Privacy-enabled Recommendations for Software Vulnerabilities

Karlsson, Linus; Paladi, Nicolae

Published in:
The 17th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2019)

DOI:
10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00111

2019

Document Version:
Peer reviewed version (aka post-print)

Privacy-enabled Recommendations for Software Vulnerabilities

Linus Karlsson
Dept. of Electrical and Information Technology, Lund University
Lund, Sweden
linus.karlsson@eit.lth.se

Nicolae Paladi
Dept. of Electrical and Information Technology, Lund University
Lund, Sweden
nicolae.paladi@eit.lth.se
RISE Research Institutes of Sweden

***Abstract*—New software vulnerabilities are published daily. Prioritizing vulnerabilities according to their relevance to the collection of software an organization uses is a costly and slow process. While recommender systems were earlier proposed to address this issue, they ignore the security of the vulnerability prioritization data. As a result, a malicious operator or a third party adversary can collect vulnerability prioritization data to identify the security assets in the enterprise deployments of client organizations. To address this, we propose a solution that leverages isolated execution to protect the privacy of vulnerability profiles without compromising data integrity. To validate an implementation of the proposed solution we integrated it with an existing recommender system for software vulnerabilities. The evaluation of our implementation shows that the proposed solution can effectively complement existing recommender systems for software vulnerabilities.**

1. Introduction

Modern enterprise software systems are increasingly complex. Organizations commonly use a plethora of software systems - running either in-house or in public clouds - running on hundreds or thousands of devices. A simple and carefully implemented software library may run in isolation for extended periods of time without maintenance. However, as more and more software applications are interconnected, they must be adapted to support new communication protocols, updated to process new types of input data, and patched to fix vulnerabilities. Software patching is costly, since it requires specialist human effort and must be done shortly after vulnerabilities are discovered - often during weekends and evenings - in order to minimize the risk of exploits. Since software vulnerabilities can be discovered on any layer of the software stack, the cost is compounded by the complexity of selecting and prioritizing the patches.

Information about software vulnerabilities is collected from several sources, such as public or private security information providers, security researchers or software vendors. Vulnerabilities discovered by the vendor of the respective software are often treated as software bugs and corrected in the regular release cycle. Other vulnerabilities, discovered by software users or security researchers, are either communicated directly to the vendors, publicly released, or sold as zero-day exploits. Publicly known vulnerabilities are assigned a common vulnerability and exposure (CVE) identifier, along with high-level attributes such as: perceived impact on confidentiality, integrity, and availability; attack complexity; attack vector; and required privileges. With new vulnerabilities published daily, software users must review the relevant vulnerabilities, understand their impact and prioritize them. This makes prioritizing software vulnerabilities a stringent issue for both software vendors and software users.

Recommender systems [1] were earlier proposed as solutions for vulnerability prioritization [2]–[4]. However, *vulnerability profiles*, i.e. the priority ranking of vulnerability entries, reveal valuable information about the software libraries, packages, applications and protocols that are part of the user's *software bundle*, i.e. the collection of software applications, services and protocols an organization uses. Access to this information allows an adversary to build an intimate profile of an organization's internal systems - including specific software packages, versions and configurations. This enables highly effective, targeted attacks using zero-day exploits. While the recommender systems proposed earlier address to some extent vulnerability prioritization, they do not implement mechanisms to protect the vulnerability profiles. We address this by describing the design, implementation and evaluation of a privacy-preserving mechanism for software vulnerability recommender systems. We leverage commodity isolated execution environments to protect vulnerability profiles using an approach derived from earlier work on differential privacy [5], yet without compromising the integrity of vulnerability data. Our contribution is as follows:

- We describe a privacy protection mechanism to protect client vulnerability profiles; to the best of our knowledge, this is the first work addressing the privacy of client vulnerability profiles.
- we describe the implementation and evaluation of a prototype solution, implemented using Intel SGX enclaves and tested with a functioning vulnerability recommender system.

The remainder of this paper is organized as follows. We

introduce the background, including the system model and the threat model in section 2. We discuss the solution space for protecting vulnerability profile privacy in Section 3, describe the implementation in Section 4 and present the evaluation results in Section 5. Finally, we discuss the related work in Section 6 and conclude in Section 7.

2. Preliminaries

We follow earlier work and define *software vulnerabilities* as exploitable software flaws in software systems that pose security risks [2], [6]. Frei et al. describe several distinct phases of the vulnerability life-cycle: creation, discovery, exploit availability, public disclosure, patch availability, and patch installation. The life-cycle events are grouped into three risk exposure phases: pre-disclosure, post-disclosure, and post-patch [6]. In the pre-disclosure phase, neither vendors nor users can affect the impact of an externally discovered security vulnerability. Once a vulnerability is disclosed (either by the vendor itself or externally), vendors and users can develop patches to correct the vulnerability, or alternatively mitigate it through workarounds. In the scope of this paper, we consider software vendors as the only legitimate source of software patches. Considering the large number of software vulnerabilities released on a daily basis, assessing the impact of a vulnerability on a particular software bundle is a tedious task. While the community proposed a variety of approaches for vulnerability risk assessment [7] they are often ad-hoc, do not scale, and are not sufficiently robust to reflect the complexity of enterprise environments.

Automating vulnerability prioritization can reduce the duration and the risk exposure of the post-disclosure phase. Rapid prioritization of software vulnerabilities can both speed up patch development and patch installation. However, despite increasing productivity, software vulnerability recommenders pose data privacy risks as they may leak client vulnerability profiles, including information about software vulnerabilities that users and vendors consider of primary interest. We address this by proposing an approach to protect client vulnerability profiles in software vulnerability recommenders. We next describe the system model of automated vulnerability prioritization followed by the treat model we consider in this work.

2.1. Automating vulnerability prioritization

Automatic tools to provide software vulnerability recommendations have been described earlier [3], [4]. Such tools help enterprise customers with recommendations and are often provided in a Software-as-a-Service (SaaS) delivery model. A key component in this scenario is that the service provider must have knowledge both about the customers' software bundles and of customer data such as user preferences from historic interactions with the system.

We consider the scenario described below. The service provider offers two distinct features: *vulnerability identification*, and *vulnerability prioritization*. The output of vulnerability identification is a list of vulnerabilities, encoded as CVE identifiers. In the following step, the vulnerability prioritization step, the list of CVE identifiers is passed to a recommender, which then ranks the list of CVEs according to user preferences. Output from the identification phase can be obtained from various sources [3], [4], [8]. In this paper we focus on the vulnerability prioritization phase.

Clients require a profile shared with a service provider in order to enable the recommender system to provide personalized ranking in the prioritization phase. This profile contains the client's individual preferences, including client considerations while assessing the severity of a vulnerability. We consider the scenario illustrated in Figure 1. A client requests recommendations for a given set of CVE identifiers from the recommender system. To get personalized recommendations, the client also attaches its client profile p to the request. The recommender returns a personalized ranking r of the given vulnerabilities. Note that in this scenario, the recommender can trivially map a client to its profile; there is no data privacy.

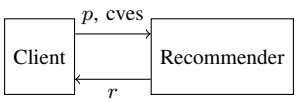

Figure 1. Vulnerability prioritization: a client requests recommendations for a set of CVEs, using client profile p

2.2. Isolated execution

In this paper we propose a mechanism to protect the privacy of client requests to the recommender. This mechanism requires confidentiality of the data, requirements that can be satisfied using isolated execution. We use SGX enclaves [9]–[12] to create commodity trusted execution environments (TEEs) during operating system run-time. We use TEEs to protect the privacy of client vulnerability profiles. SGX enclaves rely on a trusted computing base of code and data loaded at enclave creation time, processor firmware, and processor hardware. Program execution within an enclave is opaque to the underlying operating system and other mutually distrusting enclaves on the platform. Enclaves operate in a dedicated memory area called the Enclave Page Cache, a range of dynamic random access memory that cannot be accessed by system software or peripherals [10]. The CPU firmware and hardware are the root of trust of an enclave. Isolation features implemented in firmware and hardware prevent access to the enclave's memory by the operating system and other enclaves. While we use Intel SGX enclaves in our implementation of the privacy-preserving service, alternative commodity TEEs [13], [14] may be used.

2.3. Threat Model

To construct a correct and relevant threat model in the scenario described above, we conducted in-depth interviews with software vendors and users that operate enterprise deployments. Software vendors highlight the importance of

protecting information about relevant software vulnerabilities during the pre- and post-disclosure phases, i.e. while assessing the vulnerability impact and prioritizing it, as well as before releasing a software patch (see Figure 2). We aim to protect the confidentiality of the vendor's software vulnerability priority ranking, since it reveals information about the severity of the vulnerability (as perceived by the vendor) and can be used to guide exploit development. We assume clients have no information about unpublished vulnerabilities. Hence, we must protect the confidentiality of the relevance and priority of the released security patches during the post-disclosure (denoted as t_{pd}) and post-patch (denoted as t_{pp}) phases, i.e. up to the point when software users patch their systems.

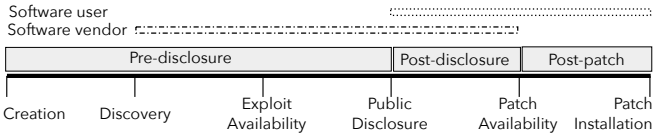

Figure 2. Vulnerability confidentiality window

We assume the duration of the post-disclosure and post-patch phases is constant, equal to T:

$$t_{pd} + t_{pp} = T \tag{1}$$

The proposed architecture contains three main components - Client, Intermediary and Recommender. We describe the security assumptions about the components of the proposed architecture.

Client. The Client is potentially malicious, and may submit queries containing requests for arbitrary CVE identifiers. The goal of the malicious Client is to reveal the privacy-preserving algorithm implemented in the Intermediary and distinguish the vulnerability profiles of other Clients from the vulnerability profiles generated by the Intermediary. A set of Clients may collude to achieve this purpose.

Recommender. The Recommender is an *honest-but-curious* entity that aims to reveal the vulnerability profile of a specific Client. The Recommender is assumed to have access to powerful computation capacity and is not sensitive to changes in the computation load of the queries received from the Client. A collusion between the Client and the Recommender reveals the vulnerability profile of the colluding Client and is out of the scope of this model.

Intermediary. The Intermediary is potentially malicious and aims to reveal the vulnerability profile of a specific Client. A collusion between the Client and the Intermediary reveals the vulnerability profile of the colluding Client and is out of the scope of this model. However, we consider that the isolated execution enclave that hosts the privacy protection mechanism is trustworthy; its trustworthiness can be established through remote attestation by any of the components participating in the protocol. Isolated execution enclaves used in the implementation of the Intermediary are vulnerable to side-channel attacks [15]; we explicitly exclude side-channel attacks from this threat model, since they can be mitigated through improved software implementation.

We consider a remote adversary with attack capabilities on the network and platform level. The adversary is capable to observe and interact with the network communication between Client and the Recommender, as well with the communication among the internal services of the Recommender. Furthermore, the adversary can launch arbitrary processes and obtain root access on the hosts running the Recommender.

3. Vulnerability Profile Privacy

As discussed in Section 2.3 above, our goal is to protect the privacy of individual Client vulnerability profiles. To design a solution for this task, we consider the following defining aspects: *(i)* Client profile updates are sparse (e.g. one request every period T); *(ii)* vulnerability profiles evolve after every period T; *(iii)* the value of an arbitrary snapshot of the vulnerability profile decays in time T, equal to the length of software user's confidentiality window. These are aspects that are sound in the context of vulnerability prioritization, but may not hold for other recommender systems.

While a myriad of definitions, approaches and techniques have been proposed for privacy protection, it remains an elusive goal. One reason is the discrepancy between the assumptions required for a solution to offer data privacy protection in the presence of other related data sets. Another reason is the trade-off between data privacy and data utility present in most privacy-preserving solutions [5], [16].

To protect the privacy of Client vulnerability profiles, we design an approach based on a combination of k-anonymity [17], [18] and local differential privacy [19]–[21]. A combination of the two approaches guarantees that queries derived from Client vulnerability profiles are released only in large enough batches that contain additional queries derived from statistically different pseudo vulnerability profiles. This approach prevents the direct release of information about subgroups and provides differential privacy guarantees by adding pseudo-random noise (in the form of fake queries) to the genuine queries based on individual Client vulnerability profiles.

This approach, adapted to the vulnerability recommender system described in [4], allows us to substitute the privacy-utility trade-off with a privacy-cost relation. Intuitively, larger proportions of noise accompanying genuine queries lead to stronger privacy of the respective Client vulnerability profile. While increasing computational cost of the recommender, this allows to proportionally decrease the utility of information observed by the adversary, without utility loss for the client.

3.1. k-anonymous vulnerability profiles

We next describe the mechanism for protecting the anonymity of queries derived from Client vulnerability pro-

files. Our approach is based on two fundamental aspects: *(1)* issuing queries in large enough batches, and *(2)* mixing queries derived from genuine Client vulnerability profiles (called *genuine queries)* with pseudo-random noise expressed as queries derived from statistically different pseudo vulnerability profiles (called *pseudo queries*).

We next present the approach of issuing a single genuine query per batch. Each Client vulnerability profile consists of exactly n different properties, where each property describes the Client's preference for a certain aspect of a vulnerability. Preferences are represented as a vector p describing the Client profile, where each individual property is denoted v_i.

$$p = \{v_1, v_2, \ldots, v_n\} \tag{2}$$

For example, a Client vulnerability profile may be built by ranking three different vulnerability properties: confidentiality impact, integrity impact, and availability impact. A client preference could be that the Client considers that vulnerabilities affecting confidentiality have higher priority, while vulnerabilities with an impact on integrity or availability have lower priority. A profile is then represented as: $\{0.9, 0.2, 0.2\}$, where a higher value v_i implies that a property is considered more important.

To reduce the utility of information disclosed to the adversary through queries, the pseudo queries should fulfill two requirements: *(i)* introduce sufficient noise regardless of the cardinality of the profile $|p|$ and of the values of the elements v_i in the profile; *(ii)* have the same dimensions and be indistinguishable from the genuine query. Different from database anonymization approaches [17], [18], [22], low variability in the data set does not satisfy our privacy requirements, since it leaks Client vulnerability profile data. Therefore we add a third condition that the pseudo queries must satisfy, along with conditions *(i)*, *(ii)* above: *(iii)* the query data set should display high variability (or high standard deviation in statistical terms).

Thus, on every period T, the Client submits a query Q_g to the Intermediary. The Intermediary derives $k-1$ pseudo queries Q_p that along with Q_g are submitted to the Recommender. The Recommender processes all of the queries and returns k different responses based on the received queries. As a result, neither the adversary observing the network, nor the potentially malicious Recommender are aware which of the k (profile, response)-pairs belongs to the Client. Upon receiving k replies the Intermediary discards all pseudo queries Q_p and returns to the Client the reply to query Q_g. The variability of the k profiles is sufficiently large to make it unfeasible to draw any conclusions about the contents of the genuine profile. The size $k-1$ is configurable, depending on the computational cost acceptable for the Recommender and the privacy guarantees requested by the Client.

Having described the Client vulnerability profile anonymization principles, we next discuss the use of isolated execution to provide platform security guarantees of the Intermediary.

3.2. Privacy protection with isolated execution

The privacy-preserving mechanism described above can be implemented by any component of the proposed architecture: Client, Intermediary, or Recommender (see Section 2.3). A straightforward client implementation is possible; however, this places a large load on the client and generate excessive network traffic. Alternatively, a client may choose to instead send only a single request to the intermediary. The intermediary is then responsible for sending multiple requests to the Recommender. From the Client's point of view, a *trustworthy* intermediary allows to reduce external network traffic, since the intermediary can be placed close to the recommender system. The privacy-preserving mechanism can also be implemented by the Recommender provider itself, thus reducing the network latency between the Intermediary and the Recommender to near zero. However, this is incompatible with the honest-but-curious Recommender described in Section 2.3.

Trustworthy intermediary functionality can be implemented using the functionality of a TEE. The TEE should support remote attestation, isolated execution, and sealing of sensitive data. In the description below, a solution designed on Intel SGX is described, but we stress that alternative commodity TEEs may be used instead. We consider a scenario with the following requirements:

- A malicious Intermediary or Recommender system should not be able to know if a certain client profile is the genuine client profile for a particular Client.
- The Client should be able to attest the integrity of the Intermediary before sending sensitive data.
- The Recommender should be oblivious to the privacy-preserving measures taken by the Intermediary: changes to the privacy preserving mechanism in the Intermediary should not require any modifications of the Recommender.

We propose a solution as depicted in Figure 3. The Client starts by attesting the integrity of the Intermediary through remote attestation. During this stage, the Client receives proof that the TEE of the Intermediary has not been tampered with, and negotiates a shared secret S, which can be used to send encrypted data to the TEE. The key S is confined within the TEE and cannot be read from the outside.

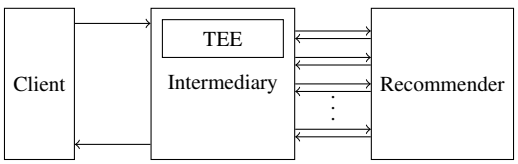

Figure 3. Proposed design with an Intermediary running in a TEE

Next, the Client either updates its client profile, or requests recommendations from the Recommender. In both cases, the Client sends a single request to the Intermediary, encrypted with the shared secret S. The TEE of the Intermediary can then decrypt the request.

For a request to update the profile, the TEE generates $k-1$ new pseudo profiles, to be used as described in Section 3.1.

For a request for recommendations, the TEE looks up k different profiles, of which only one is the genuine profile. These profiles are then used to generate k queries to the Recommender. All k responses are then returned to the Intermediary. The Intermediary next sends all results into the TEE, which picks the result corresponding to the genuine client profile. This result is encrypted with S and sent back to the Client, where it can be decrypted. This design achieves the following properties:

- The decrypted data with the genuine client profile is never available alone to the Intermediary *outside* the TEE. Only the collection of k profiles is available, and it is not possible to distinguish the genuine profile from pseudo profiles.
- Following the definition of a TEE, an entity on the outside can never read data inside the TEE.
- If the adversary modifies the TEE, the attestation phase will fail, and no sensitive data will be sent to the TEE.
- The Recommender remains unaware of the Intermediary and simply receives extra requests made with several different profiles.

Together, these properties fulfill the requirements described earlier in this section. We will next describe implementation details in Section 4.

4. Implementation

To evaluate our proposed design and demonstrate its viability, we implemented a proof-of-concept prototype. The implementation uses Intel SGX to provide the TEE of the Intermediary, and was evaluated with hardware support for Intel SGX.

The implementation includes three major components: Client, Intermediary, and Recommender, (see Figure 3). In this implementation our focus is the Client and the Intermediary. We assume that the Recommender is already implemented, but without any privacy-enabling technologies, as explained in Section 2.1 and [4]. The three main features that the implementation must support are remote attestation, profile management, and recommendation generation. We next describe their implementation details.

4.1. Remote attestation

The first step before the Client can trust the Intermediary is to attest the Intermediary's TEE, in this case an SGX enclave. The remote attestation implementation uses the suggested design from Intel [23], using a modified Sigma protocol to derive a shared secret in the attestation phase.

We next briefly describe the procedure (see [23] for more details). First, the Client starts the attestation process by contacting the Intermediary, which initiates the remote attestation process inside the enclave. The Client proceeds by retrieving a signature revocation list from Intel Attestation Services (IAS), and sends this to the enclave together with other data. The enclave proceeds by returning a quote, which can then be verified by the Client. This verification is done by first contacting IAS, to verify that the quote is made by an enclave on trusted hardware. After this, the hash value of the enclave's code can be read from the quote. If this matches the expected value, the Client can be certain that the enclave has not been tampered with.

After this point, both the enclave and the Client has a shared secret S that can be used to secure further communications. Note that this secret is only available to the Client and the enclave, *not* the Intermediary outside the enclave.

4.2. Profile management

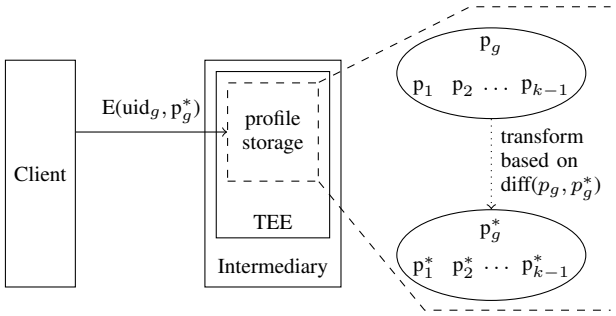

Figure 4. Enclave profile management during profile update

The profile management is located within the trusted enclave. This ensures that its behavior can be verified by the Client, such that it does not leak information to an attacker.

Each client profile p_i, has a corresponding id uid$_i$, used in communication between the different entities. Furthermore, for each *genuine* client profile p_g, there are $k-1$ *pseudo* profiles. Using a different set of pseudo profiles for each genuine profile ensures that colluding clients cannot find pseudo profiles for other clients. The profile management keeps a record over the mapping between genuine and pseudo profiles, such that the same set of pseudo profiles is used during profile update or recommendation generation for a specific genuine profile.

As the user's preferences change, the profile stored in the enclave needs to be updated. An overview of this is shown in Figure 4. During the profile update stage, the Client sends an encrypted updated genuine profile p^*_g to the Intermediary. The profile is decrypted inside the enclave, which then applies a transformation function as described below. The transformation is applied to each one of the $k-1$ pseudo profiles. This ensures that an outside observer can only see that all profiles have been updated, but still cannot know which one that is the genuine profile.

There are two main events in the life cycle of pseudo profiles: the initial pseudo profile generation, and updates of the pseudo profile. First, when a new genuine profile is created for the first time, $k-1$ new pseudo profiles

must also be created. Based on requirements listed in Section 3.1, the pseudo profiles should be indistinguishable from a genuine profile. To achieve this during initial pseudo profile generation, we select a profile such that its properties are distributed according the distribution of each property's value over all existing profiles, inspired by the work on t-closeness [24]. This ensures that the newly generated pseudo profiles is non-distinguishable from genuine profiles.

Second, when a profile should be updated, following the terminology from Figure 4, we want to implement a `diff()` function that updates the pseudo profiles based on the update of the genuine profile. This function should: *(i)* hide *which* property of the profile that was updated, and *(ii)* hide the exact *value difference* between the new and the old property. Without loss of generality, we can assume that during update of a genuine profile p_g to its new value p_g^*, only a single property v_i of the profile is modified[1].

To hide which property v_i that is updated, for every pseudo profile, we randomly select a property v_j from that profile ($1 \leq j \leq n$), whose value is updated. The result is that different properties are updated, and since an outside observer does not know the genuine profile, it is not possible to find out which actual property that was updated.

To hide the exact value difference between the old property v_i and the new property v_i^*, we suggest a solution similar to differential privacy [5]. While the genuine profile is updated to (the exact) new property value v_i^*, noise is added to the pseudo profile. The noise is based on the difference $v_i - v_i^*$, such that the exact value of the difference is hidden. The distribution from which to draw the noise may be varied, in our proof-of-concept we base it on the Laplace distribution commonly used in ϵ-differential privacy.

4.3. Recommendation generation

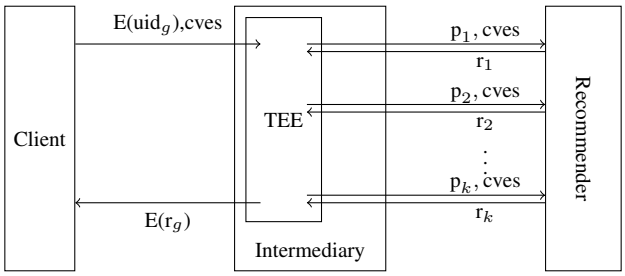

Figure 5. Recommendation generation

We illustrate the flow for recommendation generation in Figure 5. When the Client wishes to request recommendations, it sends a request to the Intermediary. The request contains an encrypted id (uid$_g$), and a list of CVEs to rank. The enclave decrypts the id, and returns k different profiles to the Intermediary, outside of the enclave. One of these profiles is the genuine client profile (p$_g$), but to the outside observer, all profiles are indistinguishable.

1. A single profile update modifying multiple properties can be converted to several consecutive updates, each modifying a single property.

For each of the k profiles, the Intermediary sends one request to the Recommender, which returns k different responses. The responses are forwarded to the enclave which selects only response r$_g$ corresponding to the genuine user profile p_g, encrypts it, and returns it to the Client.

An implementation must consider several aspects to avoid leaking information. First, the order of profiles must be randomized, such that the position of the genuine profile is not known. Second, even though the id and response is encrypted, the size of the ciphertext may still leak information, if different user profiles and responses from the Recommender have different sizes. It is therefore important for the enclave to ensure that all ids have identical size, as well as verifying that responses from the Recommender do not differ in size. In practice, this does not limit the functionality of the system: both the id and recommender response can be padded inside the enclave to ensure equal size. Note that since it is the communication between client and enclave that is padded, the Recommender does not have to be modified.

5. Evaluation

To evaluate the performance overhead of the proposed privacy-enabling mechanism, we measured the response time for recommendation generation.

Consider the setup in Figure 3, with each entity running on a different host, connected to the same local network. The Recommender is an actual implementation of a recommender as described in [4], and the Intermediary's TEE is on a CPU with hardware support for Intel SGX. We performed the following measurements. First, three random sets of CVEs were constructed, containing 30, 100, and 1000 different CVEs, respectively. Second, for each such set, we perform a test without the Intermediary as a baseline; in this test the Client connects directly to the Recommender, without any privacy protection. This can be used as a reference when comparing to the other measurements. Third, again for each set of CVEs, we performed five tests with different privacy levels, i.e. different values of k. Recall that k determines the number of profiles that are sent to the Recommender, so for e.g. $k = 8$, there is one genuine and seven pseudo profiles being sent to the Recommender. Each test was repeated 100 times, and the resulting mean, median, and standard deviation are presented in Table 1. Note that the baseline measurement, in which the Client connects directly to the Recommender, is denoted by k set to *none*.

The evaluation of the prototype implementation highlights the relation between the privacy guarantees and the response time of the recommender system. Requests to recommender systems for vulnerability prioritization are expected to be sparse and potentially asynchronous. Therefore, we consider the increase in response time detailed in Table 1 acceptable, considering the added benefit of data privacy.

TABLE 1. RESPONSE TIMES FOR RECOMMENDATION GENERATION FOR VARIOUS NUMBER OF CVES AND VARIOUS VALUES OF k

#CVEs	k	Mean (ms)	Median (ms)	St.dev (ms)
30	*none*	94.2	93.8	3.2
	1	106.7	106.4	7.2
	8	324.0	323.0	21.1
	16	589.3	583.4	21.0
	32	1117.6	1114.5	31.1
	64	2154.7	2159.6	32.6
100	*none*	122.0	121.6	2.6
	1	135.2	135.0	2.5
	8	378.0	371.7	33.0
	16	662.5	650.5	39.9
	32	1206.7	1198.0	49.9
	64	2281.7	2277.3	44.0
1000	*none*	442.6	441.6	5.5
	1	463.6	462.9	6.5
	8	813.4	777.5	108.6
	16	1536.7	1502.8	153.1
	32	2682.3	2661.1	136.1
	64	4881.5	4836.1	154.9

6. Related Work

In this paper we address the challenge of protecting the privacy of Client profiles in a vulnerability recommender service. While this topic was not addressed earlier, we base our approach on a rich body of privacy and anonymity research. We next review the related work.

6.1. Privacy for recommender systems

McSherry et al. described the design and implementation of a platform for privacy-preserving data analysis for SQL-like queries [25]. While this approach allows to write applications that provide privacy guarantees in an *honest-but-curious* threat model, it is not backward-compatible with existing applications and requires native implementations in the Privacy Integrated Queries platform. Abadi et al. described algorithmic techniques for learning and a refined analysis of privacy costs within the framework of differential privacy [26]. The approach is geared towards training deep neural networks with non-convex objectives. The solution enables this functionality under a modest privacy budget and at a manageable cost in software complexity and model quality. However, the approach is not suitable for the *honest-but-curious* threat model, since it relies on a trustworthy implementation of the framework on the data processing end. Our approach introduces a privacy-protection layer that is independent of the implementation of the data processing (in this case a recommender system) and can therefore be applied to a wide range of applications.

Ohrimenko et al. described an approach for oblivious multi-party machine learning on trusted processors [27]. The approach relies of a set of custom machine learning algorithms for trusted processors that make use of general-purpose oblivious primitives. For further security, the multi-party machine learning mechanism is implemented in trusted execution environments (namely Intel SGX enclaves). The Prochlo [28] implementation likewise uses Intel SGX enclaves to implement the the Encode, Shuffle, Analyze architecture for privacy-preserving software monitoring. The architecture is tailored for anonymizing data streams from many heterogeneous sources and allows to expose anonymized data to third parties. In this paper, we similarly rely on SGX enclaves to create trusted execution environments to run an implementation for query anonymization. We address a different use case, where the privacy of a single profile using the recommender system is preserved against an adversary capable to observe the queries and the internals of the recommender system.

6.2. Vulnerability selection

Vulnerability *detection* precedes and is closely related to vulnerability *selection*. In [29] the authors address the challenge of shifting vulnerability detection from a human-centric to a computer-centric approach. In particular, the paper presents a design and implementation for a human-assisted automated vulnerability analysis system. VULCON (VULnerability CONtrol) is a vulnerability management strategy described in [3]. It is based on two metrics, namely time-to-vulnerability remediation and total vulnerability exposure. Based on inputs such as vulnerability scan reports, metadata about the discovered vulnerabilities, asset criticality, and personnel resources VULCON prioritizes vulnerabilities for patching. Both vulnerability *detection* and vulnerability *selection* may require anonymity and privacy guarantees for the Client profiles in privacy-sensitive settings. Our work addresses this by describing a privacy protection mechanism for Client vulnerability profiles.

7. Conclusion

Automated vulnerability prioritization and patch selection become increasingly necessary in order to cope with the growing complexity of corporate software environments. Earlier research on recommender systems for vulnerability prioritization and patch selection did not address the privacy of client vulnerability profiles. In this work we presented a privacy-preserving mechanism that helps protect client vulnerability profiles in the context of recommender systems for vulnerability prioritization and patch selection; to the best of our knowledge, this is the first work addressing this aspect. We implement a prototype of the proposed solution using Intel SGX enclaves and a functioning recommender system for vulnerability prioritization and patch selection. Our evaluation of the prototype implementation reveals that the response time increases along with the proportion of pseudo queries issues with each request, but remains acceptable considering that requests are expected to be sparse. The evaluation result highlights that the proposed mechanism is practical and can complement existing recommender systems for vulnerability prioritization and patch selection.

Acknowledgment

This work was financially supported by the Swedish Foundation for Strategic Research, grant RIT17-0035.

References

[1] C. C. Aggarwal, *Recommender Systems*. Springer, 2016.

[2] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: Learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10. New York, NY, USA: ACM, 2010, pp. 105–114.

[3] K. A. Farris, A. Shah, G. Cybenko, R. Ganesan, and S. Jajodia, "Vulcon: A system for vulnerability prioritization, mitigation, and management," *ACM Trans. Priv. Secur.*, vol. 21, no. 4, pp. 16:1–16:28, Jun. 2018. [Online]. Available: http://doi.acm.org/10.1145/3196884

[4] A. Cobleigh, M. Hell, L. Karlsson, O. Reimer, J. Sönnerup, and D. Wisenhoff, "Identifying, prioritizing and evaluating vulnerabilities in third party code," in *IEEE 22nd International Enterprise Distributed Object Computing Workshop*. IEEE, 2018.

[5] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating Noise to Sensitivity in Private Data Analysis," in *Theory of Cryptography*, S. Halevi and T. Rabin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 265–284.

[6] S. Frei, D. Schatzmann, B. Plattner, and B. Trammell, "Modeling the security ecosystem-the dynamics of (in) security," in *Economics of Information Security and Privacy*. Springer, 2010, pp. 79–106.

[7] V. Sridharan and D. R. Kaeli, "Quantifying software vulnerability," in *Proceedings of the 2008 Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies*, ser. WREFT '08. New York, NY, USA: ACM, 2008, pp. 323–328.

[8] GitHub, "About security alerts for vulnerable dependencies," https://help.github.com/en/articles/about-security-alerts-for-vulnerable-dependencies.

[9] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. ACM, June 2013, p. 10.

[10] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative Instructions and Software Model for Isolated Execution," in *Proc. 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. ACM, June 2013, pp. 10:1–10:1.

[11] B. C. Xing, M. Shanahan, and R. Leslie-Hurd, "Intel Software Guard Extensions (Intel SGX) Software Support for Dynamic Memory Allocation Inside an Enclave," in *Proc. 2016 Hardware and Architectural Support for Security and Privacy*, ser. HASP '16. ACM, June 2016, pp. 11:1–11:9.

[12] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel Software Guard Extensions (Intel SGX) Support for Dynamic Memory Management Inside an Enclave," in *Proc. 2016 Hardware and Architectural Support for Security and Privacy*, ser. HASP '16. ACM, June 2016, pp. 10:1–10:9.

[13] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A Comparison Study of Intel SGX and AMD Memory Encryption Technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '18. New York, NY, USA: ACM, 2018, pp. 9:1–9:8.

[14] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "SANCTUARY: ARMing TrustZone with User-space Enclaves," in *Proc. 2019 Network and Distributed Systems Security Symposium*, ser. NDSS'19, 2019.

[15] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 2421–2434.

[16] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[17] P. Samarati and L. Sweeney, "Generalizing data to provide anonymity when disclosing information (abstract)," in *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ser. PODS '98. New York, NY, USA: ACM, 1998, pp. 188–.

[18] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, Nov 2001.

[19] R. Bassily, K. Nissim, U. Stemmer, and A. Guha Thakurta, "Practical Locally Private Heavy Hitters," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 2288–2296. [Online]. Available: http://papers.nips.cc/paper/6823-practical-locally-private-heavy-hitters.pdf

[20] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, "Privacy loss in apple's implementation of differential privacy on macos 10.12," *CoRR*, vol. abs/1709.02753, 2017. [Online]. Available: http://arxiv.org/abs/1709.02753

[21] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An Oblivious and Encrypted Distributed Analytics Platform," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 283–298.

[22] N. Holohan, S. Antonatos, S. Braghin, and P. M. Aonghusa, "(k, ϵ)-anonymity: k-anonymity with ϵ-differential privacy," *CoRR*, vol. abs/1710.01615, 2017. [Online]. Available: http://arxiv.org/abs/1710.01615

[23] J. M. (Intel), "Code sample: Intel software guard extensions remote attestation end-to-end example," July 2018, https://software.intel.com/en-us/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example.

[24] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *2007 IEEE 23rd International Conference on Data Engineering*, April 2007, pp. 106–115.

[25] F. McSherry and I. Mironov, "Differentially private recommender systems: Building privacy into the netflix prize contenders," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 627–636.

[26] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 308–318.

[27] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 619–636.

[28] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong Privacy for Analytics in the Crowd," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 441–459.

[29] Y. Shoshitaishvili, M. Weissbacher, L. Dresel, C. Salls, R. Wang, C. Kruegel, and G. Vigna, "Rise of the hacrs: Augmenting autonomous cyber reasoning systems with human assistance," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 347–362.