RepTrap: A Novel Attack on Feedback-based Reputation Systems

Yafei Yang†, Qinyuan Feng†, Yan Lindsay Sun† and Yafei Dai†
†University of Rhode Island, Kingston, RI, USA
Email: {yafei, yansun}@ele.uri.edu
-CNDS Lab, Peking University, Beijing, China
Email: {fqy,dyf}@net.pku.edu.cn

ABSTRACT

Reputation systems are playing critical roles in securing today’s distributed computing and communication systems. Similar to all other security mechanisms, reputation systems can be under attack. In this paper, we report the discovery of a new attack, named RepTrap, against feedback-based reputation systems, such as those used in P2P file-sharing systems and E-commerce websites (e.g. Amazon.com). We conduct an in-depth investigation on this new attack, including analysis, case study, and performance evaluation based on real data and realistic user behavior models. We discover that the RepTrap is a strong and destructive attack that can manipulate the reputation scores of users, objects, and even undermine the entire reputation system. Compared with other known attacks that achieve the similar goals, the RepTrap requires less effort from the attackers and causes multi-dimensional damage to the reputation systems.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed Systems; C.2.0 [Computer-Communication Networks]: General—Security and Trust

General Terms
Security, Attacks, Algorithms

Keywords
Reputation System, Feedback, Trust

1. INTRODUCTION

Word-of-mouth, one of the most ancient mechanisms in the history of human society, is gaining new significance in the Internet [7, 17]. The online reputation systems, also known as the on-line feedback mechanisms, are creating large scale, virtual word-of-mouth networks in which individuals share opinions and experiences on a wide range of topics, including products, companies, digital content and even other people. For example, Epinions.com encourages Internet users to rate practically any kind of businesses. Citysearch.com solicits and displays user feedback on restaurants, bars, and performances. YouTube.com recommends video clips based on viewers’ feedback and some P2P file-sharing systems [25] does the same for the files uploaded by the users.

The reputation systems play significant roles in (1) assisting users’ decision-making, (2) encouraging trustworthy behavior, and (3) deterring participation by unskilled or dishonest users. Reputation systems are having increasing influence on purchasing decision of consumers, online digital content distribution, and even political campaigns [7].

Meanwhile, the manipulation of such systems is rapidly growing. Firms post biased ratings and reviews to praise their own products or bad-mouth the products of their competitors. Political campaigns promote positive video clips and hide negative video clips by inserting unfair ratings at YouTube.com. There is ample evidence that such manipulation takes place [9]. In February 2004, due to a software error, Amazon.com’s Canadian site mistakenly revealed the true identities of some book reviewers. It turned out that a sizable proportion of those reviews were written by the books’ own publishers, authors, and competitors [13]. The scammers are creating sophisticated programs that mimic legitimate YouTube traffic and provide automated feedback for videos and other content they wish to promote [14]. Some eBay users are artificially boosting their reputation by buying and selling feedbacks [3]. Collaborative manipulation of feedback based reputation systems is a growing threat. This threat is hurting consumers and will also hurt the business hosting such systems [4].

In the current literature, the research on attacks against reputation systems is still immature. The existing threat models are rather straightforward [15, 23]. The known attacks can be classified according to their attack goals: self-promoting (i.e. falsely increasing reputation),slandering (i.e. falsely reducing reputation), whitewashing (i.e. repairing reputation after bad behaviors), and denial-of-service (i.e. making the system wrongly accuse honest users or objects). Attackers achieve the above goals through a variety of methods [15]. Each attacker can acquire multiple identities through a sybil attack [10, 30, 29]. All the identities under the control of the attackers can (1) provide positive feedbacks for self-promoting, (2) provide negative feedbacks for slandering; (3) behave honestly to the objects that the attackers are not interested in or simply register as a new user for whitewashing [20, 21]; and (4) subvert the underlying detection algorithms to make honest users/objects look suspicious [23].

Whereas current research primarily focuses on individual attack strategies, the intelligent human attackers can create sophisticated attacks by integrating different manipulation strategies. In this paper, we report the discovery of a new attack, named RepTrap, against feedback-based reputation...
systems, and conduct an in-depth investigation on this new attack. RepTrap is applicable to reputation systems that calculate key reputation scores based on feedbacks as well as metrics describing whether users provide honest feedbacks. These metrics are often referred to as feedback reputation. Although RepTrap is applicable to a broad range of systems, we choose P2P file-sharing system as the evaluation platform for three reasons. First, there are real trace data[19] and mature models describing user behaviors in P2P networks[12, 27]. Second, P2P networks, such as Kazaa [2], eMule [1] and Maze[19], are becoming the most important infrastructure to share content and services among users in a distributed way. Third, reputation systems play critical roles in P2P to share content and services among users in a distributed way. It is important to point out that many applications provide services in a distributed manner, such as P2P file-sharing systems in which users can directly interact with each other. However, the reputation system can be centralized. That is, there are one or several central authorities collecting feedbacks, publishing reputation scores, and detecting abnormalities. The central authority is referred to as Trust Manager (TM). It does not mean the system must have a central server. The central authority can be formed by a group of users. For example, even in a pure distributed P2P file-sharing network, we can build a reputation system with DHT[22]. In this paper, we focus on the usage of RepTrap in centralized reputation systems. Its extension to distributed reputation systems will be investigated in the future work.

2. REPUTATION SYSTEM REVIEW AND RELATED WORK

Reputation systems have been used in many application scenarios. Instead of studying a specific reputation system for a specific application, we would like to broaden the scope of this investigation by taking the following 3 steps.

- Step 1: We summarize the common properties of prevalent application scenarios of reputation systems, and build architecture of reputation systems without specifying the algorithms for reputation calculation.
- Step 2: Based on this architecture, we discover the new attack that can undermine the foundation of reputation systems.
- Step 3: To demonstrate the performance of the attack, specific algorithms and application scenario are put back into the picture. We will quantitatively evaluate the attack in a P2P file-sharing system based on real trace data.

Step 1 and 2 will be described in this section. Step 3 will be discussed in Section 3 and demonstrated in Section 4.

2.1 Application Scenarios

Feedback-based reputation systems are used in many online applications as discussed in Section 1. These application systems have two common properties. 

- First, they have reputation mechanisms that provide information to users for assisting their decision-making. For example, P2P file-sharing systems can provide scores representing the quality of files such that users can decide which file to download[25]; product rating systems provide rating scores describing the quality of the products to guide users’ purchasing decisions; eBay maintains the reputation scores of sellers/buyers such that one can decide whether to trust a seller/buyer. All above scores can be viewed as reputation, and are referred to as the primary reputation in this work. In addition, the party, whose quality is described by the primary reputation score, is referred to as the object. An object in P2P file-sharing, product rating, and user rating system is a file, a product, and a user, respectively. In other words, as the first common property, these application systems calculate and maintain primary reputation scores describing the quality of the objects. So in the rest of the paper, the primary reputation is referred to as the object quality reputation.

- Second, users can provide feedbacks about the quality of the objects, and the feedbacks are the primary source to determine the object quality reputation.

As illustrated in Figure 1, the object quality reputation is determined by feedbacks from users, feedback reputation of users, and object quality reputation. On the other hand, the system must provide incentive to users such that they will provide feedbacks. More importantly, the usage of feedbacks leads to feedback-based attacks against reputation systems. In particular, malicious users can provide dishonest feedbacks and mislead the calculation of object quality reputation. More details will be discussed in Section 2.3.

The feedback reputation is calculated based on the users’ previous feedback behaviors. The feedback behavior is often described by:

- the number of honest feedbacks given by this user previously, denoted by $G_H$;
- the number of dishonest feedbacks given by this user previously, denoted by $B_D$;
- time when the feedbacks were given.

As a summary, our contributions are (1) discovery of a new and powerful attack against feedback-based reputation systems, which reveals a severe vulnerability in prevalent reputation systems; (2) formal attack strategies to maximize the effectiveness of this attack; and (3) an in-depth study on the impact of the attack in P2P file-sharing systems.

The rest of the paper is organized as follows. An overview of feedback-based reputation systems and related work are presented in Section II. Section III presents the RepTrap attack, including basic ideas, case studies and detailed attack methods. Section IV demonstrates the performance evaluation. Some additional discussions are provided in Section V, followed by the conclusion in Section VI.
The algorithm that calculates user feedback reputation is referred to as the FR Algorithm.

Evidence collection: The Evidence Collection (EC) module provides inputs to the QR algorithm and the FR algorithm. The module needs to judge whether a user’s feedback is honest and whether an object has high quality.

It is important to point out that the system usually does not know the ground truth about whether an object is high-quality or low-quality. Therefore, the EC often needs to estimate the ground truth. The common practice is to assume that the object quality reputation calculated by the reputation system is accurate. As we will discuss in the next section, this assumption introduces vulnerabilities into the reputation system.

Abnormal detection: This module detects low-quality objects, malicious users, attacks against application systems, attacks against reputation systems, and any other abnormalities.

Finally, we would like to point out that the current literature already provides ample choices for QR and FR algorithms. Examples include the beta-function based method in [16], the simple summation method in eBay [8], the entropy-based methods in [26]. An overview of more algorithms can be found in [17]. Meanwhile, the research on abnormality detection is still in its early stage.

2.3 Related Work

Generally speaking, the basic goal of malicious attacks against a reputation system is to boost or reduce the reputation of certain objects. There is an arms race between the attack and defense efforts.

Attack: To boost or reduce the object quality reputation, the most straightforward method is to insert dishonest feedbacks. This is referred to as the bad-mouthing attack [23]. (This attack has other names. For example, in eBay-like systems, ballot stuffing refers to a seller collecting positive feedbacks from fake transactions with friends; bad-mouthing refers to deliberately lowering others’ reputation through negative feedbacks.) In this paper, we adopt the broader definition of bad-mouthing in [15], and use RepSelf to denote this attack.

Defense: To defeat the bad-mouthing attack, the reputation system adopts the feedback reputation [24]. Feedback reputation also has different names, such as “users’ overall reputation score” in [18] and “trust in raters” in [28].

Attack: If feedback reputation is used, the attackers need to maintain their feedback reputation after providing dishonest feedbacks. The attackers can provide honest feedbacks to the objects that they are not interested in. This is referred to as reputation recovery [23] or self-promoting [15]. To avoid confusion, we use RepSelf to denote this attack in this paper.

Defense: To reduce the effectiveness of feedback reputation self-promoting, the system can limit the maximum number of feedbacks per user in a given time interval. This is reasonable because regular users do not use/evaluate a large number of objects in a short period of time. Therefore, the attackers must balance the usage of the limited number of feedbacks between attack and self-promoting, i.e. either to (1) provide a large number of dishonest feedbacks and have not-so-good feedback reputation or (2) to provide a small number of dishonest feedbacks and maintain a good feedback reputation. By limiting the number of feedbacks per user, the reputation system can suppress either the impact of each dishonest feedback or the number of dishonest feedbacks.

3. REPTRAP: A NEW ATTACK AGAINST REPUTATION SYSTEMS

3.1 Basic Ideas

In self-promoting, the attackers only increase their own feedback reputation. How about reducing the honest users’ feedback reputation? Can the attackers hurt honest users’ feedback reputation and improve their own feedback reputation at the same time? In this paper, we move the attack-defense arms race one step further and propose a new attack strategy, called Reputation Trap (RepTrap).

RepTrap Attack: Attacking for some high-quality objects that have a small number of feedbacks. Then, attackers provide a large number of negative feedbacks to these objects such that these objects are marked as low-quality by the system. That is, the system makes wrong judgement about the object quality. As a consequence, the system thinks the attackers’ negative feedbacks agree with the object quality and the feedbacks from honest users disagree with the object quality. Therefore, the feedback reputation of the attackers will be increased while the feedback reputation of honest users will be reduced.

RepTrap is applicable under three conditions. First, there exist high quality objects with a small number of feedbacks. Many existing applications, such as P2P file-sharing systems and Amazon.com, satisfy this condition. Second, the reputation system uses feedback reputation to calculate object quality reputation. Third, the system does not know the ground truth about the object quality and has to estimate the ground truth based on the object quality reputation. This is true in many large scale open systems. It is also noted that this condition may not be satisfied in some semi-centralized systems where a number of experts are hired to evaluate product quality and the expert opinions override users’ feedbacks.

As a summary, the RepTrap attack hurts the feedback reputation of honest users and boosts the feedback reputation of attackers by undermining the system’s estimation of object quality. The RepTrap attack also has other effects, such as hurting good users’ incentive to collaboration, which will be discussed in Section 5.

3.2 Case Studies

To illustrate the basic idea of RepTrap, we create an example in a simplified reputation system. In this section, we first describe this simple reputation system, then present the example, and finally compare the effectiveness of various attacks in this example scenario.

3.2.1 A Simple Reputation System

This simple reputation system takes binary feedbacks. Positive feedback indicates high quality and negative feedback indicates low quality. One user can provide at most one feedback to one object. The beta-function based trust model
is adopted to calculate reputation scores in the QR algorithm and FR algorithm. The EC module uses thresholding method. Particularly,

1. The feedback reputation of user \( u_j \) is calculated as
   \[
   R^{fr}(u_j) = \frac{G_{fb}(u_j) + 1}{G_{fb}(u_j) + B_{fb}(u_j) + 1},
   \]
   where \( G_{fb} \) and \( B_{fb} \) are defined in Section 2.2.

2. The quality reputation of object \( O_k \) is calculated as a weighted average of feedback values with the weight factors as users’ feedback reputation.
   \[
   R^{fr}(O_k) = \sum_{u_j \in U^k_{pos}} R^{fr}(u_j) + \sum_{u_i \in U^k_{neg}} R^{fr}(u_i),
   \]
   where \( U^k_{pos} \) and \( U^k_{neg} \) denote the set of users who give object \( O_k \) positive and negative feedbacks, respectively.

3. Update the \( G_{fb} \) and \( B_{fb} \) values using the following rules.
   - If \( R^{fr}(O_k) \geq threshold \), \( O_k \) is marked as a high-quality object; if \( R^{fr}(O_k) < threshold \), \( O_k \) is marked as a low-quality object.
   - If a user \( u_j \) provides a positive (or negative) feedback to an object marked with high (or low) quality, the value of \( G_{fb}(u_j) \) is increased by 1. Otherwise, if the feedback and the object quality reputation do not match, \( B_{fb}(u_j) \) is increased by 1.

4. If the values of \( G_{fb} \) and \( B_{fb} \) for any users are updated, go to step 1.

3.2.2 A Simple Application Scenario

We create a simple scenario with 5 objects: \( O_1, O_2, O_3, O_4, \) and \( O_5 \), and 3 honest users: \( u_1, u_2, \) and \( u_3 \). All the objects have high quality. \( u_1 \) gives four feedbacks; while \( u_2 \) and \( u_3 \) each gives two feedbacks.

\[
\begin{align*}
O_1 & \leftarrow u_1, O_2 \leftarrow u_1, O_3 \leftarrow u_1, O_4 \leftarrow u_1 \\
O_1 & \leftarrow u_2, O_2 \leftarrow u_2, O_3 \leftarrow u_3, O_5 \leftarrow u_3
\end{align*}
\]

where \( O_k \leftarrow u_j \) denotes that user \( u_j \) provides a feedback for object \( O_k \). The relationship between objects and users is illustrated in Figure 2. To concisely describe the status of an object, we introduce the following notations.

\[
R^{fr}_{pos}(O_k) = \sum_{u_j \in U^k_{pos}} R^{fr}(u_j), \quad R^{fr}_{neg}(O_k) = \sum_{u_i \in U^k_{neg}} R^{fr}(u_i).
\]

Then, (1) becomes

\[
R^{fr}(O_k) = \frac{R^{fr}_{pos}(O_k)}{R^{fr}_{pos}(O_k) + R^{fr}_{neg}(O_k)}. \quad (2)
\]

For the example in this section, we set \( threshold = 0.3 \). And we can then calculate the reputations scores using the algorithms described in Section 3.2.1. When there are no malicious users, the feedback reputation of the honest users is:

\[
u_1 : 0.83, \quad u_2 : 0.75, \quad u_3 : 0.75
\]

and the object quality reputation is

\[
O_1 : 1.00, \quad O_2 : 1.00, \quad O_3 : 1.00, \quad O_4 : 1.00, \quad O_5 : 1.00
\]

Thus, all of the objects are marked as high quality.

3.2.3 Goal of Attacker

In this example, the attacker’s goal is to make the system mark \( O_1 \) as a low-quality object while minimizing the attack effort.

The attack effort is described by two values \((K, F)\). Here, \( K \) is the number of user IDs under the attacker’s control. These user IDs are also referred to as malicious users. \( F \) is the total number of feedbacks from the malicious users. Higher are the \( K \) and \( F \) values, more resources the attacker needs to spend. Since acquiring a user ID is usually harder than providing a feedback, reducing the \( K \) value has higher priority than reducing the \( F \) value, from the attacker’s point of view. Thus, the attacker first finds the minimal \( K \) value, and then finds the minimum \( F \) value given the minimal \( K \) value.

The attacker’s goal is then translated into finding a way to provide feedbacks such that (1) the \( K \) value is minimized and (2) the \( F \) value is minimized given the minimal \( K \) value, under the constraint

\[
R^{fr}(O_1) < threshold \quad (3)
\]

3.2.4 Comparison among Different Attack Strategies

In this subsection, we compare RepTrap with RepBad and RepSelf, which are defined in 2.3. From (2) and (3), we derive the necessary and sufficient condition for making the reputation system mark an object \((O_k)\) as low quality. The condition is:

\[
R^{fr}_{neg}(O_k) > R^{fr}_{pos}(O_k) \cdot \frac{1 - threshold}{threshold}. \quad (4)
\]

In this case, if the attacker wants \( O_k \) to be marked as low-quality when there are no previous negative feedbacks given to \( O_k \), the attacker needs to make some malicious users provide negative feedbacks to \( O_k \) and the summation of the malicious users’ feedback reputation needs to be higher than \( \frac{1}{2} R^{fr}_{pos}(O_k) \). We define the value of \( \frac{1}{2} R^{fr}_{pos}(O_k) \) as the hardness value of an object. In this example, the hardness values of the objects are:

\[
O_1 : 5.44; \quad O_2 : 3.69; \quad O_3 : 1.94; \quad O_4 : 1.94; \quad O_5 : 1.75
\]

---

**RepBad**: In the RepBad attack, the feedback reputation for each malicious user is \( 0.5 \), and the hardness of \( O_1 \) is 5.44. Since 5.44/0.5 = 10.88, the attacker needs at least 11 malicious users to successfully attack \( O_1 \). In this case, the attack power is \((N = 11, K = 11)\).

**RepSelf**: In the RepSelf attack, the malicious users first provide honest feedbacks to the objects that they do not want to attack. In this example, they can provide positive feedbacks to \( O_2, O_3, O_4 \), and accumulate their feedback reputation up to \((4 + 1)/(4 + 2) = 0.83\). Since
3.3 Basic Concepts in RepTrap

As introduced in Section 3.1, the core idea of RepTrap is to degrade the feedback reputation of honest users by undermining the system’s estimation on object quality. In Section 3.2, a case study shows the effectiveness of RepTrap. To fully understand this new attack, we must develop a systematic procedure to conduct the RepTrap attack. Whereas the formal procedure will be described in Section 3.4, we introduce the basic concepts in this subsection.

3.3.1 Object-User Graph

Figure 2 is created to represent the relationship between objects and users. On the left side of the graph, each vertex represents an object. On the right side of the graph, each vertex represents a user. Each object has a popularity score \((a, b)\), where \(a\) is the number of positive feedbacks and \(b\) is the number of negative feedbacks given to this object. If a user provides feedback to an object, a link is created between the two objects. The weight of the link is \(-1\) if the feedback is negative, and 1 if the feedback is positive. The graph is referred to as the object-user graph.

3.3.2 Correlation Between Objects

Based on the object-user graph, we can calculate the correlation between object \(O_i\) and \(O_j\), denoted by \(\text{Cor}(O_i, O_j)\). This correlation describes how \(O_i\)’s reputation is affected if \(O_i\) becomes a trap. \(\text{Cor}(O_i, O_j)\) is calculated in three steps.

1. Between \(O_i\) and \(O_j\), find all the paths with length 2. For example, if user \(u_k\) provides feedbacks to both \(O_i\) and \(O_j\), there exists a path \(O_i - u_k - O_j\) and the path length is 2.

Table 1: Reputation and hardness values in case study

<table>
<thead>
<tr>
<th>Initial state</th>
<th>After 1st round</th>
<th>After 2nd round</th>
<th>After 3rd round</th>
<th>After 4th round</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R^i_{\text{tr}}(u_1))</td>
<td>0.83</td>
<td>0.83</td>
<td>0.67</td>
<td>0.50</td>
</tr>
<tr>
<td>(R^i_{\text{tr}}(u_2))</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>(R^i_{\text{tr}}(u_3))</td>
<td>0.75</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>(R^i_{\text{tr}}(X_1))</td>
<td>0.50</td>
<td>0.67</td>
<td>0.75</td>
<td>0.80</td>
</tr>
<tr>
<td>(R^i_{\text{tr}}(X_2))</td>
<td>0.50</td>
<td>0.67</td>
<td>0.75</td>
<td>0.80</td>
</tr>
<tr>
<td>(R^i_{\text{tr}}(X_3))</td>
<td>0.50</td>
<td>0.67</td>
<td>0.67</td>
<td>0.67</td>
</tr>
<tr>
<td>(R^i_{\text{tr}}(X_4))</td>
<td>0.50</td>
<td>0.67</td>
<td>0.67</td>
<td>0.67</td>
</tr>
<tr>
<td>(\sum_{i=1}^{4} R^i_{\text{tr}}(X_i))</td>
<td>2.00</td>
<td>2.68</td>
<td>2.92</td>
<td>3.07</td>
</tr>
<tr>
<td>Hardness((O_1))</td>
<td>3.60</td>
<td>3.60</td>
<td>3.31</td>
<td>2.92</td>
</tr>
<tr>
<td>Hardness((O_2))</td>
<td>1.94</td>
<td>1.94</td>
<td>1.56</td>
<td>#</td>
</tr>
<tr>
<td>Hardness((O_3))</td>
<td>1.94</td>
<td>1.94</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>Hardness((O_4))</td>
<td>1.75</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
</tbody>
</table>

5.44/0.83 = 6.55, at least 7 malicious users are needed to successfully attack \(O_1\). Not all 7 malicious users need to provide 4 honest feedbacks. After enumerating all possible ways to perform self-promoting, we find that the malicious users need to provide at least 25 feedbacks. To achieve this, we use RepTrap. The attack power of this case study is larger than the hardness of \(O_1\). Thus, totally 18 honest feedbacks and 7 dishonest feedbacks are given in this case. In this attack, the attack power is \((N = 7, K = 25)\).

RepTrap: In the RepTrap attack, the malicious users provide dishonest feedbacks to unpopular high-quality objects. If a high-quality object is marked as low quality by the reputation system, this object is turned into a ‘trap’, which will mislead the feedback reputation of good users who give honest feedback to this trap.

It is difficult to find the optimal way to conduct the RepTrap attack. Instead, we just show one method to successfully attack \(O_1\) using RepTrap. The attack power of this method is \((N = 4, K = 18)\).

1. The initial feedback reputation of the four malicious users is 0.5. Since 0.5 × 4 = 2 is larger than the hardness of \(O_1\) (i.e. 1.75), they can turn \(O_1\) into a trap by providing negative feedbacks to \(O_1\). This has three consequences. First, the feedback reputation of the malicious users increases to \((1 + 1)/(1 + 2) = 0.67\). Second, the feedback reputation of good user \(u_3\) is reduced to \((1 + 1)/(2 + 2) = 0.5\). Third, the value of \(R^i_{\text{tr}}(O_1)\) and the hardness of \(O_1\) is reduced. The results are shown in the second column in Table 1, and we use \(X_1, X_2, X_3\) and \(X_4\) to denote the four malicious users.

2. After the first step, three malicious users can turn \(O_4\) into a trap because the summation of their feedback reputation is larger than the hardness of \(O_4\), i.e. 0.67 × 3 > 1.94. Then, the feedback reputation of these three malicious users is increased to \((2 + 1)/(2 + 2) = 0.75\), and the feedback reputation of the honest user \(u_1\) is reduced to \((3 + 1)/(4 + 2) = 0.67\). The hardness of \(O_1, O_2\) and \(O_3\) is also reduced, because their hardness depends on the feedback reputation of \(u_1\). See the third column in Table 1 for details.

3. The three malicious users in the second step can further turn \(O_3\) into a trap because 0.75 × 3 = 2.25 is larger than the current hardness of \(O_3\). Then, the feedback reputation of these three malicious users is increased to \((3 + 1)/(3 + 2) = 0.8\), the feedback reputation of \(u_1\) is reduced to \((2 + 1)/(4 + 2) = 0.5\), and the hardness of \(O_1\) and \(O_2\) continues to drop.

4. Similarly, all four malicious users can turn \(O_2\) into a trap. Then, the feedback reputation of malicious users becomes 0.83 or 0.75, the feedback reputation of \(u_1\) becomes 0.33, and the hardness of \(O_1\) is reduced to 3.11.

5. Finally, the summation of feedback reputation of the four malicious users is 0.83 × 3 + 0.75 = 3.24, which is larger than 3.11. This means that the malicious users can now make the system mark \(O_1\) as low quality. In total, the malicious users give 4 + 3 + 3 + 4 + 4 = 18 feedbacks.

In this case study, RepTrap reduces the requirement on the number of malicious users by 64% when compared with RepBad, and by 43% when compared with RepSelf. RepTrap also requires 28% less feedbacks than RepSelf. In other words, RepTrap is a much stronger attack than the existing ones in manipulating the reputation system.
2. For each path, calculate the path_effect value as the product of the weights of the two links. Thus, the path_effect value can be either 1 or \(-1\). When \(\text{path\_effect} = 1\), the quality reputation of \(O_i\) will move opposite to its original value if \(O_i\) is turned into a trap. That is, high object reputation will become lower, and low object reputation will become higher. When \(\text{path\_effect} = -1\), the object reputation will be maintained. That is, the effect of this path is to make high object reputation become higher and low object reputation become lower. The truth table (Table 2) supports the above statements, which will be explained later.

3. Let \(N_1\) denote the number of paths with positive path_effect values, and \(N_2\) denote the number of paths with negative path_effect values. Then, the correlation is calculated as

\[
\text{Cor}(O_i, O_j) = N_1 - N_2.
\]

It is noted that \(\text{Cor}(O_i, O_j)\) can be used for trap selection. If \(\text{Cor}(O_i, O_j) > \text{Cor}(O_j, O_i)\), turning \(O_i\) into a trap is more likely to have a greater impact on the reputation of \(O_j\) than turning \(O_j\) into a trap. In the example shown in Figure 2, \(\text{Cor}(O_2, O_1) = 2\) and \(\text{Cor}(O_5, O_1) = 1\).

To understand table 2, we examine the path \(O_2 - u_1 - O_1\) shown in Figure 2 with the elements in the first row of Table 2 explained one by one, from left to right.

1. Initially, \(O_2\) is marked as high quality by the system;
2. \(u_1\) gives an honest feedback to \(O_2\) which is positive;
3. Since \(u_1\)'s feedback agrees with \(O_2\)'s quality reputation, the weight of link \(O_2 - u_1\) is \(1\);
4. If \(O_2\) is turned into a trap, the system will mark \(O_2\) as low quality and also think \(u_1\) gives a wrong feedback, which leads to a reduction in \(u_1\)'s feedback reputation;
5. Initially, \(O_1\) is marked as high quality by the system;
6. \(u_1\) gives an honest feedback to \(O_1\) which is positive;
7. Since \(u_1\)'s feedback agrees with \(O_1\)'s quality reputation, the weight of link \(u_1 - O_1\) is \(1\);
8. Since the feedback reputation of \(u_1\), who gives a positive feedback, is reduced (see column 4), the object quality reputation of \(O_1\) can also be reduced;
9. Making \(O_2\) into a trap will make a high object quality reputation lower, that is, the object quality reputation of \(O_1\) moves toward the opposite direction.
10. The path_effect value which is the product of two link weights, is \(1\).

The eight rows in Table 2 represent all different cases. It is seen that the path_effect value agrees with the direction of reputation change. It is a good indicator showing the effect of the traps.

### 3.3.3 Object-targeted RepTrap and User-targeted RepTrap

The RepTrap attack can mislead the object quality reputation as well as the feedback reputation of users.

- When the primary attack goal is to mislead the quality reputation of one or multiple specific objects, we call it the object-targeted RepTrap.
- When the primary attack goal is to mislead the feedback reputation of one or multiple specific users, we call it the user-targeted RepTrap.

Using the object-user graph, we can unify these two types of RepTrap attacks. In the user-targeted RepTrap, we create a virtual object. This virtual object is linked to all the users who are the target of the attacker. The weight of the links is \(1\). It is easy to see that reducing the feedback reputation of the users is equivalent to reducing the quality reputation of the virtual object. By creating the virtual object, we can convert the user-targeted RepTrap attack into the object-targeted RepTrap attack. Therefore, in the rest of the section 3, we only describe how to conduct the object-targeted RepTrap.

#### 3.3.4 Knowledge Level of Attacker

In different application scenarios, the attackers may have different amount of knowledge about the reputation systems, which affects their ways to conduct the RepTrap attack. In this subsection, we discuss a few typical cases about the attackers’ knowledge level.

The attacker wants to know all of the following information in the reputation system, including

- \(\text{info}_1\): specific algorithms and parameters (especially thresholds) used in the reputation system
- \(\text{info}_2\): the information in the object-user graph
- \(\text{info}_3\): feedback reputation of malicious users
- \(\text{info}_4\): feedback reputation of other users

First of all, we cannot assume that the algorithms used in the reputation systems are secrets. In other words, the security and robustness of the reputation system should not rely on the secrecy of the algorithms. This is a general principle in security research. Therefore, we assume that the attacker knows the algorithms in the reputation system.

Second, we assume that the attacker can judge whether an object has been marked as low-quality by the system. This assumption is reasonable because the system must take some actions to deal with low-quality objects, such as removing them from the system or ranking them extremely low compared to other normal objects. Based on this assumption, the attacker can insert positive and negative feedbacks to a newly published object, and examine how the reputation changes and when this object is detected as low-quality. By doing this for multiple times, the attacker can estimate the detection thresholds. Therefore, it leads to another assumption that the attacker can estimate the parameters in the algorithms, e.g. threshold in (3).

Third, in many reputation systems, the information about who provides what feedbacks to which objects is open to the public. In addition, the object quality information is obviously known to all users. In those systems such as Epinion, the object-user graph is known to the attacker. Of course, there are systems that do not publish details about the feedback information.

Furthermore, since the attacker knows the algorithms and the behavior of malicious users, the attacker surely knows the feedback reputation of malicious users. Some systems publish information about the trustworthiness of users, which is directly related to the users’ feedback reputation. Examples include Epinion and Amazon. In some other systems, the attacker can easily dig out the history of users giving feedbacks. Knowing this history and the algorithms, the attacker can estimate the feedback reputation of good users.
Of course, it is possible that the attackers cannot obtain this information in some systems. Based on the above discussion, we define three knowledge levels:

- Knowledge Level 1 (KL1): the attacker knows info\textsubscript{1} and info\textsubscript{2}.
- Knowledge Level 2 (KL2): the attacker knows info\textsubscript{1}, info\textsubscript{2}, and info\textsubscript{3}.
- Knowledge Level 3 (KL3): the attacker knows info\textsubscript{1}, info\textsubscript{2}, info\textsubscript{3}, and info\textsubscript{4}.

In Section 3.4, we will first present the specific attack procedure for the KL3 case, and then extend the procedure to KL2 and KL1 cases.

### 3.4 RepTrap Attack Procedure

Before we give a formal RepTrap Attack procedure, some important terminologies and notations are summarized.

- **Object Correlation** between object \( O_i \) and \( O_j \) is denoted by \( \text{Cor}(O_i, O_j) \) (see Section 3.3.2).

- **Malicious Users** are the user IDs under the control of the attacker. We use \( m_i \) to denote a malicious user.

- **Attack Capability of a Malicious User** describes the impact of the feedback from this particular user. Let \( a_i \) denote the attack capability of the malicious user \( m_i \). For example, we can choose \( a_i \) as the feedback reputation of \( m_i \).

- **Attack Capability of a Set of Malicious Users** describes the overall impact when every user in this set gives feedback to the same object. Let \( A(S) \) denote the attack capability of user set \( S \). The simplest way to calculate \( A(S) \) is \( A(S) = \sum_{i,m_i \in S} a_i \).

- **Hardness Value** of an object describes the difficulty level in turning this object into a trap. Let \( H(O_k) \) denote the hardness of object \( O_k \). When the object quality reputation is calculated using (2), one reasonable way to calculate \( H(O_k) \) is

\[
H(O_k) = R^r_{\text{pos}}(O_k) \cdot \frac{1 - \text{threshold}_{\text{pos}}}{\text{threshold}_{\text{pos}}} - R^r_{\text{neg}}(O_k).
\]

where \( R^r_{\text{pos}}(O_k) \) (or \( R^r_{\text{neg}}(O_k) \)) is the summation of feedback reputation of the users who have provided positive (or negative) feedbacks to \( O_k \). In the case of KL3, the attacker can calculate \( H(O_k) \) using (6).

- **Malicious User Set**, denoted by \( S_m \), contains all the malicious users that are under the control of the attacker.

- **Target Object Set**, denoted by \( S_t \), is the set of objects that are the attacker’s target. That is, the attacker wants to reduce or boost the reputation of the objects in \( S_t \).

- **Related Object Set**, denoted by \( S_r \), contains the objects that have positive correlation with at least one object in \( S_t \). In other words, turning an object in \( S_t \) into a trap will help to attack the objects in \( S_t \).

- **Candidate Object Set**, denoted by \( S_c \), contains the objects that satisfy two conditions:
  - They are marked as high-quality objects.
  - The objects’ hardness values are smaller than the attack capability of the malicious user set. That is, if \( O_k \in S_c \), then \( H(O_k) < A(S_m) \).

In other words, the attacker only turns high-quality objects into traps, and the attacker does not try to create a trap unless the attacker has enough attack capability. Note that all traps are marked as low-quality. Since regular users usually do not use low-quality objects, traps will not receive many new feedbacks from honest users. On the other hand, if the system marks a low-quality object as high-quality, this object will be used by many users and quickly receives a large number of negative feedbacks. The system will correct its wrong judgement quickly.

- **Effectiveness Value** of an object describes how much influence the attacker can have on the target set if the attacker turns this particular object into a trap. The effectiveness value of the object \( O_k \) can be calculated as

\[
E(O_k) = \sum_{j \in S_t} \text{Cor}(O_k, O_j)
\]

- **Feedback Allocation** describes the specific behavior of the malicious users. From the attacker’s point of view, the most challenging job is to determine how many negative feedbacks should be given to which objects in order to create traps, i.e. how to allocate the dishonest feedbacks. We introduce the notation

\[
F(A(S_c; S_{\text{neg}}(O_k)) \text{ for } k \in S_c)
\]

which means that the malicious users in set \( S_{\text{neg}}(O_k) \) provide negative feedbacks to \( O_k \), and \( O_k \) is in the candidate set \( S_c \).

- **Round** is the time unit in which the attacker can adjust the feedback allocation. Without losing generality, one malicious user ID can provide up to one feedback in each round. The concept of round simplifies the description of the RepTrap attack procedure.

<table>
<thead>
<tr>
<th>( R^r(O_k) ) feedback to ( O_k )</th>
<th>( u_k )’s feedback to ( O_k )</th>
<th>weight of link ( O_i - u_k )</th>
<th>if ( O_i ) becomes trap, change in ( R^r(O_k) )</th>
<th>( u_k )’s feedback to ( O_k )</th>
<th>weight of link ( O_i - u_k )</th>
<th>change in ( R^r(O_k) ) due to change in ( R^r(u_k) )</th>
<th>maintain original ( R^r(O_k) )?</th>
<th>path effect value</th>
</tr>
</thead>
<tbody>
<tr>
<td>high positive 1 reduced high positive 1 reduced high negative 1 negative 1 increased</td>
<td>opposite 1</td>
<td>Opposite 1</td>
<td>Opposite 1</td>
<td>Opposite 1</td>
<td>Opposite 1</td>
<td>Opposite 1</td>
<td>Opposite 1</td>
<td>Opposite 1</td>
</tr>
</tbody>
</table>

Table 2: Truth table for verifying that path\_effect value represents the effects of a trap upon \( O_i \)’s object reputation.
The core task of the RepTrap attack is to determine the feedback allocation in each round such that the quality reputation of the objects in the target set ($S_t$) is affected most. One round of the RepTrap attack is conducted in 9 steps.

In **Step 1**, construct the related object set ($S_t$) based on the definition given previously.

In **Step 2**, calculate the hardness value for each object in $S_t$, i.e. $H(O_j)$, for $j : O_j \in S_t$.

In **Step 3**, calculate the attack capability of each malicious user, as well as the malicious user set, i.e $A(S_m)$.

In **Step 4**, construct the candidate object set ($S_c$) by comparing candidate $A(S_m)$ and the hardness values of the objects in $S_t$.

In **Step 5**, for each object in the candidate set ($S_c$), calculate its correlation with all other objects in the target object set ($S_t$).

In **Step 6**, calculate the effectiveness value for each object in $S_c$, i.e. $E(O_j)$ for $j : O_j \in S_c$.

In **Step 7**, determine the feedback allocation in this round by solving the following optimization problem.

**Procedure 1 Feedback Allocation Algorithm**

1: Set $S_{neg}(O_k) = \phi$ for $k : O_k \in S_c$.
2: Let $S_o$ denote the set of unused malicious users. Set $S_a = S_m$.
3: Set $S_o$ denote the set of remaining objects that can be turned into traps. Set $S_o = S_c$.
4: Calculate the benefit value as $B_k = E(O_k)/H(O_k)$, for $k : O_k \in S_o$.
5: **while** $A(S_o)$ is not empty **do**
6: From $S_o$, select the object that has the largest benefit value. Assume that this object is $O_e$.
7: From $S_a$, select a set of the malicious users, denoted by $S_m$, such that $A(S_m) > H(O_e)$ and the number of malicious users in $S_m$ is minimized.
8: **if** the selection of $S_o$ is successful **then**
9: set $S_{neg}(O_e) = S_m$
10: remove $O_e$ from $S_o$
11: **end if**
12: Remove $O_e$ from $S_o$
13: **Remove the objects whose Hardness is not less than $A(S_o)$ from $S_o$**
14: **end while**

The optimization problem in step 7 is NP-hard. Therefore, we developed a sub-optimal solution as shown in Procedure 1.

In **Step 8**, the attacker inserts feedbacks according to the feedback allocation calculated in step 7.

In **Step 9**, at the end of each round, the attacker checks whether the traps are successfully created. If $O_i \in S_t$ is turned into a trap, set the trap indicator $T_i$ as 1. (The default $T_i$ value is 0). If $O_i$ is not successfully turned into a trap for some reason, $O_i$ is put into the candidate set in the next round. This might happen only with a very small probability.

After the end of the round, the attacker can go back to step 1 and do more rounds, until the ending condition is met.

After reducing the good users' feedback reputation and enhancing their own feedback reputation through the RepTrap attack, the attacker is ready to attack the objects in the target set $S_t$. According to the specific attack goals, the attacker provides either positive or negative feedbacks to the objects in $S_t$.

In this 9-step procedure, two issues are not specified: the ending condition and the solution to the optimization problem in step 7. Next, we address these two issues.

The **ending condition** depends on the specific goal of attacker. Here are two examples.

- If the attacker’s goal is to reduce or boost the quality reputation of the objects in $S_t$ as much as possible, the ending condition is
  - the candidate set $S_c$ is empty;
- or the attacker estimates that he can achieve the attack goal given the users' feedback reputations in a new round.

If the attacker has knowledge KL3, he can easily perform this estimation. If the attacker has knowledge KL1 and KL2, he may try to attack the objects in $S_t$ and see how the object reputation changes.

The optimization problem in step 7 is NP-hard. Therefore, we developed a sub-optimal solution as shown in Procedure 1.
feedback reputation of the malicious users is increased. If no, there is no room for improvement. Otherwise, the attacker runs a few enhancement rounds to boost the feedback reputation of malicious users, which will improve their attack capability. Then, the attacker conducts regular RepTrap rounds until the ending condition is met again. The usage of the enhancement rounds can slightly improve the effectiveness of RepTrap.

### 3.5 RepTrap Attack for KL1 and KL2 Cases

The RepTrap attack procedure in Section 3.4 is obviously suitable for the case that the attacker has KL3. In this section, we briefly discuss how to conduct RepTrap for the KL1 and KL2 cases.

When the attacker does not know who provide feedbacks to the target set $S_t$ (i.e. KL1), it is difficult to determine an effective strategy for creating traps. In this case, the attacker can simply try to create as many traps as possible. Note that creating traps not only make the objects in $S_t$ easier to attack, but also make the system publish wrong object quality reputation.

When the attacker does not know the feedback reputation of good users, the attacker cannot calculate the hardness value $H(O_k)$ accurately. Instead, the attacker needs to estimate $H(O_k)$. For example, the attacker can estimate $H(O_k)$ as

$$\hat{H}(O_k) = N(O_k) \cdot \frac{R_{avg}^{fr}}{\text{threshold}},$$

where $N(O_k)$ is the number of users who have provided feedbacks to $O_k$ (excluding the malicious users), and $R_{avg}^{fr}$ is the average feedback reputation of good users. In most systems, $N(O_k)$ is known and the attacker only needs to estimate $R_{avg}^{fr}$. To estimate $R_{avg}^{fr}$, the attacker can use some additional user IDs that provide honest feedbacks to some other objects, just like good users. These additional users’ feedback reputation can be used as a guideline for estimating $R_{avg}^{fr}$. Usually, $R_{avg}^{fr}$ should be overestimated to ensure the success of creating traps.

### 4. PERFORMANCE EVALUATION

#### 4.1 Experiment Description

As discussed in Section 1, we evaluate the performance of RepTrap attack in P2P file-sharing systems. We build a simulator using models similar to [12] and the parameters from Maze [19]. Maze is a popular P2P file-sharing system with more than 2 million registered users and 40,000 users online at peak time. In the simulation, there are 1,000 good users and 100 high quality files. Users follow Poisson process to download files and an important Zipf parameter, which is 0.676, is used to drive the downloading frequency. While another Zipf parameter, which is 0.890, is used to drive the popularity of the files [12]. Without loss of generality, we assume good users always provide honest feedbacks after successfully file downloads. (Of course, we can assume that good users provide honest feedbacks with a certain probability, but this will not have meaningful impact on the simulation results.) Each user can give no more than one feedback to a file. The reputation system model in the simulator is the same as that in Section 3.2.

We compare RepTrap with two other schemes: RepBad and RepSelf, which are described in Section 3.2. The attacks are performed in rounds and 50 malicious users are added. In one round of attack, each malicious user can only provide one feedback. With the RepBad strategy, malicious users attack the target files by providing false feedbacks in the first round. With the RepSelf strategy, malicious users give honest feedbacks to non-targeted files and gain feedback reputation. After getting enough feedback reputation, they provide false feedbacks to the target files. With the RepTrap strategy, malicious users create traps first and then attack the target files.

#### 4.2 Attack model 1: Targeting a popular file

In a P2P file-sharing system, the popularity of files is driven by the Zipf parameter. More popular is a file, more feedbacks from good users are given to this file, and therefore more difficult for malicious users to conduct the attack. In this experiment, We attack the 1st (i.e. top 1) and the 10th (i.e. top 10) popular files and aim to make the system identify them as low quality files (i.e. faked files or pollution).

If the malicious users start an attack at the beginning of the simulation, the attack is much easier to succeed. This is because there are few feedbacks on each file and even a small number of false feedbacks can easily slander a good file. As the time goes, more feedbacks are provided to the files and the attack gets harder and harder to succeed with limited number of malicious users. We therefore change the starting time of the attack to show the attack performance measured by success probability. For each starting time, we conduct 1,000 simulations and success probability is calculated as the number of successful attacks divided by 1,000.

Figure 3 and 4 are for the scenarios that malicious users start the attack at the 1st and the 10th popular files, respectively. The horizontal axis is attack starting time and the vertical axis is the success probability. It is obvious that the proposed scheme has the best performance and RepBad has the worst performance. RepBad can only successfully attack the 1st popular file within 2 days and the 10th popular file within 10 days after the file is published. After that time, the success probability decreases sharply. RepSelf has better success probability than RepBad because in RepSelf malicious users can improve their feedback reputation and have more attack power. However, the performance of RepSelf is still much worse than RepTrap. For example, when RepSelf has only 10% of success, RepTrap has 60% and 80% of success in Figure 3 and 4 respectively. These results clearly show
the advantage of reducing honest users’ feedback reputation, which can only be achieved in RepTrap.

In the above experiment, both RepTrap and RepSelf need several attack rounds for the malicious users to gain feedback reputation. To compare the efficiency of RepTrap and RepSelf, we examine the round number, which is the number of rounds needed to gain enough reputation feedback to successfully attack a file. Please note that in each attack round, 50 malicious users provide 50 feedbacks (each user provides one feedback). The number of attack rounds therefore can represent the total amount of feedbacks provided by the malicious users.

Figure 5 and Figure 6 show the number of rounds needed to successfully attack the 1st and the 10th files, respectively. Three schemes are compared: RepSelf, RepTrap in KL3 case and RepTrap in KL2 case. Recall that KL2 and KL3, defined in Section 3.3.4, represent different knowledge levels of the attacker. In KL2, the attacker has to estimate the good users’ feedback reputation that affects the hardness value calculation. We assume it overestimates the hardness values of the objects by 20% in KL2. Compared with RepSelf, both RepTrap methods need much less number of rounds. This means that the RepTrap attackers spend much less resources. Compared with the KL3 case, the KL2 case needs slightly more rounds. It indicates that imperfect knowledge about the good users’ feedback reputation will not largely reduce the power of the RepTrap attack.

4.3 Attack model 2: Blind attack

In this experiment, we consider the KL1 case, in which malicious users know neither the object-user graph nor the good users’ feedback reputation. In this case, RepTrap does not target any specific files or users. Instead, it aims to disturb the entire reputation system by creating as many traps as possible. The malicious users will continue to make traps, starting from the least popular files to more popular files, until they cannot make any more traps.

Figure 7 shows the number of traps created by the 50 malicious users with different attack starting time. It is seen that at the beginning, all files can be turned into traps. The trap number decreases gradually with the starting time. However, even after day 50, more than 80% of the files can be trapped. This is because the popularity of the files in P2P file-sharing systems is driven by the power law distribution, so there are a large number of files downloaded by only a few users. From Figure 3, 4 and 7, we can conclude that RepTrap is a very powerful attack, which can attack most of the files no matter the starting time. The attack performance is only sensitive to the starting time when the attack targets very popular files.

Through analysis and simulations, we have seen that RepTrap can severely degrade the quality reputation of objects and feedback reputation of users. Comparing RepBad and
RepSelf, RepTrap needs the fewest user IDs to get the best achievements. On the other hand, the kind of Sybil attack has been confined and it would significantly increase the cost for an attacker to control more user IDs without detection\[30, 29\]. In this case, it would be wise to maximize the usage of the existing user IDs rather than to create new ones continually.

Meanwhile, in most of the current reputation systems, users give the negative feedbacks much more frequently than the positive feedbacks due to lack of incentive \[7, 17\]. Therefore it is not easy for the system to detect RepTrap based on a large number of negative feedbacks. The effective detection techniques are still highly expected.

In fact, RepTrap has an even broader influence. And we use the P2P file-sharing system as an example to illustrate. First, RepTrap hurts the users’ incentive to contribute. Nobody would like to see that the files they published are marked as fake files. When the system has many traps, fewer people will have strong motivation to publish high-quality files.

Second, many reputation systems calculate the reputation of the users in terms of publish high quality files. This reputation can be referred to as the user service reputation. RepTrap obviously damage the user service reputation by making the system mark high-quality files as low-quality.

Third, many reputation systems have mechanisms to encourage good users and punish bad users. The target users of RepTrap would not receive the reward they deserve, especially the malicious users will not be punished. This will undermine the incentive for participation in the service or system.

Fourth, in this paper, we study the reputation systems using feedback reputation. If RepTrap is not solved, the system may have to give up using feedback reputation, which opens doors to simple bad-mouthing attacks. Even if the system does not use feedback reputation, RepTrap still hurts the object quality reputation by slandering good files directly.

Therefore, a thorough understanding on this attack, as well as developing defense mechanisms, will have significant impact on the security and usability of reputation systems used today and in the future. This paper is the first effort toward understanding and solving this security problem in reputation systems. More research on defense is expected.

6. CONCLUSION

In this paper, we reported the discovery of a new and powerful attack, named RepTrap, against feedback-based reputation systems. Through an in-depth investigation, we presented the scenarios that this attack can be applied to, the ways to effectively conduct this attack, and the consequence of the attack. The performance evaluation is based on real data and realistic user models in a popular P2P file-sharing system. The simulation results as well as the case studies have demonstrated that the RepTrap attack can significantly reduce the resources required to attack popular objects. With the same resources, the success probability of attacks was greatly increased. The broader impact of this attack and several variations of RepTrap were also discussed. The investigation in this paper has moved the arms race between attack and defense in reputation systems to a new level.

7. REFERENCES


