Towards a Formal Approach for Data Minimization in Programs (Short Paper)

Florian Lanzinger✉[0000−0001−8560−6324]⋆ and
Alexander Weigl[0000−0001−8446−4598]

lastname@kit.edu
Institute of Information Security and Dependability (KASTEL)
Karlsruhe Institute of Technology (KIT)

Abstract. As more and more processes are digitized, the protection of personal data becomes increasingly important for individuals, agencies, companies, and society in general. One principle of data protection is data minimization, which limits the processing and storage of personal data to the minimum necessary for the defined purpose. To adhere to this principle, an analysis of what data are needed by a piece of software is required. In this paper, we present an idea for a program analysis which connects data minimization with secure information flow to assess which personal data are required by a program: A program is decomposed into two programs. The first projects the original input, keeping only the minimal amount of required data. The second computes the original output from the projected input. Thus, we achieve a program variant which is compliant with data minimization. We define the approach, show how it can be used for different scenarios, and give examples for how to compute such a decomposition.

Keywords: Secure information flow, Data minimization, GDPR

1 Introduction

Privacy and data protection are urgent topics as more and more processes in are digitized. One consequence of this digitization is the increasing amount of stored and processed personal data. To ensure that users' privacy is respected and their data protected, the European legislator reacted with the General Data Protection Regulation (GDPR) [3] which regulates the collection and processing of personal data. One important principle is *data minimization*, which limits the collection and storage of personal data to those which are "necessary in relation to the purposes for which they are processed" [3, §5(1)(c)].

Therefore, authorities and companies must analyze which personal data are necessary for the conduct of their operation. This not only includes inspecting the accessed and used personal data, but also analyzing how they are involved in the

⋆ This work was supported by funding of the Helmholtz Association (HGF) through the Competence Center for Applied Security Technology (KASTEL)

process and what decisions, if any, depend on them. For example, an authority may store the exact income of a person, but if it only needs to know that the income is above a certain threshold, it can just store this Boolean information. As a result of such an analysis, only the minimal necessary information should be collected.

Contribution. We sketch a formal approach to analyze and assess the data usage of algorithms (Sect. 2). The idea of the analysis is to decompose a given program into two parts: The first part projects the original input data onto the minimal set of data from which the result can be computed. The second part computes the original output from this minimal data. We discuss how this decomposition can be applied in different scenarios (Sect. 3) and which additional constraints are added to the base problem. Moreover, we give two examples of how such a decomposition can be computed with different precisions using syntactic or semantic analyses.

2 A Formal Approach to Data Minimization

For the presentation of the approach, we model a program as a function $f\colon I \to O$ which maps the given input $i \in I$ to an output $f(i) = o \in O$. The input space $I = H \times L$ is a combination of personal data H, which are **H**ighly confidential, and **L**ow confidential data L like configuration parameters. The high and low categorization is defined by the author or auditor. This modeling assumes that programs are side-effect free and that the output o is computed only by considering the inputs in $H \times L$. Fig. 1 shows an example program whose output is the tax rate, whose confidential inputs H are the `age` and `income` of a person, and whose non-confidential input L is a base tax rate, from which the actual taxe rate is computed.

```
tax_rate(age, income: H,
         base: L) {
  if (age<18) return 0*base;
  if (age<25 && !(income>1000))
    return base;
  if (income<=1000)
    return 2*base;
  return 3*base; }
```

Fig. 1. An example program which computes someone's tax rate based on their age and income.

Basic idea. Given a program $f : H \times L \to O$, we want to find a decomposition into two programs (π, ρ) such that $f(h, l) = \rho(\pi(h), l)$ for all inputs $(h, l) \in H \times L$ and such that the information provided by $\pi(h)$ is minimal. We call π the *projection* and ρ the *reduction*.

The information provided by $\pi(h)$ can be described as the information leakage, a notion commonly used in quantitative information flow. Leakage is the amount of secret information (here, the amount of information about h) that is disclosed by a program [7]. The leakage is given in bits and computed from the entropy. Different entropy definitions reflect different attacker capabilities. For simplicity, we consider the min-entropy which reflects how well the input can be guessed by

an attacker in one try. This leads to a minimization of the number of possible images of π for deterministic programs: $\min_\pi |\{\pi(h) \mid h \in H\}|$.

Our approach is also related to secure information flow, which analyses whether an attacker learns secret information by observing the exposed output. In some situations, secret information is declassified. For example, paper reviews are confidential until the final acceptance decision is drawn. By computing the decomposition, we want to infer the minimal amount of secret information that is declassified by f. This declassified information is represented by the image of π.

An additional benefit of our approach is that it allows for a comparison between programs. Given two programs f, f', we can compute both decompositions and compare the images of the projections π and π' to find out which version requires less information.

We assume that the decomposition of f into (π, ρ) is computed either by the author of f who honestly wants to observe the principle of data minimization or by an independent auditor with access to f's source code.

Unwanted degenerate solution. If all parameters are in H (e.g., if we replaced the parameter `base` in Fig. 1 with a constant), there is a trivial solution to the stated problem definition, where $\pi(h) = f(h)$ projects all inputs h to the output $f(h)$ and ρ is just the identity function $\rho = \mathrm{id}$. In our tax example, this represents a decomposition in which the applicants compute their tax rates by themselves and only submit these final tax rates to the authority. This solution is undesirable for various reasons: First, it does not provide any insight into which information is required by the computation. Second, from the viewpoint of the authority, the decision might not be auditable later, especially if the required personal information has changed. Collecting a rudimentary amount of personal data may be required to simplify later validation.

The given formalization thus requires further constraints to give useful results. The constraints depend on the operational purpose and the usage context in which the personal data should be minimized.

3 Usage Scenarios

In this section, we consider some usage scenarios, each of which leads to different additional constraints.

Paper-based form. Public authorities often ask applicants for personal data using paper-based forms. These forms are of course static; their contents cannot change depending on the user's input. Also, they should not require the applicant to do very complex computations.

Let us again consider Fig. 1. It shows a simple program which computes the tax rate based on the applicant's `age` and `income`, which are both in H. The authority wants to collect the minimal amount of personal information required by this program.

To keep the projection simple enough to compute for the person filling out the form, and to prevent degenerate solutions, we add the following constraint:

The projection π is only allowed to contain basic unary and binary expressions over a single variable, e.g., $x < 2$ or $\neg x$. In our example, the function which maps `age` and `income` to the three truth values $\texttt{age} < 18, \texttt{age} < 25, \texttt{income} > 1000$ is a valid projection.

The reduction ρ then only has to know the values of these Boolean expressions instead of the exact age and income. Indeed, it is common in tax applications that applicants only need to confirm that they qualify for some taxation rule without giving the exact reason why they qualify. Only in doubt does the tax office request further evidence to check the confirmation.

Web form. Our second scenario uses the same setting, but instead of a paper-based form, we use a web form. In this case, the decomposition represents the system borders: π represents the web form in the browser, ρ the program on the authority's servers. The user can enter their complete personal information, but the browser will only send the required information to the authority. This allows π to synthesize expressions that would be too complex to put on a paper form. However, the actual decision-making should still be done in ρ, not in π. We thus modify the previously stated constraint: The program π is allowed to contain Boolean and arithmetic expressions over multiple variables , e.g., $x < y \wedge y < z$, $x + y$, or $x + y > 0$. This allows for a smaller information leakage between π and ρ, but the requirement that π only contain basic expressions (and not, for example, **`if`** or **`while`** statements) ensures that the actual computation is still done in ρ. One reason behind this requirement is that π runs on a non-trusted computer system, i.e., the browser of the customer or applicant. To ensure a proper evaluation of the personal data, we still need to transfer them to the trusted server where ρ is executed. Thinking beyond, we might want to decompose f into three programs, one that runs on the non-trusted platform and minimizes the data to be transferred to the server, one that runs on the server and minimizes the data to be stored, and one which computes the final decision from the storage. In general, we can extend the approach to a decomposition into an arbitrary number of program parts, as long as it is defined which computations must be done in which part.

Outsourcing computation to the cloud. We consider a company which wants to outsource expensive computations to the cloud while exposing as little internal data and computation as possible. The decomposition into π and ρ represents different environments. The program π is executed on the company's system—a trusted environment where sensitive computations can be executed—and ρ is executed in the cloud. Therefore, ρ should contain expensive computations on the minimal required sensitive data. Thus, in this scenario, π can be any arbitrary program. Additionally, we assign computation costs to each statement or expression in f. We want to keep sensitive computations from f from being moved into the cloud computation ρ. We thus receive new constraints: The program f is split into π and ρ, such that the computation cost of π is minimal, and no (or only a certain amount of) sensitive computation is done in ρ.

4 Approaches to the Computation of the Decomposition

In this section, we sketch some ideas how such a partitioning can be (approximately) computed. For now, we assume that f is loop-free and that all expressions are side-effect-free. We also assume that f is free of local variables; i.e., the only variables that occur are the parameters. We give some ideas of how these restrictions can be lifted at the end of this section.

The first approach works syntactically by collecting the set E of all expressions in f which contain at least one sensitive parameter in H. Choose a set $\Pi \subseteq E$ such that all expressions in E can be built up only from expressions in Π, low parameters in L, literals, and the basic binary operators. E.g., from the expressions $\mathtt{a}$ and $\mathtt{b}$, we can build $\mathtt{a} + \mathtt{b} > 0$ and also $\mathtt{a} + \mathtt{c} > 0$ if $\mathtt{c} \in L$. A function that maps f's parameters to such a subset Π is a valid projection π. The reduction ρ belonging to π is the function whose body is equal to f's body, except that every expression e in Π is substituted by a new variable, which represents the value of e as computed by π.

For example, a possible solution for Fig. 1 is $\Pi = \{\mathtt{age} < 18, \mathtt{age} < 25, \mathtt{income} > 1000\}$. The program π maps `age` and `income` to the values of the expressions in Π, and ρ is identical to f except that all expressions in Π have been replaced by new variables, e.g., the expression $\mathtt{age} < 18$ has been replaced by a new variable `ageUnder18`. If we allow π to contain expressions over multiple variables, as in our second scenario, we can also choose $\Pi = \{(\mathtt{age} < 18), (\mathtt{age} < 25 \wedge \neg \mathtt{income} > 1000), (\mathtt{income} > 1000)\}$.

The second approach works semantically: We collect the path conditions and returned expressions for each path by using symbolic execution, a common technique for program analysis and widely available for different programming languages. For Fig. 1, this yields the four conditions

$$R1 = \mathtt{age} < 18;$$
$$R2 = \neg(\mathtt{age} < 18) \wedge \mathtt{age} < 25 \wedge \neg(\mathtt{income} > 1000)$$
$$R3 = \neg(\mathtt{age} < 18) \wedge \neg(\mathtt{age} < 25 \wedge \neg(\mathtt{income} > 1000)) \wedge \mathtt{income} \leq 1000$$
$$R4 = \neg(\mathtt{age} < 18) \wedge \neg(\mathtt{age} < 25 \wedge \neg(\mathtt{income} > 1000)) \wedge \neg(\mathtt{income} \leq 1000))$$

and the four return expressions `0`, `base`, $2 * \mathtt{base}$ and $3 * \mathtt{base}$. We choose a set of expressions Π from whose values the value of the path conditions and return expressions of all reachable paths can be computed (e.g., from `a` and `b`, we can compute $\mathtt{a} + \mathtt{b} > 0$, but also $\mathtt{a} + \mathtt{b} + \mathtt{c} - \mathtt{c} > 0$ for any `c`) and construct π, ρ as before.

For Fig. 1, this leads to the same solution(s) as the syntactic approach. However, the semantic approach has several advantages in comparison to the syntactic approach. First, it is robust against misleading syntactical constructs. A programmer may try to change a program's syntax to increase

```
tricky(a, b: H) {
  if(a != 0)
    if(b < 0) return a-b;
  else
    if(a != 0) return b;
  return a; }
```

Fig. 2. An example with unreachable code and misleading syntax which are undetectable by the syntactical approach.

the required information in favor of
the company. For example in Fig. 2, the programmer split an expression `a!=0` and `b<0` into two if-statements. A syntactic analysis would include both expressions in the projection, missing the fact that they can be joined into `a!=0 && b<0`, Moreover, working on a semantic level makes the analysis aware of unreachable code. For example, the `return b;` in Fig. 2 is not reachable. A syntactic approach would include `b` in the projection, while a semantic approach could notice that this return statement is unreachable.

In general, semantic approaches track and extract the required information precisely. The disadvantage is the complexity and low scalability of the analysis. The halting problem is reducible to the computation of the exact minimizing decomposition. Thus, the exact decomposition is undecidable in general.

To allow local variables, we can transform a program into its *static single assignment* (*SSA*) form, in which every local variable is only assigned to exactly once. The SSA form allows us to expand the definitions of all variables and then delete all unnecessary assignments; e.g., the statements `a=b;c=a` can be transformed into `c=b`. We can then apply our—slightly adapted—approaches on this transformed program. For programs containing loops, we cannot expand variable definitions like this, since the correct expansion depends on which loop iteration we are in. One way to solve this may be to choose an upper bound for the number of loop iterations and unroll all loops.

5 Limitations

Our proposed approach has several limitations. First, we do not assess whether the given data are "adequate, relevant, and limited to what is necessary" for the stated purpose, as required by GDPR §5(1)(c). We only consider whether the data are required to compute the output of the given program. For example, the gender of a person may not be (legally) relevant for some application process, but if the program's output depends on it, our approach will mark it as required. This difference was discussed by Biega et al. [2], where you can also find a discussion on the wording of GDPR §5(1)(c), including the legal meaning of the term *relevant*.

Second, the approach is not aware of statistical correlations or dependencies between parameters. If two parameters are correlated, e.g., `address` and `income`, the approach is unable to infer that knowing a person's address may also allow one to draw conclusions about their income. Datta et al. [4] show how systems can be validated against this *proxy discrimination*.

Furthermore, this is a white-box approach which requires access to the source code and the program structure. On the one hand, this gives us the possibility to output either witnesses that show why certain data are indeed required by a program or formal guarantees that the date are not required. On the other hand, it limits the scalability of the approach, as it needs to scale with large source code. Also, we need to adapt the approach in situations where the source code may not be available (e.g., built-in library functions). As discussed in Sect. 4, using the source code of the program makes our approach, especially the syntactic variant,

vulnerable to adversaries. Programmers may be able to exploit the program structure to increase the minimal required amount of information.

6 Related Work

Pfitzmann and Hansen [9] identify five concepts for data minimization: anonymity, unlinkability, undetectability, unobservability, pseudonymity, and authenticity. We follow a different approach and only consider whether personal data are required for the execution of a program. In detail, we try to find a *similar* program which is functionally equivalent and requires less personal data. Goldsteen et al. [5] present a method for the minimization of the required personal data in a machine learning model. Their approach removes and generalizes the input features. The features which promise the least degeneration of accuracy are identified in the learned model. Later in the application phase of the trained machine, only the survived features are collected. The other features are removed. Therefore, the training still requires personal data. A similar approach is followed by Biega et al. [2]. They investigate data minimization for recommendation systems under a global and per-user minimization strategy. Ramadan et al. [10] present a framework for modeling business processes that evaluates the security, data-minimization, and fairness requirements, and allows the detection of conflicts between them based on a catalog of domain-independent anti-patterns.

Program Slicing [1] is a technique to omit unnecessary statements from a program. A statement is unnecessary if it does not influence the output. We can compute a program's slice, in which every statement influences the output. In such a slice, the required data is easier to identify. Our approach goes further: Instead of just removing completely unnecessary statements, it is also able to replace expressions with less informative variants and thus actually decrease the amount of required information.

Kammüller [6] demonstrates the formalisation of the GDPR using the *Decentralized Label Model* (DLM) [8] in Isabelle. In DLM, data is labeled with owner and reader lists which define access rights. DLM allows multiple principles, i.e., users who are granted access rights. In contrast to DLM, our approach consumes only one algorithm as input and input categorization. It can be extended for multiple principles: If we consider each principle's information use separately, we can compute the minimal amount of required information for each principle. Then, instead of granting a principle read access to the complete information, they only receive the required amount of information. In contrast to Kammüller, who checks for GDPR compliance, we focus on a program transformation which makes a program GDPR-compliant.

7 Conclusion

We present a novel connection between data minimization and secure information flow. The idea is to split a given program into two programs: one which projects the specified personal data to the minimal needed amount of data, and one which

computes the original output from the minimized data. We elaborate various additional constraints on this formalization depending on different usage scenarios and also discuss the feasibility of the approximation of such a flow-minimizing program split.

In the future, we plan to extend our approach to programs with loops. We also plan to implement both a syntactic and a semantic approach as outlined in Sect. 4, which automatically computes (minimal or approximately minimal) projections and appropriate reductions for programs in a real-world programming language.

References

1. Beckert, B., Bormer, T., Gocht, S., Herda, M., Lentzsch, D., Ulbrich, M.: SemSlice: Exploiting Relational Verification for Automatic Program Slicing. In: Polikarpova, N., Schneider, S.A. (eds.) IFM 2017, Proc. LNCS, vol. 10510, pp. 312–319. Springer (2017). https://doi.org/10.1007/978-3-319-66845-1_20
2. Biega, A.J., Potash, P., III, H.D., Diaz, F., Finck, M.: Operationalizing the legal principle of data minimization for personalization. In: Huang, J., Chang, Y., Cheng, X., Kamps, J., Murdock, V., Wen, J., Liu, Y. (eds.) SIGIR 2020, Proc. pp. 399–408. ACM (2020). https://doi.org/10.1145/3397271.3401034
3. Council of the European Union: General Data Protection Regulation (2016), https://eur-lex.europa.eu/eli/reg/2016/679
4. Datta, A., Fredrikson, M., Ko, G., Mardziel, P., Sen, S.: Proxy non-discrimination in data-driven systems. CoRR (2017), http://arxiv.org/abs/1707.08120
5. Goldsteen, A., Ezov, G., Shmelkin, R., Moffie, M., Farkash, A.: Data minimization for GDPR compliance in machine learning models. CoRR (2020), http://arxiv.org/abs/2008.04113
6. Kammüller, F.: Formal modeling and analysis of data protection for GDPR compliance of iot healthcare systems. In: IEEE SMC, Proc. pp. 3319–3324. IEEE (2018). https://doi.org/10.1109/SMC.2018.00562, https://doi.org/10.1109/SMC.2018.00562
7. Klebanov, V., Manthey, N., Muise, C.J.: Sat-based analysis and quantification of information flow in programs. In: Joshi, K.R., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013, Proc. LNCS, vol. 8054, pp. 177–192. Springer (2013). https://doi.org/10.1007/978-3-642-40196-1_16
8. Myers, A., Liskov, B.: Complete, safe information flow with decentralized labels. In: IEEE S&P, Proc. pp. 186–197 (1998). https://doi.org/10.1109/SECPRI.1998.674834
9. Pfitzmann, A., Hansen, M.: A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management (2010), http://dud.inf.tu-dresden.de/Anon_Terminology.shtml
10. Ramadan, Q., Strüber, D., Salnitri, M., Jürjens, J., Riediger, V., Staab, S.: A semi-automated BPMN-based framework for detecting conflicts between security, data-minimization, and fairness requirements. Softw. Syst. Model. **19**(5), 1191–1227 (2020). https://doi.org/10.1007/s10270-020-00781-x