Finding risk patterns in cloud system models*

Florian Kunz and Zoltán Ádám Mann
paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Essen, Germany

***Abstract*—The risk of unauthorized access to confidential data is a major problem in cloud computing. In previous work, the notion of risk patterns was introduced to capture configurations of cloud systems that are prone to data protection issues. In this paper, we devise a program for the automatic detection of risk patterns in cloud system models. Our program makes use of the Eclipse Modeling Framework, the model transformation library Henshin, and the modeling workbench Sirius to (i) enable security experts to describe cloud risk patterns in a compact way, (ii) enable the efficient automatic detection of risk patterns in the model of a cloud system, and (iii) support cloud experts in experimenting with the security implications of different cloud configurations. A case study and experiments demonstrate the applicability and scalability of the proposed approach.**

***Index Terms*—cloud computing; security; risk management; data protection; graph pattern matching; run-time model**

I. Introduction

Using cloud services usually involves storing and processing data in the cloud. Some of those data may be confidential: for example, personal data that must be protected according to the EU General Data Protection Regulation (GDPR [6]), or trade secrets that must be protected from competitors. Protecting confidential data in the cloud is very challenging: customers of cloud services lose control of their data when the data are uploaded to the cloud, and numerous other parties – e.g., the service provider and other tenants – may succeed in obtaining unauthorized access to the data [16], [8].

Protecting data in the cloud is made difficult by the complexity of cloud systems, which consist of many hardware and software components [7]. These components are accessible by different stakeholders and host different pieces of data. A malicious attacker having access to one component may be able to find a path of components through which it can get access to confidential data. For instance, an Infrastructure-as-a-Service (IaaS) provider may abuse its access to physical servers to get control of applications running in virtual machines hosted on those servers, and then use such an application to access confidential data stored in a remote database. If such an attack is anticipated, security engineers can protect against it by using appropriate security controls (e.g., access control or encryption). Each security control may block some potential attack paths. However, the higher the complexity of a cloud system, the more difficult it becomes to verify that every potential attack path has been blocked. Thus, it is difficult to assess the risk that certain types of attacks would be successful in gaining unauthorized access to confidential data.

A further issue that complicates data protection is the dynamic nature of cloud systems. Workloads change, prices change, servers fail, new servers become available, software is upgraded etc., leading to frequent changes in the cloud. All changes may impact the risk of data protection violations: a change may eliminate a vulnerability, or may introduce a new one. Hence, assessing data protection risks of a cloud system is not a one-off activity, but must be carried out continually. Moreover, it has to be automated to enable the quick detection of newly arising data protection risks during system operation.

A promising approach for automatically detecting data protection risks in cloud systems uses so-called *risk patterns* that capture sub-structures of cloud configurations associated with high data protection risks [13]. Moreover, a model of the cloud system is used and continually updated based on run-time monitoring of the system to ensure that the model is always in line with the real cloud system. [13] suggested that graph pattern matching algorithms could be used to automatically search for instances of the risk patterns in the model of the cloud system. However, the approach was not implemented, leaving several important questions open:

- Can graph pattern matching indeed be used for finding risky cloud configurations?
- Can risk patterns be captured in a concise format?
- Is the search for risk patterns fast enough for online use?

In this paper, we address these gaps. We provide an implementation of the risk pattern approach using the Eclipse Modeling Framework (EMF), the model transformation library Henshin, and the modeling workbench Sirius. We show that

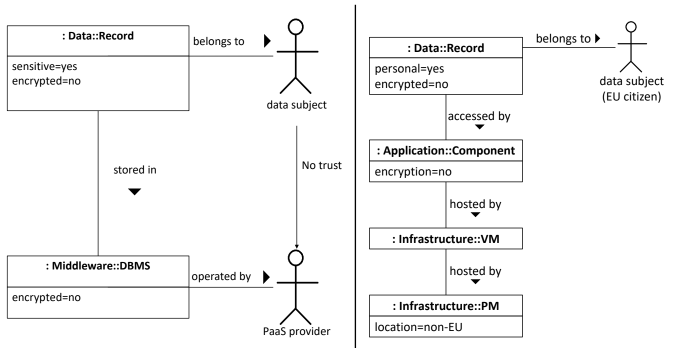
- Graph pattern matching is not enough to capture all relevant information. However, using more powerful EMF-based model matching, we were able to implement the automatic search for risk patterns.
- Risk patterns can be concisely captured in the form of Henshin rules, which are both human-readable and machine-readable.
- The search for risk patterns is quite fast: for cloud models of up to 17,000 nodes, the search took less than 0.4 seconds on a commodity computer.

Therefore, our implementation yields significant new insights regarding the applicability of the risk pattern approach.

II. Preliminaries

In this section, we summarize information about the risk pattern approach suggested in [13], [9]. The aim of the approach is to detect cloud configurations that are associated with an unacceptably high risk of data protection violations.

* This paper was published in: *IEEE 12th International Conference on Cloud Computing (IEEE CLOUD)*, pp. 251-255, 2019



(a) Risk Pattern A (b) Risk Pattern B

Fig. 1. Example risk patterns from [13]

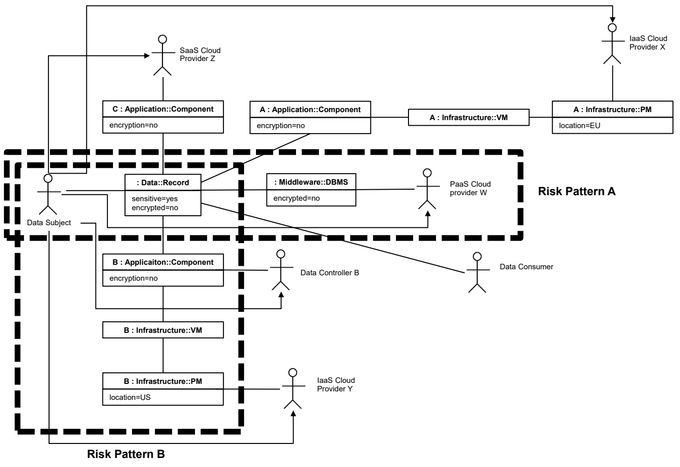


Fig. 2. Example cloud system model, in which the found risk pattern instances are marked by dashed frames [13]

To achieve this aim, a model-based approach is suggested. A *cloud system model* represents all entities that may be relevant for data protection, including infrastructure elements like physical machines (PM) and virtual machines (VM), middleware elements like application servers, application components, databases, and stakeholders [9]. It is assumed that run-time monitoring of the cloud continually updates the model so that it is always in line with the actual cloud configuration.

Beside the cloud system model, the other key artefact is a set of *risk patterns*. A risk pattern captures a problematic sub-structure of a cloud configuration which, if contained in the cloud system model, would lead to unacceptably high data protection risks. Risk patterns are model fragments using the same entities as the cloud system model, i.e., they are based on the same meta-model, but have different semantics. While the cloud system model describes a specific cloud configuration, a risk pattern captures a problematic situation that may or may not be found in the models of different cloud configurations. A risk pattern specifies model elements (entities, relations, attribute values) that must be present in a model to match the given pattern, as well as model elements that must not be present in a match. For elements not specified in the risk pattern, their presence does not matter.

In [13], two sample risk patterns were presented, see Fig. 1. The first risk pattern describes a situation in which sensitive data of a data subject are stored in unencrypted form in a database which is operated by a Platform-as-a-Service (PaaS) provider, and the provider is not trusted by the data subject. This is a high data protection risk since the untrusted provider may gain unauthorized access to the sensitive data. The arc labeled "No trust" is an example of the absence of a model element being prescribed by the risk pattern, i.e., this risk pattern will only match cloud models in which there is no trust relation between the data subject and the PaaS provider. The second risk pattern captures the situation in which sensitive data about an EU citizen are processed by an application component hosted outside the EU. This is problematic because the GDPR stipulates that personal data should only be processed within the EU[1]. The risk pattern captures the chain of relations from the data through the application component and a VM to the PM, the location of which causes the problem.

In [13], the model of a real cloud system was used for validation (see Fig. 2). The two risk patterns were manually searched for in the model, and one instance of each of them was found. This result was in line with the evaluation by security experts.

The risk pattern approach is promising as it can capture complex configurations that lead to data protection risks. This can be seen on the two presented example risk patterns: the data protection issues arise when the depicted configurations, including the given entities, attributes, and relations, can be found in a cloud model. However, existing work on the risk pattern approach is on a high level of abstraction, in particular lacking an implementation of the pattern matching process.

III. Implementation

For implementing the risk pattern approach, we need more than just a graph pattern matching algorithm. As can be seen from Figures 1–2, we need the ability to reason about not only graph structures (vertices and edges), but also object types, attribute values, and relationship types. The types can also form an inheritance hierarchy. E.g., it was shown in [12] that it is useful to extend the meta-model with two new types `AtomicComponent` and `CompoundComponent`, both inheriting from `Component`. The `Component` element of Risk Pattern B in Fig. 1 should then match any object of type `Component` or of any of its sub-types.

Therefore, we decided to use the Eclipse Modeling Framework (EMF [15]). EMF is a widely used modeling framework based on the Eclipse environment and supported by a variety of tools. In particular, Henshin is a model transformation library,

[1]Actually, the GDPR allows processing of personal data also in certain countries outside the EU. In this respect, Risk Model B of [13] is not fully accurate. The location attribute of the PM should be something like "prohibited country" instead of "non-EU"

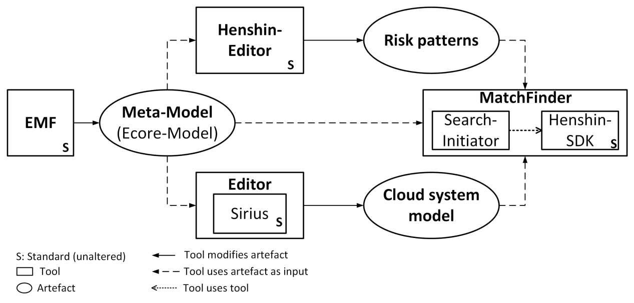


Fig. 3. Overview of our implementation of the risk pattern approach

also supporting pattern matching in EMF models [3]. EMF also allows direct model manipulation with custom Java code.

Fig. 3 shows the interaction of the used tools (EMF, Henshin, Sirius, MatchFinder) and the artefacts created or used by the tools (meta-model, cloud system model, risk patterns). With the help of EMF, we created a meta-model of cloud configurations. This meta-model is stored in EMF's native format (Ecore). The risk patterns are modelled with the Henshin editor in the form of Henshin rules. Pattern matching and the handling of found matches are performed by the MatchFinder tool, which is a combination of custom Java code and the Henshin engine. To foster experimentation, visualization, and manual exploration of design alternatives during design time, we also created an editor with which the cloud system model can be displayed and edited. For this purpose, the Sirius plug-in was used [17].

a) Meta-model in Ecore: We translated the meta-model from [12] to Ecore format. This required some small technical changes, like the insertion of a root node. We also discovered some small inconsistencies between the meta-model of [12] and the cloud system model and risk patterns of [13]; e.g., the meta-model of [12] contained no relation between actors that would represent a trust relationship, which is needed in the example risk patterns of [13]. We fixed the inconsistencies by making appropriate changes to the meta-model, the cloud system model, or the risk patterns. Such inconsistencies can remain undiscovered in a manual validation such as in [13], [9], [12]; an implementation and validation like we perform here has a much higher chance of discovering such issues.

b) Risk patterns as Henshin rules: Henshin provides an editor for the creation of Henshin rules based on an Ecore model. A Henshin rule specifies a model transformation. Conceptually, a Henshin rule consists of two parts: the left-hand side (LHS) specifies a pattern to be found in the input model, while the right-hand side (RHS) specifies how the found pattern is to be changed in the output model. The Henshin editor presents the LHS and RHS together in an integrated object model, in which Henshin-specific annotations are used to mark the role of the model elements (objects, relations, attributes) with respect to the LHS and RHS. For specifying risk patterns, we use two of Henshin's annotations: `preserve` and `forbid`. Every model element annotated with `preserve` has to be found in the input model for

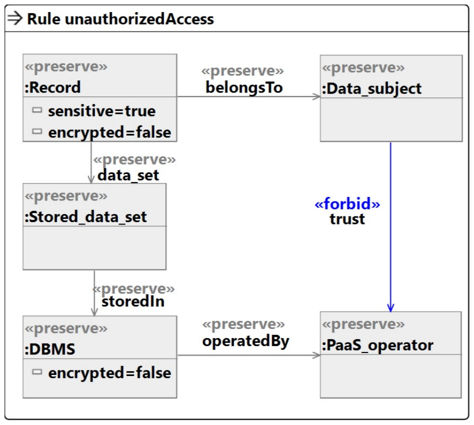


Fig. 4. Realization of Risk Pattern A from Fig. 1 as Henshin rule

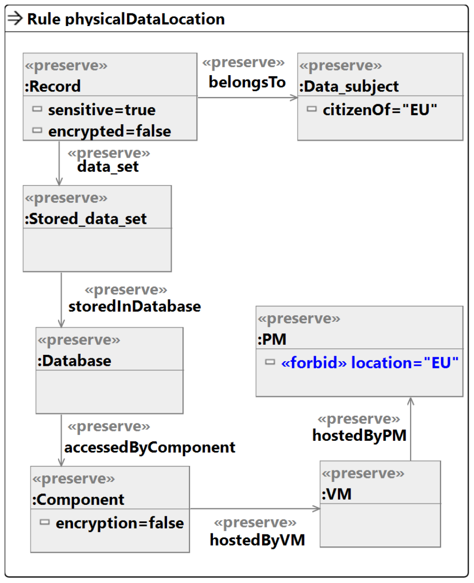


Fig. 5. Realization of Risk Pattern B from Fig. 1 as Henshin rule

a match. Model elements marked with `forbid` must not be present for a match to be found. Figures 4–5 show the realization of the two example risk patterns from Fig. 1 as Henshin rules. As can be seen, the annotations `preserve` and `forbid` are used to mark model elements that must be present, respectively must not be present in a match.

c) Editor for cloud system models: Beside the risk patterns, a cloud system model is needed in which the risk patterns are searched for. The cloud system model can be a design-time artefact that a cloud security analyst works with to analyze different cloud configurations, or a run-time model updated through monitoring reflecting the current configuration of the cloud system. We created an editor for working with cloud system models, which may be used in the design-time setting by the cloud security analyst. In a run-time setting, this tool may be used to visualize the current system configuration to a human operator.

We used Sirius [17] to create a graphical editor, in which

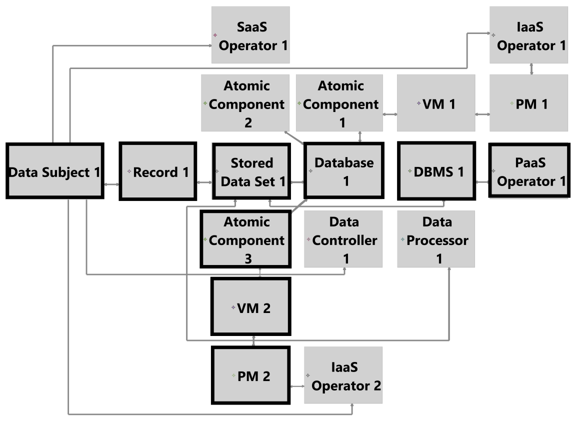


Fig. 6. Cloud system model with the found risk pattern instances marked

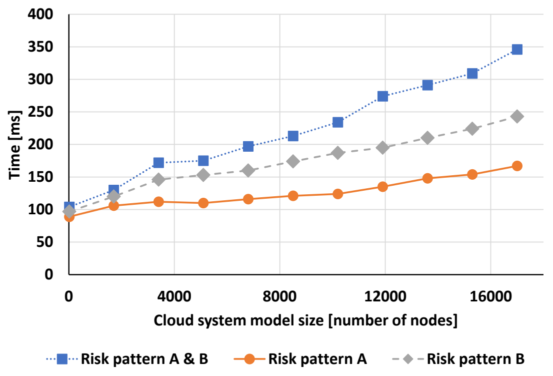


Fig. 7. Duration of pattern matching for growing cloud system models

objects, their attributes and relationships can be created in accordance with the meta-model, displayed on a canvas, and edited. We constructed creators for the instantiable types of the meta-model, which are made available to users in the form of a toolkit. For the relations, providing a separate creator for each association in the meta-model would have led to a huge number of creators, resulting in decreased usability. Hence we implemented a generic creator for relations, which automatically infers the type of the relation based on the types of the start and end nodes.

d) The control logic: MatchFinder is a Java project that integrates the other pieces. It uses the risk patterns in form of Henshin rules, the cloud system model created with the Sirius-based editor, and the meta-model in Ecore format. MatchFinder controls the pattern matching process and handles the results. The SearchInitiator within MatchFinder reads and converts all input models and initiates the pattern matching using the Henshin tool set. The result of the pattern matching is a (possibly empty) set of matches. For each match, MatchFinder displays a human-readable textual description of the match. The matches are also graphically shown within the cloud model in the Sirius-based editor.

IV. Validation

To validate our implementation, we used the same example risk patterns and cloud model as in [13], except for some small changes required to ensure consistency with the meta-model. Figures 4–5 show our realization of the example risk patterns as Henshin rules. We found it straight-forward to model risk patterns in the form of Henshin rules. Henshin rules provide a concise representation of risk patterns with well-defined syntax and semantics, which we found easy to understand.

Fig. 6 shows the realization of the example cloud system model with the help of our Sirius-based editor. The Sirius-based editor made it easy to create the cloud system model or to make changes to it.

After running MatchFinder, we can check both in the textual output and in the Sirius-based editor that exactly one instance of each of the two risk patterns was found, just like in [13]. Fig. 6 shows the output in the Sirius-based editor, where the objects of the cloud model participating in at least one found match are marked. It is important to note that, while Risk Pattern B contains an object of type `Component` (cf. Fig. 5), this matches the object of type `AtomicComponent` in the cloud model (cf. Fig. 6), since `AtomicComponent` is a sub-type of `Component` in the meta-model.

We also performed additional tests, where small changes were made to the cloud system model. In all these tests, all present instances of the risk patterns were found, as we verified manually. Furthermore, our validation study also showed that the manipulation of all important artefacts (meta-model, cloud system model, risk patterns) was simple and intuitive with the implemented toolset, suggesting that the used tools are appropriate and together form a good technical basis for the management of data protection risks in cloud systems.

V. Empirical results

We performed controlled experiments to investigate the scalability of our approach. To generate larger models with a realistic structure, we enlarged the example cloud model from Section IV in a sequence of 10 steps. After each step we checked the duration needed by MatchFinder to search for (i) Risk Pattern A, (ii) Risk Pattern B, (iii) both risk patterns together. The enlarged models consist of several copies of the example model which share the same data subject node. The model of the first step consists of the basic example model with 18 nodes. Each further step adds the example model 100 times except the data subject node. Thus the model of the second step consists of $18 + 100 \cdot (18 - 1) = 1,718$ nodes and the model of the 10th step contains $18+10\cdot100\cdot(18-1) = 17,018$ nodes. The measurements were performed on a Surface Pro 3 laptop with Core i5-4300U CPU at 2900MHz, 4GB RAM, Windows 10 Professional, and Java 9.

Fig. 7 shows the duration of MatchFinder for the models of increasing size. As can be seen, the execution time scales roughly linearly with the size of the model. Looking for Risk Pattern A which consists of only 5 nodes needs less time than looking for Risk Pattern B which consists of 7 nodes. Looking for both risk patterns at the same time needs more time than looking for just one; however, if both risk patterns have to be looked for, then it is faster to look for them together than to look for them separately. This is probably due to the overhead of setting up the search itself, which is incurred only once if the two risk patterns are searched for together.

Most importantly, Fig. 7 shows that the duration is quite low even for the biggest models tested. In the first step, the search took 104 ms while the duration in the last step is 346 ms for a model with over 17,000 objects. Thus the duration increases by around 3.3 times while the size of the model increases by around 945 times. These findings indicate that the presented implementation scales well even for big cloud systems.

We also investigated how execution time scales with the number of edges in the model. We took one of the models from the previous experiments and added random edges to it, thus keeping the number of nodes constant while increasing the number of edges. We repeated this experiment with several starting models of different size. Also, we considered two approaches for adding new edges: (i) only adding edges inside the copies of the original example model of 18 nodes, (ii) adding edges between copies of the original example model. Our experience from these experiments is that increasing the number of edges hardly influences the time needed for the pattern matching. The duration remains low even for the biggest models considered (more than 100,000 edges).

VI. Related work

A large body of research addresses specific vulnerabilities of cloud systems and specific techniques to prohibit attacks. Examples include access control [10], trust management [11], security certification [2], and privacy-preserving analytics [14]. The risk pattern approach considered in this paper is fundamentally different from these works, as it focuses on the detection of cloud configurations with high data protection risks in general, instead of specific security controls.

The risk pattern approach was suggested in [13] and extended with a meta-model for cloud system models in [9]. A catalog of risk patterns modeling known attacks was presented in [12]. This paper is the first that presents an implementation and thus validates the applicability of the approach.

SecVolution [5] is also a model-based approach for detecting system configurations that may violate some security requirements, also using EMF and Henshin. However, SecVolution is focused on software, whereas the risk pattern approach includes all relevant layers of a cloud system. The risk pattern approach allows more accurate reasoning about cloud vulnerabilities. SecVolution aims at supporting manual enhancement of software by semi-automatically changing design artefacts, while the risk pattern approach aims at the fully automatic and quick detection of risky configurations.

Some works addressed the management of security and privacy risks in cloud settings. E.g., [4] elaborated on the necessity of quantified risk assessment for cloud-based processes, but also on the difficulties and challenges of such assessment, and [1] assessed the risks associated with migrating a process to the cloud. These approaches are complementary to our approach, as they can be used for determining the cloud configurations that must be avoided, which can be captured in the form of risk patterns.

VII. Conclusions

This paper has provided an implementation of the risk pattern approach for automatically detecting forbidden cloud configurations, thus validating the applicability of the approach, and leading to new insights about the requirements on the used pattern matching algorithm. Risk patterns can be represented as Henshin rules in a compact manner which is both human-readable and machine-readable. The approach is fast for even large models, allowing it to be used automatically at run time. An important area for further research is the automatic mitigation of the found risks by appropriate adaptations of the cloud configuration.

Acknowledgments. This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grant 731678 (RestAssured).

References

[1] A. A. L. Abdul Rahman, S. Islam, C. Kalloniatis, and S. Gritzalis, "A risk management approach for a sustainable cloud migration," *Journal of Risk and Financial Management*, vol. 10, no. 4, 2017.

[2] M. Anisetti, C. Ardagna, E. Damiani, and F. Gaudenzi, "A semi-automatic and trustworthy scheme for continuous cloud service certification," *IEEE Transactions on Services Computing*, 2017.

[3] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: advanced concepts and tools for in-place EMF model transformations," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2010, pp. 121–135.

[4] V. Bellandi, S. Cimato, E. Damiani, G. Gianini, and A. Zilli, "Toward economic-aware risk assessment on the cloud," *IEEE Security & Privacy*, vol. 13, no. 6, pp. 30–37, 2015.

[5] J. Bürger, D. Strüber, S. Gärtner, T. Ruhroth, J. Jürjens, and K. Schneider, "A framework for semi-automated co-evolution of security knowledge and system models," *Journal of Systems and Software*, vol. 139, pp. 142–160, 2018.

[6] Council of the European Union, "General Data Protection Regulation," http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf, 2016.

[7] Z. Á. Mann, "Resource optimization across the cloud stack," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 1, pp. 169–182, 2018.

[8] Z. Á. Mann and A. Metzger, "Optimized cloud deployment of multi-tenant software considering data protection concerns," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 2017, pp. 609–618.

[9] Z. Á. Mann, A. Metzger, and S. Schoenen, "Towards a run-time model for data protection in the cloud," in *Modellierung 2018*. Gesellschaft für Informatik e.V., 2018, pp. 71–86.

[10] M. Narouei, H. Khanpour, H. Takabi, N. Parde, and R. Nielsen, "Towards a top-down policy engineering framework for attribute-based access control," in *Proceedings of the 22nd ACM Symposium on Access Control Models and Technologies*. ACM, 2017, pp. 103–114.

[11] T. H. Noor, Q. Z. Sheng, L. Yao, S. Dustdar, and A. H. Ngu, "CloudArmor: Supporting reputation-based trust management for cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 367–380, 2016.

[12] A. Palm, Z. Á. Mann, and A. Metzger, "Modeling data protection vulnerabilities of cloud systems using risk patterns," in *International Conference on System Analysis and Modeling*, 2018, pp. 1–19.

[13] S. Schoenen, Z. Á. Mann, and A. Metzger, "Using risk patterns to identify violations of data protection policies in cloud systems," in *Service-Oriented Computing – ICSOC 2017 Workshops*. Springer, 2017, pp. 296–307.

[14] S. Sharma, J. Powers, and K. Chen, "PrivateGraph: Privacy-preserving spectral analysis of encrypted graphs in the cloud," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[15] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: eclipse modeling framework*, 2nd ed. Addison-Wesley, 2008.

[16] J. Tang, Y. Cui, Q. Li, K. Ren, J. Liu, and R. Buyya, "Ensuring security and privacy preservation for cloud data services," *ACM Computing Surveys*, vol. 49, no. 1, 2016.

[17] V. Viyović, M. Maksimović, and B. Perisić, "Sirius: A rapid development of DSM graphical editor," in *18th International Conference on Intelligent Engineering Systems (INES)*. IEEE, 2014, pp. 233–238.