Redacted for double-blind review

Anonymous Author(s)

ABSTRACT

In recent years, we have witnessed a dramatic increase in the application of Machine Learning algorithms in several domains, including the development of recommender systems for software engineering (RSSE). While researchers focused on the underpinning ML techniques to improve recommendation accuracy, little attention has been paid to make such systems robust and resilient to malicious data. By manipulating the algorithms' training set, i.e., large open-source software (OSS) repositories, it would be possible to make recommender systems vulnerable to adversarial attacks. This paper presents an initial investigation of adversarial machine learning and its possible implications on RSSE. As a proof-of-concept, we show the extent to which the presence of manipulated data can have a negative impact on the outcomes of two state-of-the-art recommender systems which suggest third-party libraries to developers. Our work aims at raising awareness of adversarial techniques and their effects on the Software Engineering community. We also propose equipping recommender systems with the capability to learn to dodge adversarial activities.

1 INTRODUCTION

To cope with their everyday programming tasks, developers access and browse various information sources [5]. Given the abundance of sources of formal and informal documentation, the problem is not the lack of information, but, instead, its overload [18, 19]. Recently, many studies have been conducted to develop methods and tools — recommender systems for software engineering (RSSE) — to provide developers with automated assistance [10, 21, 24]

The development of RSSE encompasses several phases including the design of the underpinning algorithms or the reuse of existing ones. Machine Learning (ML) techniques are amongst the natural choices that developers take when new recommender systems have to be conceived [25]. This means, for example, that the recommendation of code elements (e.g., snippets or APIs) is learned from existing code bases or informal documentation. As a result, the quality of the provided recommendation depends on the quality of the underlying data, whose noisiness has been previously reported [14].

Even worse, online data used to train recommenders can be exploited for malicious purposes. Adversarial Machine Learning [13, 32] (AML) is a field of study that focuses on security issues in ML systems and, specifically, in recommender systems too [7]. Research has been done to identify probable threats and seek out adequate countermeasures [1, 34]. For example, Anelli *et al.* [1] use shilling attack [30] to manipulate a collaborative-filtering recommender system operated with Linked Data. Also, Wang and Han [34] propose an improvement of the Bayesian personalized ranking (BPR) technique by exploiting the adversarial training-based mean strategy (AT-MBPR) in the domain of collaborative filtering-based recommender systems. To protect a system against threats, in the first place, it is necessary to be knowledgeable of various types of adversary activities [16]. Such activities generate perturbations to deceive and disrupt systems by causing a malfunction, compromising their recommendation capabilities.

The ultimate aim of these attacks is to manipulate target items, thus creating either a negative or positive influence on the final recommendations [11], depending on the attacker's intention. For instance, in image classification, an attacker crafts an input image by adding non-random noise in a way that it will be falsely classified by ML models, whilst still being properly recognized by humans [20]. As an example, a panda was recognized as a ribbon by cutting-edge deep neural networks once the input image had been padded with noise, meanwhile humans still correctly perceived the panda from the forged image [9].

AML has been studied in a wide range of domains, e.g., online shopping systems [16] or image classification [20], and addresses both risks and countermeasures. However, to the best of our knowledge, AML has not been investigated in the context of SE applications of ML yet.

In this paper, we report the first study on threats that cause harm or danger to RSSE suggesting third-party libraries and API function calls. We select these two types of recommendations as they are representative of scenarios in which *(i)* the recommendation is learned from OSS repositories; and *(ii)* the outcome of a malicious recommendation, e.g., the usage of a library or an API, can result in severe security holes [2].

Based on thorough observations and on the analysis of the existing literature, we realized that most of the efforts to improve RSSE have been made to enhance their prediction accuracy. To the best of our knowledge, no work has investigated the problem of using intentionally falsified data to compromise recommender systems' capabilities, as well as conceptualizing the corresponding countermeasures. In this respect, our work is the first one that brings in the issue of AML in RSSE.

Most RSSE heavily rely on open data sources, such as GitHub, the Maven Central Repository, or Stack Overflow, which can be easily steered by adversaries [35]. In other words, these systems are vulnerable to attacks equipped with forged input data. Therefore, there is the need for comprehending the likely threats, with the ultimate aim of conceiving counteractions to increase the resilience of RSSE.

We first provide an overview of state-of-the-art API and third-party library recommenders, discussing how they could be potentially affected by AML. Through a literature search from premier venues in Software Engineering for adversarial techniques, we show that there are considerably evident threats to RSSE. Then, we perform a preliminary examination of two existing systems for recommending third-party libraries. The experimental results reveal a worrying outcome: by a simple manipulation, we can seamlessly spoil the final recommendations, putting software clients at risk.

This paper makes the following contributions:

– A discussion on the possible adversarial attacks to state-of-the-art RSSE for third-party libraries and API calls;

Table 1: Notable RSSE for mining libraries and APIs.

	System	Venue	Year	Data source
Library rec.	LibRec [31]	WCRE	2013	GitHub
	LibCUP [27]	JSS	2017	GitHub
	LibD [15]	ICSE	2017	Android markets
	LibFinder [23]	IST	2018	GitHub
	CrossRec [21]	JSS	2020	GitHub
	LibSeek [12]	TSE	2020	Google Play, GitHub, MVN
API rec.	MAPO [36]	ECOOP	2009	SourceForge
	UP-Miner [33]	MSR	2013	Microsoft Codebase
	DeepAPI [10]	ESEC/FSE	2016	GitHub
	PAM [8]	ESEC/FSE	2016	GitHub
	FINE-GRAPE [28]	EMSE	2017	GitHub
	FOCUS [22]	ICSE	2019	GitHub, MVN

– A preliminary evaluation on two library recommender systems to perturb their recommendations.

The paper is structured as follows. Section 2 provides background notions about adversarial attacks and their effects on RSSE. Section 3 reports an initial evaluation on two RSSE. Section 4 makes an overview of related work, whereas Section 5 outlines future work and concludes the paper.

2 ATTACKS AND THREATS TO RSSE

In the following subsections, we first (Section 2.1) provide an overview of various types of attacks to recommender systems, and then (Section 2.2) review state-of-the-art RSSE suggesting third-party libraries and API function calls, while analyzing the AML threats they might be exposed to.

2.1 Classification of attacks

Attacks to recommender systems are classified into two main categories as follows [7]:

- *Poisoning attacks* spoil an ML model by falsifying the input data;
- *Evasion attacks* attempt to avoid being detected by hiding malicious contents, which then will be classified as legitimate by ML models.

With poisoning attacks, there are two possible interventions:

- *Push attacks* favor the targeted items, thus increasing the possibility of being recommended;
- In contrast, *nuke attacks* try to downgrade/defame the targeted items [1, 16], compelling them to disappear from the recommendation list.

In the scope of this paper, we focus on poisoning attacks as they are easy to conduct, yet effective. The remaining attacks are left to our future work.

2.2 Potential risks to RSSE for mining libraries and API calls

We review notable RSSE that support the development of software projects by delivering third-party libraries and API function calls. Table 1 lists the considered systems according to their functionality in chronological order. By investigating their internal design, we discuss possible vulnerabilities according to the previously-given categorization of attacks.

▷ **Library recommendation.** LibRec [31] recommends libraries using a combination of rule mining and a collaborative-filtering technique to mine libraries from projects similar to the one being developed. LibCUP [27] suggests libraries that have strong ties by using a clustering approach to identify and recommend co-usage patterns. LibD [15] provides libraries to Android apps using a clustering technique. First, it decompiles applications to build a control flow graph composed of packages, classes, and methods belonging to the projects. Then, the graph is used to extract features, and grouped by a similarity function. Ouni *et al.* propose LibFinder [23], providing libraries based on a multi-objective search-based algorithm. Being built with a collaborative-filtering technique, CrossRec extracts libraries from similar projects [21]. LibSeek [12] relies on a matrix factorization technique to deliver relevant libraries for mobile apps, obtained by collecting neighborhood information, i.e., characteristics of similar libraries.

As it can be seen, all the considered systems leverage open sources, e.g., GitHub or Android markets, for training. Moreover, they mine libraries using similarity-based measures, either a similarity function, or a clustering technique. Thus, they are exposed to perturbations with malicious content hidden in OSS projects. We therefore conjecture that, by fabricating projects with bogus data, attackers can favor (push attack) or defame (nuke attack) a library [1, 7, 30]. In other words, they can make a good/useful library out of being recommended, or even worse, promote a bad/malicious library to a higher rank in the recommendation list, so that users of the recommender system would unintendedly adopt it.

▷ **API recommendation.** MAPO [36] recommends API patterns by extracting API related information from the developer's context. The resulting data is clustered and ranked according to their similarity with the client code. In this respect, the system can be fooled with malicious code intentionally inserted into similar projects. Wang *et al.* propose UP-Miner [33] exploiting SeqSim and BIDE to mine from source code. Since UP-Miner relies on a similarity measure, it may recommend to developers malicious code embedded in projects disguised as similar. DeepAPI [10] generates relevant API sequences starting from a natural language query. It employs an RNN Encoder-Decoder to encode words in context vectors used to train the model. As the corpus is collected from GitHub projects, a hostile user can easily inject perturbations during the data gathering phase, i.e., feeding the system with interfered projects. PAM [8] has been proposed to extract relevant API patterns from client code by using the structural Expectation-Maximization (EM) algorithm to infer the most probable items. The mined API patterns are then ranked according to their probability. Push and nuke attacks could easily modify the final ranking obtained by the tool, i.e., operating on terms' occurrences to favor or defame a certain pattern. FINE-GRAPE [28] delivers relevant APIs by relying on the history of the related files. It parses GitHub projects, discovers, and ranks the relevant API calls according to their history, i.e., methods, annotations, and classes from every API version. FINE-GRAPE is prone to manipulations which forge an artificial history of API calls in GitHub projects. FOCUS [22] suggests APIs by encoding projects in a tensor and using a collaborative-filtering technique. Since it works on data mined from similar projects, FOCUS is not immune from poisoning attacks, i.e., an adversary can create fake projects with toxic APIs and pose them as legitimate to trick FOCUS into recommending these calls.

3 PROOF OF CONCEPT

In this section, we report a preliminary investigation into the implication of AML on two existing RSSE. The study has been conducted on two library recommenders, i.e., LibRec [31] and CrossRec [21] (cf. Table 1). Such tools are chosen due to the following reasons. LibRec is a well-established tool, considered to be the first library recommender system. CrossRec is a more recent approach, which has been shown to outperform LibRec. Both tools have the replication package gratefully made available by their authors, allowing us to fully make use of the available source code and tailor it to the study needs.

3.1 Problem statement

We consider a scenario in which, when developing new software, programmers rely on a recommender system to find and use in their code third-party libraries that offer the desired features. For instance, the developer is searching for libraries for parsing JSON documents and the system should be able to suggest those that are used by similar projects. We simulate an attacker attempting to inject a phoney library with malicious code (e.g., by referring to the previous example, a malicious JSON parser), let it be lib^*, into the active projects. Given that, under normal circumstances, the systems would never recommend lib^*, since the library has not been invoked anywhere. To this end, we forcefully favor lib^* using push attacks (cf. Section 2.1), so that, in the end, the library will be suggested to projects.

While in many cases experienced developers would simply rely on well-known libraries, sometimes a malicious library, which is properly documented (i.e., providing useful features), and (artificially) upvoted, may still be adopted by some. Moreover, while our evaluation in this paper is limited to library recommendation, a similar attack scenario could happen for recommenders that suggest code snippets, e.g., MUSE [17], PAM [8], or FOCUS [22].

The name lib^* has been chosen for illustration purposes only, so as to facilitate the reading. In practice, to avoid being detected, an attacker probably gives the library a copycat name, i.e., one that closely resembles a popular library, aiming to disguise it better. In fact, typosquatting has been reported in recent work [6], and this indeed suggests an evident threat to the adoption of third-party libraries. For instance, a malicious library has been named as "*jeIlyfish*" to deceive developers into believing it "*jellyfish*," the authentic library.[1] In this way, developers would adopt the disguised library without hesitation, once it has been suggested by the recommendation engine.

Summary. Altogether, this is to stress that the risk of being fooled by malicious libraries in practice is present. As such, software developers who use recommender systems may suffer if the issue of dealing with this type of attack is not adequately studied.

3.2 Study design and dataset

Both LibRec and CrossRec suggest to active projects, i.e., the ones under development, by searching for libraries from the most similar projects in the training data [21].

[1]https://bit.ly/394uETI

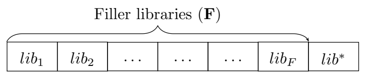

Figure 1: Manipulation of library usages for a fake project.

When LibRec and CrossRec learn from existing projects to provide recommendations, they treat a project's library usage as a vector, where each entry corresponds to a library: 1 means that the library is included in the project, 0 otherwise [21, 31]. To attack a recommender system, there are various approaches to populate ratings for fake user profiles [3, 16]. Unfortunately, they cannot be directly applied to forge projects, since they deal with multi-level ratings, while in library recommendation there are only two *ratings* (0 and 1). Thus, to create a training set for the LibRec and CrossRec tools, we have to devise a new method to produce fake projects as follows. A project's vector is divided into two independent parts, namely *filler libraries* and *target library* (cf. Fig. 1). We fill in the former with a set of libraries from training data; while the latter is set to lib^*. The manipulation aims to achieve two goals: *(i)* making the fake project be similar to active projects; and *(ii)* boosting the presence of lib^*.

We come across the following question: "*Which libraries should be chosen as fillers, so that the fake project will be incorporated into the recommendation?*" In fact, recommenders heavily rely on similarity [29], thus we come up with the selection of popular libraries as fillers. Our intuition is that, a fake project with libraries widely used by other projects is likely to get attention from the recommendation engine. A list of libraries in the training projects is populated and sorted in descending order of the number of caller projects. A certain number of the most popular libraries is then randomly selected.

On the attacker's side, α is the ratio of fake projects to the total number of projects (in %); γ is the number of fillers. For LibRec and CrossRec, k is the number of similar projects and N is the cut-off value for the ranked list of recommendations [21, 31]. We conducted experiments using ten-fold cross-validation on a dataset crawled from GitHub consisting of 1,400 projects and 13,497 libraries. For a testing project, we randomly split its libraries into two equal parts, the first part is used as a query, while the second one is ground-truth data. To measure the effectiveness of the push attacks, we use *Hit ratio HR@N*, defined as the ratio of projects being recommended with lib^* to the total number of testing projects [1]. To measure the accuracy, we use success rate *SR@N*, i.e., the ratio of projects getting at least a matched library (the size of the intersection of the recommended list with the ground-truth data is larger than 0) to the number of testing projects.

We experimented with α={5%, 10%} as attack size; k={5, 10, 15, 20}; γ={3, 4, 6, 8, 10} as fillers' size. The calibration of these parameters allows us to simulate *stress tests*, aiming to investigate the systems' resilience under different working conditions.

The following research questions are addressed to study the performance:

- **RQ$_1$**: *Are LibRec and CrossRec prone to push attacks?* By experimenting the two systems on a real dataset collected from GitHub, we are interested in understanding if the push attacks pose a threat to them.

Table 2: Hit ratio @N for LibRec.

		HR@10					HR@15				
		γ=3	γ=4	γ=6	γ=8	γ=10	γ=3	γ=4	γ=6	γ=8	γ=10
α = 5%	**k=5**	—	0.154	0.177	0.242	0.219	—	0.154	0.189	0.252	0.237
	k=10	—	0.198	0.273	0.428	0.410	—	0.198	0.317	0.455	0.442
	k=15	—	0.199	0.307	0.541	0.546	—	0.199	0.368	0.580	0.580
	k=20	—	0.177	0.316	0.608	0.637	—	0.177	0.388	0.658	0.670
α = 10%	**k=5**	—	0.247	0.240	0.242	0.210	—	0.247	0.256	0.251	0.237
	k=10	—	0.380	0.418	0.428	0.320	—	0.380	0.449	0.455	0.442
	k=15	—	0.439	0.505	0.541	0.390	—	0.439	0.554	0.580	0.580
	k=20	—	0.443	0.547	0.608	0.431	—	0.443	0.605	0.657	**0.670**

Table 3: Hit ratio @N for CrossRec.

		HR@10					HR@15				
		γ=3	γ=4	γ=6	γ=8	γ=10	γ=3	γ=4	γ=6	γ=8	γ=10
α = 5%	**k=5**	0.159	0.158	0.145	0.113	0.103	0.178	0.177	0.163	0.138	0.120
	k=10	0.140	0.141	0.165	0.149	0.140	0.184	0.190	0.201	0.185	0.174
	k=15	0.154	0.163	0.188	0.198	0.189	0.200	0.227	0.235	0.250	0.229
	k=20	0.112	0.135	0.207	0.188	0.194	0.168	0.191	0.268	0.235	0.250
α = 10%	**k=5**	0.356	0.346	0.267	0.235	0.210	0.386	0.373	0.291	0.265	0.248
	k=10	0.440	0.430	0.348	0.347	0.320	0.496	0.478	0.388	0.393	0.357
	k=15	0.427	0.445	0.427	0.407	0.390	0.487	0.497	0.488	0.475	0.438
	k=20	0.455	0.424	0.457	0.454	0.431	0.515	0.525	**0.530**	0.514	0.486

- **RQ$_2$**: *How do α and γ affect the attacks' effectiveness?* Both parameters can be tuned by attackers. Thus, it is also important to see how the parameters impact on the efficacy of the attacks. By varying the parameters, we study the corresponding influence on the recommendation outcomes of both systems.

3.3 Experimental results

We answer the research questions by referring to Table 2 and Table 3. LibRec does not work on projects with a small number of libraries, this is why there are no results for γ=3. Instead, we can run CrossRec with any $\gamma > 0$.

3.3.1 ***RQ$_1$****: Are LibRec and CrossRec prone to push attacks?* According to Table 2, we see that regardless of the internal configuration of both systems (i.e., k and N), the push attacks always succeed in injecting lib^* to projects. In general, an attack becomes more effective when a larger value of k is used. We analyze the results obtained by running the tools as follows.

With α=5%, for LibRec, we can see that the system recommends the malicious library at different hit ratios. For instance, with γ=4, k=5 we get HR@10 of 0.154, and the hit ratio gradually increases when we use a larger number of libraries. Given that 10 libraries are used, HR@10 reaches 0.219. Moreover, the hit ratio also increases alongside k, i.e., the number of neighbor projects used for recommendations. Remarkably, when k=20, we get a hit ratio HR@10 of 0.637. Similarly, CrossRec also recommends the malicious library by all the experimental settings.

As we have shown in Section 3.1, developers would voluntarily adopt the recommended libraries, especially when the fake library is disguised with a typosquatting name, and the ranked list is short, e.g., N=10 or N=15. The adoption of a phoney library indeed poses a threat to the software system under development. Altogether, we conclude that adversarial attacks to library recommender systems are highly probable, and we should neither underestimate nor neglect them.

Answer to RQ$_1$. Even with a simple fabrication, the resilience of LibRec and CrossRec is considerably compromised. Both systems recommend to developers the malicious library, which can put software clients at risk once being invoked.

3.3.2 ***RQ$_2$****: How do α and γ affect the attacks' effectiveness?* Adversaries pay attention to α and γ, which are under their control and can be tuned to make attacks more effective. The results in Table 2 and Table 3 show that by adding more fake projects, one can increase the hit ratio by both systems.

For example, with LibRec, for (α=5%, k=10, γ=6) HR@10 is 0.273; and for (α=10%, k=10, γ=6) the corresponding hit ratio HR@10 is 0.418. Similarly, with CrossRec (cf. Table 3), for (α=5%, k=20, γ=6), HR@10 is 0.207, while the hit ratio HR@10 for α=10% reaches a value of 0.457. The maximum hit ratio is obtained, i.e., HR@15 = 0.530 when γ=6 and k=20.

For CrossRec, the hit ratio decreases when $\gamma > 6$, this means that an attacker should use a small γ to get a higher hit ratio. For LibRec, the maximum HR@15 is 0.670 and obtained when γ=10.

We ran LibRec and CrossRec in two modes, i.e., with and without fake projects, and realized that there is almost no difference in their success rate. Due to space limit, we report only the differences as average *SR@N* in two modes, which is 2.6% and 2% for LibRec and CrossRec, respectively.

This implies that the inadvertent inclusion of fake projects in the recommendation is difficult to notice, and cannot be detected by simply looking at variations in the accuracy values.

Answer to RQ$_2$. By adding more fake projects (α), attackers can increase the number of infected clients in both systems. A small γ is more dangerous to CrossRec, while a large γ causes more harm to LibRec. The introduction of the malicious library does not greatly impact the recommendation accuracy, thus being an imperceptible incident.

3.4 Discussions

This subsection discusses the perspective of an attacker who wants to *compromise the security of a system (marked with ✗)*, and that of an administrator who *defends the system against hostile attempts (marked with ✔)*.

While in the evaluation we performed only push attacks, the same methodology can be applied to conduct nuke attacks, i.e., removing a useful library from the recommendations (✗). Moreover, we tried with only one malicious library, however the evaluation can easily be extended to a set of libraries.

We considered different settings for LibRec and CrossRec, and this aims at studying the systems' capability in various conditions. In practice, an attacker may not be aware of such internal design, what matters is how she populates fake projects so that the malicious library will be recommended (✗), regardless of the systems' configurations.

To simplify the evaluation, we generated fake projects at the metadata level, i.e., we assume that all projects have been successfully fetched from GitHub.[2] In practice, it is necessary to plant these projects in GitHub to make them appear as legitimate (✗). In fact, such an incident has been recently reported, where hundreds of GitHub repositories promoting malware apps have been discovered.[3] To our knowledge, research conducted to combat this type of abuse is still in its infancy [26].

Aiming for credible sources, a recommender usually chooses to crawl only from repositories with a considerably large number of stars and/or forks, or whatever criterion a recommender uses for selecting its training set (✔) [21]. In this case, an attacker may need to disguise a planted project by falsifying this information, e.g., by starring and forking with forged accounts (✗). Thus, there is a need for mechanisms to detect this type of fabrication (✔).

To penetrate a system, the introduction of a library is just the first step. The second step is to trick developers into invoking the library by calling its functions. Thus, an adversary needs also to steer API or code recommenders (✗), e.g., the ones recently published [8, 22, 28]. We conjecture that a technique similar to what described in Section 3.2 can be used for injecting toxic APIs.

4 RELATED WORK

The topic of AML has garnered attention from the research community in recent years. In image classification, several adversarial techniques have been introduced to add perturbations to images [9, 20], which are then correctly recognized by humans. However, DNNs are prone to these attacks as they fail to classify the crated images.

[2]We mined library usage directly from *pom.xml* files.
[3]https://zd.net/3bg3CK9

For recommender systems, there are various studies about AML [7, 11, 16]. Recently, Anelli *et al.* [1] present a very first work on using shilling attack [30] to manipulate a collaborative-filtering recommender system operated with Linked Data. In our work, we performed similar push attacks on CrossRec which runs with data from GitHub. However, we need to devise our own method to curate fake projects.

Wang and Han [34] propose an improvement of the Bayesian personalized ranking (BPR) technique by exploiting the adversarial training-based mean strategy (AT-MBPR) in the domain of collaborative filtering-based recommender systems. By exploiting user's ratings, the underpinning MBPR model extract and classify user's feedback into three categories. After, the algorithm injects adversarial perturbations in the initial training data to construct a noisy dataset. An iterative process then optimizes the objective function.

To defend against attacks, there have been several approaches [16]. An attack-agnostic detection model has been proposed [4] to support recommender systems based on reinforcement learning (RL). The model is based on an attention classifier that aims to separate adversarial examples from the normal ones by measuring the probability that input data suffer from perturbations introduced using a constructed attack model.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented an initial investigation of the topic of AML in RSSE. While RSSE suggesting libraries and API calls have gained momentum, we showed how they can be susceptible to adversarial attacks with bogus input data. By spoiling the tools' recommendation list, the attack paves the way for unauthorized access to software clients.

As future work, we plan to devise and evaluate effective countermeasures that can ward off attacks tailored to RSSE. This will be done by studying learning algorithms being aware of adversarial attacks and seize them. Moreover, adversarial attempts should be turned into features for the training process, i.e., RSSE should not only learn from useful patterns, but also be able to learn how to avoid hostile patterns. Besides recommenders leveraging GitHub, we plan also to investigate to what extent RSSE leveraging discussions from Q&A forums such as Stack Overflow (SO) are susceptible to adversarial attacks. For example, Prompter [24] leverages code and comments written by a developer in the IDE to recommend relevant posts from SO. Recommenders based on SO could be subject to attacks, if one hides malicious code in SO posts, though moderation and voting mechanisms might make such attacks more challenging. Another line of research we plan to investigate is related to online learning systems for API recommendation and code completion. We plan to assess to what extent this kind of recommender systems are vulnerable to adversarial attacks and identify possible countermeasures to mitigate related consequences.

REFERENCES

[1] Vito Walter Anelli, Yashar Deldjoo, Tommaso Di Noia, Eugenio Di Sciascio, and Felice Antonio Merra. 2020. SAShA: Semantic-Aware Shilling Attacks on Recommender Systems Exploiting Knowledge Graphs. In *The Semantic Web*. Springer International Publishing, Cham, 307–323.

[2] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable Third-Party Library Detection in Android and Its Security Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna,

Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 356–367. https://doi.org/10.1145/2976749.2978333

[3] Runa Bhaumik, Chad Williams, Bamshad Mobasher, and Robin Burke. 2006. Securing Collaborative Filtering Against Malicious Attacks Through Anomaly Detection. In *Proceedings of ITWP'06*. Held at AAAI 2006, Boston, Massachusetts. http://www.aaai.org/Press/Reports/Workshops/ws-06-10.php

[4] Yuanjiang Cao, Xiaocong Chen, Lina Yao, Xianzhi Wang, and Wei Emma Zhang. 2020. Adversarial Attacks and Detection on Reinforcement Learning-Based Interactive Recommender Systems. In *Proceedings of the 43rd ACM SIGIR*. ACM, Virtual Event China, 1669–1672. https://doi.org/10.1145/3397271.3401196

[5] Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard, and Jacqueline P. de Vries. 2010. Moving into a New Software Project Landscape. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1* (Cape Town, South Africa) *(ICSE '10)*. ACM, New York, NY, USA, 275–284. https://doi.org/10.1145/1806799.1806842

[6] Alexandre Decan, Tom Mens, and Eleni Constantinou. 2018. On the Impact of Security Vulnerabilities in the Npm Package Dependency Network. In *Proceedings of the 15th International Conference on Mining Software Repositories* (Gothenburg, Sweden) *(MSR '18)*. Association for Computing Machinery, New York, NY, USA, 181–191. https://doi.org/10.1145/3196398.3196401

[7] Yashar Deldjoo, Tommaso Di Noia, and Felice Antonio Merra. 2020. Adversarial Machine Learning in Recommender Systems: State of the art and Challenges. *ArXiv e-prints* (May 2020). http://www-ictserv.poliba.it/publications/2020/DDM20a/Survey_AML_RecSys.pdf Under Review.

[8] Jaroslav Fowkes and Charles Sutton. 2016. Parameter-free Probabilistic API Mining Across GitHub. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, New York, 254–265. https://doi.org/10.1145/2950290.2950319

[9] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6572

[10] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API Learning. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, New York, 631–642. https://doi.org/10.1145/2950290.2950334

[11] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. 2014. Shilling Attacks against Recommender Systems: A Comprehensive Survey. *Artif. Intell. Rev.* 42, 4 (Dec. 2014), 767–799. https://doi.org/10.1007/s10462-012-9364-9

[12] Q. He, B. Li, F. Chen, J. Grundy, X. Xia, and Y. Yang. 2020. Diversified Third-party Library Prediction for Mobile App Development. *IEEE Transactions on Software Engineering* (2020), 1–1.

[13] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. 2011. Adversarial Machine Learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence* (Chicago, Illinois, USA) *(AISec '11)*. Association for Computing Machinery, New York, NY, USA, 43–58. https://doi.org/10.1145/2046684.2046692

[14] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. 2014. The promises and perils of mining GitHub. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*. 92–101.

[15] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo. 2017. LibD: Scalable and Precise Third-Party Library Detection in Android Markets. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 335–346.

[16] B. Mobasher, R. Burke, R. Bhaumik, and J. J. Sandvig. 2007. Attacks and Remedies in Collaborative Recommendation. *IEEE Intelligent Systems* 22, 3 (2007), 56–63.

[17] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrian Marcus. 2015. How Can I Use This Method?. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. 880–890.

[18] Gail C. Murphy. 2009. Attacking information overload in software development. In *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2009, Corvallis, OR, USA, 20-24 September 2009, Proceedings*. 4.

[19] Emerson R. Murphy-Hill, Gail C. Murphy, and William G. Griswold. 2010. Understanding context: creating a lasting impact in experimental software engineering research. In *Proceedings of FoSER 2010, at FSE 2010, Santa Fe, NM, USA, November 7-11, 2010*. 255–258.

[20] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images.. In *CVPR*. IEEE Computer Society, 427–436. http://dblp.uni-trier.de/db/conf/cvpr/cvpr2015.html#NguyenYC15

[21] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. 2019. CrossRec: Supporting Software Developers by Recommending Third-party Libraries. *Journal of Systems and Software* (2019), 110460. https://doi.org/10.1016/j.jss.2019.110460

[22] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, Lina Ochoa, Thomas Degueule, and Massimiliano Di Penta. 2019. FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, Piscataway, NJ, USA, 1050–1060. https://doi.org/10.1109/ICSE.2019.00109

[23] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M. German, and Katsuro Inoue. 2017. Search-based Software Library Recommendation Using Multi-objective Optimization. *Inf. Softw. Technol.* 83, C (March 2017), 55–75. https://doi.org/10.1016/j.infsof.2016.11.007

[24] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2016. Prompter - Turning the IDE into a self-confident programming assistant. *Empirical Software Engineering* 21, 5 (2016), 2190–2231. https://doi.org/10.1007/s10664-015-9397-1

[25] Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann (Eds.). 2014. *Recommendation Systems in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. http://link.springer.com/10.1007/978-3-642-45135-5 DOI: 10.1007/978-3-642-45135-5.

[26] Md Omar Faruk Rokon, Risul Islam, Ahmad Darki, Evangelos E. Papalexakis, and M. Faloutsos. 2020. SourceFinder: Finding Malware Source-Code from Publicly Available Repositories. *ArXiv* abs/2005.14311 (2020).

[27] Mohamed Aymen Saied, Ali Ouni, Houari Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, and David Lo. 2018. Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software* 145 (2018), 164 – 179. https://doi.org/10.1016/j.jss.2018.08.032

[28] Anand Ashok Sawant and Alberto Bacchelli. 2017. fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage. *Empirical Software Engineering* 22, 3 (June 2017), 1348–1371. https://doi.org/10.1007/s10664-016-9444-6

[29] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. *Collaborative Filtering Recommender Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 291–324. https://doi.org/10.1007/978-3-540-72079-9_9

[30] Mingdan Si and Qingshan Li. 2020. Shilling attacks against collaborative recommender systems: a review. *Artif. Intell. Rev.* 53, 1 (2020), 291–319. https://doi.org/10.1007/s10462-018-9655-x

[31] Ferdian Thung, David Lo, and Julia Lawall. 2013. Automated library recommendation. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 182–191. https://doi.org/10.1109/WCRE.2013.6671293

[32] J. D. Tygar. 2011. Adversarial Machine Learning. *IEEE Internet Computing* 15, 5 (2011), 4–6.

[33] J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang. 2013. Mining Succinct and High-coverage API Usage Patterns from Source Code. In *10th MSR*. IEEE, Piscataway, 319–328. https://doi.org/10.1109/MSR.2013.6624045

[34] Jianfang Wang and Pengfei Han. 2020. Adversarial Training-Based Mean Bayesian Personalized Ranking for Recommender System. *IEEE Access* 8 (2020), 7958–7968. https://doi.org/10.1109/ACCESS.2019.2963316 Conference Name: IEEE Access.

[35] Yiming Zhang, Yujie Fan, Shifu Hou, Yanfang Ye, Xusheng Xiao, Pan Li, Chuan Shi, Liang Zhao, and Shouhuai Xu. 2020. Cyber-guided Deep Neural Network for Malicious Repository Detection in GitHub. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*. 458–465. https://doi.org/10.1109/ICBK50248.2020.00071

[36] Hao Zhong, Tao Xie, Lu Zhang, Jian Pei, and Hong Mei. 2009. MAPO: Mining and Recommending API Usage Patterns. In *23rd European Conference on Object-Oriented Programming*. Springer, Berlin, Heidelberg, 318–343. https://doi.org/10.1007/978-3-642-03013-0_15