Understanding the Manipulation on Recommender Systems through Web Injection

Yubao Zhang, Jidong Xiao, *Member, IEEE*, Shuai Hao, *Member, IEEE*, Haining Wang, *Senior Member, IEEE*, Sencun Zhu, *Senior Member, IEEE*, and Sushil Jajodia, *Fellow, IEEE*

***Abstract*— Recommender systems have been increasingly used in a variety of web services, providing a list of recommended items in which a user may have an interest. While important, recommender systems are vulnerable to various malicious attacks. In this paper, we study a new security vulnerability in recommender systems caused by *web injection*, through which malicious actors stealthily tamper any unprotected in-transit HTTP webpage content and force victims to visit specific items in some web services (even running HTTPS), *e.g.*, YouTube. By doing so, malicious actors can promote their targeted items in those web services. To obtain a deeper understanding on the recommender systems of our interest (including YouTube, Yelp, Taobao, and 360 App market), we first conduct a measurement-based analysis on several real-world recommender systems by leveraging machine learning algorithms. Then, web injection is implemented in three different types of devices (*i.e.*, computer, router, and proxy server) to investigate the scenarios where web injection could occur. Based on the implementation of web injection, we demonstrate that it is feasible and sometimes effective to manipulate the real-world recommender systems through web injection. We also present several countermeasures against such manipulations.**

***Index Terms*— Recommender systems, recommendation manipulation, web injection.**

I. Introduction

IN THE era of information explosion, people have faced with an overwhelming array of choices when looking for information of their interests. There is a need to allocate the attention efficiently, and thereby recommender systems have become increasingly popular in recent years. Since recommender systems are able to allocate the attention and help users locate the information of their interests, they have been used in a wide variety of popular web services, such as YouTube, Amazon, and Yelp. Those web services are increasingly relying on recommender systems to improve the quality of their customers' experience, given that a recommender system is capable of making recommendations that match users' preferences. Moreover, recommender systems allow web service providers to increasingly shape users' selection and make the maximum profit. Therefore, recommender systems have become an essential component in a wide range of web services.

Despite the importance of a recommender system in the web ecosystems, it is vulnerable to malicious attacks. Xing *etal.* [1] proposed *pollution attacks* to *user-item* recommender systems. The attack could spoof the recommender system to recommend any items to a victim user, but the recommended items do not match the victim user's preference. Yang *et al.* [2] proposed *fake co-visitation attacks* to *item-item* recommendations, in which a recommender system could be tricked into recommending an item that may be entirely irrelevant to the viewing item.

Meanwhile, the economics of the web ecosystem is heavily dependent on web traffic. However, some of them (*e.g.*, HTTP traffic) are insecure and could be modified, referred to as *web injection*. Web injection manifests in a spate of ways, but fundamentally occurs when attackers tamper directly with web traffic. It can make profit from a variety of ways, such as inserting rogue tracking pixel and injecting ads.

In this paper, we make a first attempt to understand the manipulation on recommender systems caused by web injection. More specifically, there are basically two steps in the manipulation: (1) web injection is executed by tampering with in-transit HTTP traffic in public WiFi portals or proxy servers, and the malicious payload is injected into the HTTP webpage; (2) the tampered HTTP webpage will afterwards force victims to visit the targeted items in a recommender system so as to manipulate the recommender system (even running HTTPS) through browser sessions. In such an attack scenario, malicious actors can take advantage of browser sessions in two different ways. First, the large scale of browser sessions enables malicious actors to launch a fake

Manuscript received April 24, 2019; revised September 14, 2019 and November 1, 2019; accepted November 2, 2019. Date of publication November 20, 2019; date of current version July 17, 2020. This work was supported in part by the Army Research Office under Grant W911NF-13-1-0421 and Grant W911NF-19-1-0049 and in part by the National Science Foundation under Grant CNS-1618117 and Grant CNS-1822094. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Wei Yu. *(Corresponding author: Yubao Zhang.)*

Y. Zhang is with the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE 19716 USA (e-mail: ybzhang@udel.edu).

J. Xiao is with the Department of Computer Science, Boise State University, Boise, ID 83725 USA.

S. Hao is with the Department of Computer Science, Old Dominion University, Norfolk, VA 23529 USA, and also with the Center for Cyber Security Education and Research, Old Dominion University, Norfolk, VA 23529 USA.

H. Wang is with the Department of Electrical and Computer Engineering, Virginia Tech, Arlington, VA 22203 USA.

S. Zhu is with the Department of Computer Science and Engineering, Penn State University, University Park, PA 16802 USA.

S. Jajodia is with the Center for Secure Information Systems, George Mason University, Fairfax, VA 22030 USA.

Digital Object Identifier 10.1109/TIFS.2019.2954737

co-visitation attack. Co-visitations from a group of users would facilitate to establish a connection between two items. Second, each single browser session can be exploited to pollute the profile of a user, which enables miscreants to recommend any target items to the user. Moreover, such manipulations could exacerbate the impact of Cross-Site Request Forgery (CSRF) [3], [4] when CSRF vulnerabilities are exploited in real-world web services.

We first conduct a measurement study on real-world recommender systems and leverage machine learning to characterize their activities, which pave the way for further analysis. In particular, we collect the recommendation data from several popular services, including YouTube, Yelp, Taobao, and 360 Android App market. Using machine learning methods, we study the features that are publicly available in real-world recommender systems, such as "views count" in YouTube videos. We rank the importance of the features and predict whether item x (called *target item*) will display in the recommendation list of item y (called *anchor item*).

We then implement web injection in three different devices to demonstrate the potential impact of web injection on recommender systems. These three devices are a personal computer, a router, and a proxy server, each of which corresponds to an attack scenario. Attackers can use a personal computer to set up a WiFi portal and inject attack payload into the HTTP response of victims that connect to the portal. The implementations of web injection on routers and proxy servers could impose way more severe impact on recommender systems, since attackers can reach much more users through routers and proxy servers.

To demonstrate the feasibility of exploiting web injection for manipulation, we perform promotion attacks on real-world recommender systems. A promotion attack is to promote a specific target item and make it shown in an anchor item's recommendation list. More specifically, promotion attacks include three different kinds of attacks, *i.e.*, profile pollution attack, fake co-visitation attack, and CSRF-based attack. In light of ethical concerns, a router with implementation of web injection is only used to evaluate a profile pollution attack against a few accounts created by ourselves; while automatic attack platforms are implemented using Python to evaluate fake co-visitation and CSRF-based attacks, respectively. Thus, the impact of our experiments on real-world systems and users are controllable.

Finally, we present the countermeasures against the web injection based manipulation on recommender systems, including defense against web content tampering, blocking web service access, and limiting fake co-visitations.

In summary, we highlight our major contributions as follows:

- We conduct a measurement study on four real-world recommender systems and characterize their activities with respect to the publicly available features by using machine learning methods.
- We reveal that recommender systems are vulnerable to web injection for malicious manipulation, and we implement web injection in three different platforms to show three different scenarios where web injection could occur.
- We demonstrate the feasibility of the manipulation on recommender systems by launching three different kinds of promotion attacks on the real-world recommender systems. We also discuss the countermeasure to prevent the web injection based manipulation.

The rest of the paper is organized as follows. Section II provides a general overview and related works on recommender system and web injection. Section III presents our measurement study on real-world recommender systems and their behavior characterization based on machine learning. Section IV introduces the implementation of web injection and Section V explores specific attacks that are launched against real-world recommender systems. Section VI describes the countermeasures against the malicious manipulation. We discuss other potential attacks based on web injection in Section VII, and finally, we conclude in Section VIII.

II. Background and Related Work

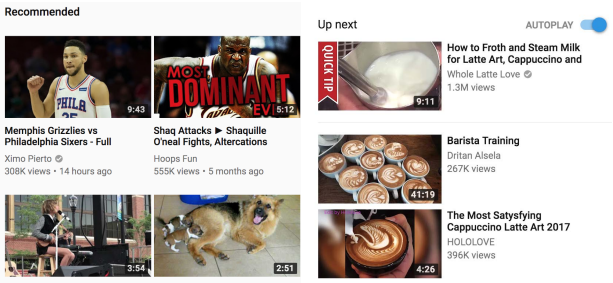
In this section, we present a brief overview of recommender systems as they play an important role in popular web services (e.g., YouTube [5], [6] and Amazon [7]), as well as mobile application markets [8], [9]. We then survey existing attacks against recommender systems and present a new attack vector, which exploits insecure HTTP traffic to manipulate recommender systems.

A. Recommender System

Web services are increasingly using recommender systems to deliver information or products to users who would be interested in. Many recommender systems have been developed in the past, including content based recommender systems [10], [11] and collaborative filtering based recommender systems [12], [13]. We introduce two widely used classifications of recommender systems, *i.e.*, filtering method and recommendation tasks.

There are several filtering methods to generate recommendations [14]. The content-based filtering method makes recommendations based on a user's preference. It seeks items that have similar content to a user's visiting history. The collaborative filtering method finds a set of users with similar preferences (user-item collaborative filtering) or searches a set of items that are similar to user visiting history (item-item collaborative filtering) [15]. The demographic filtering method recommends items to users based on their personal attributes (sex, age, country, etc.). Moreover, hybrid filtering is commonly used in real-world recommender systems [16].

The two popular recommendation tasks are *user-item recommendation* and *item-item recommendation*. The user-item recommendation takes user visiting history as input, while the item-item recommendation takes an item that is being visited as input. In a user-item recommendation, a recommender system recommends top-N items to a user based on his/her visiting history, which includes both implicit and explicit visiting history. In an item-item recommendation, a recommender system recommends top-N items that are similar to those items having been visited. Fig. 1 shows a user-item recommendation and an item-item recommendation in YouTube, respectively.



(a) User-item recommendation (b) Item-item recommendation

Fig. 1. User-item recommendation vs. Item-item recommendation.

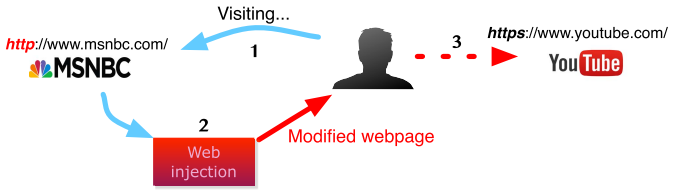


Fig. 2. An example of web injection. When a victim is visiting the MSNBC news website, web injection tampers with the in-transit HTTP content of MSNBC news webpage by injecting JavaScript to stealthily visit YouTube videos. The victim's browser executes the injected script to automatically visit YouTube videos in a stealthy manner.

Existing Attacks against Recommender System: Collaborative filtering recommender systems help users select a item they might like given a set of items. In a typical collaborative filtering recommender system, an $n \times m$ user-item matrix ($I_{n\times m}$) is created, where I_{ij} represents ith user's preference about jth item. A *Profile injection attack* (also called *shilling attack*) can introduce a bias into a collaborative filtering recommender system by inserting fake user ratings [17], [18]. In a profile injection attack, an attacker first registers a large number of fake accounts. With these fake accounts, the attacker interacts with the recommender system to build within it a large number of fake profiles to mislead the system's recommendations. Researchers have proposed several techniques to detect profile injection attacks, including a statistical method [19], a classification method [20], [21], and a data reduction-based method [22].

User-item recommender systems personlize items to users based on their profile (*e.g.*, browsing history). A *Pollution attack* [1] injects fake information into a user's profile via CSRF [3], such that an attacker can spoof the system to recommend the target item to a user. Xing *et al.* [1] demonstrated that YouTube, Amazon, and Google search are vulnerable to such attacks. However, a pollution attack is not applicable to item-item recommender systems.

Co-visitation recommender systems recommend items to users according to a *co-visitation graph*, which is a data structure for recording the co-visitation relationship among items. 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. Here, *co-visitation* relationship among two items means that the visitations of these two items appear successively within a certain period, *e.g.*, a user watches two videos on YouTube successively within ten minutes. Co-visitation recommendation has been implemented in several popular web services due to its simplicity, such as Amazon and YouTube. Yang *et al.* [2] proposed a *fake co-visitation injection attack*, which injects fake co-visitation into the co-visitation graph. In comparison with [2], our work exploits web injection to conduct not only fake co-visitation attack, but also profile pollution attack, and CSRF-based attack on recommender systems.

Additionally, Calandrino *et al.* [23] proposed a *privacy attack* to infer a user's profile by analyzing the publicly available recommendations from the recommender systems. They demonstrated that several popular web services (*e.g.*, Amazon and Last.fm) are vulnerable to the attack.

B. Web Injection

Malicious and unwanted actors could directly tamper with browser sessions and modify webpage content that reaches a browser, referred to as *web injection* [24]. Web injection manifests in many forms. Among these forms, *ad injection* reigns as one of the most lucrative strategies for monetizing web traffic. Public WiFi portals (*e.g.*, airport WiFi hotspots) and ISP routers could be compromised to tamper with in-transit HTTP content to inject ads [25], [26]. In addition, modifying a website through a man-in-the-middle attack could compromise the anonymity of some Tor users [27].

Our work presents a potential attack to recommender systems by tampering in-transit HTTP content in public WiFi portals or even ISP routers. Fig. 2 shows an example of manipulating the recommender system of YouTube via web injection. A victim visits the MSNBC news website. Web injection tampers with the in-transit HTTP content by injecting JavaScript to stealthily visit YouTube videos. Note that the attack just redirects victims to YouTube, instead of tampering with YouTube webpage content. The victim's browser executes the injected script to automatically visit YouTube videos in a stealthy manner, such as creating an invisible window on background. If the victim has logged in YouTube, YouTube counts the visitations as logged-in actions of the victim. If a large number of YouTube accounts visit YouTube unconsciously in this way, it could challenge the YouTube's recommendation mechanism.

III. Measurement Analysis of Recommender Systems

Due to various recommendation algorithms and diverse recommendation goals, practical recommender systems are complicated. Although there are plenty of works on theoretical analysis of a specific recommendation algorithm (*e.g.*, collaborative filtering), it is difficult to perform systematic analysis when more than one recommendation algorithms work together in practice [14], [16]. Different evaluation metrics (*e.g.*, coverage and novelty) are employed to guarantee the quality of recommendations.

Instead of theoretically analyzing real-world recommender systems, we conduct a measurement-based analysis on recommender systems by using machine learning methods. To this end, we first collect recommendation data from several services, including YouTube, Yelp, Taobao, and 360 Android App market. We extract features from publicly available

TABLE I
DATASET SUMMARY

	Size	Dimensions	Features
YouTube	22,762	Video title; View count; Publisher; Rating; Review comments	Title similarity; Ratio of view count; Same publisher; Ratio of rating; Comment similarity; Same commenter count; Same recommendation count
Taobao	46,551	Product title; Sales; Seller; Product description	Title similarity; Ratio of sales; Same seller; Description similarity; Same recommendation count
360 App Market	30,074	App title; Rate score; Download count; Review comments; Good rating; Fair rating; Bad rating Description	Title similarity; Ratio of rate score; Ratio of download count; Comment similarity; Ratio of good rating; Ratio of fair rating; Ratio of bad rating; Description similarity; Ratio of comment count; Same commenter count; Same recommendation count
Yelp	1,205	Place title; Review count; Review comments; Rating	Title similarity; Ratio of review count; Comment similarity; Ratio of rating; Same commenter count; Same recommendation count

recommendation data and train binary classifiers to mimic how a recommender system determines whether to recommend the target item to a user given an anchor item. Then we can rank the importance of the features and predict whether the target item will display in the recommendation list of an anchor item. Note that we only analyze the item-item recommendation in this section, while Xing *et al.* [1] investigated user-item recommendation (personalization).

A. Datasets

We collect data from YouTube [28], Yelp [29], Taobao [30], and 360 Android App market [31], respectively. We utilize Selenium to handle the dynamic content of webpages. The datasets are summarized in Table I and detailed below.

YouTube is a video-sharing website. It recommends 20 videos on the right hand side of the video that is being viewed. We collect a video titles, the number of views, publisher names, ratings, the review comments, and the recommendations from the anchor video that is being viewed. The videos in the recommendations are then viewed and crawled.

Taobao is a Chinese online shopping website similar to Amazon. It recommends five "People also viewed" products downwards the product webpage that is being browsed. We collect product titles, sales, seller names, descriptions, and recommendations. Similar to YouTube collection, the products in the recommendations are then browsed and crawled.

360 App Market is a Chinese App market similar to Google play. It recommends seven "People also liked" Apps on the right hand side of the App that is being viewed. We collect App titles, rate scores, the number of downloads, review comments, and recommendations, as well as good, fair, and bad ratings. Similarly, the Apps in the recommendations are then browsed and crawled recursively.

Yelp is a crowd-sourced local business review and social networking site. It recommends three "People also viewed" business places (*e.g.*, restaurants) on the right hand side of the anchor business place that is being viewed. We collect place names, the number of review, review comments, rating, and recommendations. Similarly, places are crawled recursively from the recommendations.

B. Features

We group positive samples by selecting pairs of items that one item is in the recommendation list of the other. The recommendation relationship is recorded during the data collection. Negative samples are grouped by randomly choosing pairs of items without recommendation relationship between each other. For each data sample, we consider the following features.

Same publisher/seller. It is set to 1 if two items are from the same publisher/seller; otherwise, 0. Some recommender systems tend to recommend items from the same publisher/seller.

Title similarity. We measure the similarity between two titles by computing their edit distance. Larger edit distance means that two titles are less similar to each other.

Comment/description similarity. We first aggregate comments/descriptions and remove stop words. Then, we utilize scikit-learn toolkit [32] to tokenize the document and obtain a vector of token counts. We measure comment/description similarity by computing the similarity of two token vectors.

Numeric features. For numeric values of items, we calculate the ratio of two numeric values of each pair. The detailed features are listed in Table I. The feature *same recommendation count* between items a and b therein means the number of recommendations that are displayed in the recommendation list of both items a and b.

C. Evaluation Results

We employ three different classifiers to evaluate the prediction accuracy, including Gaussian Naïve Bayes (GNB), Support Vector Machine (SVM), and Neural Network model (NN). GNB and SVM classifiers are built using scikit-learn toolkit, while NN is implemented using Keras [33]. Specifically, we use default *priors* in GNB, which is calculated from the datasets. RBF kernel is used in SVM. We utilize *GridSearch* to optimize two hyperparameters γ and C of RBF kernel SVM. The NN model we implemented here is a multi-layer feed forward neural network model. We train the model using Stochastic Gradient Descent(SGD) and use *crossentropy* loss function for the network. Our results are obtained through the 10-fold cross-validation.

The dataset size and prediction performance are listed in Table II. Each dataset has equal positive and negative samples. We observe that all the three classifiers have excellent

TABLE II
PREDICTION PERFORMANCE

	Size	GNB			SVM			DL		
		Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
YouTube	156,242	94.42%	**97.99%**	**90.70%**	94.79%	99.43%	90.09%	**94.84%**	94.47%	87.71%
Taobao	593,318	50.22%	50.11%	**99.65%**	**91.65%**	**97.51%**	85.48%	93.86%	91.86%	82.78%
360 App Market	217,432	85.44%	96.42%	73.60%	**93.88%**	**96.83%**	**90.73%**	93.86%	92.8%	87.95%
Yelp	2,460	77.89%	78.28%	**79.02%**	83.74%	**91.10%**	**74.88%**	**84.11%**	84.28%	74.62%

performance in four different datasets, except GNB on the Taobao dataset. The performance consistency suggests that our machine learning methods are, to a certain extent, effective to demonstrate how recommender systems work. Especially, feature importance can indicate the key points of recommender systems.

We utilize ensembles of decision tree with *sklearn* to investigate the relative importance of the features for each recommender system. The results are listed in Table III. We observe that the number of same recommendations for two items is the most important feature for all recommender systems, which is straightforward to understand on how recommender systems measure the similarity of items. If two items have more similar recommendations, they are more likely to be similar to each other.

Except the number of same recommendations, feature importance ranking of different recommender systems varies. We enumerate the features for each recommender system based on feature importance ranking.

YouTube promotes the recommendations from the same publisher. The other features play less important role in the recommendation. Taobao promotes the products with similar titles. Sales and descriptions are not important in the recommendation. Note that the feature of same seller is 0 because Taobao only recommends products from the same seller. For 360 App market, the other features are way less important than the number of same recommendations, which implies that the recommendation highly relies on the item similarity. Yelp makes recommendations based on similar comments. It also prefers the recommendations with the similar number of reviews, which means similar popularity.

It is worth to note that our analysis can only reveal how recommender systems work to a certain extent. However, our machine learning methods can take input other information (*e.g.*, co-visitation graph and user profile) to further improve prediction accuracy.

TABLE III
RELATIVE FEATURE IMPORTANCE

	Feature importance	
YouTube	Same recommendation count:	0.47;
	Same publisher:	0.216;
	Title similarity:	0.098;
	Ratio of view count:	0.059;
	Ratio of rating:	0.057;
	Comment similarity:	0.051;
	Same commenter count:	0.048
Taobao	Same recommendation count:	0.833;
	Title similarity:	0.138;
	Ratio of sales:	0.026;
	Description similarity:	0.003;
	Same seller:	0
360 App Market	Same recommendation count:	0.713;
	Ratio of good rating:	0.046;
	Ratio of comment count:	0.039;
	Ratio of fair rating:	0.039;
	Ratio of bad rating:	0.039;
	Comment similarity:	0.033;
	Ratio of rate score:	0.032;
	Title similarity:	0.024;
	Ratio of download count:	0.022;
	Description similarity:	0.014;
	Same commenter count:	0
Yelp	Same recommendation count:	0.416;
	Comment similarity:	0.224;
	Ratio of review count:	0.193;
	Title similarity:	0.073;
	Ratio of rating:	0.05;
	Same commenter count:	0.043

D. Selection of Anchor Items

Although the analysis above cannot uncover how those recommender systems work completely, it demonstrates some of principles about how those recommender systems work. Therefore, it aids us to select anchor items efficiently, such that the target items can be displayed in their recommendation lists with less cost. Taking input the target item, k nearest neighbors algorithm can be employed to select the anchor items.

First, we measure the similarity between items a and item b by computing cosine similarity of their feature vectors:

$$sim_{a,b} = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|_2 * \|\vec{B}\|_2} \quad (1)$$

where $\vec{A}$ and $\vec{B}$ represent the feature vectors of a and b, respectively. Then, we find the top n items that are most similar to the target item using k nearest neighbors algorithm. Additionally, we can construct the recommendation structure with the crawled dataset and leverage the structure information to improve the quality of selected anchor items.

IV. IMPLEMENTING WEB INJECTION

In this section, we implement web injection in three different scenarios, including personal computer, router, and HTTP proxy. We also discuss the injection vectors to implement web injection on webpage.

A. HTTP Tampering

We take into account three general scenarios where attackers could sniff HTTP traffic and modify webpage content. First, an attacker can use personal computer to set up a hotspot. Second, public wireless routers in business places (*e.g.*, coffee shop) could be utilized to perform HTTP traffic tampering. Third, proxy server is also an option for attackers for web traffic tampering. Corresponding to the above three scenarios,

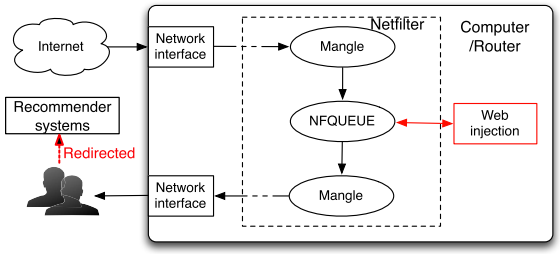


Fig. 3. Web injection in computer/router.

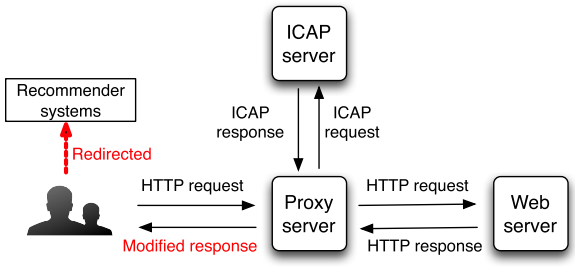


Fig. 4. Web injection in web proxy.

we implement HTTP content tampering in personal computer, router, and proxy server, respectively.

1) Personal Computer: We implement web injection in a Ubuntu system (Ubuntu 16.04 LTS), which acts as a WiFi hotspot, as Fig. 3 shows. The implementation of web injection is on packet level and based on packet firewall. For the packets coming in through network interface, a specific firewall rule checks if the packet is a TCP packet and its source port is 80 or 8080 in the mangle table. If the packet matches, it jumps to the target NFQUEUE, where packets are queued for a user space process. We employ *libnetfilter_queue* library [34] to receive the queued packets via a dedicated socket and then manipulate the payload. Also, we recalculate both the TCP and the IP checksums after manipulation, because otherwise the injected packet would be dropped by the network.

2) Router: We implement web injection in a router with LEDE operating system. Since LEDE is a Linux operating system, the implementation is almost the same as in Ubuntu system. The router with web injection implementation is employed to perform profile pollution attack in our following experiments.

3) HTTP Proxy: HTTP proxy is widely used since it can facilitate access to web content and provide anonymity, as well as bypassing IP address blocking. It has been reported that a number of HTTP proxies are actively modifying the relayed content [35]–[37]. To implement web injection in HTTP proxy, we leverage Internet Content Adaptation Protocol (ICAP), which is often used to implement content filters in transparent proxy servers. At the core of ICAP is a cache that proxies client transactions and process them through the ICAP server. In our implementation, we set up Squid proxy server and ICAP server in Ubuntu system (Ubuntu 16.04 LTS). The ICAP server is configured to modify HTTP response relayed by the proxy server, as shown in Fig.4.

B. Injection Vectors

To launch various web injection attacks against recommender systems, we manipulate the packet payload by injecting scripts, *e.g.*, tags. This enables us to automatically introduce actions upon the target items of a recommender system, including viewing the web pages of target items and adding comments to target items. The script injected into HTTP webpage is capable of making the browser rendering the webpage automatically take actions upon target items without being noticed by web users. For example, when a victim visits an HTTP webpage, under injection attacks, his browser will execute some injected script (*e.g.*, a frame tag), and hence automatically view YouTube videos by creating an invisible window on background. If users have logged in or have automatic log-in enabled, the unaware visits can not only change users' own profile, but also generate valid user-item connections. This connection plays key role in collaborative filtering recommendation algorithm.

Injected script can conduct stealthy actions upon target items in many ways. For example, we are able to redirect users to target items of a recommender system by stealthy view-based actions (*e.g.*, via *window.open* or *iframe*). Such actions can be executed in background without client's awareness. For example, an *iframe* of very small size can play a mute YouTube video that would be ignored by victims. Additionally, Content Security Policy (CSP) could block the cross-site redirection operations due to domain restriction. However, we can modify CSP configuration on packet level at the very beginning.

V. Attacking Real-World Systems

In this section, we demonstrate three different attacks mentioned in Section IV against real-world recommender systems based on web injection.

A. Experiment Overview

We evaluate promotion attacks on real-world recommender systems on several popular websites, including YouTube (the feature "Up Next video", the feature "Recommended" in users' homepage), 360 App market (the feature "People also liked"), Taobao (the feature "People also liked", the feature "You may like" in users' homepage), and Yelp (the feature "People also viewed"). Our attack goal is to make the target item display in the recommendation list of an anchor item or a user. Specifically, three different promotion attacks are evaluated in two modes: the logged-in mode and the anonymous mode. In the logged-in mode, there will be a logged-in user of the web service and hence the recommendations are tailored for the logged-in user; while in the anonymous mode, there is no such a logged-in user and therefore the recommendation systems select the items in various ways, e.g., user's geolocation or language preference.

Moreover, for the logged-in mode, we also inspect two scenarios: purging user history and not purging user history. In general, the recommendations would be significantly impacted by the presence of user history, which explicitly indicates the user's preference. However, purging user history

will force the recommendation system into selecting the items in different ways, e.g., similar to the anonymous mode or based on other invisible factors such as established user profiles stored in web services.

First, fake co-visitation attack can affect the co-visitation graph and hence manipulate the item-item recommendations that recommender systems make. We implement an automatic co-visitation attack platform using Python and utilize the open source web crawler Scrapy to collect item information from websites. To avoid the injected co-visitations being filtered by the web services, we inject the co-visitations with random time intervals, as well as utilizing Tor to disguise IP address of our platform and change IP address periodically. An anchor item is successfully attacked if the target item is shown in its recommendation list. Note that, we perform injections in both logged-in and anonymous scenarios. Comparing with fake co-visitation attack in [2], fake co-visitation attack via web injection can not only manipulate co-visitation graph by the co-visitations sent by victims' browsers, but also affect the item-item recommendations to these victims themselves.

Second, profile pollution attack is to manipulate the personalized items for a certain user. In other words, its goal is to spoof the user-item recommendations of recommender systems. It is different from profile injection attack. It pollutes the profile of an existing user, while profile injection attack is injecting fake new profiles. We conduct an automatic profile pollution with a smart router running web injection for a few accounts that we create for the experiments. An attack is successful if the target item is shown in user-item recommendations, *e.g.*, the feature "Recommended" in users' YouTube homepage.

Third, CSRF-based attack is more powerful than the above two attacks. It is capable of creating valuable user-item connection by conducting purchasing or commenting actions and hence manipulating recommendations dramatically. We restrain our experiments to only make non-misleading comments using a very small number of accounts. An anchor item is successfully attacked if the target item is shown in its recommendation list. Note that we merely simulate CSRF-based attack, instead of exploiting CSRF vulnerabilities of web services.

Ethical Considerations: Our work is to demonstrate the potential hazard of HTTP traffic tampering to recommender systems. We understand that our experiments are likely to impose certain impact on real-world recommender systems, though we want to emphasize that our experiments have quite low risks to service providers and users. We take several measures to mitigate the risks. First, we limit our experiments to a small scale. As we mentioned, our objective is to demonstrate the potential hazard,so a small-scale attack is capable of achieving our goal. Second, we create a small number of accounts to conduct profile pollution and CSRF-based attacks. The attacks would merely affect these users themselves, without impact on other users and service providers. Third, our experiments strictly follow the policy for vulnerability testing. Fourth, we do not perform experiments on any public router. We only use a private router to perform profile pollution attack on a few accounts created by ourselves.

TABLE IV
PROMOTION RATE OF OUR ATTACKS

	Attack	Promotion rate
YouTube	Fake co-visitation (anonymous)	7%
	Fake co-visitation (logged-in)	67.89% (on average)
	CSRF-based attack	15%
360 App Market	Fake co-visitation (anonymous)	82%
	Fake co-visitation (logged-in)	88%
Taobao	Fake co-visitation (anonymous)	30%
	Fake co-visitation (logged-in)	44%
Yelp	Fake co-visitation (logged-in)	4%

B. Overall Results

We measure the effectiveness of our attacks with the metrics *promotion rate*, which is defined as the number of successfully attacked anchor items over the total number of chosen anchor items for each attack. Table IV shows the promotion rate of our attacks. The details will be described later on.

We observe that the promotion rate in 360 App Market is much higher than other websites, which indicates that the recommendation algorithm of 360 App Market is the weakest against attacks. For all websites, the attacks in the logged-in mode have higher promotion rate than anonymous mode, since the attacks in the logged-in mode aim at the recommendations of an individual and the attacks in anonymous mode aim at the recommendations systemwide.

C. YouTube

We crawled the metadata information of over 20 thousand videos, as described in Section III. For each attack, we randomly select 100 target videos and choose one corresponding anchor video for each target video using the method in Section III-D. For simplicity, we call a target video and its corresponding anchor video *a pair of* videos. Throughout our experiments, we examine the recommendation list of each anchor item right before we start the attack and make sure that a target item is not in the recommendation list of its corresponding anchor item yet (otherwise, there is no need to promote).

1) Fake Co-Visitation Attack: To inject a fake co-visitation, a pair of videos are viewed in the same session. The viewing duration of each video is two minutes and the interval between sessions is one minute on average. This attack lasted one week.

Since fake co-visitation attack can result in either the target video showing in the recommendation list of its anchor item for all users, *i.e.*, global promotion, or only for those users who co-visit the pair of videos, *i.e.*, individual promotion. To mount the attack globally and achieve global success, we select 100 pairs of videos and insert fake co-visitations anonymously; while to test individual success, we select another 100 pairs of videos and insert fake co-visitations with logged-in users.

In the anonymous scenario, seven pairs among the selected pairs are promoted successfully. This low promotion rate could be attributed to (1) YouTube can detect fake co-visitation to some an extent and (2) we do not deliberately select unpopular videos to improve the promotion rate. Some of videos are highly popular. View count is used to measure the popularity of videos. For the selected and successfully promoted anchor

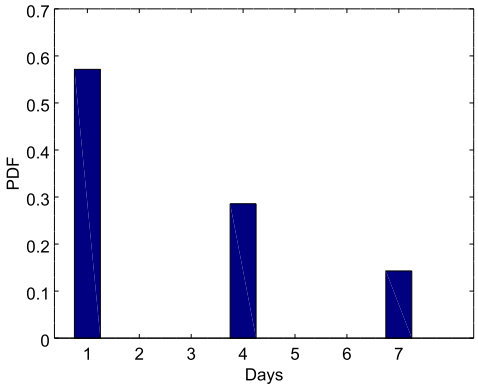


Fig. 5. YouTube promotion rate in the anonymous scenario.

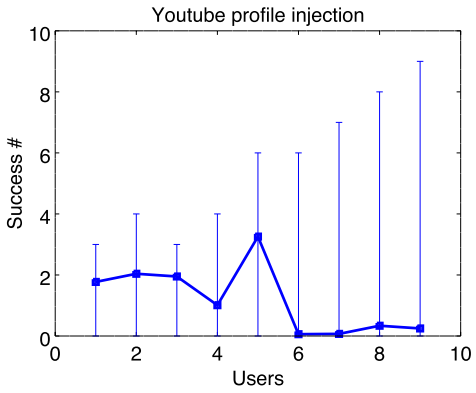


Fig. 6. Number of successfully promoted videos over nine accounts.

videos, the means of view count are 2, 430, 000 and 75, 700, respectively, which could imply that unpopular videos are more vulnerable to this fake co-visitation attack. Fig. 5 depicts the promotion rate against time. Four pairs are promoted in the first day. Two are promoted till the forth day and one is promoted till the seventh day.

For the logged-in scenario, the average promotion rate over nine users is 67.89%, which is much higher than that of the anonymous scenario. Especially, there are three users whose promotion rate are more than 95%. The higher promotion rate is attributed to the fact that we only focus on the logged-in users themselves. While global promotion has way more benefit than individual promotion, it requires much more resource to achieve. For the selected and successfully promoted anchor videos, the means of view count are 3.25M and 4.27M, respectively. We observe that anchor videos of highly popular can be easily promoted in the logged-in scenario.

2) Profile Pollution Attack: The feature "Recommended" in a user's YouTube homepage recommends 10 videos to the user. To attack the feature, we select 100 pairs of videos, in which each target video is already in the recommendation list of its anchor video, and inject the anchor videos into the watching history of 9 users. Each user's homepage recommendation is checked 100 times. Fig. 6 shows the number of successfully promoted videos with profile pollution attack over nine accounts. Blue square indicates the average number of successfully promoted videos of each account. The first 5 users' watching histories are purged before the attack, while the watching histories of the last 4 users remain intact. We observe that the promotion rate is much higher if the previous watching history is purged. This indicates that newcomers with less watching records are more vulnerable to such a profile pollution attack.

3) CSRF-Based Attack: As mentioned before, we do not exploit CSRF vulnerabilities of any real-world web services. To simulate CSRF-based attacks, we randomly select 100 pairs of videos and use nine accounts to make simple comments (*e.g.*, "I like it.") and click the like on both videos in a pair. Within the 100 pairs, we succeed to promote 15 target videos globally with the promotion rate of 15%, which is much more effective than anonymous co-visitation. For the selected and successfully promoted anchor videos, the means of view count are 2.37M and 3M, respectively. It implies that CSRF-based attacks (*e.g.*, making comment and like) could be highly capable to successfully promote popular videos.

4) Summary: First, promoting an anchor video globally is difficult, while targeting an individual victim is much more easier. The attack vector via web injection in routers or even ISPs is able to reach a large number of users and spoof the recommendations of these victims. Second, profile pollution attack via web injection can effectively spoof user-item recommendations. Due to the high exposure of the feature "Recommended" in the YouTube homepage, the attack could have a strong impact upon recommender systems. Third, CSRF-based attack on a recommender system is powerful, since it can promote a target video to a popular video that has good exposure. Fortunately, the attack relies on CSRF, which makes it hard to mount in reality.

D. 360 App Market

We crawled information of more than 30 thousand Apps. Since there is no user-item recommendation feature and comments are not allowed through the web portal, we only launch fake co-visitation attacks on 360 App market.

1) Fake Co-Visitation Attack: To inject a fake co-visitation, a pair of App webpages are visited in the same session. We choose one corresponding anchor App for each target App using the method in Section III-D. The interval between sessions is one minute on average. The attack lasts one week. Similarly, we consider two scenarios for fake co-visitation attack, *i.e.*, anonymous and logged-in. Each of them has 50 pairs of Apps.

In the anonymous scenario, App webpages are visited anonymously. Within 50 anchor Apps, 41 of them are promoted successfully on a global scale with the promotion rate of 82%. This high promotion rate means that the recommender system of 360 App market is considerably vulnerable to anonymous fake co-visitation attacks. Fig. 7 shows the promotion rate against time. Most of anchor Apps are successfully promoted within the beginning two days. We utilize download count to measure the popularity of anchor Apps, which could impact the promotion rate. For the selected and successfully promoted anchor Apps, the means of download count are 18.24M and 14.29M, respectively. It may imply

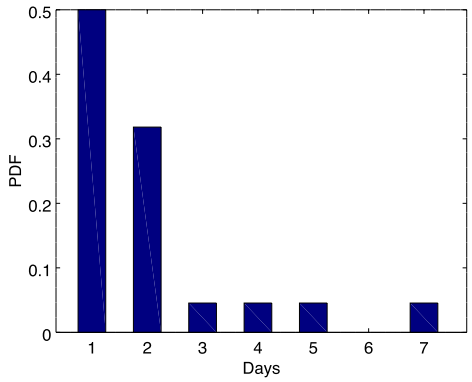


Fig. 7. 360 App Market promotion rate in anonymous scenario.

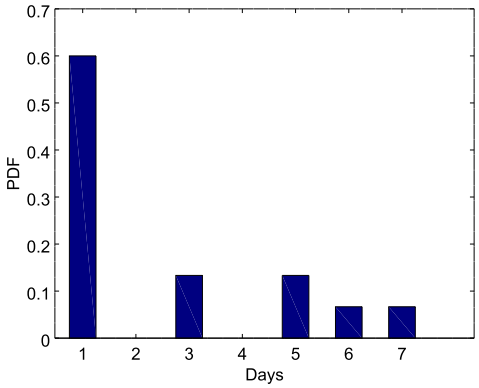


Fig. 8. Taobao promotion rate in anonymous scenario.

that 360 App market is susceptible to this attack regardless of App popularity.

In the logged-in scenario, each pair of App webpages are visited with a logged-in user. The attack lasts two weeks. Within 50 anchor Apps, 44 of them are promoted successfully for the user with a promotion rate of 88%. For the selected and successfully promoted anchor Apps, the means of download count are 16.89M and 27.15M, respectively. It implies that highly popular Apps are also susceptible to the promotion attack, which is quite similar to anonymous fake co-visitation attacks.

2) Summary: The recommender system of 360 App market is rather vulnerable to both anonymous and logged-in fake co-visitation attacks. Moreover, popular Apps with high download count are also susceptible to such attacks.

E. Taobao

We crawled information of more than 44 thousand items. Since the comments are anonymous, we launch fake co-visitation attack and profile pollution attack on Taobao.

1) Fake Co-Visitation Attack: To inject a fake co-visitation, a pair of item webpages are visited in the same session. We choose one corresponding anchor item for each target item using the method in Section III-D. The interval between sessions is one minute on average. The attack lasts one week. Similarly, we consider two scenarios for fake co-visitation attacks, *i.e.*, anonymous and logged-in. For each of them, 50 pairs of items are randomly selected.

In the anonymous scenario, item webpages are visited anonymously. Within 50 anchor items, 15 of them are promoted successfully on a global scale. Fig. 8 shows the promotion rate against time. Nine anchor items are successfully promoted in the first day. We utilize sold count to measure the popularity of anchor items. For the selected and successfully promoted anchor items, the means of sold count are 2.4 and 0.8, respectively. The sold count of anchor items is quite low because plenty of anchor items have never been sold.

In the logged-in scenario, each pair of item webpages are visited with a logged-in user. Within 50 anchor items, 22 of them are promoted successfully for the user with a promotion rate of 44%. For the selected and successfully promoted anchor items, the means of sold count are 7.9 and 12.7, respectively. Even though the sold count is still quite small, it could conjecture that items with sold count less than a certain value (*e.g.*, 100) might be susceptible to such a attack, which represents a large number of items in Taobao due to their heavy-tailed distribution.

2) Profile Pollution Attack: The feature "You may like" in a user's Taobao homepage recommends a group of items to the user. To attack the feature, we select 50 pairs of videos in which the target item is already in the recommendations of the anchor item and inject the anchor items into the history of a user. The homepage recommendations is checked 100 times. However, none of the target items is recommended in the feature "You may like". Instead, this feature is employed to promote the items that Taobao itself needs to promote.

3) Summary: The recommender system of Taobao is somewhat vulnerable to both anonymous and logged-in fake co-visitation attacks. However, it is resistant to profile pollution attacks.

F. Yelp

We crawled information of more than one thousand restaurants. Since there is no user-item feature and fake comments could be misleading, we only launch fake co-visitation attacks on Yelp.

1) Fake Co-Visitation Attack: To inject a fake co-visitation, a pair of restaurant webpages are visited in the same session. We choose one corresponding anchor restaurant for each target restaurant using the method in Section III-D. The interval between sessions is one minute on average. The attack lasts one week. Similarly, we consider two scenarios for fake co-visitation attack, *i.e.*, anonymous and logged-in. For each of them, 50 pairs of restaurants are randomly selected. Review count is used to measure restaurant popularity.

In the anonymous scenario, restaurant webpages are visited anonymously. Within 50 anchor restaurants, none of them are promoted successfully on a global scale. For the selected anchor restaurants, the mean of review count is 256.8.

In the logged-in scenario, each pair of restaurant webpages are visited with a logged-in user. The attack lasts one week. Within 50 anchor restaurants, two of them are promoted successfully for the user. For the selected and successfully promoted anchor restaurants, the mean of review count are 221.8 and 967, respectively. Two successfully promoted anchor

restaurants are both considerably popular with a large number of reviews.

2) Summary: The recommender system of Yelp is very resistant to both anonymous and logged-in fake co-visitation attacks. We conjecture that CSRF-based attacks could affect the recommender system of Yelp, since Yelp emphasizes the quality of reviews all the time. However, we do not launch CSRF-based attacks (*e.g.*, making fake comments) due to ethical consideration.

VI. Countermeasures

Exploiting web injection to attack web services is quite covert from the viewpoints of both web service providers and clients. Therefore, we discuss different countermeasures from the perspectives of both parties.

A. Implementing HTTPS on Web Services

HTTPS leverages the SSL/TLS to create an encrypted connection between a web server and a web browser. If SSL/TLS is used by a web server, web content tampering is hardly possible.

However, while the adoption of HTTPS is increasing, the practical usage of HTTPS is still at a relatively low level over the entire Internet [38]. The reasons include the lack of awareness of growing importance of secure connections, the cost and perceived complexity of switching to HTTPS, as well as the potential Search Engine Optimization (SEO) impact.

B. Limiting Fake Co-Visitations

Yang *et al*. proposed leveraging CAPTCHA, anomaly detection, and distinguishing registered users to limit fake co-visitations [2]. However, our study considers different attack scenarios from them. The attacks we explore include a set of registered users who are hijacked to make co-visitations, instead of individual attackers who attempt to promote or demote some items via making a burst of co-visitations. Therefore, based on the countermeasures proposed in [2], web service providers should take the IP addresses of visitors into consideration and detect anomalous behaviors in certain IP addresses. Moreover, analyzing user behaviors on the basis of IP address could also be able to detect potential profile pollution attacks. If a spate of users with the same IP address (probably through the same router in a public place) conduct the same action (*e.g.*, watching the same video), these users are probably exposed to profile pollution attacks.

C. Securing User Browsers

In order to study the mitigation strategies on the client side, we develop the extensions on two widely used browsers, Chrome and Firefox. The extensions detect and block a fake co-visitation attack by searching the patterns of the invisible co-visitation, *e.g.*,

"window\.open\("(http(s)?:\/\/)?((w)3.)?YouTube?(.com)?\/.+"\)"

or "<iframe.*?src="(http(s)?:\/\/)?((w)3.)?YouTube?(.com)?\/.+".

The evaluation on our testbed demonstrates that our extensions can effectively identify and prevent the activities of injected co-visitation. Furthermore, the extensions can also detect suspicious CSRF-based attacks by matching keywords in the URL path (*e.g.*, "action", "comment", and "submit") and cooperate with existing CSRF defense mechanisms, such as synchronizer tokens, double cookie defense, encrypted token pattern, and custom header [39], to improve the effectiveness against recommendation manipulation.

VII. Discussion

Although we only evaluate the promotion attacks on real-world recommender systems, the attacks that we demonstrate also applied to the demotion attacks, *i.e.*, obliterating target item from anchor item's recommendation list. Yang *et al.* [2] presented the feasibility of exploiting the fake co-visitation to perform demotion attacks.

In addition, web injection could be widely applied to perform a variety of attacks, such as ad injection, targeting ad pollution, and malware download.

A. Ad Injection

As presented in [25], [26], several practical instances of ad injection are illustrated, *e.g.*, public WiFi portals that tamper with in-transit HTTP content to inject ads. Ad injection by tampering is a straightforward manner of monetizing browser traffic. However, such activities would not only degrade user experience, but also harm the reputation of website [24]. Similarly, adversaries can inject malicious code to automatically download malware.

B. Ad Targeting

Ad targeting can be used by advertisers to target users who are interested in some categories [40] and maneuver the ad selection. Ads that might relate to the user's online interest are selected according to the observation from her browsing pattern, which is profiled by ad exchange and other third party trackers via installing third party JavaScript tracking code. Tampering with in-transit HTTP content can load polluted content in a camouflaged manner and impact the user's profile, such that attackers can influence the ad selection process.

In comparison with these attacks mentioned above, the attacks on recommender systems via web injection are more stealthy. It attributes to the reason that attacks on recommender systems are based on the behavior of visiting the recommender system itself, rather than tampering with webpage appearance like ad injection does. Such stealthiness could make the attacks more powerful with respect to both duration and scale.

VIII. Conclusions

In this paper, we revealed that recommender systems are vulnerable to web injection, which is exploitable for malicious manipulation in a recommender system. To deeply understand this security threat, we first conducted a measurement-based analysis on several real-world recommender systems by using

machine learning methods. Then we implemented web injection in three different devices (*i.e.*, computer, router, and proxy server) to investigate the potential impact of web injection. Based on these implementations, we further demonstrated that it is feasible and sometimes effective to manipulate the real-world recommender systems through web injection. Finally, we presented the countermeasures against the malicious manipulations on recommender systems, including defenses against web content tampering, limiting fake co-visitations, and securing user browsers.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their insightful comments, which help improve the quality of this paper.

REFERENCES

[1] X. Xing, W. Meng, D. Doozan, A. C. Snoeren, N. Feamster, and W. Lee, "Take this personally: Pollution attacks on personalized services," in *Proc. USENIX Secur. Symp.*, 2013, pp. 671–686.

[2] G. Yang, N. Z. Gong, and Y. Cai, "Fake co-visitation injection attacks to recommender systems," in *Proc. NDSS*, 2017, pp. 1–15.

[3] W. Zeller and E. W. Felten, "Cross-site request forgeries: Exploitation and prevention," *New York Times*, pp. 1–13, Oct. 10, 2008.

[4] A. Sudhodanan, R. Carbone, L. Compagna, N. Dolgin, A. Armando, and U. Morelli, "Large-scale analysis & detection of authentication cross-site request forgeries," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Apr. 2017, pp. 350–365.

[5] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for YouTube recommendations," in *Proc. 10th ACM Conf. Recommender Syst.*, 2016, pp. 191–198.

[6] J. Davidson *et al.*, "The YouTube video recommendation system," in *Proc. 4th ACM Conf. Recommender Syst.*, 2010, pp. 293–296.

[7] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, Jan./Feb. 2003.

[8] H. Zhu, H. Xiong, Y. Ge, and E. Chen, "Mobile app recommendations with security and privacy awareness," in *Proc. 20th Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 951–960.

[9] B. Liu, D. Kong, L. Cen, N. Z. Gong, H. Jin, and H. Xiong, "Personalized mobile app recommendation: Reconciling app functionality and user privacy preference," in *Proc. 8th ACM Int. Conf. Web Search Data Mining*, 2015, pp. 315–324.

[10] K. Lang, "NewsWeeder: Learning to filter netnews," in *Machine Learning Proceedings*. Amsterdam, The Netherlands: Elsevier, 1995, pp. 331–339.

[11] R. J. Mooney and L. Roy, "Content-based book recommending using learning for text categorization," in *Proc. 5th ACM Conf. Digit. Libraries*, 2000, pp. 195–204.

[12] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An open architecture for collaborative filtering of netnews," in *Proc. ACM Conf. Comput. Supported Cooperat. Work*, 1994, pp. 175–186.

[13] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 40, no. 8, pp. 30–37, Aug. 2009.

[14] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowl.-Based Syst.*, vol. 46, pp. 109–132, Jul. 2013.

[15] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, 2001, pp. 285–295.

[16] C. Porcel, A. Tejeda-Lorente, M. Martínez, and E. Herrera-Viedma, "A hybrid recommender system for the selective dissemination of research resources in a technology transfer office," *Inf. Sci.*, vol. 184, no. 1, pp. 1–19, 2012.

[17] I. Gunes, C. Kaleli, A. Bilge, and H. Polat, "Shilling attacks against recommender systems: A comprehensive survey," *Artif. Intell. Rev.*, vol. 42, pp. 767–799, Dec. 2014.

[18] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams, "Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness," *ACM Trans. Internet Technol.*, vol. 7, no. 3, p. 23, Oct. 2007.

[19] N. Hurley, Z. Cheng, and M. Zhang, "Statistical attack detection," in *Proc. 3rd ACM Conf. Recommender Syst.*, 2009, pp. 149–156.

[20] R. Burke, B. Mobasher, C. Williams, and R. Bhaumik, "Detecting profile injection attacks in collaborative recommender systems," in *Proc. 8th IEEE Int. Conf. Enterprise Comput., e-Commerce, e-Services*, Jun. 2006, p. 23.

[21] C. A. Williams, B. Mobasher, and R. Burke, "Defending recommender systems: Detection of profile injection attacks," *Service Oriented Comput. Appl.*, vol. 1, no. 3, pp. 157–170, Nov. 2007.

[22] S. Zhang, Y. Ouyang, J. Ford, and F. Makedon, "Analysis of a low-dimensional linear model under recommendation attacks," in *Proc. 29th ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2006, pp. 517–524.

[23] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov, "'You might also like:' Privacy risks of collaborative filtering," in *Proc. IEEE Symp. Secur. Privacy*, May 2011, pp. 231–246.

[24] K. Thomas *et al.*, "Ad injection at scale: Assessing deceptive advertisement modifications," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 151–167.

[25] D. Kravets. (2014). *Comcast Wi-Fi Serving Self-Promotional Ads Via JavaScript Injection*. [Online]. Available: http://arstechnica.com/tech-policy/2014/09/why-comcasts-javascript-ad-injections/threaten-security-net-neutrality/

[26] D. Coldewey. (2014). *Marriott Puts an End to Shady Ad Injection Service*. [Online]. Available: http://techcrunch.com/2012/04/09/marriott-puts-an-end-to-shady-ad-injection-service/

[27] T. G. Abbott, K. J. Lai, M. R. Lieberman, and E. C. Price, "Browser-based attacks on Tor," in *Proc. Int. Workshop Privacy Enhancing Technol.* Springer, 2007, pp. 184–199.

[28] Youtube. [Online]. Available: https://www.youtube.com/

[29] Yelp. [Online]. Available: https://www.yelp.com/

[30] Taobao. [Online]. Available: https://www.taobao.com/

[31] *360 App Market*. [Online]. Available: http://zhushou.360.cn/

[32] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[33] F. Chollet *et al.* (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras

[34] H. Welte. (2015). *Libnetfilter_Queue Project*. [Online]. Available: https://www.netfilter.org/projects/libnetfilter_queue/index.html

[35] G. Tsirantonakis, P. Ilia, S. Ioannidis, E. Athanasopoulos, and M. Polychronakis, "A large-scale analysis of content modification by open HTTP proxies," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[36] D. Perino, M. Varvello, and C. Soriente, "ProxyTorrent: Untangling the free HTTP (S) proxy ecosystem," in *Proc. Conf. World Wide Web*, 2018, pp. 197–206.

[37] A. Mani, T. Vaidya, D. Dworken, and M. Sherr, "An extensive evaluation of the Internet's open proxies," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, 2018, pp. 252–265.

[38] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS adoption on the Web," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1323–1338

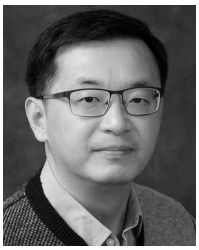
[39] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proc. 15th ACM Conf. Comput. Commun. Secur.*, 2008, pp. 75–88.

[40] W. Meng, X. Xing, A. Sheth, U. Weinsberg, and W. Lee, "Your online interests: Pwned! A pollution attack against targeted advertising," in *Proc. 2014 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 129–140.

Yubao Zhang received the bachelor's and master's degrees from the National University of Defense Technology, China. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Delaware. His research interests lie in online social networking and web security.

Jidong Xiao received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2016. He is currently an Assistant Professor with Boise State University, Boise, ID, USA. His research focuses on cybersecurity, with a particular emphasis on operating system security and cloud security.

Shuai Hao received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2017. He worked as a Post-Doctoral Researcher with CAIDA, UC San Diego. He is currently an Assistant Professor with the Department of Computer Science, Old Dominion University (ODU), Norfolk, VA. His research interests lie in the measurement and security of internet infrastructure and networking systems.



Haining Wang received the Ph.D. degree in computer science and engineering from the University of Michigan at Ann Arbor in 2003. He was a Professor of ECE with the University of Delaware. He is currently a Professor with the Department of Electrical and Computer Engineering, Virginia Tech. His research interests lie in the areas of security, networking systems, cloud computing, and Internet-of-Things (IoT) systems.

Sencun Zhu received the B.S. degree in precision instruments from Tsinghua University, Beijing, China, in 1996, the M.S. degree in signal processing from the Graduate School, University of Science and Technology of China, Beijing, in 1999, and the Ph.D. degree in information technology from George Mason University, Fairfax, VA, USA, in 2004. He is currently an Associate Professor with Penn State University. His research interests include wireless and mobile security, network and systems security, and software security. Among many academic services, he is also the Editor-in-Chief of *EAI Transactions on Security and Safety* and an Associate Editor of the IEEE TRANSACTIONS ON MOBILE COMPUTING (TMC).

Sushil Jajodia is currently a University Professor, a BDM International Professor and the Director of Center for Secure Information Systems with George Mason University. Prior to joining Mason, he held permanent positions at NSF, NRL, and University of Missouri-Columbia. He has sustained a highly active research agenda spanning database and cyber security for over 30 years. He has authored or coauthored 7 books, edited 52 books and conference proceedings, and published more than 500 technical articles in refereed journals and conference proceedings. He holds 23 U.S. patents, and has received a number of prestigious awards in recognition of his research accomplishments. According to the Google Scholar, he has over 42 ,000 citations and his H-index is 105.