Statistical Model Checking meets GDPR.*

Eduard Baranov$^{1[0000-0002-7357-705X]}$, Kim G. Larsen$^{2[0000-0002-5953-3384]}$, and Axel Legay$^{1[0000-0003-2287-8925]}$

[1] Université Catholique de Louvain
`firstname.lastname@uclouvain.be`
[2] Aalborg University
`kgl@cs.aau.dk`

Abstract. Software systems are incorporated into various aspects of human society. However, their integration brings a set of challenges, especially when software operates on personal data. The systems must be correct and provide the desired functionality while maintaining privacy and security of personal data. Verification techniques can support software system development and provide mathematical evidence of their correctness and security.
This work considers two recent applied and collaborative national/EU projects from different domains. Both projects involve processing personal data and sharing it among multiple individuals and organizations. Therefore, ensuring security and data privacy guarantees, as mandated by the General Data Protection Regulation (GDPR), is crucial. We explore the applicability of formal methods and demonstrate the utility of Statistical Model Checking to ensure security and privacy in real-world projects. The goal is to validate specific aspects of GDPR compliance for both projects.

Keywords: Statistical Model Checking · UPPAAL · GDPR

1 Introduction

In recent years software systems have been incorporated into almost every aspect of human society offering various improvements including efficiency augmentation, management simplification, and easier collaboration between different solutions or organisations. However, in many cases the development of such systems must be rigorous, the systems have to guarantee the satisfaction of a set of requirements. A classic example is critical systems: the consequences of error can be catastrophic. Yet in last years another set of requirements has become extremely important: security and privacy of personal data.

An immense number of software systems collect or process personal data. For example, healthcare systems work with medical data, Enterprise Resource

* This work was supported by Walloon Region (Belgium) projects DeepConstruct (Convention n°8560) and Cyber Excellence (Convention n°2110186) and the Villum Investigator Grant S4OS.

Planning systems store official documents including personnel identity documents. Unfortunately, data leaks caused by design issues, bugs in software, or system configurations became commonplace affecting millions of people. For example, in 2019 an error in API of the third largest Indian mobile operator exposed personal data of 300 million customers [2]. In 2020, badly designed coronavirus app in Slovakia was exposing data of almost 400000 users [3]. To reduce the risk of data leaks, correctness and security checks are necessary for a large spectrum of systems.

Data privacy guarantees are also required for legal reasons. In 2016 EU adopted (effective from 2018) a General Data Protection Regulation (GDPR) on information privacy [1]. GDPR imposes a set of requirements on systems processing personal data, in particular it requires to minimize the collected data and to utilise security measures for data protection. Several other countries such as Australia, India, Japan, etc. have adopted similar regulations.

Additional challenge arises from the movement towards distributed systems. Different parts of the system can be developed and maintained in different locations, sometimes in different countries. Data can be divided and stored in several organisations, while its processing might be performed by yet another organisation. Both distributions make data transfer inevitable. From the security and the data privacy perspective, such distributed setup requires to verify not only each component or data storage, but the system as a whole including all communication channels and interactions.

In this work we present our experience in providing data privacy guarantees in two projects from two domains, healthcare and construction. The first project is EU Horizon 2020 project Serums[3] [24]. The project goal is to improve healthcare provision in Europe through the proposal of a secure and transparent data sharing platform able to ensure privacy when accessing patient data [23]. Data sharing aspect is the centre of Serums. For example, a person travelling to another country might require medical services during the trip. The establishment of a correct diagnosis and the quality of a suggested treatment can require knowledge of a patient's medical history. Serums proposed a system for medical data sharing where a user can control the shared data by setting up access rules. Data privacy guarantees are extremely important not only for legal purposes but for user trust as well.

The second project is DeepConstruct[4] from the construction domain. Its goal is to build a management system for activities, equipment, and documents. The system facilitates the collaboration and information exchange between multiple organisations. In addition, the system automatically manages the communications with the government agencies providing them legally required documents and collecting their feedback. From the data protection perspective, correct access control is mandatory: DeepConstruct operates with fiscal information and personal documents, such as passports, work permissions, and diplomas.

[3] `www.serums-h2020.org`
[4] `www.multitel.be/projets/deep-construct`

From the perspective of our projects, we are focused on a data sharing aspect. According to Article 5 of GDPR, the systems must ensure "appropriate security of personal data, including protection against unauthorised or unlawful processing and against accidental loss" and demonstrate compliance with the requirement. Article 25 requires protection by design and by default, so that security is considered during the whole system life cycle. Security measures have to be considered at the system design level as well. Thus, verification is an important element of checking data security at the design level and its demonstration with mathematical evidence of the result. In addition, Article 25 requires minimal necessary accessibility of the data by default. For our projects this requirement can be formulated as follows:

Property 1. The access to a newly created document with user's personal data is limited to the user and a person or an organisation responsible for the user.

In case of Serums, access could be granted by default to a personal doctor and a hospital where the data is stored. For DeepConstruct that would be a company the user is working at and a manager of a construction site. The introduction of security measures against unauthorised access is required by Article 32. In particular, personal data should not be shared with other persons without the individual's intervention. However, legal obligations allow the processing and sharing as stated in Article 6. For example, worker documents must be approved by Social Security, thus the documents must be shared with them. As a result, there is a need not for a general consent for data sharing but for a fine-grained access control. This gives us the following property for verification:

Property 2. Without legal obligation data cannot be accessed by users or organisations unless the access has been granted by the data owner.

Several GDPR articles impose a set of regulations for transferring data to third countries, but such scenarios are not considered in the projects, therefore we leave them out of the scope of this work.

Data privacy is an important property and multiple techniques can be applied to ensure its satisfaction. Various testing methods are extremely useful for checking system's implementation. Considering the protection by design required by GDPR Article 25, formal methods and, in particular, model checking are extremely helpful to ensure correctness of system's design and to provide mathematical evidence for the result. Different techniques can be used for the verification, however exhaustive model checking can terminate on a small much simplified subsystem, the check of the complete system is infeasible [8]. Therefore, we decided to apply Statistical Model Checking (SMC) that is a simulation-based technique with high scalability. SMC can be efficiently used with highly detailed models of complete systems.

There exist a few works that formalise and verify systems compliance with some parts of GDPR. In [32], authors with the help of legal experts built a UML model of the GDPR requirements. Another set of rules representing the entire set of regulations has been created in RIO logic in [12]. However, in both works the application of the defined rules to the compliance checking is left as a future work. Authors of [11] built a Uppaal model of a simple online shop and formulated the properties checking the presence of user's consent and encryption for data transfer. In [10] smart contracts have been used to monitor the presence of consent and encryption during data transfer for IOT devices. A runtime monitoring of GDPR compliance is proposed in [4].

In this work we are providing the experience of checking GDPR compliance focusing on the data sharing aspect. We are considering a fine-grained access control rather than a general consent to do anything with the personal data. We show the formalisation and verification of the Properties 1 and 2 within two projects, Serums and DeepConstruct.

The paper is structured as follows. We start with background information on Statistical Model Checking in Section 2. Then we provide details on the systems designed for each project and their models in Section 3. Properties and details of their checking are described in Section 4. Section 5 concludes the paper.

2 Background

Formal methods is a set of techniques capable of verifying system properties and providing mathematical evidence of the result. They have been applied in multiple projects, e.g. [25, 16, 27, 21, 14]. Model checking is one of the formal methods utilising a formal representation of a system which is called a model. One of the common representations is based on transition systems [5] containing a set of systems' states and a set of transitions describing how the states can be changed during the system execution. Properties to be verified require a formal representation as well. They are usually expressed with one of the temporal logics [7] which extend Boolean logic with temporal operators allowing reasoning on sequences of events.

Statistical Model Checking (SMC) [22, 28, 17, 29, 30] is a verification method combining ideas from classical model checking and statistics. While exhaustive model checking explores the full state-space of the model that is susceptible to a state-space explosion problem, SMC uses simulations for the state-space exploration. The core idea of SMC is to run a large number of simulations on which a property under validation is checked and to use statistical methods to compute the probability of the property being satisfied. Confidence level of the result is a parameter that can be selected at the beginning of the verification. Being simulation-based, SMC verification with high degree of confidence is fast and requires a small amount of memory, as a result it scales to real-world projects. Multiple projects in different areas used SMC for verification, e.g. [9, 13, 20, 26, 31, 33].

Uppaal [15] and its SMC extension Uppaal SMC [18] are efficient tools for modelling and verification. Systems are modelled with a network of stochastic timed automata. Transitions in automata are equipped with a set of labels controlling its behaviour: guards enabling or disabling the transition depending on the current evaluation of variables, simple C-like functions updating the variables during the state change, and channels that are used for synchronisation between automata. Transitions of multiple automata labelled with the same channel are synchronised. Note that for Uppaal SMC only broadcast channels can be used. Shared variables are used for data transfer between automata. To model several automata with identical behaviour, for example users of the system, Uppaal defines a model with a set of templates. Each template can instantiate one or several automata which share common behaviour but act independently during simulation. This feature simplifies modelling of different scenarios, for example the number of users can be easily changed.

For the specification of properties Uppaal SMC uses queries written with an extension of Metric Interval Temporal Logic (MITL). Its SMC engine runs a large number of simulations and checks queries on each of trace. Basic temporal operators are $\Box\ p$ and $\Diamond\ p$ checking that p holds in all or at least one state in the trace respectively. There several types of queries supported by Uppaal SMC. The first query type computes a probability of a property to be satisfied. It is specified with a formula $Pr[\# \leq N]\ F$, where F is a property specified with MITL, N is a maximal trace length and # indicates that trace length is computed as a number of transitions taken rather than elapsed time. The result of such query would be an interval $[x - \epsilon, x + \epsilon]$ with a confidence α, where ϵ and α are selected parameters. Another query type that we use in this work computes an expected value of an expression $Expr$ and is specified with a formula: $E[\# \leq N,\ X]\ Expr$, where N is a maximal trace length and X is a number of simulations. We used an expression $max:\ var$ that returns a maximal value of a variable var encountered during the simulation. This request is useful to determine which trace length is sufficient for multiple expected visits of a system part that is currently being checked.

3 Modelling Systems

Architectures in both SERUMS and DeepConstruct projects consist of several components interacting with each other and with users. Their modelling is straightforward: behaviour of each component is modelled with one or several automata and their interactions are modelled with channels and shared variables. Users are included in the model; several templates are created for modelling different user roles in the systems. In this section we provide a brief description of models created for the two projects.

3.1 Serums model

Serums addresses the need to securely share medical data to allow healthcare provision across different healthcare providers. The high-level architecture of the Serums platform is shown in Fig. 1. Smart Health Centre System (SHCS) is a central component managing interactions between users and other components. In order to interact with the system, users have to sign-up and login. An authentication component is responsible for these interactions and is built

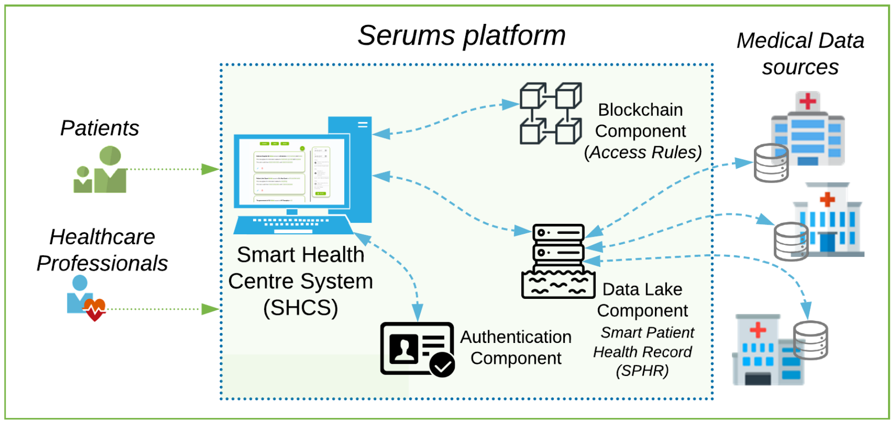

Fig. 1: Serums architecture [8].

from two sub-components, front-end and back-end. The former interacts with users and controls the execution of authentication steps while the latter performs background checks and is not supposed to interact with users directly. Patients can specify who could access their data via fine-grained access rules that are stored and managed in smart contracts in a Blockchain component. For example, access to general information such as name and blood type can be granted to all medical personnel while specific test results can be visible to the treating doctor only. Healthcare professionals[5] can request patients' data. Such requests are first transferred to the Blockchain component in order to verify access level. If some data is allowed to be retrieved, the request is passed to a Data Lake component that is responsible for collecting and processing of requested patient's data from connected hospitals.

During the project we built a model of the Serums system and continuously evolved it following the introduction of new features to the system. The model has 11 templates: 7 are modelling Serums components and the remaining 4 represent users of the system and hospitals. There are multiple users and hospitals interacting with the system, therefore there are multiple automata in the model instantiated from their templates.

Fig. 2 shows the automaton for the central Serums component SHCS mediating between users and the rest of the platform. Component's behaviour can be summarised with several sequences of interactions starting from the state in the centre of the automaton. The left part of the figure corresponds to the login procedure that starts with a request from a patient or a doctor and checks the presence and the validity of an authentication token. If the check fails, the user is transferred to the Authentication component. The right part of the figure is responsible for the creation and modification of access rules: user requests are

[5] In the remainder of the paper we would reference healthcare professionals as doctors.

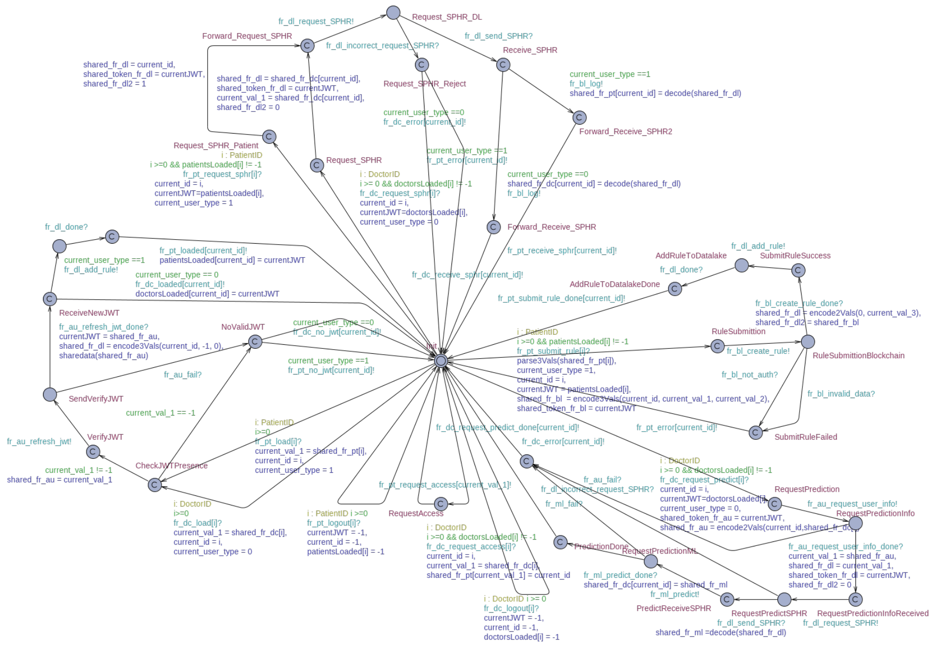

Fig. 2: Serums automaton of the SHCS component.

forwarded to the Blockchain component, and the result is returned to the user. The upper part performs requests for patient data. The request can be done by a patient for his/her personal data (GDPR Article 15). Alternatively, a doctor can request data for a patient. In both cases the Blockchain component checks access rules and the Data Lake collects the allowed information. The remaining transitions in the bottom correspond to logout procedures and doctors requesting data access from a patient.

Other parts of the Serums systems have been modelled as well at different level of abstraction. Blockchain has been abstracted to a blackbox, only its API have been modelled while access rights have been represented by a matrix. The Authentication component having a large number of different interactions has more details in the model and includes 4 automata. Fig. 3 shows an automaton for one user type representing a doctor. The left part of the automaton defines different interactions related to authentication and the right part corresponds to the data access operations.

3.2 DeepConstruct model

A goal of DeepConstruct is to build and verify a platform for managing personnel and resources in the construction domain. It is a centralised solution with an application server that performs most of the processing with a connected database. Several interfaces are provided to the users: a connected web platform (CWP)

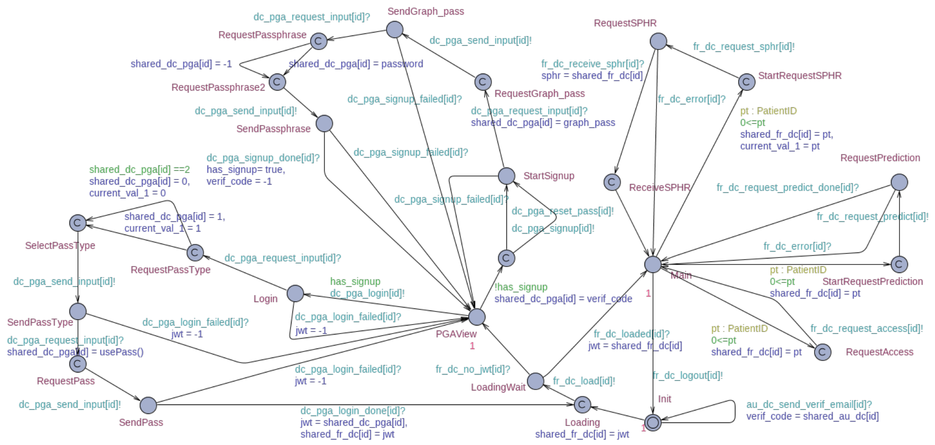

Fig. 3: Serums automaton of a doctor.

designed for organisation resource management and 3 mobile applications for the on-site management of presence control and work updates. Several types of users with different level of access are considered: a worker can manage and update his/her data in a mobile application, an on-site manager can have some access to the data of workers at the construction size, and a higher-level manager or human resources can manage all the workers of the organisation and have access to the CWP. In addition, the platform provides services for telemetry and geolocation of vehicles and tools that are collected and stored in the database as well as the automatization of interactions with external services such as Enterprise Resource Planning systems and governmental services. High-level architecture of DeepConstruct is shown in Fig. 4. One of the important features is an integration with an optical character recognition (OCR) tool. All the documents can be imported via scans or photos and OCR gathers data from the images. Document sharing between organisations is set up by access rights: by default, only one organisation can access a document and it can grant or revoke access to other organisations.

A model of DeepConstruct platform has 14 templates including 3 types of users, 3 mobile applications, and 2 external services. The majority of automata are simple and do not involve complex behaviour since almost all interactions are independent requests to the database via the application server. A more complex communication sequence involves document recognition with OCR. A user can upload an image or take a photo with one of the mobile apps, which in its turn would connect to the OCR service (authentication is done with JWT), and the recognised text would be validated and sent to the database. This sequence is shown in the top-left part of the mobile application automaton in Fig. 5. Other parts of the automaton correspond to the login procedure (right part) and time sheets submission (bottom-left part).

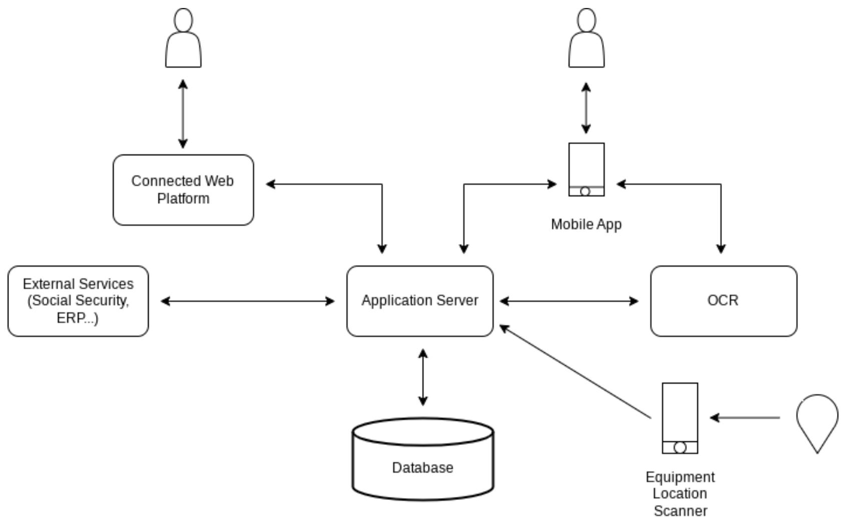
Connected Web Platform
Mobile App
External Services (Social Security, ERP...)
Application Server
OCR
Database
Equipment Location Scanner

Fig. 4: DeepConstruct architecture.

4 Verification of GDPR Properties

Developed models allow us to perform formal analysis of the systems. We use Uppaal SMC for the verification in the Serums and DeepConstruct projects. Contrary to the exhaustive model checking, SMC easily scales to large models. Queries for Uppaal SMC have to be expressed in MITL. In this work we consider GDPR related properties, focusing on the ones defined in Section 1. Both properties have been encoded with a set of queries. The structure of the encoding in similar in the two projects, however there are differences in the final formulas due to particularities of each project.

Property 1 verifying that by default personal data has minimal access rights can be checked in two ways. The first option applicable to our models is to check access rules. In the Serums project, access rules are stored in the Blockchain component that is abstracted to a matrix. Since all data is stored in local hospital databases rather than in the Serums platform, there is no notion of the new data creation, therefore we check that for each patient p patient's data can only be accessed by the personal doctor before any access rule has been added by the patient. The corresponding query is

$$Pr[\# <= N]\ (\Box\ (p.rulesCreated > 0\ ||\ \bigwedge_{d:D\setminus p.doctor} !\ blockchain.rules[p][d])),$$

where N is a desired length of a simulation trace, p is a patient, $rulesCreated$ is a counter incremented during an access rule creation, and the conjunction is taken over all doctors in the system excluding the personal doctor of p.

Contrary to Serums, in DeepConstruct documents are stored in the central database, therefore we check access rights after a new document is added to the

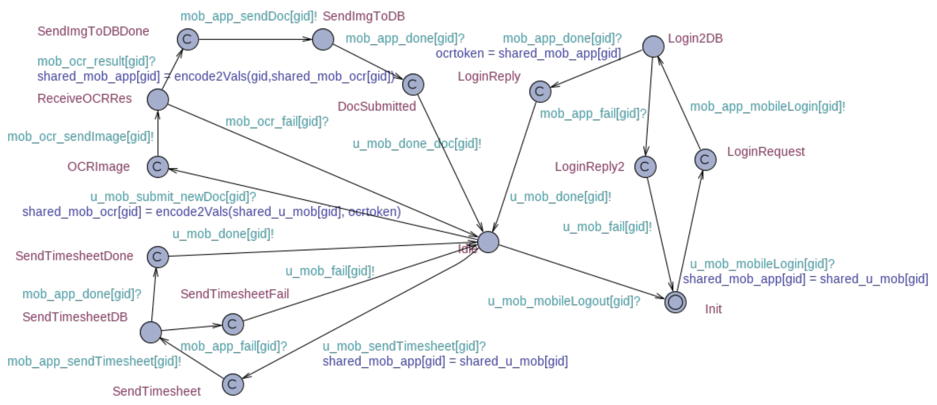

Fig. 5: DeepConstruct automaton of a main mobile application.

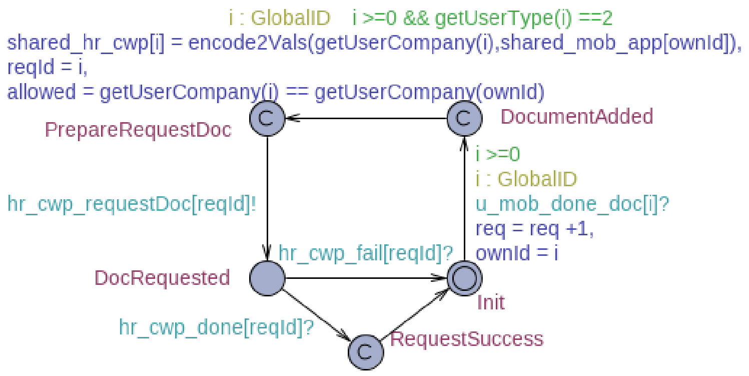

Fig. 6: Monitor automaton for DeepConstruct.

system. Whenever a user creates a document, it is expected that only user's organisation can access the document by default. This can be checked with a query

$$Pr[\# <= N]\ (\Box\ (!db.NewDoc\ ||\ \bigwedge_{c:Company} (c == getCompany(db.requestor)$$
$$||\ !\ db.hasAccess(c, db.lastDoc)))),$$

where db is a database automaton, $NewDoc$ is a state where a document is added, $getCompany$ is a function returning user's company, $hasDocAccess$ is a function checking access right, and $lastDoc$ is an id of the last document.

An alternative way to check Property 1 is to attempt to request data before making modifications to the access rules. This can be done by adding a "monitor" automaton that detects the addition of a new data and attempts to access it as one of the users. An example of the automaton for DeepConstuct is shown in Fig. 6. At the end of the document creation, the automaton remembers its owner and then selects a user at random and sends a request to retrieve the document. The *RequestSuccess* state is reached if the document have been received. We check the query $Pr[\# <= N]\ (\diamond(mon.RequestSuccess\ \&\&\ !\ mon.allowed))$ where *allowed* is a boolean variable with the expected access right. It is possible to select users that are not supposed to have the access and check the reachability of the *RequestSuccess* state. In Serums, the corresponding automaton sends a set of requests at the beginning of the simulation.

Note that the first type of queries without the additional monitor only checks the correct access rights but not the privacy of the data. Indeed, it is necessary to ensure that the system respects access rules and it is not possible to violate them. This is guaranteed if the Property 2 checking the access control is satisfied. For the verification we add a query per user that can request some data; the query checks data received by the user. In case of Serums we have the queries with the following structure:

$$\begin{aligned} Pr[\# <= N]\ (\square\ (!d.ReceiveSPHR\ ||\ (d.data! = -1\ \&\& \\ blockchain.rules[getOwner(d.data)][d]))), \end{aligned}$$

where d is a doctor, $ReceiveSPHR$ is a state after receiving data, comparison with -1 checks that the received data is not empty, and *getOwner* function returns the patient to which the data belongs. In case of a patient's request, instead of access rules we check that the patient is the owner of the data. In addition, we check that granted access allows the user to receive data by checking reachability of such state:

$$Pr[\# <= N]\ (\diamond\ (d.ReceiveSPHR\ \&\&\ (d.data! = -1))).$$

Note that we do not add a constraint with access rule since the state can only be reached with granted access assuming that the preceding query is verified. Queries for DeepConstruct are similar and can be obtained by replacing automata names, states, and the access rule format with the corresponding values.

The queries above cover access control for the requests sent by official API and check the correctness of the system under verification. However, security aspect must also be considered. We need to check that malicious users that are not restricted with normal behaviour cannot access personal data. For this verification we make two assumptions. The first one is that no one has unauthorised access to the personal data outside of the system under verification, for example physical access to its database. Indeed, any security measure in the system would not be able to prevent such access. The second assumption is that authentication credentials are not leaked. If this assumption is violated, a malicious user can login with leaked credentials and access personal data.

In order to add security checks to the verification process, we added malicious users or attackers to the models. For example, such user would attempt to login to the system without knowing correct credentials. A corresponding query to check that such user cannot access any data is a simple reachability of a state of successful login to the system. Another part of verification is related to the utilisation of ”internal” API, i.e. non-public communication endpoints used for interaction between different components. In our projects, systems are distributed and part of the communication goes not inside an internal network but through internet, therefore such endpoints can potentially be discovered. Therefore, it is important to check that the components perform sender authentication. Attacker models for this part of verification attempt to send requests to such endpoints (we add a transition to the recipient automaton to synchronise with the attacker request). As for a previous attacker, a query checks reachability of the state where some data is received. This part of the verification is especially important for Serums since the Blockchain component storing access rules and the Data Lake component collecting data can have different locations. In DeepConstruct due to its centralised architecture, there are only a few internal endpoints to check.

We checked the desired properties in the projects with the formalised queries. To find out the required simulation trace length, we added simple counters that are incremented an action important for the verification is taken (e.g. $rulesCreated$ in the query for Property 1 or req variable in Fig. 6) and checked a query

$$E[\# <= N;\ X]\ (max\ :\ counter),$$

which outputs an expected value of the counter at the end of the trace of length N after X simulations. We used $X = 100$. By testing various lengths, we note how often the actions are taken and select trace length accordingly. For our projects traces of lengths between 10000 and 20000 (depending on a query) were considered sufficient. Another technique we used during verification is modification of the initial state of the system. For example, for the GDPR properties we assumed that all users are initially logged in to the systems, thus making the transitions corresponding to data creation and access to appear earlier in simulations.

On a laptop with i7-8650U CPU, each query check takes about 1 minute on DeepConstruct model and about 4 minutes on Serums model with a confidence 0.999. For both project the verification required just 50MB of RAM. Note that exhaustive model checking would quickly run out of any reasonable amount of memory on such big models. Uppaal SMC returns an interval for the probability of the property to be satisfied. The interpretation of the result depends on a property type. For safety queries such as ones used to check Property 1, the interval must have the right bound equal to 1 in order to be considered satisfied. A different result would mean that there has been at least one simulation on which the query was violated. For liveness queries checking reachability of some state the interval must not be close to 0, so that at least one simulation reached the state.

For the security part of the verification, we found a potential issue in one of the early designs of the Serums system. A data request had been split into two steps: at the first step access rights were requested from the Blockchain component, and the response was added to the second request to the Data Lake component. If an attacker has had a capability to request the Data Lake component directly, it would have had possible to forge the Blockchain response and to obtain access to all data. In the latter versions, the Data Lake component was responsible for the access rights requests, thus removing the possibility to forge the Blockchain reply.

5 Conclusion

While software systems aim to improve human society, their utilisation of personal data is inevitable. Therefore, verification becomes an essential part of software development to ensure correctness of systems and privacy of data. Regulations such as GDPR impose legal requirements to the software systems. In this work we showed how part of the requirements can be formalised with temporal logic. In particular we focus on properties related to data access and sharing. Statistical Model Checking being an efficient methodology can be applied to perform the verification on real-world projects and provide statistical guarantees of the result. For the future work we are going to explore the applicability of hyperproperties [19, 6] for the verification of GDPR requirements. Hyperproperties reason on sets of traces and can be used to express properties such as anonymity and side-channel information leakage. This may lead to a higher coverage of GDPR by Statistical Model Checking.

References

1. Regulation 2016/679 of the european parlament and of the council (general data protection regulation) (2016), `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679`
2. Indian airtel: Bug meant users' personal data was not secure (2019), `https://www.bbc.com/news/world-asia-india-50641608`
3. Critical vulnerability in my ezdravia application (2020), `https://nethemba.com/sk/kriticka-zranitelnost-v-aplikacii-moje-ezdravie-unik-databazy-pacientov-testovanych-na-covid-19/`
4. Arfelt, E., Basin, D., Debois, S.: Monitoring the gdpr. In: Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24. pp. 681–699. Springer (2019)
5. Arnold, A.: Finite transition systems - semantics of communicating systems. Prentice Hall international series in computer science, Prentice Hall (1994)
6. Arora, S., Hansen, R.R., Larsen, K.G., Legay, A., Poulsen, D.B.: Statistical model checking for probabilistic hyperproperties of real-valued signals. In: Legunsen, O., Rosu, G. (eds.) Model Checking Software - 28th International Symposium, SPIN 2022, Virtual Event, May 21, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13255, pp. 61–78. Springer (2022)
7. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
8. Baranov, E., Bowles, J., Given-Wilson, T., Legay, A., Webber, T.: A secure user-centred healthcare system: Design and verification. In: 10th International Symposium From Data to Models and Back (2021)
9. Baranov, E., Given-Wilson, T., Legay, A.: Improving secure and robust patient service delivery. In: Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles. pp. 404–418. Springer (2020)

10. Barati, M., Rana, O., Petri, I., Theodorakopoulos, G.: Gdpr compliance verification in internet of things. IEEE access **8**, 119697–119709 (2020)
11. Barati, M., Theodorakopoulos, G., Rana, O.: Automating gdpr compliance verification for cloud-hosted services. In: 2020 International Symposium on Networks, Computers and Communications (ISNCC). pp. 1–6 (2020)
12. Bartolini, C., Lenzini, G., Santos, C.: An agile approach to validate a formal representation of the GDPR. In: Kojima, K., Sakamoto, M., Mineshima, K., Satoh, K. (eds.) New Frontiers in Artificial Intelligence - JSAI-isAI 2018 Workshops, JURISIN, AI-Biz, SKL, LENLS, IDAA, Yokohama, Japan, November 12-14, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11717, pp. 160–176. Springer (2018)
13. Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. International Journal on Software Tools for Technology Transfer **14**(1), 53–72 (2012)
14. ter Beek, M.H., Borälv, A., Fantechi, A., Ferrari, A., Gnesi, S., Löfving, C., Mazzanti, F.: Adopting formal methods in an industrial setting: the railways case. In: International Symposium on Formal Methods. pp. 762–772. Springer (2019)
15. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Formal methods for the design of real-time systems. pp. 200–236. Springer (2004)
16. Cerone, A., Elbegbayan, N.: Model-checking driven design of interactive systems. Electronic Notes in Theoretical Computer Science **183**, 3–20 (2007)
17. D'Argenio, P.R., Legay, A., Sedwards, S., Traonouez, L.: Smart sampling for lightweight verification of markov decision processes. Int. J. Softw. Tools Technol. Transf. **17**(4), 469–484 (2015)
18. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal SMC tutorial. International Journal on Software Tools for Technology Transfer **17**(4), 397–415 (jan 2015)
19. Dobe, O., Schupp, S., Bartocci, E., Bonakdarpour, B., Legay, A., Pajic, M., Wang, Y.: Lightweight verification of hyperproperties. In: André, É., Sun, J. (eds.) Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14216, pp. 3–25. Springer (2023)
20. Ellen, C., Gerwinn, S., Fränzle, M.: Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains. International Journal on Software Tools for Technology Transfer **17**(4), 485–504 (2015)
21. Harrison, M.D., Masci, P., Campos, J.C.: Formal modelling as a component of user centred design. In: Federation of International Conferences on Software Technologies: Applications and Foundations. pp. 274–289. Springer (2018)
22. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Implementations, LNCS, vol. 2937, pp. 73–84. Springer Berlin Heidelberg (2004)
23. Janjic, V., Bowles, J., Vermeulen, A., et al.: The serums tool-chain: Ensuring security and privacy of medical data in smart patient-centric healthcare systems. In: 2019 IEEE Int. Conf. on Big Data. pp. 2726–2735 (December 2019)
24. Janjic, V., Vinov, M., Given-Wilson, T., Legay, A., Blackledge, E., Arredouani, R., Stylianou, G., Huang, W., Bowles, J.K.F., Vermeulen, A.F., Silvina, A., Belk, M., Fidas, C., Pitsillides, A., Kumar, M., Rossbory, M.: The SERUMS tool-chain: Ensuring security and privacy of medical data in smart patient-centric healthcare systems. In: Baru, C.K., Huan, J., Khan, L., Hu, X., Ak, R., Tian, Y., Barga, R.S.,

Zaniolo, C., Lee, K., Ye, Y.F. (eds.) 2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019. pp. 2726–2735. IEEE (2019)

25. Jetley, R., Iyer, S.P., Jones, P.: A formal methods approach to medical device review. Computer **39**(4), 61–67 (2006)
26. Kalajdzic, K., Jégourel, C., Lukina, A., Bartocci, E., Legay, A., Smolka, S.A., Grosu, R.: Feedback control for statistical model checking of cyber-physical systems. In: International Symposium on Leveraging Applications of Formal Methods. pp. 46–61. Springer (2016)
27. Kwiatkowska, M., Lea-Banks, H., Mereacre, A., Paoletti, N.: Formal modelling and validation of rate-adaptive pacemakers. In: 2014 IEEE International Conference on Healthcare Informatics. pp. 23–32. IEEE (2014)
28. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: International Conference on Runtime Verification. pp. 122–135. Springer (2010)
29. Legay, A., Lukina, A., Traonouez, L.M., Yang, J., Smolka, S.A., Grosu, R.: Statistical model checking. In: Computing and Software Science, pp. 478–504. Springer (2019)
30. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: 17th International Conference on Computer Aided Verification, LNCS, vol. 3576, pp. 266–280. Springer Berlin Heidelberg (2005)
31. Ter Beek, M.H., Legay, A., Lafuente, A.L., Vandin, A.: A framework for quantitative modeling and analysis of highly (re) configurable systems. IEEE Transactions on Software Engineering **46**(3), 321–345 (2018)
32. Torre, D., Soltana, G., Sabetzadeh, M., Briand, L.C., Auffinger, Y., Goes, P.: Using models to enable compliance checking against the gdpr: An experience report. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS). pp. 1–11 (2019)
33. Zuliani, P.: Statistical model checking for biological applications. International Journal on Software Tools for Technology Transfer **17**(4), 527–536 (2015)