Poster: Reducing Data Leakage on Personal Data Management Systems

Robin Carpentier, Iulian Sandu Popa, Nicolas Anciaux

Poster: Reducing Data Leakage on Personal Data Management Systems

Robin Carpentier
Univ. Versailles St-Q.-en-Yvelines, France
robin.carpentier@uvsq.fr

Iulian Sandu Popa
Univ. Versailles St-Q.-en-Yvelines, France
iulian.sandu-popa@uvsq.fr

Nicolas Anciaux
Inria Saclay, France
nicolas.anciaux@inria.fr

1. Introduction

Over the past decade, successive steps have been taken to empower individuals with new legal and technical means, from smart disclosure initiatives to the right to data portability of the GDPR and the new notion of data altruism enacted in the EU [5]. The ultimate goal is to enable individuals to collect and share their personal information, for their own good and broader societal benefits, unlocking innovative usages when crossing multiple data sources from one or many users. The technical corollary of this movement is the emergence of personal data management systems (PDMS) that allow individuals to assemble their personal data under their control, with products such as Digi.me, Cozy Cloud or Solid/PODS, as well as initiatives like Mydata.org, supported by data protection agencies.

A viable PDMS solution needs to fulfill two major (but conflicting) objectives: security and extensiveness. Indeed, given the particular ecosystem that the PDMS paradigm creates by concentrating all personal data on the user's side, security is of paramount importance. PDMS must protect data from typical privacy threats (e.g., data snooping and secondary uses of data). As well, an important specificity of PDMS is that applications "move" to the data, as opposed to personal data migrating to remote services –as in existing cloud services– which also raises security concerns since leakages of personal data by malicious applications on end-users devices are common.

Extensiveness is equally important, as a viable PDMS solution should cover all major functionalities of the personal data life cycle, from collection to storage, retrieval, individual and collective exploitation of personal data. Extensiveness should also allow individuals to benefit from novel and advanced data-driven processing. Big personal data computing lies in the cross-analysis of various types of personal data (e.g., data series –location and quantified-self data–, images) by means of data mining, machine learning and artificial intelligence algorithms. Hence the need to support an open ecosystem of applications running on a potentially untrusted environment.

On the bright side, one can leverage Trusted Execution Environments (TEEs) to achieve a trustworthy platform. With their recent democratization (e.g., Intel SGX, ARM TrustZone, AMD SEV), TEEs are prime candidates for building an extensive and secure PDMS [1], on which trusted user-side processing can be established, particularly due to the security properties they offer in terms of *isolation* [7]. Recent solutions have been proposed to preserve TEE isolation for data-intensive computations in the presence of side channels (e.g., memory, timing) attacks, resorting to oblivious memory access structures (e.g., [8]) or differential privacy (e.g., [9]). As well, several SQL database engines have recently been adapted to SGX enclaves (e.g., EnclaveDB [6], StealthDB [4]) to provide data confidentiality, integrity and freshness. In our work, we also assume a basic data management engine to store personal data and regulate the access and the data flow based on simple access control policies.

However, there are important limitations since these works assume that the code running inside the TEE is trusted, which contradicts with an extensible platform. Our goal is to control the potential information leakage in the legitimate query results, computed by untrusted code extensions running data-oriented functions with access to large portions of the personal database. We favor a generic approach, independent of any semantic analysis of the processing code used or the result produced.

In Section 2, we propose an architecture dealing with the tension between security and extensiveness and formulate the security hypotheses and the corresponding threat model. Different from existing SGX-based solutions, we achieve extensiveness for advanced data processing through code extensions running inside other enclaves under the control of a trusted core engine.

Our second contribution, in Section 3, is to discuss possible security mechanisms that can be automatized within a PDMS platform to further reduce the private data leakage through legitimate computation results with basic aggregate computations, and analyze their security benefits, limitations and cost with an SGX implementation. To our knowledge, this is the first work to propose such solutions which are complementary to solutions based on code or application auditing. Finally, Section 4 proposes some further work stemming from our current results.

2. PDMS Architecture and Trust Model

Designing an extensive and secure PDMS architecture preserving usage and compliant with the expected security properties defined above is a significant challenge, given the fundamental tension between security (small trusted code base) and extensiveness (rich data-driven tasks). In particular, the code extensions cannot be part of a trusted code base (TCB) for the PDMS. To solve this problem, we propose a three-layer architecture where *Untrusted Apps* on which no security assumption is made, communicate with a minimal *Secure Core* implementing basic operations on personal data extended with *Isolated Data Tasks*.

Secure Core. The Core is a secure subsystem that constitutes a TCB. It is ideally minimal, inextensible (po-

tentially proven correct by formal methods) and isolated from the rest of the system. It must provide all basic operations required to ensure the confidentiality, integrity, and resilience of PDMS data. It must be the sole entry point to manipulate data. The Core must thus implement a data storage module, a policy enforcement module to control access to data by other layers of the architecture and a communication manager for securing data exchanges between layers, with Apps and third parties.

Isolated Data Tasks. Data tasks are introduced as a means to support code extensions needed to deal with application-specific personal data management outside of the Core. The idea is to control the execution of extensions by (1) splitting their execution into different isolated Data tasks to maintain control on the data accessed by the Core and the results delivered to the Apps to limit side effects in terms of data leakage, and (2) scheduling and verifying the execution of the different Data tasks by the Core so that security and privacy can be enforced globally.

Untrusted Apps. The need to run a wide and diverse range of applications in user environments (e.g., a web browser), naturally leads to not making any security assumptions on the applications. They consequently only manipulate authorized data resulting from Data tasks, but have no privilege on raw personal data.

Security hypotheses. To ensure the security of our architecture, we make three hypotheses.

H1. Isolated enclave execution. All architecture components manipulating sensitive data (i.e., the Core and all Data tasks) run inside TEE enclaves.

H2. Confined Data task execution. Besides running in an isolated enclave, the Data task programs are confined using an in-enclave sandbox which prevents *voluntary data leakages* from the Data task to the untrusted execution environment. Such in-enclave sandbox can be obtained using existing software such as Ryoan [2] or SGXJail [10] which provide means to restrict enclave operations (bounded address space and filtered syscalls).

H3. Secure data exchange. Each Data task program is authenticated by the Core using TEE attestation ensuring that the Core communicates only with the expected Data task running within a sandboxed enclave.

H1 prevents the untrusted environment from accessing sensitive PDMS data at runtime. H2 prevents a malicious Data task from any voluntary –direct– information disclosure outside the PDMS (e.g., by writing to untrusted memory). H1, H2 and H3 ensure that the only channel between the PDMS and Apps is through the Core.

Thus, the Core is the only module capable of declassifying the result of a Data task's computation. However, given the minimality requirement, we cannot assume the Core is capable of checking code or results semantics.

Threat model. We consider an attacker, controlling the untrusted execution environment and the code of Apps and Data tasks, whose goal is to obtain the maximum amount of personal database objects from a PDMS. Since the only data communication channel is through results declassification, the attacker will exploit it as a secondary channel to conduct the attack.

3. Proposed Solution for Reducing Leakage

Our goal is to provide techniques for reducing the attack surface of the legitimate declassification channel of results between the Core and Apps. We do not impose any usage limitations (e.g., obfuscated results or fixing a query budget for Apps), but for simplicity we focus on the case of functions producing fixed-size results like aggregates.

3.1. Computation Model

The personal database securely hosted by the Core is a set $\mathcal{O}$ of data objects of any type (tabular, time series, GPS traces, documents, images). Applications are granted results of aggregate functions evaluated over $\mathcal{O}$ according to predefined access control policies $\{< a, f, \sigma >\}$ accepted by the PDMS owner and enforced by the Core, where a is a given App with granted execution on a given aggregate function f on a subset $\mathcal{O}_\sigma \subseteq \mathcal{O}$ of database objects. σ stands for a selection predicate on objects metadata, e.g., date/type, evaluated by the Core access control module. To evaluate f, the Core launches a Data task DT_f running the code of f and provides it with $\mathcal{O}_\sigma$. DT_f achieves the computation and returns a result r to the Core which can be declassified to a.

The aggregate functions f that we consider can be represented by $f(\mathcal{O}_\sigma) = agg\left(\{cmp(o)\}_{o\in\mathcal{O}_\sigma}\right)$, where cmp is a potentially complex computation applied on each object of $\mathcal{O}_\sigma$ and agg is an aggregate (or similar) function that produces fixed-length results. For instance, cmp can compute the length of a GPS trace or the integral of a time series indicating the electricity consumption, while agg can be a typical aggregate function (sum, average, min) or similar (top-k or k-means). Although basic, the considered computation covers interesting use cases in the PDMS context, e.g., computing the user's electricity bill, statistics concerning physical activities, etc.

We consider aggregates producing results of n bits with $n \ll |O|_b$, where $|O|_b$ is the size of any object in bits. Hence, an attacker has to access (many) successive results to obtain large sets of private database objects.

In the basic approach, cmp and agg are evaluated by a single Data task $DT_{agg,cmp}$ receiving $\mathcal{O}_\sigma$ as input. A *leak* occurs each time $DT_{agg,cmp}$ produces a result deemed legitimate by the Core –then sent to a– and from which objects in $\mathcal{O}_\sigma$ can be deduced. Since the attacker controls the function code, instead of expected $comp$ and agg, a leakage function f_{leak} can be implemented, to maximize the leakage produced by consecutive queries. For example, f_{leak} can (i) sort the input set $\mathcal{O}_\sigma$, (ii) identify the n bits next to the –previous– $leak$ and consider them as the –new– $leak$; (iii) send $leak$ as the result to the Core. The attacker then reconstructs $\mathcal{O}_\sigma$ by assembling the received $leak$ resulting form consecutive evaluations.

3.2. Security Building Blocks

We introduce three security elements for a comprehensive solution to prevent leakage of large data sets.

Stateless Data tasks. An important lever that a malicious Data task can exploit is data persistency, as keeping a "state" between successive executions maximizes leakage –in the example above *leak* is kept to avoid leaking

the same data twice–. In our system, persistent states can be stored without hurting the security hypotheses, e.g., via the PDMS database or secure file system.

A first element is to rely on stateless Data tasks – without negative impact on usage as database queries are evaluated independently–. Stateless nature extends confinement (see *H2*) by preventing access to the PDMS database or secure storage (e.g., SGX protected file system library). A stateless Data task is instantiated for the sole purpose of answering a specific function call/query, after which it must be terminated and its RAM wiped. On SGX, this can be achieved by destroying the Data task's enclave.

Deterministic Data tasks. An attacker can also exploit randomness to increase leakage. For example, at each execution, f_{leak} can select a random fragment of $\mathcal{O}_\sigma$ to produce a new $leak$ with high probability –even if the same query is run on the same input–.

A second element is thus to impose reproducible Data tasks, i.e., producing the same result for the same function code (agg and cmp) run on the same input ($\mathcal{O}_\sigma$). The in-enclave sandbox can be leveraged to provide this property by preventing access to any source of randomness, e.g., syscalls to random APIs or timer/date. Virtual random APIs are provided to preserve legitimate uses, e.g., the need for sampling, and are "reproducible", e.g., random numbers are forged in the Core using a seed based on function code and input set ($seed = hash(agg||cmp||\mathcal{O}_\sigma)$). For efficiency, the Core could store results of Data task calls to respond to subsequent invocations of the same Data tasks on the same inputs.

Circumscribed leakage at objects level. Since the same function can be evaluated on different inputs by changing the selection predicate σ, the attacker can leak new data with each new query, regardless if Data tasks are stateless or deterministic. To preclude this kind of attack, we enforce a deterministic leakage at the object level.

A third security element is to split the execution into several Data tasks scheduled by the Core. First, a set of Data tasks $DT^i_{cmp} = DT_{cmp}(\mathcal{P}_i)$, $i \in [1,m]$, are instantiated and each evaluates cmp on one partition $\{\mathcal{P}_i\}_{i\in[1,m]}$ of $\mathcal{O}_\sigma$. Each DT^i_{cmp} produces a result set $res^i_{cmp} = \{cmp(o)\}_{o\in\mathcal{P}_i}$. We suppose that $|cmp(o)|_b < |o|_b$. Second, a Data task DT_{agg} is used to aggregate $\bigcup_{i\in[1,m]} res^i_{cmp}$.

The remaining question is how many partitions m to create. Reducing m leads to fewer Data tasks which improves performances (see Section 3.3). On the other hand, increasing m leads to smaller partitions thus smaller leakage "surface". In the most secure scenario, each DT^i_{cmp} receives a single object o^i and produces $res^i_{cmp} = cmp(o^i)$. In this case the leak: (i) is a fixed value for each object since DT^i_{cmp} is deterministic; (ii) no object can be leaked entirely since $|cmp(o^i)|_b < |o|_b$.

3.3. Performance Assessment

The proposed PDMS architecture provides increased security and extensiveness, but with the security building blocks, it introduces performance overheads. Our goal is to provide an initial assessment. We implement a PDMS platform on Intel SGX using OpenEnclave SDK [3], using SQLite as personal database hold in the Core.

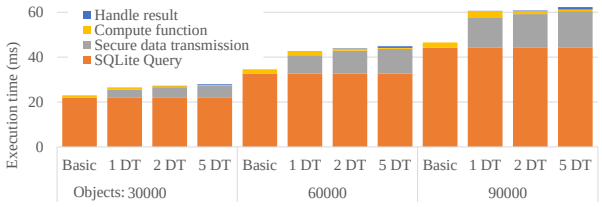

Figure 1. Performance evaluation overview.

Figure 1 shows execution times for a typical aggregation query on tabular data with different data set sizes. As baseline, we use a single enclave running agg, cmp and the Core. This is an unrealistic approach as it assumes trusted code for the extensions agg and cmp. The other three implementations correspond to our approach which separates extensions from the trusted Core, within one or more untrusted Data tasks. The results indicate that: (i) the separation between Core and Data tasks introduces a relatively small overhead (25% on average); (ii) the execution time is largely dominated by the database query execution at the Core level, followed by the secure data transmission between Core and Data tasks; (iii) the use of more Data tasks slightly increases the cost of data transmission, but not the overall computation time. Our conclusion is that the overhead remains acceptable compared to the security benefits, which validates our approach.

4. Conclusion and Roadmap

The next step is to quantify the potential leakage and explore performance trade-offs related to partition size, number of enclaves allocated at query time and reuse of previous results stored by the Core. Our solution already applies to simple use cases. Other types of computations need to be studied, e.g., the case of parametric queries when parameters are supplied by the App, which offers an additional side channel for the attacker.

References

[1] N. Anciaux et al., "Personal Data Management Systems: the security and functionality standpoint", Inf. Syst., 80, 2019.

[2] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: a distributed sandbox for untrusted computation on secret data", ACM Trans. on Computer Syst. (TOCS), 35(4), 2018.

[3] OpenEnclave SDK, https://openenclave.io.

[4] D. Vinayagamurthy, A. Gribov, and S. Gorbunov, "StealthDB: a scalable encrypted database with full SQL query support", Privacy Enhancing Technologies, pp.370–388, 2019.

[5] EU Commission, "Proposal for a Regulation of the European Parliament and of the Council on European data governance (Data Governance Act), COM/2020/767", 25 October 2020.

[6] C. Priebe, K. Vaswani, M. Costa, "EnclaveDB: a secure database using SGX", IEEE Sym. on Security and Privacy (S&P), 2018.

[7] M. Sabt, M. Achemlal, A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not", TrustCom, 2015.

[8] K. Ren et al., "Hybridx: New hybrid index for volume-hiding range queries in data outsourcing services", ICDCS, 2020.

[9] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers, "Shrinkwrap: efficient sql query processing in differentially private data federations", VLDB, 12(3), 2018.

[10] S. Weiser, L. Mayr, M. Schwarz, and D. Gruss, "SGXJail: Defeating Enclave Malware via Confinement", Int. Symp. on Research in Attacks, Intrusions and Defenses (RAID), 2019.