Fake Co-visitation Injection Attacks to Recommender Systems

Guolei Yang
Iowa State University
yanggl@iastate.edu

Neil Zhenqiang Gong
Iowa State University
neilgong@iastate.edu

Ying Cai
Iowa State University
yingcai@iastate.edu

***Abstract*—Recommender systems have become an essential component in a wide range of web services. It is believed that recommender systems recommend a user items (e.g., videos on YouTube, products on Amazon) that match the user's preference. In this work, we propose new attacks to recommender systems. Our attacks exploit fundamental vulnerabilities of recommender systems and can spoof a recommender system to make recommendations as an attacker desires. Our key idea is to inject fake co-visitations to the system. Given a bounded number of fake co-visitations that an attacker can inject, two key challenges are 1) which items the attacker should inject fake co-visitations to, and 2) how many fake co-visitations an attacker should inject to each item. We address these challenges via modelling our attacks as *constrained linear optimization problems*, by solving which the attacker can perform attacks with maximal threats. We demonstrate the feasibility and effectiveness of our attacks via evaluations on both synthetic data and real-world recommender systems on several popular web services including YouTube, eBay, Amazon, Yelp, and LinkedIn. We also discuss strategies to mitigate our attacks.**

I. Introduction

In the era of information explosion, people face an overwhelming number of choices when looking for information of their interests on the Internet. "*... a wealth of information creates a poverty of attention and a need to allocate that attention efficiently ...*" [1]. *Recommender systems* play a curial role to allocate user attention and help users locate relevant information in a wide range of web services such as YouTube, eBay, and Amazon.

In a recommender system, we have a set of *users* (e.g., registered users, unregistered visitors) and *items* (e.g., videos on YouTube, products on eBay). Two widely used recommendation tasks are *user-to-item* recommendation and *item-to-item* recommendation. In a user-to-item recommendation, the system recommends items to a user based on the user's profile (e.g., the browsing history, the items the user liked or disliked). In an item-to-item recommendation, a list of items are recommended to a user when the user is visiting an item. This recommendation is commonly known as features like "People who viewed this also viewed". One particular category of recommender system to implement the two recommendation tasks, which we call *co-visitation recommender system*, is likely being widely used by web service providers (e.g., YouTube [2], Amazon [3]) due to its effectiveness and simplicity. Co-visitation recommender systems leverage co-visitation information between items, and the key idea is that two items that were frequently co-visited in the past are likely to be co-visited in the future.

It was widely believed that recommender systems should recommend a user items that match the user's preference. However, Xing et al. [4] recently proposed *pollution attacks* to *user-to-item* recommendation, in which the recommender system is spoofed to recommend any *target item* (e.g., a video advertisement on YouTube) to a victim user. Their key idea is to inject fake information, which is related to the target item, into the victim user's profile via cross-site request forgery (CSRF) [5] attacks. However, pollution attacks suffer from the following limitations: 1) pollution attacks rely on CSRF, which makes it hard to perform the attacks at a large scale, and 2) pollution attacks are not applicable to item-to-item recommendation because the attacker cannot change the item that the user is currently visiting.

In this work, we propose new attacks to spoof recommender systems to make recommendations as an attacker desires. Our attacks do not rely on CSRF, can be performed at a large scale, and are applicable to both user-to-item and item-to-item recommendations. In particular, we focus on co-visitation recommender systems. Our key idea is to inject fake co-visitations to the system, and we call our attacks *fake co-visitation injection attacks*. We note that attacking co-visitation recommender systems via injecting fake co-visitations is a natural idea. Our key contribution is to perform the first formal and systematic study on fake co-visitation injection attacks.

First, we propose a novel threat model. In our threat model, we define two attacks to recommender systems. They are *promotion attacks* and *demotion attacks*. A promotion attack is to spoof the recommender system to recommend a target item (e.g., a video advertisement on YouTube, a product on eBay) to as many users as possible. Recommending a target item to more users increases the item's *user impression*, which in turn could lead to more user visits/clicks of the item and eventually purchases of certain products. On the contrary, a demotion attack is to spoof the recommender system to recommend an item to as few users as possible. An attacker can use demotion attacks to demote its competitors' items. Moreover, we consider three categories of attackers with different background knowledge (i.e., *high knowledge*, *medium knowledge*, and *low knowledge*). These background knowledge model a variety of

NDSS '17, 26 February - 1 March 2017, San Diego, CA, USA
Copyright 2017 Internet Society, ISBN 1-1891562-46-0
http://dx.doi.org/10.14722/ndss.2017.23020

web services and attack scenarios. For instance, in a high knowledge scenario, the attacker knows the recommendation system's model details, which represents an upper bound of what an attacker can achieve; in a low knowledge scenario, the attacker only knows the publicly available recommendation lists made by the system.

Second, we propose fake co-visitation injection attacks to implement promotion and demotion attacks for different background knowledge. Our key idea is to use scripts to automatically co-visit a target item and some items, which we call *anchor items*, such that the target item appears in the anchor items' item-to-item recommendation lists. In practice, the number of fake co-visitations that an attacker can inject is often bounded (though it is still large) due to resource constraints and some mitigation techniques deployed by the service providers [6]. Given a bounded number of fake co-visitations, two key challenges for an attacker are 1) which items should be selected as anchor items, and 2) how many fake co-visitations should be injected between the target item and each anchor item, such that the threat of the attack is maximized, e.g., the target item is recommended to the largest number of users for promotion attacks. We address these challenges via modelling our attacks as *constrained linear optimization problems*, by solving which an attacker obtains the anchor items and the number of fake co-visitations for each anchor item.

Third, we demonstrate the feasibility and effectiveness of our attacks via performing extensive evaluations on both synthetic recommender systems and real-world recommender systems. In particular, we demonstrate that the recommender systems used by several popular web services including YouTube, eBay, Amazon, Yelp, and LinkedIn are vulnerable to our attacks. For instance, in the experiment of YouTube, using a single computer with moderate computing power, we are able to promote 20 target videos within three weeks, each of which appears in the item-to-item recommendation lists of more than 200 anchor videos on average; for each target video, the total number of views of the anchor videos is more than 6×10^5 on average. We note that, after our attacks, a target video is going to be shown to any user who views any of these anchor videos. Moreover, our attack can promote a target video to be in the user-to-item recommendation list of a newly registered user with a small number of fake co-visitations.

Finally, we discuss strategies to mitigate our attacks. For instance, for web services like YouTube, one mitigation strategy to balance between security against our attacks and usability is to hide the exact number of views for a video and only shows its range. We demonstrate, via evaluations on synthetic data, that this strategy can mitigate our attacks significantly.

We summarize our main contributions as follows:

- We present the first formal and systematic study about fake co-visitation injection attacks to co-visitation recommender systems.
- We propose a novel threat model to cover a variety of attackers with different goals and background knowledge. We formulate the fake co-visitation injection attacks as constrained linear optimization problems, by solving which an attacker can perform attacks with maximal threats.
- We demonstrate the feasibility and effectiveness of our attacks on both synthetic and real-world recommender systems used by several popular web services. We also discuss strategies to mitigate our attacks.

II. Background and Related Work

A. Co-visitation Recommender Systems

Recommender system has become an essential component in many web services (e.g., YouTube, eBay, Amazon, and Yelp). In a recommender system, we have a set of *users* and *items*. A user could be a registered user or an unregistered visitor of a web service. Items are different on different web services, e.g., items are videos on YouTube, while they are products on Amazon. The goal of a recommender system is to recommend a user items that match the user's preference.

Many recommender systems (e.g., content-based systems [7, 8] and collaborative filtering based systems [2, 3, 9–12]) have been developed in the past two decades. We refer readers to surveys [13, 14] on recommender systems for details. Among these systems, one particular collaborative filtering based system, which we call *co-visitation recommender system*, is likely being widely used by web services because of its effectiveness and simplicity. For instance, co-visitation recommender system is used by YouTube to recommend videos according to Google's official report [2], and it is used by Amazon to recommend products according to Amazon's publication [3]. In this work, we focus on co-visitation recommender systems.

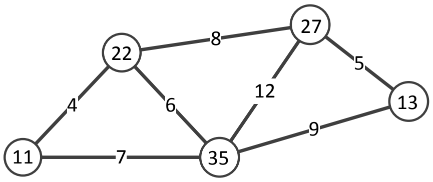
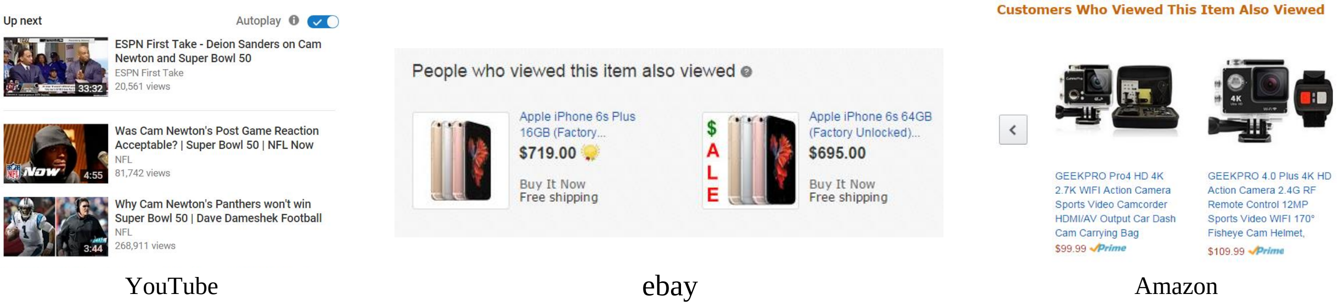


Fig. 1: Illustration of a co-visitation graph. The weight on edge (i, j) is the number of times that items i and j were co-visited, while the weight on node i is the total number of times that i was visited.

Co-visitation graph: Two items were *co-visited* by a user if the user visited both of them. For instance, on YouTube, two videos are co-visited by a user if the user watched one video after watching the other one in the same browser session [2]. The key component of a co-visitation recommender system is a data structure that we call *co-visitation graph*. Fig. 1 illustrates an example co-visitation graph. We denote a co-visitation graph as $G = (V, E)$, where each node i is an item and an edge (i, j) means that items i and j were co-visited by at least one user. Each edge (i, j) in the co-visitation graph has a weight, which is the number of times that i and j were co-visited. We call the number of times that an item i was visited the *popularity* of i. In the co-visitation graph, we represent the popularity of an item i as the *node weight* of i. Note that, in the co-visitation graph, the node weight (i.e., popularity) of a node i is no less than its *weighted degree*, which is the sum of the weights of its edges. This is because users could visit



Up next

Autoplay

ESPN First Take - Deion Sanders on Cam Newton and Super Bowl 50
ESPN First Take
20,561 views

Was Cam Newton's Post Game Reaction Acceptable? | Super Bowl 50 | NFL Now
NFL
81,742 views

Why Cam Newton's Panthers won't win Super Bowl 50 | Dave Dameshek Football
NFL
268,911 views

YouTube

People who viewed this item also viewed

Apple iPhone 6s Plus 16GB (Factory...
$719.00
Buy It Now
Free shipping

Apple iPhone 6s 64GB (Factory Unlocked)...
$695.00
Buy It Now
Free shipping

ebay

Customers Who Viewed This Item Also Viewed

GEEKPRO Pro4 HD 4K 2.7K WIFI Action Camera Sports Video Camcorder HDMI/AV Output Car Dash Cam Carrying Bag
$99.99 Prime

GEEKPRO 4.0 Plus 4K HD Action Camera 2.4G RF Remote Control 12MP Sports Video WIFI 170° Fisheye Cam Helmet,
$109.99 Prime

Amazon

Fig. 2: Item-to-item recommendation in YouTube, eBay, and Amazon.

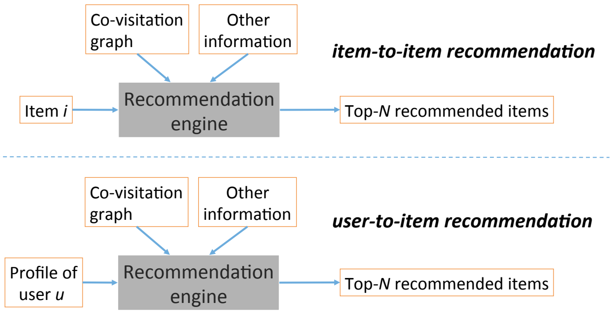


Fig. 3: Item-to-item recommendation vs. user-to-item recommendation.

the item i without visiting other items. We denote by w_{ij} and w_i the weights of edge (i, j) and node i, respectively.

A co-visitation recommender system mainly leverages such co-visitation graph to recommend items to a user. The key intuition is that items that were frequently co-visited in the past are likely to be co-visited in the future. Specifically, two popular recommendation tasks are *item-to-item recommendation* and *user-to-item recommendation*. In an item-to-item recommendation, when a user is visiting an item i, the system shows the top-N recommended items that are similar to i. In a user-to-item recommendation, the system recommends top-N items to a user via considering the user's visiting history. The visiting history could include all items the user has visited if the user logs in the web service, or it could include items the user has visited in a browser session if the user does not log in or the user is an unregistered visitor. Fig. 3 compares item-to-item recommendation and user-to-item recommendation.

Item-to-item recommendation: The service provider computes the *similarity* between each co-visited pair of items (each edge in the co-visitation graph corresponds to such a pair) via the co-visitation graph. Intuitively, items i and j are more similar if they are more frequently co-visited; given the number of co-visitations between i and j, they tend to be less similar if they are more popular. To capture such intuitions, the similarity s_{ij} between item i and item j is calculated as follows [2]:

$$s_{ij} = \frac{w_{ij}}{f(w_i, w_j)}, \qquad (1)$$

where $f(w_i, w_j)$ is a function of w_i and w_j. Different web services might use different such functions. For instance, YouTube [2] uses $f(w_i, w_j) = w_i \cdot w_j$. Amazon [3] uses *Cosine Similarity* between the view vectors (i.e., an entry of the vector is 1 if the corresponding user viewed the corresponding item, otherwise the entry is 0) of two items as their similarity. Since the entries of the view vectors have binary values, Cosine Similarity between two items is reduced to be Equation 1 with $f(w_i, w_j) = \sqrt{w_i \cdot w_j}$.

Given an item i that a user is visiting, the system first ranks the items using their similarities with i and then recommends the top-N items with the largest similarities to the user. We denote the top-N recommended items for the item i as a sorted list L_i. Note that this item-to-item recommendation method favours unpopular items. Specifically, an item with a small popularity is more likely to be recommended than an item with a large popularity if they have the same number of co-visitations with the item i. YouTube's co-visitation recommender system [2] avoids recommending highly unpopular items via excluding items whose popularities are smaller than a popularity threshold τ when preparing the top-N recommendation list.

We note that, although co-visitation graph is the core information that is leveraged by co-visitation recommender systems, other information (e.g., item diversity [2]) could also be considered to tune the recommended items. However, in this work, we focus on co-visitation graph and as we will demonstrate, manipulating co-visitation graph is sufficient to attack co-visitation recommendation systems at scale.

Fig. 2 shows item-to-item recommendations in YouTube[1], eBay, and Amazon. Although the details of the recommender systems used by eBay and Amazon are not publicly known, from their service names (e.g., "People who viewed this also viewed"), we suspect that they are very likely using co-visitation recommender systems. The parameters N are usually 20, 5, and 4 in the three web services, respectively.

User-to-item recommendation: The profile of a user consists of the items that the user has visited. User-to-item recommendation considers the user profile when making recommendations. The details of how to leverage user profile might be different for different web services. For instance, on

[1] According to Xing et al. [4], for logged-in users, at most two of the top-N recommended items on YouTube are chosen by user-to-item recommendation instead of item-to-item recommendation. For simplicity, we treat all of them as item-to-item recommendation.

YouTube [2], for each item i that was visited by the user, the system computes its top-N recommended items L_i via item-to-item recommendation. Then YouTube treats the union of these items L_i as a candidate set and recommends the user top-N items among the candidate set. To increase diversity of recommended items, YouTube enlarges the candidate set via repeatedly adding in the top-N recommended items of the current items in the candidate set [2]. Again, apart from the core co-visitation graph, other information could be considered to tune the top-N recommended items.

This work focuses on item-to-item recommendation, but our attacks are also applicable to user-to-item recommendation.

B. Attacks to Recommender Systems

1) Security Attacks: We first review existing security attacks to recommender systems.

Pollution attacks to user-to-item recommender systems: Xing et al. [4] recently proposed *pollution attacks* to the user-to-item recommendation and demonstrated that YouTube, Amazon, and Google search are vulnerable to the attacks. The goal of pollution attacks is to spoof the system to recommend a target item to a specific victim user. The key idea is to inject fake information into the victim user's profile (e.g., browsing history) via cross-site request forgery (CSRF) [5]. Pollution attacks suffer from two key limitations: 1) pollution attacks rely on CSRF, which makes it hard to perform the attacks at a large scale, and 2) pollution attacks are not applicable to item-to-item recommendation because the attacker cannot change the item that the user is visiting. Our attacks do not rely on CSRF, can be performed at scale, and are applicable to both item-to-item recommendation and user-to-item recommendation.

Profile injection attacks to recommender systems using user-item rating matrices: A few studies [15–17] have demonstrated that recommender systems (e.g., [3, 9, 10]) leveraging a *user-item rating matrix* are vulnerable to *profile injection attacks* (also called *shilling attacks*). Specifically, in a user-item rating matrix, each row corresponds to a registered user and each column corresponds to an item; an entry in the matrix is the rating score that the corresponding user gave to the corresponding item; a rating score represents the corresponding user's preference to the corresponding item; and most entries of the matrix are missing since a user only provides feedback about a small number of items. Given such a matrix, these recommender systems infer the values of the missing entries and then recommend users items with the largest inferred values.

Profile injection attacks aim to make a target item be recommended to more users. Specifically, in a profile injection attack, an attacker first registers a large number of fake accounts in the service. Then each fake account gives certain rating scores to a carefully chosen subset of items. These profile injection attacks are not applicable to co-visitation recommender systems that do not rely on the rating matrices.

2) Privacy Attacks: Calandrino et al. [18] proposed privacy attacks to infer a user's profile (e.g., the products that the user purchased on Amazon) via analyzing the publicly available recommendations that are made by the recommender system. Specifically, in their privacy attacks, an attacker first obtains a partial profile of a user. For instance, on Amazon, some users will review the products that they purchased. Through collecting these publicly available reviews, the attacker can obtain a subset of products that the target user purchased. Then the attacker monitors the temporal changes of item-to-item recommendation lists of these products. If an item appears in the recommendation lists of a large number of products that are in the target user's partial profile, the attacker infers that the user purchased the item. The authors demonstrated that this privacy attack is feasible on various popular web services including Amazon, LibraryThing, Hunch, and Last.fm.

III. Problem Definition

TABLE I: Categorization of an attacker's background knowledge

Scenario	Explanation
High knowledge	Co-visitation graph G, popularity threshold τ
Medium knowledge	Recommendation lists L, item popularities W
Low knowledge	Recommendation lists L

A. Attacker's Background Knowledge

We consider three scenarios where attackers can access different background knowledge of the recommender system.

High knowledge: In this scenarios, an attacker has access to the co-visitation graph G and the popularity threshold τ that is used to tune the top-N recommended items. This represents a strong attacker because the attacker knows the key components of the co-visitation recommender system. An attacker could obtain these information from an *insider* of the web service through underground market or the attacker itself could be an insider. This scenario represents an upper bound of the threats introduced by our attacks. We represent high knowledge as a pair (G, τ).

Medium knowledge: In this scenario, an attacker writes a web crawler to collect some items and their item-to-item top-N recommendation lists from the web service. We note that web services often make the item-to-item recommendation lists publicly available so unlogged-in visitors can also see them. Therefore, an attacker can collect these recommendation lists. Recall that we denote by L_i the item-to-item top-N recommendation list of an item i. We denote the recommendation lists collected by an attacker as a set $L = \{L_1, L_2, \cdots, L_m\}$, where m is the number of items whose recommendation lists are collected by the attacker.

Some web services show items' popularity to users/visitors, and thus an attacker has access to items' popularities. For instance, YouTube shows visitors the number of views (i.e., popularity) of a video. For convenience, we denote by a set $W = \{(i, w_i) | i \in I\}$ the popularities of all the items that the attacker encountered when collecting L, where the set I consists of all the items that the attacker encountered (i.e., the m items whose recommendation lists were collected by the attacker and items in these recommendation lists), and w_i is the popularity of item i. We represent medium knowledge as a pair (L, W).

Low knowledge: In this scenario, an attacker writes a crawler to collect $L = \{L_1, L_2, \cdots, L_m\}$, the item-to-item top-$N$ recommendation lists of m items. However, we assume the service does not provide item popularities, and thus the attacker does not have access to them. This scenario represents the *least* knowledge that a co-visitation recommendation system can leak to an attacker. For instance, eBay and Amazon belong to this category of services.

B. Definition of Attacks

We consider two families of attacks to co-visitation recommender systems, namely *promotion attacks* and *demotion attacks*. In a promotion attack, an attacker aims to make a *target* item (e.g., a video on YouTube) be recommended to as many users as possible, while the attacker's goal is to make a target item be recommended to as few users as possible in a demotion attack. Formally, we define them as follows:

Definition 1 (Promotion Attacks): Given a target item and an attacker with certain background knowledge about the co-visitation recommender system, a promotion attack is to abuse the recommender system so that it recommends the target item to as many users as possible.

Definition 2 (Demotion Attacks): Given a target item and an attacker with certain background knowledge about the co-visitation recommender system, a demotion attack is to abuse the recommender system so that it recommends the target item to as few users as possible.

Limited resources: We consider an attacker injects fake co-visitations between the target item and other selected items to perform promotion and demotion attacks. We assume the number of fake co-visitations that an attacker can inject is limited. We adopt this threat model for two reasons. First, an attacker could have limited resources, e.g., IP addresses, computing resources. If an attacker injects a very large amount of fake co-visitations from a single IP address, the service provider can easily detect the attack and block the attacker [6]. Therefore, the attacker can inject a bounded number of fake co-visitations without being detected, though this bounded number could still be large. Second, suppose an attacker deploys our attacks *as a service*. An organization wants to use this service to promote its product, but this organization has a limited budget to pay for the service. In this scenario, the limited budget can be translated to a bounded number of fake co-visitations. An attacker's goal is to maximize the threats of the promotion or demotion attacks, when the number of fake co-visitations that can be injected is fixed.

C. Evaluation Metrics

1) Number of Items: One natural metric to measure promotion attacks and demotion attacks is to use the increased (for promotion attacks) or decreased (for demotion attacks) number of items whose item-to-item recommendation lists include the target item. For instance, suppose the target item originally appears in the recommendation lists of 10 items, and the number increases to be 30 after the promotion attack, then the promotion attack's threat is 20 items.

2) User Impressions: The metric *number of items* does not consider item popularities. Appearing in the recommendation list of a more popular item means that the target item is going to be recommended to more users/visitors. Therefore, we propose new metrics, which incorporate item popularities, to evaluate the threats of promotion and demotion attacks.

Top-k user impression: When a target item i_t appears in an item i's item-to-item recommendation list, the target item is exposed to any user who visits item i. In other words, the target item obtains one *user impression* for any user visit to i. If the item i has more visits in the future, the target item will obtain more user impressions. Having more user impressions could lead to more visits of the target item, which subsequently could lead to more purchases (if the item is a product or an ads about a product).

We note the likelihood of turning a user impression to a visit or even purchase could depend on the specific ranking position of the target item in the recommendation list. For instance, the highest ranked item might have a higher visit rate than the lowest ranked item. To incorporate the impact of item ranks in the recommendation list, we define a user impression as a *top-k user impression* if the target item is ranked top-k on the recommendation list, where $k \leq N$.

Probability of top-k user impression: Measuring top-k user impressions for a target item requires knowledge about the number of visits to certain items *in the future*, which might not be available at the time of attacks. Therefore, we propose *probability of top-k user impression (UI)*, which is the probability the target item obtains a top-k user impression for a random user visit. Suppose a random user visits an item in a web service, we denote by p_i the probability that this random user visits item i. Let I_{i_t} be the set of items whose top-k recommended items include the target item i_t. Then the probability of top-k user impression of i_t is $UI = \sum_{i \in I_{i_t}} p_i$.

Although the exact p_i is not available at the time of attacks, we can estimate it using the popularity of the item i in the past. Several studies [19–22] found that many natural phenomena follow a *power law*. In our case, power law phenomena implies that an item that was popular in the past is likely to be popular in the future. More specifically, the probability p_i is proportional to the current popularity of the item i in power law phenomena. Formally, p_i is estimated as follows [23]:

$$p_i = \frac{w_i}{w_1 + w_2 + \cdots + w_n}, \tag{2}$$

where n is the total number of items and w_i is the popularity of i in the past. Therefore, we have:

$$\text{UI} = \frac{\sum_{i \in I_{i_t}} w_i}{w_1 + w_2 + \cdots + w_n}. \tag{3}$$

Intuitively, UI $= x\%$ indicates that $x\%$ of website visitors will see the item in the recommendation lists of some other items.

Measuring threat of promotion attacks: Suppose a target item i_t is originally among the top-k recommendation list in a set of items which we denote as I_{i_t}, where $k \leq N$. After a promotion attack, this set of items is enlarged to be J_{i_t}. We define the threat of this promotion attack as *increased* *probability of top-k user impression (IUI)*, which we formally represent as follows:

$$\mathrm{IUI} = \textstyle\sum_{i \in J_{i_t} - I_{i_t}} p_i \tag{4}$$

Measuring threat of demotion attacks: Suppose the set of items whose top-k recommended items include the target item is reduced to be J_{i_t} after a demotion attack. We define the threat of this demotion attack as *decreased probability of top-k user impression (DUI)*, which we define as follows:

$$\mathrm{DUI} = \textstyle\sum_{i \in I_{i_t} - J_{i_t}} p_i \tag{5}$$

An attacker's goal is to maximize the IUI or DUI for a target item with given background knowledge and resource. We note that when all item popularities are known, we will always use IUI or DUI to measure our attacks. For instance, a service provider, who has access to popularities of all its items, can calculate IUI and DUI to measure the security of its recommender system against our attacks.

IV. Co-visitation Injection Attacks

We discuss fake co-visitation injection attack strategies for attackers with different background knowledge.

A. Promotion Attacks

In promotion attacks, an attacker selects a set of items whose recommendation lists haven't included the target item yet. We call these items *anchor items*. Then the attacker injects fake co-visitations between the target item and each anchor item to make the target item appear in its top-k recommendation list. Specifically, to inject fake co-visitations between items, the attacker can write a script and use it to automatically and repeatedly visit them simultaneously or consecutively (e.g., view the two items in the same browser session). We note that, in practice, attacker's resources are bounded, and thus the number of fake co-visitations that an attacker can inject is bounded. Therefore, in promotion attacks, the attacker's goal is to maximize the increased probability of top-k user impression (IUI) for a given number of fake co-visitations that can be injected. Two key challenges in promotion attacks are: 1) how to select the anchor items, and 2) how many fake co-visitations should be injected for each anchor item.

We first show how to solve the challenges with high knowledge; then we transform a medium-knowledge attack to a high knowledge attack by estimating the missing parameters; and finally we transform a low-knowledge attack to a medium-knowledge attack by estimating the item popularities.

1) High Knowledge: With this background knowledge, an attacker can access the co-visitation graph G and the popularity threshold τ used to filter out unpopular items when producing the recommendation lists. Suppose the attacker can inject totally m fake co-visitations. The probability p_i that a random user visits the item i is determined by Equation 2.

Attacking one anchor item: Suppose j is a selected anchor item and L_j is the top-k recommendation list for j. Furthermore, we denote by k_j the ranked k-th item in L_j. We note that if we add visitations to j while the number of co-visitations between j and the items in L_j keeps unchanged, then the recommendation list L_j and the relative rankings of the items in L_j keep unchanged. In order to make the target item i_t appear in the top-k recommendation list of j, the attacker needs to inject m_{jk} fake co-visitations between the items i_t and j, where m_{jk} satisfies two conditions:

$$s'_{ji_t} > s'_{jk_j} \tag{6}$$

$$w_{i_t} + m_{jk} \geq \tau, \tag{7}$$

where s'_{ji_t} is the similarity between j and the target item i_t, and s'_{jk_j} is the similarity between j and the k-th ranked item k_j after the attack. Formally, we have $s'_{ji_t} = (w_{ji_t} + m_{jk}) / f(w_j + m_{jk}, w_{i_t} + m_{jk})$ and $s'_{jk_j} = w_{jk_j} / f(w_j + m_{jk}, w_{k_j})$, where w_{ji_t} is the number of co-visitations between j and i_t, w_j is j's popularity, w_{i_t} is i_t's popularity before the attack, and the function f is the normalization factor that we discussed in Section II-A. In our formulation, we assume that the number of co-visitations between the item j and each item in its recommendation list does not increase significantly during the attacking process.

Intuitively, Equation 6 guarantees the similarity between the item j and the target item is larger than that between j and the k-th ranked item in j's recommendation list after the attack, while Equation 7 guarantees the target item's popularity passes the threshold testing. The two conditions can be transformed to linear constraints on m_{jk} for various normalization functions f, e.g., the widely used product normalization function (i.e., $f(w_i, w_j) = w_i \cdot w_j$) and sqrt-product normalization function (i.e., $f(w_i, w_j) = \sqrt{w_i \cdot w_j}$). Details of such transformations are given in Appendix A. These two linear constraints enable us to compute the minimum value of m_{jk} that is required to attack the anchor item j.

Attacking multiple anchor items: The attacker selects a set of anchor items that can be successfully attacked using bounded resources to maximize the threat. For convenience, we use a binary variable a_j to represent whether the item j is selected as an anchor item or not, i.e., $a_j = 1$ means j is an anchor item. With these variables, we formulate the promotion attack as an optimization problem:

$$\text{maximize } IUI = \textstyle\sum_{j \in V_k} a_j \cdot p_j \tag{8}$$

$$\text{subject to} \textstyle\sum_{j \in V_k} a_j \cdot m_{jk} \leq m \tag{9}$$

$$s'_{ji_t} > s'_{jk_j}, \qquad \forall j \in V_k \tag{10}$$

$$w_{i_t} + m_{jk} \geq \tau, \qquad \forall j \in V_k \tag{11}$$

$$a_j \in \{0, 1\}, \qquad \forall j \in V_k \tag{12}$$

where V_k is the set of items whose top-k recommendation lists do not include the target item.

Intuitively, in our formulation, Equation 8 indicates that the attacker aims to maximize the IUI; Equation 9 encodes the resource constraint; Equation 10 and 11 are the constraints that the number of fake co-visitations m_{jk} should satisfy to attack anchor item j. We transform the optimization problem to a *linear programming* problem. Specifically, we first derive the minimum value of m_{jk} using Equation 10 and 11. Then, we replace the variable m_{jk} with its minimum value in Equation 9. The resulting problem is a standard linear programming problem, which has been studied extensively and can be solved efficiently by various algorithms (e.g., Ellipsoid method [24]).

- **Step 1:** The attacker solves the optimization problem in Equation 8, e.g., using Ellipsoid method [24]. After this step, the attacker obtains the set of anchor items (i.e., an item with $a_j = 1$ is an anchor item) and the number of fake co-visitations that the attacker needs to inject to each anchor item.
- **Step 2:** The attacker uses a script to automatically inject m_{jk} fake co-visitations between the target item i_t and each anchor item j.

2) Medium Knowledge: With medium knowledge, the attacker can access the popularity of each item and see its recommendation list, but the attacker cannot obtain the number of co-visitations between items (i.e., edge weights of the co-visitation graph) nor the popularity threshold τ. Once the attacker has access to the popularity threshold τ and the similarity s_{jk_j} for each item j in the set of items that the attacker has collected, the attacker can use our attacks that we develop for high knowledge. Therefore, our key idea is to transform attacks with medium knowledge to attacks with high knowledge via estimating the missing parameters.

Specifically, we estimate *upper bounds* of the missing parameters, which gives a lower bound of the threat an attacker can achieve with a given number of fake co-visitations that can be injected. First, we estimate the popularity threshold τ as the popularity of the least popular item on the recommendation lists. Second, we have $s_{jk_j} \leq s_{j(k-1)_j} \leq s_{j(k-2)_j} \leq \cdots \leq s_{j1_j}$, i.e., the similarity between j and the kth ranked item k_j in j's recommendation list is smaller than those between j and the $(k-1)$th, $(k-2)$th, $\cdots$, 1st ranked items of j's recommendation list. Moreover, $s_{jx} = \frac{w_{jx}}{f(w_j,w_x)} \leq \frac{\max\{w_j,w_x\}}{f(w_j,w_x)}$. Therefore, we can estimate an upper bound of s_{jk_j}. In particular, we first compute the upper bound of s_{jx} for all items x that are ranked higher than k_j, and then take the minimum of these upper bounds as an upper bound for s_{jk_j}. With these estimated parameters, the attacker follows the steps in attacks with high knowledge to perform attacks. However, since the number of injected co-visitations m_{jk} are estimated, they might be larger than what are really needed. Therefore, the attacker gradually injects co-visitations and monitors the recommendation lists of the anchor items; if the target item appears in the top-k recommendation list of an anchor item, the attacker stops injecting co-visitations to this anchor item.

We note that we assume the recommendation list is a sorted list when estimating the parameters. However, if the list is not sorted, we can still estimate upper bounds of these parameters. For details, please refer to Appendix B. Moreover, techniques like XRay [25] could also be used to estimate the missing parameters.

3) Low Knowledge: With low knowledge, the attacker only obtains the recommendation lists of a set of items. We propose to estimate the item popularities and transform the attack to an attack with medium knowledge. Specifically, previous work [26] has shown that item popularity can be represented as a function of a set of features (e.g., number of user reviews, number of purchases) about the item. Formally, we have

$$w_j = g(f_1, f_2, \cdots, f_F), \tag{13}$$

where F is the number of features. For instance, in our experiments we assume g is a linear function, which means that item popularity is a linear regression of the feature values. Specifically, $g(f_1, f_2, \cdots, f_F) = a_0 + \sum_{t=1}^{F} a_t f_t$. From a machine learning perspective, the parameters (e.g., a_0, a_1, $\cdots$, a_F for linear regression) in the function g can be learnt with a training dataset, which consists of some items with both popularities and feature values. However, with low knowledge, the attacker does not know the item popularities.

To address this challenge, we propose the attacker gradually injects fake co-visitations and monitors the changes of the recommendation lists, during which the attacker refines the parameters of g. For a web service, the attacker only needs to estimate g once and use it for future attacks.

Learning parameters of g: The attacker first collects a set of publicly available features of some items. Then the attacker starts with random initial parameter values for the function g and uses g to compute the estimated item popularities. With the estimated popularities, the attacker performs attacks with medium knowledge. However, the key difference is that the attacker injects fake co-visitations to anchor items one by one and monitors the changes of the recommendation lists. For an anchor item j, if the target item i_t appears in its top-k recommendation list before injecting m_{jk} fake co-visitations, which indicates j's popularity might be overestimated, the attacker decreases the parameters of g by half; if the target item does not appear in the top-k recommendation list after m_{jk} fake co-visitations, which means the popularity of j might be underestimated, the attacker doubles the parameters of g. Via repeatedly adjusting the parameters of g, the attacker is able to learn a predictor to estimate item popularity. With the function g, the attacker transforms an attack with low knowledge to an attack with medium knowledge, and then follows the procedure in Section IV-A2 to perform attacks.

B. Demotion Attacks

Demotion attack aims at decreasing UI of a target item i_t. We achieve this goal via removing i_t from the top-k recommendation lists of the selected anchor items. Let L_j be the recommendation list of an anchor item j that contains i_t as the uth ranked item ($u \leq k$). The attacker cannot remove existing co-visitations, but it can improve the ranking of the $(u+1)$th, $(u+2)$th, $\cdots$, $(k+1)$th ranked items until i_t is not on the top-k recommendation list. This can be viewed as a promotion attack which treats these items as target items. Therefore, we apply the promotion attacks that we discuss in the previous section to perform demotion attacks. The only difference is that the selected anchor items should contain the target item i_t. With this difference, we formulate the demotion attack (with high knowledge) as the following optimization problem:

$$\text{maximize } DUI = \textstyle\sum_{j \in V_k} a_j \cdot p_j \tag{14}$$

$$\text{subject to} \textstyle\sum_{j \in V_k} a_j \cdot \sum_{x=u+1}^{k+1} m_{jx} \leq m \tag{15}$$

$$\min_{x=u+1}^{k+1} \{s'_{jx}\} > s'_{ji_t}, \qquad \forall j \in V_k \tag{16}$$

$$\min_{x=u+1}^{k+1} \{w_x + m_{jx}\} \geq \tau, \qquad \forall j \in V_k \tag{17}$$

$$a_j \in \{0, 1\}, \qquad \forall j \in V_k \tag{18}$$

where V_k is the set of items that contain i_t in their top-k recommendation lists, Equation 16 guarantees the similarity score of any of the $(u+1)$th, $(u+2)$th, $\cdots$, $(k+1)$th ranked item is greater than that of the target item i_t, and Equation 17 guarantees that these promoted items can appear in the recommendation list. Then the attacker applies our promotion attacks with different background knowledge to perform demotion attacks with the corresponding background knowledge. The only difference is that the optimization problem in Equation 8 is replaced with Equation 14.

V. Experiments on Synthetic Data

We evaluate fake co-visitation injection attacks using synthesized datasets in this section.

A. *Experiment Design*

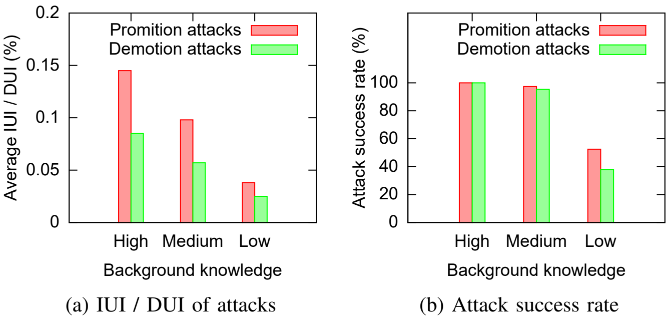
Design goals: We aim to answer the following questions:

- How does different background knowledge (i.e., high, medium, and low knowledge) impact the threats of fake co-visitation injection attacks?
- How does the structure of the co-visitation graph impact the threats of fake co-visitation injection attacks?
- How do the attacker's resources (i.e., the number of fake co-visitations that the attacker can inject) and the attacking parameter k impact the threats of fake co-visitation injection attacks?

Synthesizing co-visitation graphs: The key component of a co-visitation recommender system is the co-visitation graph. We generate a co-visitation graph with 100,000 nodes. Different structures of this graph could have different impact on our attacks. We leverage three popular graph generation models (i.e., regular graph, *Erdos-Renyi (ER) random graph* [27], and *power-law random graph* [23]), which are developed by the network science community, to generate the structure of the co-visitation graph. Specifically, in a regular graph, each node has the same degree; to generate a ER graph or a power law graph, we gradually add nodes to the graph and each new node is linked to d existing nodes. These d nodes are picked uniformly at random in ER graph model while they are picked with probabilities that are proportional to their current degrees in power-law graph model. In our experiments, we assume $d = 10$. Recall that an edge in the co-visitation graph means that the two corresponding items were co-visited at least once.

The co-visitation graph also has node weights (i.e., item popularities) and edge weights (i.e., number of co-visitations between two items). Since item popularity often follows a power-law distribution in real-world recommender systems, we generate item popularities from a power-law distribution with exponent 2 and they range from 100 to 20,000. Since an item that are co-visited with more items might be more popular, we assign a larger generated popularity to an item with larger node degree in the co-visitation graph. Then we randomly assign integer edge weights such that the weighted node degree is no larger than the node weight for each node in the graph.

Unless otherwise mentioned, we use power-law graph model to generate the co-visitation graph, the recommender system produces a top-10 item-to-item recommendation list with a popularity threshold $\tau = 500$. The attacking parameter k is set to be 5 (e.g., the attacker wants to promote a target item to be among top-5 in the recommendation list of the anchor items), and the attacker has resources to inject 5000 fake co-visitations. We assume product normalization function to compute similarity. When we study the impact of one factor (e.g., attacker's background knowledge), we will vary this factor while fixing other parameters.



(a) IUI / DUI of attacks

(b) Attack success rate

Fig. 4: Impact of the attacker's background knowledge.

Simulating background knowledge: In high knowledge, the attacker knows the co-visitation graph and the popularity threshold. In medium knowledge, the attacker knows the item popularities and the top-10 recommendation list for each item. In low knowledge, the attacker knows the top-10 recommendation list for each item; we assume the number of reviews about each item is available to the attacker, and the attacker uses it as a feature to estimate item popularities. Specifically, we randomly generate the number of reviews for each item such that the correlation coefficient between item popularities and reviews equals 0.8.

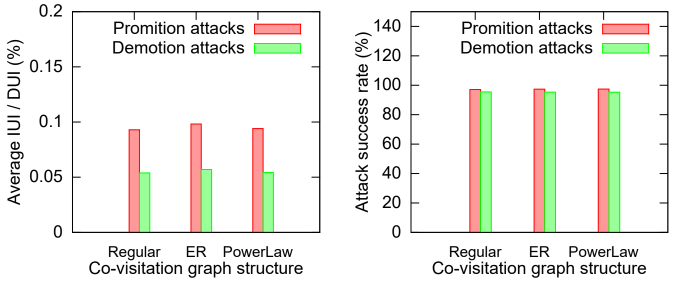
B. *Results*

We assume 10 new target items which are not in the co-visitation graph for promotion attacks, while we pick 10 items uniformly at random from the co-visitation graph for demotion attacks. Our reported results are averaged among the corresponding 10 target items.

Impact of the attacker's background knowledge: Fig. 4 shows the results for attackers with different background knowledge. The *attack success rate* is defined as the number of successfully attacked anchor items over the number of anchor items that are selected by our attacks. An anchor item is successfully attacked if the target item appears in its top-k recommendation list. Formally, we define *Attack Success Rate (AST)* as:

$$AST = \frac{\#successfully\ attacked\ anchor\ items}{\#selected\ anchor\ items} \tag{19}$$

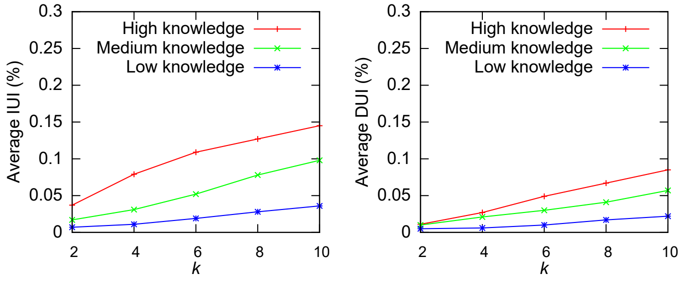
First, as expected, attacks with high knowledge achieve larger threats (i.e. IUI for promotion attacks and DUI for demotion attacks) than attacks with medium knowledge, which in turn achieve larger threats than attacks with low knowledge. However, we also find that, even if only low knowledge is available, the attacks can still achieve threats that are almost 1/3 of those achieved with high knowledge.



(a) IUI / DUI of attacks

(b) Attack success rate

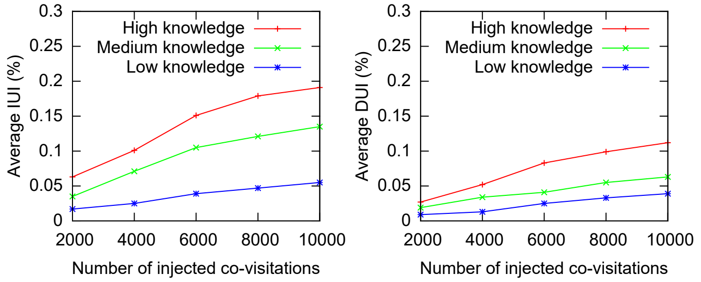
Fig. 5: Impact of the co-visitation graph's structure.



(a) Promotion attack

(b) Demotion attack

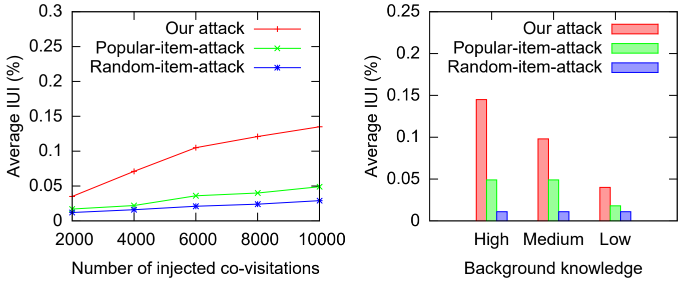
Fig. 7: Impact of k.



(a) Promotion attack

(b) Demotion attack

Fig. 6: Impact of the number of fake co-visitations.



(a)

(b)

Fig. 8: Comparing with baseline attacks.

Second, the attack success rate is 100% with high knowledge, because the attacker knows all necessary information, and our attack algorithm can accurately compute the number of fake co-visitations need to be injected. With medium knowledge, the attack success rate slightly decreases because the key parameters are estimated. With low knowledge, the attack success rate further decreases as the item popularities are estimated by a linear regression.

Impact of co-visitation graph's structure: We compared our attacks for three types of co-visitation graphs, i.e., *regular graph*, *Erdos-Renyi (ER) random graph*, and *power-law random graph*. The result is showed in Fig. 5. We find graph structures have relatively small impact on our attacks, though the attacker can achieve slightly better results when the co-visitation graph is close to a ER graph.

Impact of the attacker's resources and k: Fig. 6 shows the impact of the number of fake co-visitations that can be injected on the threats of our attacks. Not surprisingly, our attacks have larger threats when the attacker has resources to inject more fake co-visitations. Fig. 7 shows the attacking results as a function of k. A user impression is counted when the target item appears in top-k recommendation list of an anchor item. Our results verify that when k is smaller, both IUI and DUI become smaller. This is because, when the number of fake co-visitations is fixed, the attacker can attack less anchor items and each anchor item needs more fake co-visitations.

Comparing with baseline attacks: An attacker may also perform simple attacks on co-visitation recommender systems, e.g., inject co-visitations with randomly selected anchor items. We compare our promotion attacks with two baseline attacks.

- **Popular-item-attack**. This attack injects co-visitations between a target item and the most popular item until the target item appears in its top-k recommendation list, and then attacks the next most popular item, until there are no more fake co-visitations to inject.
- **Random-item-attack**. This attack randomly selects an anchor item, injects co-visitations until the target item appears in its top-k recommendation list. This process is repeated until no more fake co-visitations to inject.

Fig. 8a shows the average IUI of our promotion attack and the baseline attacks under medium knowledge. Our attack achieves the highest threat when the same amount of fake co-visitations are injected. Fig. 8b compares the performance of our proposed attack with the baseline attacks where the attackers have different background knowledge. Our attack substantially outperforms the baseline attacks in all situations.

Summary: We have the following observations:

- High-knowledge attacks are more effective than medium-knowledge attacks, which in turn are more effective than low-knowledge attacks.
- Graph structures have small impact on our attacks.
- Our attacks have larger success rates when the attacker has resources to inject more fake co-visitations or a larger k is considered as a threat.
- Our attacks achieve significantly larger success rates than the baseline attacks that simply select the most popular items or random items as anchors.

VI. Attacking Real-world Systems

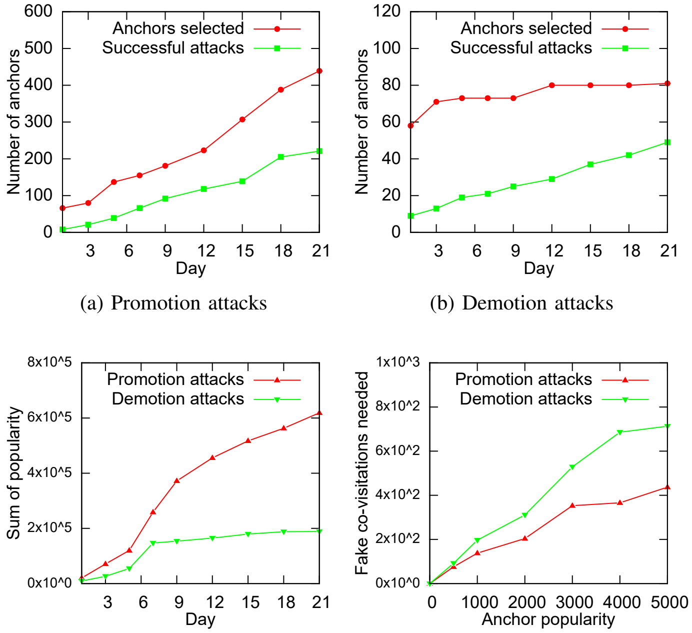
A. *Experiment Overview*

We evaluate our fake co-visitation injection attacks on real-world recommender systems of several popular websites, including YouTube (the feature "Up Next video"), eBay (the feature "People who viewed this item also viewed"), Amazon (the feature "People who viewed this also viewed"), Yelp (the feature "People also viewed"), and LinkedIn (the feature "People Also Viewed"). Among these websites, YouTube is categorized as medium knowledge because its item popularity is publicly visible. All the other websites provide only recommendation lists, and thus they are categorized as low knowledge. We consider view-based co-visitation, because injecting purchase-based co-visitation (e.g., features like "People also purchased") is too expensive as the attacker needs to purchase the items.

We implemented an automatic co-visitation injection system using C#. We integrate the open source web crawler GRUB into our system to collect item information from the websites. Our experiment platform is a windows server with Intel Xeon 64-bit 8-core CPU running on 2.93GHz and 32GB RAM. Our system automatically injects co-visitations by repeatedly opening item web pages consecutively within the same browser session (and using the same user account when login is required). Since none of the attacked web services provides high-knowledge, we first used approximately 1 week to estimate missing parameters on all these recommender systems with our proposed methods. We then continue to attack these web services for 3 weeks, and record the accumulated attacking results. We divide the 3 weeks period into multiple 12 hour attacking windows. At the beginning of each 12 hour window, our system evaluates the attacking results, updates its budget, and then re-selects anchor items if necessary. This includes selecting new anchors if the attack on some anchors failed after 2 or more attacking windows.

We also attempted to avoid the injected co-visitations being filtered out by the web services. To this end, we inject co-visitations with random time intervals to avoid any fixed patterns. Additionally, consecutively injected co-visitations are generated to include different items. We also disguised the IP address of our experiment platform. Specifically, we purchased a VPN service which provides more than 100 VPN servers with IP addresses all over the world. Our system frequently switches between these servers to visit the websites, in order to avoid an IP address being blocked due to abnormal activities. The cost of such VPN service is around 10$ per month at the time we conducted the experiments.

For each web service, we randomly select a set of 40 target items, 20 for promotion attacks and the rest for demotion attacks. Anchor items are selected among the set of items that we collected from each web service according to our attacks. Specifically, for each target item, we first generate a set of candidate anchor items. These candidates are selected by searching items containing similar keywords and/or falls in the same category as the target item. This is to make the attacks more realistic. Because it might be suspicious if users co-visit two items that are completely unrelated. We average our results over the target items for promotion and demotion attacks, respectively.



(a) Promotion attacks

(b) Demotion attacks

(c) Popularity of successfully attacked anchors

(d) Cost vs. anchor popularity

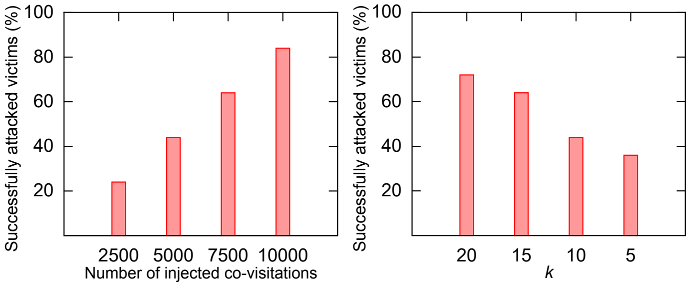
Fig. 9: Attacking YouTube.

Ethical considerations: To the best of our knowledge, there is no known methodology that could obtain our results without any effect on the real-world recommender systems, though we want to stress that our experiments have very low risks to the service providers and users. For the service providers, our attacks will affect their co-visitation graphs via changing the weights of a very small number of edges. For a user, the risk is that some items are recommended to him/her due to our attacks.

We take several actions to mitigate such risks. First, we limit our experiments to small scale attacks that are enough to demonstrate effectiveness and feasibility of our attacks. Second, we reported our attacks to the service providers. Our experiments strictly followed the responsible disclosure policy for vulnerability testing of the web services. We confirmed that our research is IRB exempt.

B. *Attacking Results*

YouTube: Before attacks, we randomly crawled information of approximately 100,000 videos using the same video selection method as introduced in [26]. We collected title, view count (popularity), and recommendation list of each video. All target items are selected from these videos. Anchor items are selected according to our attacks, but we avoid extremely popular items, e.g., items being viewed for over 1 million times. This is because attacking such items requires more resources (i.e., time or computing power) than our experiment platform has, but our attacks are also applicable to such popular items. Specifically, we select anchor items with the number of total views between 500 to 10,000. We set k to be 9 because top-9 videos in a recommendation list are shown when a video is being watched.



(a)

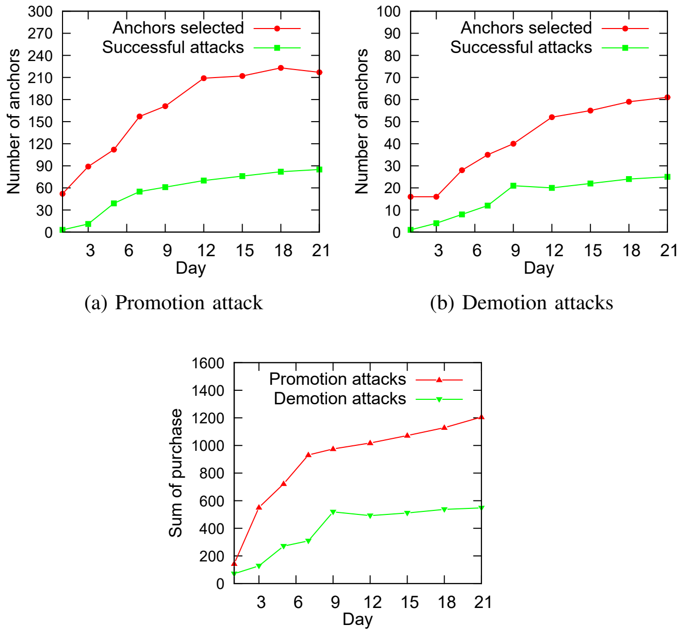
(b)

Fig. 10: Attacking user-to-item recommendation in YouTube.

We limit the number of injected co-visitations to approximately 2400 per 12 hour window. To make our injected co-visitations be more likely to be counted by YouTube, our system kept playing the opened video streams for about 3 minutes. Note that computing the exact IUI or DUI requires knowledge of precise popularity of *every* video on YouTube, which we do not have access to. Thus we report two related measurements. The first one is the number of selected anchor items and the number of anchor items that are successfully attacked. For promotion attacks, an anchor item is successfully attacked if the target item appears in its recommendation list after the attack; while for demotion attacks, it means the target item disappears from its recommendation list. The second one is sum of popularity of successfully attacked anchor items. Fig. 9 shows our results. The results are averaged over the 20 target items for promotion attacks and demotion attacks, respectively. We have observed a delay of 24-48 hours between attacks and affected recommendation lists update. Such a delay is also widely observed on other attacked web services.

We observe that more than a half of selected anchor items are successfully attacked for both promotion attacks and demotion attacks. For promotion attacks, the sum of popularity of the successfully attacked anchor videos reaches more than 6×10^5. A target item will be shown to any user who visits a successfully attacked anchor video in the future. Fig. 9d shows that more co-visitations are needed to attack anchor videos with larger popularities. This is because videos that appeared on the recommendation list of a popular anchor video are also likely to be popular, and thus have higher number of co-visitations with the anchor video. A larger number of co-visitations is needed for the target video to compete with popular videos on the recommendation list. Nevertheless, attacking popular anchor items is not our attack's goal. Our attacks aim to optimize user impressions of target items, and popular items may or may not be selected as anchors. Moreover, demotion attacks require larger number of fake co-visitations than promotion attacks, since demotion attacks need to promote multiple videos in order to exclude the target item from the recommendation list.

In addition to experiments on the item-to-item recommendation, we also evaluate our attacks for the **user-to-item recommendation** on YouTube. According to [2], the user-to-item recommendation list generated for a registered user is based on the co-visitation information of videos the user viewed before. Therefore, user-to-item recommendation is also vulnerable to our attacks. Although the exact list of videos a user viewed are not publicly available, each user does have an open list of videos he/she “liked” and “subscribed”, which can be used as anchor items in our attacks.



(a) Promotion attack

(b) Demotion attacks

(c) Purchases of successfully attacked anchors

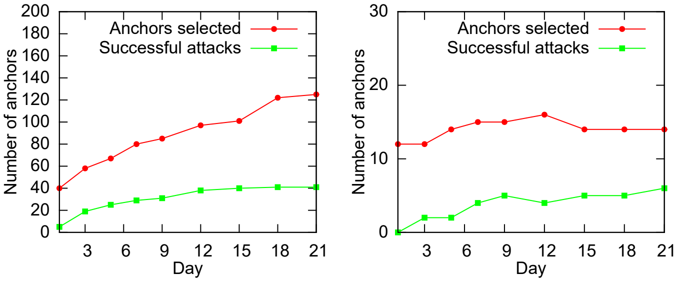
Fig. 11: Attacking eBay.

To evaluate our attack without affecting real users, we registered 25 fake accounts. We had each of them watch up to 100 randomly selected videos, as well as like and/or subscribe an arbitrary number of the watched videos. YouTube has then generated a list of recommendations for each fake account. We use these fake accounts as victims to perform attacks. The attack goal is to make a randomly selected target video to appear on top-k user-to-item recommendation lists of the victims, i.e., promoting a specific video to a targeted group of users. The attacker only requires the list of videos the victims liked and subscribed, which we demonstrate to be sufficient to launch effective attacks in our experiments.

Fig. 10a shows the fraction of successfully attacked victims whose user-to-item recommendation lists contain the target video as we inject more fake co-visitations. The k is set to be 10 since it is the size of the first page of the recommendation list. We repeated the attack for 10 times, each time using a different target video, and we report the average results. As expected, the fraction of successfully attacked victims grows as the number of fake co-visitations increases.

We also studied the impact of k value on the attack success. In this experiment, the total number of fake co-visitations is fixed to be 5000. The k value is adjusted from 20 to 5 and the result is showed in Fig. 10b. As expected, the fraction of successfully attacked victims drops when we decrease k.

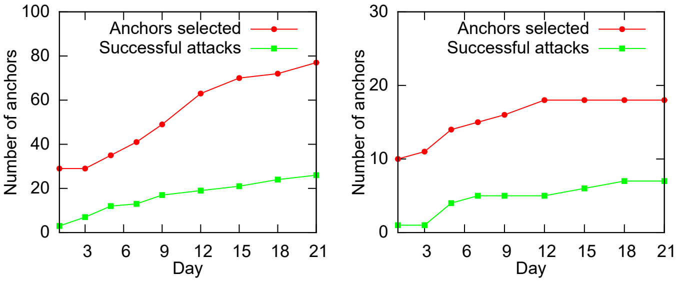
eBay: We collected information of over 7,000 items on eBay with a crawler and use them as candidates for target and anchor items. These items are randomly crawled from



(a) Promotion attacks

(b) Demotion attacks

Fig. 12: Attacking Amazon.



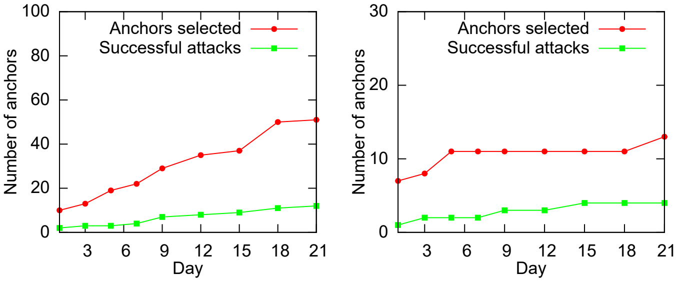
(a) Promotion attacks

(b) Demotion attacks

Fig. 13: Attacking Yelp.

all categories on eBay. When an item is sold out on eBay, it will be removed from recommendation lists. Therefore, it is meaningless to use sold out items as target or anchor. Additionally, the feature "People who viewed this also viewed" feature is not enabled for all items at all time. Taking these features into consideration, we limit our target and anchor item selection process to items with stable supply (i.e., they have more than one hundred in stock and/or being listed for more than 30 days) and with the recommendation feature enabled at the time of attack. The number of views of each item is not visible so it is estimated based on item features including the number of purchases and the number of reviews. The k value is set to be 5, which is the size of recommendation lists on eBay. Our system injects 2400 co-visitations per 12 hour. In our preliminary experiments, we observed that eBay strongly favors co-visitations generated by registered users over anonymous visitors. Thus, we manually registered 10 fake accounts to perform the attacks.

We report the number of selected anchor items and successfully attacked anchor items in Fig. 11. Since item popularity is not known, we also report the sum of purchases of successfully attacked anchor items as a related measurement for popularity (Fig. 11c). Overall, compared with YouTube, attacks on eBay demonstrate a smaller number of selected anchor items and a smaller fraction of successfully attacked anchor items. The reason is that eBay is a low-knowledge attack scenario, and some fake co-visitations are wasted on failed attacking attempts. In contrast, YouTube is a medium-knowledge attack scenario since it shows item popularity. This result indicates that limiting an attacker's background knowledge about item popularity is an useful way to mitigate our attacks.



(a) Promotion attacks

(b) Demotion attacks

Fig. 14: Attacking LinkedIn.

Amazon: We found that Amazon shows a "People who viewed this also viewed" recommendation list, but only for items that are purchased by less than about 5 times within a certain time period. Once the item sales increase, this recommendation list is replaced by a purchase-based recommendation list ("Items frequently bought together"). In our experiments, we found that some successfully attacked anchor items no longer have the view-based recommendation lists. This makes it hard to track attacking results and to perform adaptive attacks for such items. Therefore, we remove these items from our experiment statistics, and report results of items with stable recommendation lists. The result is shown in Fig. 12. k is set to be 4, and the number of injected fake co-visitations is 3000 per 12 hour.

Yelp: Items on Yelp are location-sensitive, e.g., a restaurant will appear in the recommendation list of another restaurant only if they locate in the same city. Therefore, we require the selection of target and anchor items to be in the same city. We crawled information of over 4000 restaurants in New York city, San Francisco, Los Angeles, and Chicago as item set. Yelp didn't explicitly state if the recommender system considers co-visitation from registered users only, but for the best result, we used multiple fake accounts to launch the attack. We estimate item popularity using the number of reviews as well as their rank in local restaurant list, and the number of fake co-visitations injected is 3000 per 12 hour. The size of recommendation list is only 3, thus we also set k to be 3. The attacking result is illustrated in Fig. 13. On average our attack can successfully make a target item appear in the recommendation list of more than 20 restaurants. Note that the item set of Yelp is significantly smaller comparing to YouTube or eBay, making it harder to find suitable anchor items. We also observed that some items in the recommendation list appear to be immune from our co-visitation injection attacks. We suspect that such items might be from a sponsor, who pays Yelp to always show its item in recommendation list. It is also possible that the recommendation list is not generated completely based on the number of co-visitations, but also involves other factors such as user reviews.

LinkedIn: Finally, we test our attacking system against the "People Also Viewed" list on LinkedIn. LinkedIn requires very complete personal information and valid email address. Thus, we used 5 actual user accounts for our attack. Using these 5 users as seeds, we crawled publicly available information of about 200 people, and used these people and their direct connections as item set which include about 1200 people. The result (Fig. 14) shows that our attacks are effective and increase the number of a target item's appearance in recommendation list by approximately 15 on average.

VII. Countermeasures

A. Limiting Background Knowledge

Our experiments in Section V show that limiting the attacker's background knowledge can substantially reduce the threats of our attacks. For instance, when the service provider shows the recommendation lists of the items but does not show the item popularities, i.e., the attacker has low knowledge about the recommender system, threats of our attacks are reduced, though they are still feasible and effective. This implies that service provider can hide item popularities in order to mitigate our attacks. However, in certain web services (e.g., YouTube), item popularities are useful information for users; hiding item popularities may affect user experience.

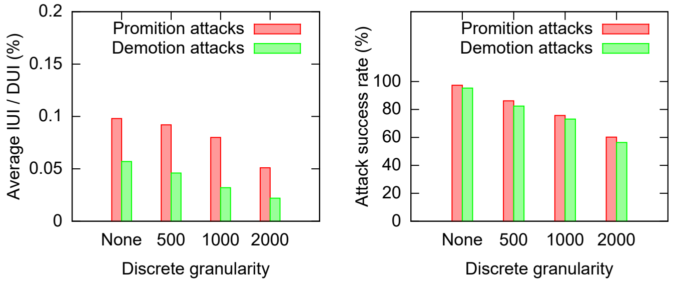
We propose to discretize item popularities and show the popularity range instead of the exact popularity for each item. This could achieve a trade-off between security of the recommender system against our attacks and user experience. Fig. 15 shows our attack results with medium knowledge when we discretize item popularities using different granularities. The co-visitation graph and experimental settings are the same with those we used in Section V. When the item popularity is discretized, our attacks sample a random number in the popularity range of an item and treat it as the item popularity. We observe that, when item popularity is discretized with a granularity of 2000, the threats of our attacks drop by about 40%, making an attack with medium knowledge similar to an attack with low knowledge.

B. Limiting Fake Co-visitations

Another direction of mitigating our attacks is to limit the number of fake co-visitations that an attacker can inject.

CAPTCHA: CAPTCHA is a widely used security technique to distinguish between human and computer. We note that none of the real-world recommender systems that we attacked has deployed CAPTCHAs. A web service can show a visitor CAPTCHA challenges if the number of visiting requests from the same IP address within a given short period of time is larger than a threshold. The threshold achieves a trade-off between user experience and attacker's success. Specifically, legitimate users/visitors might be affected by CAPTCHAs if the threshold is too small, while a too large threshold allows attackers to inject many fake co-visitations. Setting a good threshold requires analyzing behaviors of users in the recommender system. For instance, if a majority of users issue 10 visiting requests within 5 minutes, then the threshold can be set to be 10 for 5 minutes.

We note that recent studies [28, 29] demonstrated that CAPTCHA challenges can be automatically solved by machine learning techniques with relatively high accuracies, and an attacker can outsource CAPTCHA challenges to human workers using crowdsourcing platforms [30]. However, CAPTCHA is easy to deploy and it can still slow down the injection of fake co-visitations (because the attacker needs time to solve CAPTCHAs and it is challenging for the attacker to solve them with 100% accuracy) and increase the costs for attackers.



(a) IUI / DUI of attacks

(b) Attack success rate

Fig. 15: Effect of discretizing popularity of items.

Detecting fake co-visitations: Another mitigation strategy is to detect fake co-visitations. Once fake co-visitations are detected, the service provider can remove them from the co-visitation graph or reduce their importance if the detector is not very accurate. From a machine learning perspective, detecting fake co-visitations is an *anomaly detection* problem. For instance, if an unpopular item suddenly has many co-visitations with some items, then it is possible that an attacker is trying to promote this item via our fake co-visitation injection attacks. Via analyzing temporal dynamics (e.g., using similar techniques in Viswanath et al. [31]) of visits and co-visits, the service provider could detect certain fake co-visitations and mitigate our attacks. We were not able to explore this mitigation strategy since we do not have access to the visits and co-visits with temporal information.

Using co-visitations from registered users: The service provider can also choose to distinguish between visits from registered users and those from unlogged-in visitors, and give higher weights to visits from registered users. Moreover, the service provider can constrain that a registered user can only contribute to a limited number of co-visitations to each pair of items. As a result, fake co-visitation injection attacks rely on registering a large amount of fake accounts and using them to perform co-visitations. These fake accounts could be detected via *Sybil detection* methods. For instance, when social relationships between accounts are available, we can leverage SybilBelief [32] to detect fake accounts.

VIII. Discussion

Other attacks: We note that web services often provide a search functionality to help users locate relevant items. An attacker could leverage this functionality to perform attacks. For instance, an attacker could add popularly searched keywords to the title of the target item to perform promotion attacks. Such methods, usually called Search Engine Optimization (SEO) [33], is complementary to our fake co-visitation based methods, and they can be used together in practice.

Moreover, an attacker could simply visit a target item to make it more popular, and presumably it will appear in search results more frequently, which serves as a promotion attack.

However, it is hard for this method to perform optimized attacks. This is because how an item's popularity is related to its ranking in the search results is not publicly known and may vary from service to service, which implies that it is hard for an attacker to optimize the number of visits to a target item in order to promote it to be a certain rank in the search results. Furthermore, it is hard to evaluate the success of this method because how exactly users use the search functionality is also not publicly known. This limitation implies that, when an attacker develops the attacks as a service, the attacker cannot quantify the success of the attacks to an organization who pays for the service.

Attacking YouTube's deep learning based recommender systems: Google researchers recently proposed a deep learning based *user-to-item* recommender system for YouTube (especially for mobile version of YouTube) [34]. This new recommender system is much more complex than the co-visitation recommender system that we focus on. It is unclear whether our attacks are also effective at attacking such user-to-item recommender systems. Nevertheless, it is an interesting future work to study security of such deep learning based recommender systems.

IX. Conclusion and Future Work

In this work, we perform the first formal and systematic study on fake co-visitation injection attacks to recommender systems. First, we propose a novel threat model, which covers a variety of attackers with different goals and background knowledge. Second, we formulate fake co-visitation injection attacks as constrained optimization problems. An attacker can perform attacks with maximum threats via solving the optimization problems. Third, we demonstrate the feasibility and effectiveness of our attacks via evaluations on both synthetic recommender systems and real-world recommender systems used by popular web services such as YouTube, eBay, and Amazon. We plan to explore new methods to defend and mitigate our attacks in the future.

Acknowledgements

We thank the anonymous reviewers for their insightful comments, which have helped improve the paper substantially. This work is supported by the Department of Electrical and Computer Engineering of the Iowa State University through a startup package.

References

[1] Herbert A Simon. Designing organizations for an information-rich world. 1971.

[2] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[3] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[4] Xinyu Xing, Wei Meng, Dan Doozan, Alex C Snoeren, Nick Feamster, and Wenke Lee. Take this personally: Pollution attacks on personalized services. In *USENIX Security*, pages 671–686, 2013.

[5] William Zeller and Edward W. Felten. Cross-site request forgeries: Exploitation and prevention. Technical report, Princeton University, 2008.

[6] Miriam Marciel, Rubén Cuevas, Albert Banchs, Roberto Gonzalez, Stefano Traverso, Mohamed Ahmed, and Arturo Azcorra. Understanding the detection of view fraud in video content portals. In *WWW*, pages 357–368, 2016.

[7] K. Lang. Newsweeder: Learning to filter netnews. In *ICML*, 1995.

[8] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.

[9] P. Resnick, N. Iacovou, M. Sushak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW*, 1994.

[10] Y Koren, R Bell, and C Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 8:30–37, 2009.

[11] Bin Liu, Deguang Kong, Lei Cen, Neil Zhenqiang Gong, Hongxia Jin, and Hui Xiong. Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. In *WSDM*, 2015.

[12] Bin Liu, Yao Wu, Neil Zhenqiang Gong, Junjie Wu, Hui Xiong, and Martin Ester. Structural analysis of user choices for mobile app recommendation. *ACM Transactions on Knowledge Discovery from Data*, 11(2), 2016.

[13] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TKDE*, 17(6), 2005.

[14] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[15] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre. Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology*, 4(4):344–377, 2004.

[16] Shyong K Lam and John Riedl. Shilling recommender systems for fun and profit. In *WWW, 2004*, pages 393–402.

[17] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology*, 7(4):23, 2007.

[18] Joseph A. Calandrino, Ann Kilzer, Arvind Narayanan, Edward W. Felten, and Vitaly Shmatikov. "you might also like:" privacy risks of collaborative filtering. In *IEEE Symposium on Security and Privacy*, 2011.

[19] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yongyeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In *ACM/USENIX Internet Measurement Conference*, 2007.

[20] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, 1999.

[21] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data.

SIAM Rev., 51(4):661–703, 2009.

[22] Neil Zhenqiang Gong, Wenchang Xu, Ling Huang, Prateek Mittal, Emil Stefanov, Vyas Sekar, and Dawn Song. Evolution of social-attribute networks: Measurements, modeling, and implications using google+. In *IMC*, 2012.

[23] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[24] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[25] Mathias Lécuyer, Guillaume Ducoffe, Francis Lan, Andrei Papancea, Theofilos Petsios, Riley Spahn, Augustin Chaintreau, and Roxana Geambasu. Xray: Enhancing the web's transparency with differential correlation. In *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[26] Gloria Chatzopoulou, Cheng Sheng, and Michalis Faloutsos. A first step towards understanding popularity in youtube. In *IEEE INFOCOM Workshops*, pages 1–6. IEEE, 2010.

[27] P. Erdős and A. Rényi. On random graphs i. *Publ. Math. Debrecen*, 6, 1959.

[28] Elie Bursztein, Matthieu Martin, and John C. Mitchell. Text-based captcha strengths and weaknesses. In *ACM CCS*, pages 125–138, 2011.

[29] Elie Bursztein, Romain Beauxis, Hristo Paskov, Daniele Perito, Celine Fabry, and John Mitchell. The failure of noise-based non-continuous audio captchas. In *IEEE Symposium on Security and Privacy*, pages 19 – 31, 2011.

[30] Inside india's captcha-solving economy. http://blogs.zdnet.com/security/?p=1835. 2016-02-07.

[31] Bimal Viswanath, Muhammad Ahmad Bashir, Mark Crovella, Saikat Guha, Krishna P. Gummadi, Balachander Krishnamurthy, and Alan Mislove. Towards detecting anomalous user behavior in online social networks. In *Usenix Security*, 2014.

[32] Neil Zhenqiang Gong, Mario Frank, and Prateek Mittal. Sybilbelief: A semi-supervised learning approach for structure-based sybil detection. *IEEE Transactions on Information Forensics and Security*, 9(6):976–987, 2014.

[33] Harold Davis. *Search engine optimization.* " O'Reilly Media, Inc.", 2006.

[34] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys*, 2016.

Appendix

A. Formulating Linear Constrains

We show details of transforming two common normalization functions into corresponding linear constrains (as in Equation 6) in formulating the optimization problem. First, for the product normalization function, we have:

$$s'_{ji_t} > s'_{jk_j} \tag{20}$$

Via substituting with

$$s'_{ji_t} = \frac{w_{ji_t} + m_{jk}}{(w_j + m_{jk}) \cdot (w_{i_t} + m_{jk})} \tag{21}$$

$$s'_{jk_j} = \frac{w_{jk_j}}{(w_j + m_{jk}) \cdot w_{k_j}}, \tag{22}$$

we have the following linear constraint:

$$\frac{w_{ji_t} + m_{jk}}{(w_j + m_{jk})(w_{i_t} + m_{jk})} > \frac{w_{jk_j}}{(w_j + m_{jk})w_{k_j}} \tag{23}$$

$$\frac{w_{ji_t} + m_{jk}}{w_{i_t} + m_{jk}} > \frac{w_{jk_j}}{w_{k_j}} \tag{24}$$

$$\left(1 - \frac{w_{jk_j}}{w_{k_j}}\right) m_{jk} > \frac{w_{i_t} w_{jk_j}}{w_{k_j}} - w_{ji_t} \tag{25}$$

For the sqrt-product normalization function, the transformation is similar. After substituting with the sqrt-product function, we get:

$$\frac{w_{ji_t} + m_{jk}}{\sqrt{(w_j + m_{jk})(w_{i_t} + m_{jk})}} > \frac{w_{jk_j}}{\sqrt{(w_j + m_{jk})w_{k_j}}} \tag{26}$$

$$\frac{(w_{ji_t} + m_{jk})^2}{w_{i_t} + m_{jk}} > \frac{w_{jk_j}^2}{w_{k_j}} \tag{27}$$

$$m_{jk}^2 - \left(\frac{w_{jk_j}}{w_{k_j}} + 2w_{ji_t}\right) m_{jk} > \frac{w_{i_t} w_{jk_j}^2}{w_{k_j}} - w_{ji_t} \tag{28}$$

Since only non-negative real root of Equation 28 is meaningful in our context, we can approximate it as a corresponding linear constraint:

$$m_{jk} > \frac{\alpha_1 + \sqrt{\alpha_1^2 - 4\alpha_2}}{2}, \tag{29}$$

where the two constant coefficients are $\alpha_1 = \frac{w_{jk_j}}{w_{k_j}} + 2w_{ji_t}$ and $\alpha_2 = -\frac{w_{i_t} w_{jk_j}^2}{w_{k_j}} + w_{ji_t}$.

B. Unsorted Recommendation List

In our attacks with medium and low knowledge, we assume that the recommendation lists are sorted by the similarity between items. In the case that the list is unsorted, we assume the N items in the recommendation list are still the top-N items with highest similarity scores, only the orderings among them are random. The attacker's goal, however, is to make the target item appear in the top-N recommendation list. k is no longer a meaningful attacking parameter. We modify Equation 6 accordingly to reflect the new constraint:

$$s'_{ji_t} > \min_{k=1}^{N}\{s'_{jk_j}\} \tag{30}$$

Additionally, the parameter estimation process for medium knowledge attacker also needs to be modified. Without order information, the attacker can only estimate a loose upper bound for s_{jx} by $s_{jx} \leq \frac{\max\{w_j, w_x\}}{f(w_j, w_x)}$. Nevertheless, this loose upper bound can be updated during the aforementioned adaptive attacking process.