In the trust evaluation algorithms, all the users' voting information for each item shrinks into a binary value to represent its quality. So the user trust is only calculated with two values: the number of honest votes ($H(u_i)$) and the number of dishonest votes ($D(u_i)$). It treats the item with a small number of votes equally to the items with a large number of votes. In other words, a vote to a popular item and a vote to an unpopular item carry the same weight in the trust calculation. This gives the attacker an opportunity to take advantage of the lost information to make a powerful attack. In Section 5.2, we will develop the second defense scheme based on this observation.

5 Defense schemes

We develop two types of defense schemes: *knowledge hiding* and *robustness-of-evidence*, which targets the "correlated item" and "information shrinking" enabling factors of RepTrap respectively.

5.1 Defense: knowledge hiding

From the discussion in Section 4.1, we know that the effect of RepTrap heavily depends on the number of correlated attackable items that could be found. Specifically, in the heuristic trap selection procedure, the attacker needs to know the item-user graph (i.e. who give what votes to which). As the first step toward defense, the system can hide the IDs of voters for each item. This defense is referred as $Defense_{KH}$ in the rest of the paper. Without this knowledge, the attackers cannot easily calculate $C(I_j, I_T)$ and $W(I_j, I_T)$.

The cost of $Defense_{KH}$ is its potential negative impact on users' satisfaction. For example, hiding the user IDs could hurt the users' incentive to contribute. In this paper, our investigation will focus on the technical consequence of $Defense_{KH}$.

When $Defense_{KH}$ is used, the attacker can modify the trap selection procedure based on $|S^{user}(I_k)|$, defined as the number of users who vote item I_k. Larger is $|S^{user}(I_k)|$, more honest users will be affected if I_k is turned into a trap. Note that $|S^{user}(I_k)|$ cannot be hidden in any practical systems. Otherwise, the basic functionality of the system will be damaged. With the knowledge of $|S^{user}(I_k)|$, the attacker can modify the RepTrap procedure by changing equation (7) into

$$W(I_j, I_T) = |S^{user}(I_j)|.$$

We should note that without knowing the user IDs who vote on each item, the attackers cannot calculate $G(I_j)$ directly. But they can estimate this value as

$$G(I_j) \approx T_{ave} \cdot |S^{user}(I_j)|,$$

where T_{avg} is the average trust value of users in the system. Because we have the hardness margin (δ) when we use $G(I_j)$ in equation (8), so the attackers can make the trap successfully most of the time.

5.2 Defense: robustness-of-evidence

As discussed in Section 4.2, when we calculate the quality of an item with voting system, we only retain one binary to indicate its quality. So when we convert the users' voting history into two numbers: $H(u_i)$ and $D(u_i)$, too much information is lost. Let's compare the following two cases:

- $evidence_1$: u_i's vote disagrees with $Q(I_{k1})$, and item I_k1 receives total 1000 votes;
- $evidence_2$: u_i's vote disagrees with $Q(I_{k2})$, and item I_k2 receives total 10 votes.

In the existing method, $evidence_1$ and $evidence_2$ are treated equally. Both are the evidence that u_i provides a dishonest vote. However, $evidence_2$ is less robust than $evidence_1$. The attacker can easily manipulate $Q(I_{k2})$ and generate false evidence since its number of votes is small.

In this work, we introduce the concept of *robustness-of-evidence* (RE), which describes how difficult it is to manipulate the evidence. If an evidence is more robust than another evidence, it should have more impact on the user's trust, because this evidence is less likely to be manipulated.

Recall that the effectiveness of RepTrap is closely related to the parameter p, which is defined in Section 4.1 and describes how much the attacker can reduce the hardness of the targets to. If we give lower (or higher) weight to the evidence that are more (or less) likely to be manipulated by the attacker, the trust value of honest users will not drop greatly even if the attackers create traps successfully. Thus, this defense can improve p value, and therefore reduce the effectiveness of RepTrap.

It is not straightforward to calculate RE. The calculation needs to consider not only the voting system, but also possible attacks against RE calculation itself. As the first step toward understanding the usefulness of RE, we identify three properties that RE functions should have and provide a specific RE function. The design of RE functions deserve more investigation in the future work.

From the illustration of the previous example, the RE value related to item I_k, denoted by $R(I_k)$, should be a function of the number of voters for this item (i.e.$|S^{user}(I_k)|$). This function should satisfy a few properties.

- The item with more voters should have a larger RE value than the item with less voters.
- The item with much more voters should not overwhelm the items with less voters.
- The increase of the robustness should not be linear. For example, the difference between 10 and 20 voters should be larger than the difference between 1000 and 1010 voters.

In this work, we calculate $R(I_k)$ as

$$R(I_k) = log_2(|S^{user}(I_k)|) \tag{9}$$

Then, we use the RE value as the weight factors in trust calculation. In particular, when the system thinks that u_i provided one honest (or dishonest) vote to item I_k, the system will count this evidence with $R(I_k)$ as weight. Let $S^{agr}(u_i)$ denotes the set of items of which u_i's votes agree with the quality and $S^{dis}(u_i)$ denotes the set of items of which u_i's vote disagree with the quality, the calculate of $H(I_k)$ and $D(I_k)$ are as follows.

$$H(u_i) = \sum_{I_k \in S^{agr}(u_i)} R(I_k) \tag{10}$$

$$D(u_i) = \sum_{I_k \in S^{dis}(u_i)} R(I_k) \tag{11}$$

This defense scheme also affects the strategies to launch RepSelf and RepTrap. Without the robustness-of-evidence scheme, the malicious users give honest votes to randomly selected non-target files, for the purpose of gaining trust. When the robustness-of-evidence scheme is used, the malicious users can gain trust faster and easier by giving honest votes to popular items, because the popular items yield larger robustness scores. Thus,

we introduce the **modified RepSelf and RepTrap** attacks. When malicious users need to give honest votes to gain trust, they will vote the item that receives the largest number of votes among all the available items. The modified RepSelf and RepTrap are stronger than the original attacks when the robustness-of-evidence defense scheme is used. In the performance evaluation, we will use the modified RepSelf and RepTrap to replace original RepSelf and RepTrap, whenever robustness-of-evidence scheme is used.

6 EXPERIMENT

To demonstrate the consequence of RepTrap and defense schemes, we implement these approaches in the context of a P2P file-sharing network and evaluate their impact and performance in varying circumstances. In this section, we first describe the experiment setup, then compare several attack methods as well as several defense schemes and investigate the joint effect of attacks and defenses, and finally discuss the effect of ground truth and the correlation in real systems.

6.1 Experiment description

The user behavior model and file popularity model used in our simulation is very similar to the models in [7]. That is, a user's download behaviors follow a Poisson process. The download frequency of different users follows a power-law distribution. The popularity of the files, i.e. the number of downloads of files, follows another power-law distribution.

In our configuration, there are 1000 honest users and 100 files in the system. The power-law parameter of download frequency is set as $\alpha = 0.999$. That is, the download frequency of user u_j is $T \cdot \alpha^{j-1}$, which is also the arrival rate of the Poisson process that governs the download behaviors. Obviously, T and α determine the total number of honest votes within a given time. In the experiments, we first set the total number of honest votes, and then derive the proper T value that is used to generate the user download behaviors. The votes from honest users are randomly assigned to the files according to file popularity distribution. The power-law parameter of file popularity is set as 0.99. These parameters are chosen to make sure that the least active users will download at least one file and the least popular file will be downloaded by at least one user.[2]

Without loss of generality, we assume that good users always provide honest votes after successful downloads[3]. Each user cannot vote more than once for one file. The voting system described in Section 2.2 is adopted.

2. We have conducted experiments with different power-law parameters and find that the simulation results are similar. Varying the power-law parameters will not change our conclusion on the relative performance of different attacks and defense schemes. Therefore, in the rest of this session, we will not change the parameters in the user behaviors and file popularity models.

3. An alternative model is to set that good users provide dishonest votes with a small probability. However, this will not have meaningful impact on the simulation results.

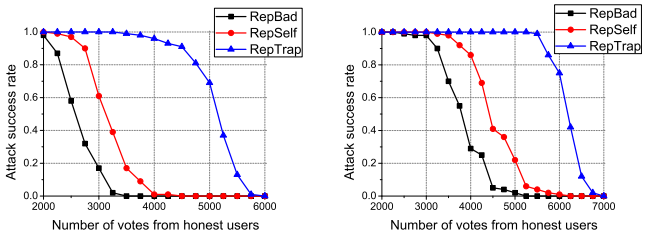
To evaluate the strengthen of **attacks**, we compare RepTrap with two other schemes: RepBad and RepSelf, which have been described in Section 2.3 and 3.3.3. Briefly speaking, with the RepBad strategy, each malicious user provides an dishonest vote to the target file. With the RepSelf strategy, malicious users first provide honest votes to randomly selected files that are not target. After they have gained enough trust, they provide dishonest votes to the target file. As we have discussed in Section 5.2, when $Defense_{RE}$ is used, we will use the modified RepSelf and RepTrap in the evaluation. Otherwise, we use regular RepSelf and RepTrap. After testing the effect of RepTrap for several times, we find that $\delta = 20\%$ is a proper choice. Due to space limitation, we move the discussion on δ to future work.

To evaluate the impact of **defense schemes**, we compare four systems: original design of majority rule based system with beta trust function; the system with knowledge hiding defense (i.e.$Defense_{KH}$), the system with robustness-of-evidence defense (i.e. $Defense_{RE}$), and the system with both $Defense_{KH}$ and $Defense_{RE}$.

In the performance evaluation, we pay attention to the following four values: the number of honest users (N_H), the total number of votes from the honest users (V_H), the number of malicious users (N_M), and the total number of votes from the malicious users (K_M). Here, the first two values describe the strength of honest users. Increasing these two values will increase the difficulty of attacks. In this work, we fix N_H as 1000 and change V_H. The other two values describe the attack resources. In the simulation, N_M changes from 20 to 50. If it is not explicitly mentioned, the default value of N_M is 50.

Evaluation will be conducted in two scenarios. In the first scenario, the number of malicious users is fixed but they can insert as many votes as possible to achieve the attack goal. In this scenario, we measure the *attack success rate*, denoted by R_{sus} and defined as the probability that the attack goal is achieved. For each configuration, we first generate the item-user graph randomly by following the power-law distributions discussed before. Specially the system will first run without malicious users until the total number of votes from honest users reaches a threshold (e.g. 3000), we then use RepBad, RepSelf and RepTrap to insert malicious votes respectively to see whether they can achieve the attack goal in such case. We run this process for 1000 times (with a different item-user graph each time) to estimate the attack success rate. We also measure the *average K_M value*, which is the average number of votes needed from the malicious users to achieve the attack goal. The attack that has higher attack success rate and/or smaller average K_M value is a more powerful attacks.

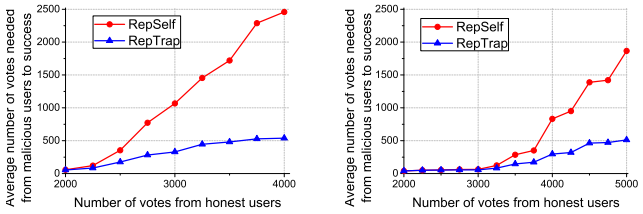
In the second scenario, we limit the number of malicious users as well as the maximum number of votes for malicious users. That is, the N_M value and the K_M value cannot exceed certain thresholds. In this scenario, the performance measure will be the attack success rate.



(a) Success rate (R_{sus}) when attacking the 1^{st} popular file

(b) Success rate (R_{sus}) when attacking the 20^{th} popular file

Fig. 3. Success rate of RepTrap, RepBad and RepSelf attacks



(a) Average number of malicious votes (K_M) needed to successfully attack the 1^{st} popular file

(b) Average number of malicious votes (K_M) needed to successfully attack the 20^{th} popular file

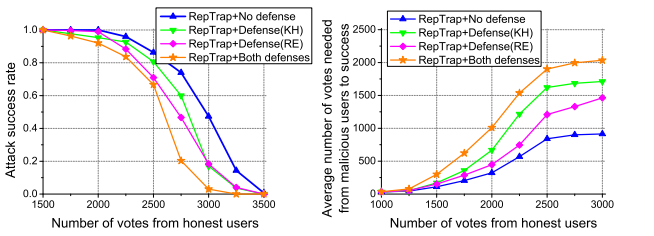
Fig. 4. Average number of votes needed from malicious users

6.2 RepTrap attack

In this set of experiments, we evaluate RepTrap, by comparing it with the known RepBad and RepSelf attacks, without considering the proposed defense schemes.

Figure 3(a) and 3(b) are for the experiments when the target file is the 1^{st} and the 20^{th} popular file, respectively. The attack goal is to make the system mark the target file as low quality. As mentioned earlier, the attack does not start at the beginning. The horizontal axis is K_H, the total number of honest votes in the system when the malicious users start to attack. The vertical axis is the attack success rate R_{sus}. In this experiment, the number of malicious users is 50 (i.e. $N_M = 50$), and the malicious users can insert as many votes as they want. Since each malicious user can vote one file at most once, they may not achieve the attack goal even if they can insert many votes, especially when there are already a large number of honest votes in the system. Therefore, the success rate is a decreasing function of K_H.

It is clearly seen that the RepTrap scheme is the strongest and the RepBad is the weakest. With a given K_H, *RepTrap has much larger chance to be successful than RepBad and RepSelf.* As shown in Figure 3(a), when attacking the 1^{st} popular file, RepBad can only be successful when $K_H < 3500$, RepSelf can be successful when $K_H < 4000$, and RepTrap can be successful when $K_H < 5800$. RepSelf has better success rate than RepBad because malicious users in RepSelf can improve their own trust. The proposed RepTrap is much stronger than RepSelf because the malicious users in RepTrap can increase their own trust and reduce the honest users' trust at the same time. For example, when RepSelf has 10% success rate, RepTrap has 100% success rate in both Figure 3(a) and Figure 3(b). When RepTrap still has 90% success rate, both of RepBad and RepSelf has almost zero



(a) Success rate (R_{sus}) when attacking the 1^{st} popular file

(b) Average number of malicious votes needed (K_M) to successfully attack the 1^{st} popular file

Fig. 5. Effectiveness of RepTrap with defense schemes

success rate.

Next, we examine the average number of votes needed from malicious users to attack successfully. Note that RepTrap and RepSelf need to insert more votes to manipulate trust values. Since the K_M value in RepBad equals to the number of malicious users, which is 50 in this case, we only compare RepTrap and RepSelf in this experiment.

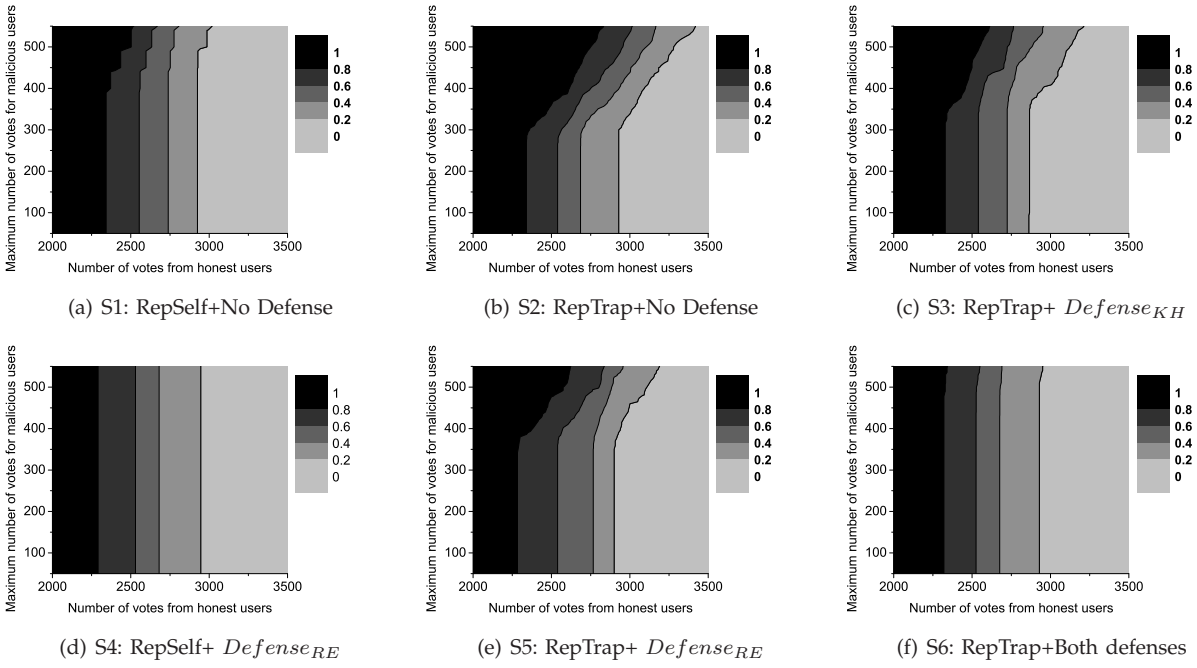
Figure 4(a) and 4(b) show the average K_M value v.s. K_H when the target is the 1^{st} and the 20^{th} popular files, respectively. Compared with RepSelf, RepTrap needs much less votes from malicious users. For example, in Figure 4(a), when the number of votes from the honest users is 4000, RepTrap can achieve the attack goal by inserting on average 500 votes, whereas the RepSelf needs to insert on average 2500 (4 times more) votes to achieve the same attack goal. When the number of votes from the honest users is more than 4000, the RepSelf cannot be successful no matter how many votes it inserts. This is why the curves stop at $K_H = 4000$ in Figure 4(a). We can conclude from Figure 4 that compared with RepSelf, RepTrap can reduce more than 50% of votes most of the time, which means that RepTrap requires much less attack resources than RepSelf.

In all above experiments, we see that attacking the 1^{st} popular file is more difficult (i.e. lower R_{sus} and higher K_M value) than attacking the 20^{th} popular file as it has more honest votes. In both cases, RepTrap has significant advantage from the attacker's point of view. Due to space limitation, we will only show the results of attacking the 1^{st} popular file in the rest of the experiments since the results for other files are similar.

6.3 Defense

In this section, we examine the effectiveness of the proposed defense schemes. In particular, we evaluate R_{sus} and average K_M value in four scenarios: (1) RepTrap without defense, (2) RepTrap with $Defense_{KH}$, (3) RepTrap with $Defense_{RE}$, and (4) RepTrap with both $Defense_{KH}$ and $Defense_{RE}$.

Figure 5(a) shows the attack success rate (R_{sus}) and Figure 5(b) shows the number of votes needed from malicious users (K_M) to attack successfully, as functions of the number of votes from honest users (K_H). In this experiment, the number of malicious users is 30 (i.e. $N_M = 30$). The following observations are made.



(a) S1: RepSelf+No Defense (b) S2: RepTrap+No Defense (c) S3: RepTrap+ $Defense_{KH}$

(d) S4: RepSelf+ $Defense_{RE}$ (e) S5: RepTrap+ $Defense_{RE}$ (f) S6: RepTrap+Both defenses

Fig. 6. Attack success rate when varying the number of K_H and the maximum number of K_M

First, both defense schemes can reduce the attack success rate R_{sus} and increase the cost of attack represented by K_M. Second, in Figure 5(a), $Defense_{KH}$ is more effective when $K_H < 2200$, $Defense_{RE}$ is more effective when $2200 < K_H < 3000$, and their effects become the same when $K_H > 3000$. Third, $Defense_{KH}$ is more effective to increase the average K_M. Forth, combining the two defense schemes leads to better defense. For example, with $K_H = 2750$, the success rate for RepTrap is 72%. $Defense_{KH}$ alone reduces the success rate to 60%, and $Defense_{RE}$ alone reduces the success rate to 46%. When both defense schemes are used, the success rate is reduced to 20%. Meanwhile, when both defense schemes are used, the average K_M value is increased from 900 to 2000, which corresponds to 122% increase. Here, $Defense_{KH}$ alone increases the K_M by 87%, and $Defense_{RE}$ alone increase K_M by 48%. In summary, each defense scheme can reduce the attack success rate by up to 30% and increase attack cost by up to 50%.

The reason behind these observations is that with $Defense_{KH}$, the traps created by the attack may not be highly correlated with the target file. The $Defense_{RE}$ reduces malicious users' capability of reducing honest users' trust from creating traps. Both schemes reduce the effectiveness and increase the cost of RepTrap. Furthermore, the ideas behind these two defense schemes are complementary. Thus, two schemes can be used together to achieve better defense effect.

6.4 Joint effect of attacks and defenses

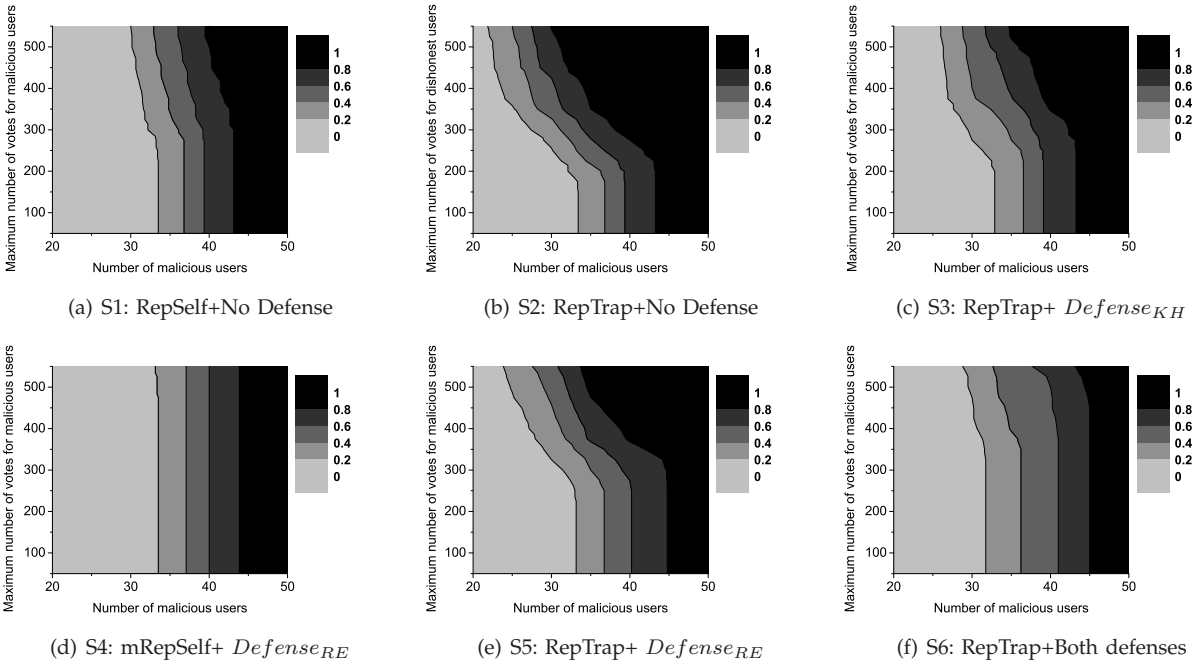
In this section, we investigate the attack success rate under different combinations of attack and defense schemes. In this comprehensive investigation, we change all key parameters, including the number of attackers (N_M), the maximum number of votes for malicious users (K_M), and the number of votes from honest users (K_H).

We have seen that the proposed defense schemes can reduce the power of RepTrap. How do they affect the RepBad and RepSelf? It is obvious that knowledge hiding does not affect RepBad or RepSelf. The robustness-of-evidence scheme doest not affect RepBad, but could change the malicious users' behaviors in RepSelf as we have discussed in Section 5.2.

In the experiments, we will compare 6 scenarios: (S1) RepSelf with no defense, (S2) RepTrap with no defense, (S3) RepTrap with $Defense_{KH}$, (S4) RepSelf with $Defense_{RE}$, (S5) RepTrap with $Defense_{RE}$, (S6) RepTrap with both $Defense_{KH}$ and $Defense_{RE}$.

In Figure 6, the x-axis is the number of votes from honest users (K_H), the y-axis is the maximum number of votes for malicious users (K_M), the darkness represents the success rate (R_{sus}). In this experiment, the target file is the 1^{st} popular file and there are 50 malicious users. To quickly grab the information in these figures, one can look at the dark area and light area. A strong attack will generate bigger dark area and smaller light area. A good defense will reduce dark area and increase light area.

The following observations are made. First, it is interesting to see that there exists a critical value such as 300 votes in S2. When K_M is smaller than this critical value, R_{sus} is determined by K_H. When K_M is larger than this critical value, R_{sus} starts to increase with K_M. The reason is that both RepSelf and RepTrap need many additional votes to manipulate trust or make traps, and this can only be achieved if they can insert enough votes. Thus, the attacks become more effective when K_M is greater than the critical value. Second, by comparing S1 and S2, we see that RepTrap yields higher R_{sus} (i.e. bigger dark area and smaller light area) than RepSelf, especially for larger K_M values. Third, by comparing S2,



(a) S1: RepSelf+No Defense (b) S2: RepTrap+No Defense (c) S3: RepTrap+ $Defense_{KH}$

(d) S4: mRepSelf+ $Defense_{RE}$ (e) S5: RepTrap+ $Defense_{RE}$ (f) S6: RepTrap+Both defenses

Fig. 7. Attack success rate when varying the number of N_M and the maximum number of K_M

S3 and S5, we see that both defense schemes work. They reduce the dark area and increase the critical value (i.e. 350 votes in S3 and 380 votes in S5). Adding S6 to the comparison, we see that the joint effect of two defense schemes is better. They greatly reduce R_{sus}. Finally, by comparing S1 and S4, we see that $Defense_{RE}$ also works for RepSelf.

Figure 7 shows R_{sus} for varying attack resources, described by the number of malicious users (N_M, x-axis) and then by the maximum number of votes for malicious users (K_M, y-axis). The number of votes from honest users is fixed as 2500. The darkness means the attack success rate. So the darker of the figure, the stronger of the attack in this scenario. These figures convey rich information. To quickly grab the key ideas, one can look at the lines separating different gray-scale areas. We can see that the attack resources required to achieve the same R_{sus} in different situations.

First, in all figures, the attack becomes stronger when N_M and/or K_M increase. When K_M is smaller than a critical value, the R_{sus} is mainly determined by N_M. This is similar to the observation in Figure 6. The underlying reasons are similar too. Second, RepTrap requests much less attack resources than RepSelf (compare S1 with S2). Third, both defense schemes can increase the required attack resources for achieving the same R_{sus} (compare S2, S3, and S5), whereas the combination of two defense methods greatly increases required N_M and K_M values (see S6). Finally, by comparing (S1) and (S4), we see that $Defense_{RE}$ works for RepSelf too.

6.5 The effect of ground truth

Some systems may know the ground truth of certain items, by hiring professional reviewers or identifying pre-trusted users. In such systems, the real quality of some items is known. Let G denote the *ground truth percentage*, defined as the percentage of items whose true quality is known by the system. To understand the effect of RepTrap in this scenario, we compare RepSelf and RepTrap with different G values.

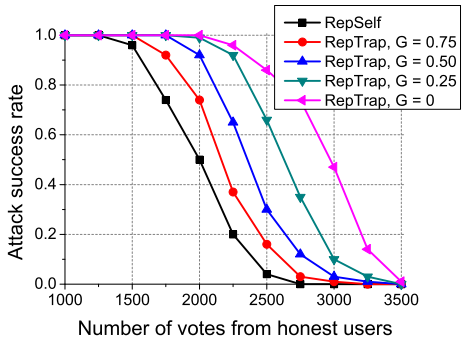


Fig. 8. Success rate under different ground truth percentage

Figure 8 shows the attack success rates under different knowledge of the ground truth. There are 1000 honest users, 100 items, and 30 malicious users. The x-axis means the number of honest votes in the system and the y-axis means the attack success rate. The five curves are the results of RepSelf and RepTrap with different ground truth percentage. When $G = 0.75$, it means that the system knows the real quality of 75% of the items and the attackers can only try to turn around the quality of the remaining 25% items. As expected, when the system knows more about the ground truth of the items (G increases), the success rate of the attack is reduced. This is because there are less items could be used to reduce the trust of honest users. However, RepTrap is still much more powerful than RepSelf, it can increase the success rate by 20% even when the system knows the real quality of 50% of items.

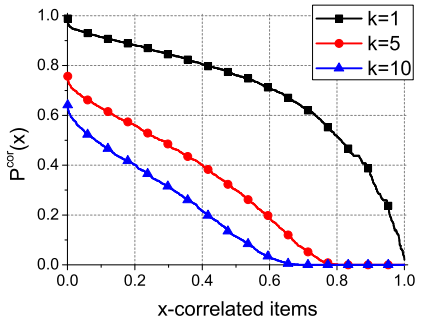


Fig. 9. Correlation between items in MovieLen dataset

6.6 Correlation in real systems

RepTrap exploits correlation between items. An interesting question to ask is how much correlation exists in real systems. To demonstrate that items in real systems are indeed correlated, we analyze the MovieLen dataset[4] which has been used by many researchers. It contains 100,000 ratings for 1682 movies from 943 users.

In this experiment, we define that two items are *correlated* if they have at least k common voters. An item is called $x\%$-*correlated* if it is correlated to at least $x\%$ of all other items. Let $P^{cor}(x)$ denote the percentage of items that are $x\%$-correlated. Higher is the $P^{cor}(x)$ value, higher is the correlation between the items.

In Figure 9, the y-axis is $P^{cor}(x)$ and the x-axis is the $x - correlated$ value. The three curves are for k=1, k=5, and k=10, respectively. The results show that there is a lot of correlation between the items. Let us take $k = 5$ for example (see red round curve), when the x-axis value is 0.6, the y-axis is 0.2. It means that 60% of items are 20%-correlated. That is, each of those items shares no less than 5 common voters with at least 20% of all other items. When $k = 10$ (see blue triangle curve), 40% of items are 20%-*correlated*. Therefore, for a particular victim item, it is highly likely that the attacker can find many correlated items to conduct RepTrap.

7 Conclusion and Discussion

Safe guarding online voting systems is gaining increasing attention in many e-Commerce and e-Business applications. In this paper, we study the vulnerabilities and countermeasures of voting systems with existing trust mechanisms. Comparing with the state of art research in literature, we make three unique contributions. First, we present RepTrap, which is more powerful than existing known attacks such as RepBad and RepSelf. Concretely, RepTrap significantly increases the attack success rate and reduces the resources required from the attackers. For example, when evaluated in a P2P file-sharing system, RepTrap can successfully attack the most popular item with more than 90% success rate, whereas RepBad and RepSelf can never succeed under the same conditions. Furthermore, RepTrap only needs to control 5% of the users IDs to accomplish the attack, and reduces the total number of votes from attackers by more than 50% compared to RepSelf. These results demonstrate that the existing defense schemes, including adding trust mechanisms and increasing the cost of inserting user IDs and votes, are not sufficient against sophisticated attacks. Second, we have identified four key factors that facilitate RepTrap: the lack of ground truth, availability of correlation knowledge, power-law item popularity distribution, and information shrinking. Third but not the least, we have developed two defense methods, knowledge hiding and robustness-of-evidence, to constrain the effect of attacks and significantly increase the difficulty of launching an successful attack. Our experimental results demonstrate that each defense scheme can reduce the attack success rate by up to 30% and increase the cost of attack by up to 50% and the effect of combining the two defense schemes is much more significant under various scenarios.

We are continuing our efforts in understanding potential attacks and developing more robust defense schemes for guarding online voting systems. First, we are interested in understanding the parameters that are critical in terms of tuning the efficiency and effectiveness of our defense schemes. Second, we are interested in further improving the robustness and attack resilience of our defense schemes. Third, we are interested in studying the effect of different attack models under different trust frameworks and scenarios.

Acknowledgement.
We would like to thank the reviewers and the associate editor Elisa Bertino for their comments that have helped improve the quality of the paper. The first author completed this work as a visiting PhD student at Georgia Institute of Technology, he is supported by China Education Ministry scholarship program, NGFR 973 program (NO.2004CB318204), and NNSF(NO.60673183). The third author is partially sponsored by grants from NSF CNS under NetSE and CyberTrust, an IBM SUR grant, an IBM faculty partnership grant, and a grant from Intel Research Council.

References

[1] R. Farquharson. Theory of Voting. Oxford, 1961.
[2] M. Faloutsos, P. Faloutsos and C. Faloutsos. On power-law relationships of the Internet topology. In Proceedings of SIGCOMM, 1999.
[3] A. Josang and R. Ismail. The beta reputation system. In Proceedings of the 15th Bled Electronic Commerce Conference, June 2002.
[4] J. R. Douceur. The sybil attack. In Proceedings of IPTPS, March 2002.
[5] B. C. Ooi, C. Y. Liau, K. L. Tan. Managing trust in peer-to-peer systems using reputation-based techniques. In Proceedings of the 4th international conference in advances in web-age information management (WAIM), Chengdu, China, August 2003.
[6] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In Proceedings of WWW, May 2003.
[7] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In Proceedings of ACM SOSP, 2003.
[8] C. Dellarocas. The digitization of word-of-mouth: Promise and challenges of online reputation systems. Management Science, October 2003.

4. http://www.movielens.org/

[9] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati. Managing and Sharing Servents' Reputations in P2P Systems. IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 4, pp. 840-854, July/August, 2003.
[10] S. Buchegger and J.-Y. L. Boudec. A Robust Reputation System for P2P and Mobile Ad-hoc Networks. In Workshop on the Economics of Peer-to-Peer Systems, Boston, MA, June 2004.
[11] M. Khambatti, P. Dasgupta and K.D. Ryu, A Role-Based Trust Model for Peer-to-Peer Communities and Dynamic Coalitions, Second IEEE International Information Assurance Workshop, Charlotte, NC, April 2004.
[12] L. Xiong and L. Liu. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. IEEE Transactions on Knowledge and Dat a Engineering, Vol.16, No. 7, July, 2004.
[13] B. Bertino, E. Ferrari, and A.C. Squicciarini. Trust-X: A Peer-to-Peer Framework for Trust Establishment. IEEE Transactions on Knowledge and Data Engineering, Vol16, no.7, July 2004.
[14] M. Fan, Y. Tan, A. B. Whinston. Evaluation and Design of Online Cooperative Feedback Mechanisms for Reputation Management. IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 2, pp. 244-254, February, 2005.
[15] Z. Liang, W. Shi. PET: A PErsonalized Trust Model with Reputation and Risk Evaluation for P2P Resource Sharing, in Proceedings of HICSS'05, January, 2005.
[16] K. Walsh and E. G. Sirer, Experience with an Object Reputation System for Peer-to-Peer Filesharing, USENIX NSDI, 2006.
[17] H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman. Sybilguard: Defending against sybil attacks via social networks. In Proceedings of ACM SIGCOMM, September 2006.
[18] J, Brown and J. Morgan. Reputation in Online Markets: Some Negative Feedback. IBER Working Paper, UC Berkeley, 2006.
[19] J. A. Chevalier, and D. Mayzlin. The effect of word of mouth on sales: Online book reviews. Journal of Marketing Research 43, 3 (Aug. 2006), 345C354
[20] Y. Sun, Z. Han, W. Yu, and K. J. R. Liu. A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks. In Proceedings of IEEE INFOCOM, April 2006.
[21] A. C. Squicciarini, E. Bertino, and E. Ferrari. Achieving Privacy with an Ontology-Based Approach in Trust Negotiations. IEEE Transaction on Dependable and Secure Computing (TDSC), 3(1):13C30 2006.
[22] M. Sensoy, P. Yolum. Ontology-Based Service Representation and Selection. IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 8, pp. 1102-1115, August, 2007.
[23] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. ACM Transaction on Internet Technology, Oct. 2007.
[24] A. C. Squicciarini, E. Bertino, E. Ferrari, F. Paci, and B. Thuraisingham. PP-Trust-X: A System for Privacy Preserving Trust Negotiations. ACM Transactions on Information and System Security (TISSEC), 10(3), July 2007.
[25] A. C. Squicciarini, A. Trombetta, E. Bertino. Supporting Robust and Secure Interactions in Open Domains through Recovery of Trust Negotiations. ICDCS 2007.
[26] A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. Decision Support System, 43(2):618-644, 2007.
[27] K. Hoffman, D. Zage, and C. Nita-Rotaru. A survey of attack and defense techniques for reputation systems. Technical Report CSD TR #07-013, Purdue University, 2007.
[28] P. Chen, S. Dhanasobhon, and M. Smith. All reviews are not created equal: The disaggregate impact of reviews and reviewers at amazon.com. In Proceedings of ICIS, 2007.
[29] R. Zhou and K. Hwang. PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing. IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No.5, May 2007.
[30] L. Vu, K. Aberer. A Probabilistic Framework for Decentralized Management of Trust and Quality. In Proceedings of the 11th international Workshop on Cooperative information Agents XI, 2007.
[31] Z, Liang. and W. Shi. Analysis of ratings on trust inference in open environments. Elsevier Performance Evaluation, Vol. 65, No. 2, pp. 99-128, Feb. 2008.
[32] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In Proceedings of IEEE Symposium on Security and Privacy, May 2008.
[33] R. Zhou and K. Hwang. Gossip Trust for Fast Reputation Aggregation in Peer-to-Peer Networks. IEEE Transaction on Knowledge and Data Engineering, Feb. 2008.
[34] L. Vu, K. Aberer. Effective Usage of Computational Trust Models in Rational Environments. Web Intelligence, 2008.
[35] Y. Yang , Y. Sun, S. Kay, and Q. Yang. Defending Online Reputation Systems against Collaborative Unfair Raters through Signal Modeling and Trust, to appear in Proceedings of ACM Symposium on Applied Computing (SAC'09), Honolulu, Hawaii, Mar. 2009.
[36] N. Tran, B. Min, J. Li, and L. Submaranian. Sybil-resilient online content voting. In Proceedings of the 6th Symposium on Networked System Design and Implementation (NSDI'09), Apr. 2009.
[37] A. Parsa, Belkin's Development Rep is Hiring People to Write Fake Positive Amazon Reviews. http://www.thedailybackground .com/2009/01/16/exclusive-belkins-development-rep-is-hiring-people-to-write-fake-positive-amazon-reviews/, 2009.
[38] D. Zarrella, Not Everything That Can be Counted Counts, http://pistachioconsulting.com/shortyawards-gaming/, 2009.
[39] P. Dewan, P. Dasgupta. P2P Reputation Management Using Distributed Identities and Decentralized Recommendation Chains. IEEE Transactions on Knowledge and Data Engineering, 06 Feb. 2009.

PLACE PHOTO HERE

Qinyuan Feng Biography text here.

PLACE PHOTO HERE

Yan Lindsay Sun Biography text here.

PLACE PHOTO HERE

Ling Liu Biography text here.

Yafei Yang Biography text here.

Yafei Dai Biography text here.