Cerberus: Automated Synthesis of Enforcement Mechanisms for Security-Sensitive Business Processes

Luca Compagna[2], Daniel Ricardo dos Santos[1,2,3(✉)], Serena Elisa Ponta[2], and Silvio Ranise[1]

[1] Fondazione Bruno Kessler (FBK), Trento, Italy
dossantos@fbk.eu
[2] SAP Labs France, Mougins, France
[3] University of Trento, Trento, Italy

Abstract. Cerberus is a tool to automatically synthesize run-time enforcement mechanisms for security-sensitive Business Processes (BPs). The tool is capable of guaranteeing that the execution constraints EC on the tasks together with the authorization policy AP and the authorization constraints AC are satisfied while ensuring that the process can successfully terminate. Cerberus can be easily integrated in many workflow management systems, it is transparent to process designers, and does not require any knowledge beyond usual BP modeling. The tool works in two phases. At design-time, the enforcement mechanism M, parametric in the authorization policy AP, is generated from EC and AC; M can thus be used with any instance of the same BP provided that EC and AC are left unchanged. At run-time, a specific authorization policy is added to M, thereby obtaining an enforcement mechanism M^* dedicated to a particular instance of the security-sensitive business process. To validate our approach, we discuss the implementation and usage of Cerberus in the SAP HANA Operational Intelligence platform.

1 Introduction

A security-sensitive business process (BP) [1] is a structured collection of tasks, defining a workflow, equipped with an authorization policy (AP) defining which users are entitled to execute which tasks, and authorization constraints such as Separation of Duties (SoD) defining that certain tasks must be executed by different users. The authorization policy and constraints are crucial to comply with regulations and prevent frauds. It is, however, of utmost importance to ensure that business continuity is not endangered, i.e. it must be possible to complete the process while satisfying the authorization policy and constraints.

As an example, consider the Voting process shown in Fig. 1. It is composed of four tasks (represented by rounded rectangles), *Request Voting* ($t1$), *Moderate e-mail discussion* ($t2$), *Moderate conference call* ($t3$) and *Validate Voting* ($t4$),

This work has been partly supported by the EU under grant 317387 SECENTIS (FP7-PEOPLE-2012-ITN).

M. Chechik and J.-F. Raskin (Eds.): TACAS 2016, LNCS 9636, pp. 567–572, 2016.
DOI: 10.1007/978-3-662-49674-9_36

and two SoD constraints (dashed lines labeled by $\neq$), which impose that the user who executes $t2$ ($t3$, resp.) cannot also execute $t3$ ($t4$, resp.). Examples of valid execution scenarios, i.e. assignments of users to tasks such that the process can terminate and constraints are satisfied, are $t1(C), t2(A), t3(B), t4(A)$ and $t1(A), t3(B), t2(C), t4(A)$. Though in this simple process it is easy to determine whether there exist valid execution scenarios, for complex BP with more constraints and expressive policies this is not the case. Establishing whether all tasks can be executed while satisfying the authorization policy and without violating any authorization constraints is known as the Workflow Satisfiability Problem (WSP), whose solution is NP-hard already in presence of one SoD constraint [9]. The problem becomes even more complex if we consider the run-time version of WSP that consists in answering user requests to execute a task while ensuring successful termination together with the satisfaction of authorization constraints. As an example consider that at run-time $t1$ has been performed by A, and B is requesting to execute $t2$. Although B is entitled to do so by the authorization policy and $t2$ is not in SoD with any task B executed in the past, granting this request would break business continuity. In fact A would be the only user entitled to execute $t3$ because of the SoD between $t2$ and $t3$, but then no user would be able to execute $t4$ without violating the SoD with $t3$. In [3] a technique was introduced to automatically synthesize, from security-sensitive BPs, enforcement mechanisms that solve the run-time WSP.

In this paper we present Cerberus [1], a tool that relies on [3] to automatically synthesize at design-time enforcement mechanisms capable of guaranteeing at run-time that the workflow can terminate while satisfying the authorization policy and the authorization constraints. The synthesized mechanisms are parametric in the authorization policy so that they can be combined at run-time with authorization policies dedicated to different instances of the process. Cerberus can be easily integrated in many workflow management systems, it is transparent to process designers, and does not require any knowledge beyond usual BP modeling. To demonstrate the tool, we integrated it into the SAP HANA Operational Intelligence platform[2] (OpInt) which offers a BPMN modeling and enactment environment.

2 Tool Architecture and Implementation

A reference architecture for Workflow Management (WFM) systems [10] is composed of the five blue elements shown in Fig. 2. *Workflow Modeling* is a graphical user interface for a *Process Designer* to create workflow models using a modeling language such as BPMN or YAWL (see, e.g., [10]). The models are stored in a *Workflow Model Repository*, while the *Workflow Engine* interprets the models

[1] Cerberus is a three-headed watchdog in Greek mythology, with the first head associated to the past, the second to the present and the third to the future. Cerberus acts as a monitor that takes into account the history of executions, the current authorization relation and future executions to grant or deny requests.

[2] https://help.sap.com/hana-opint.

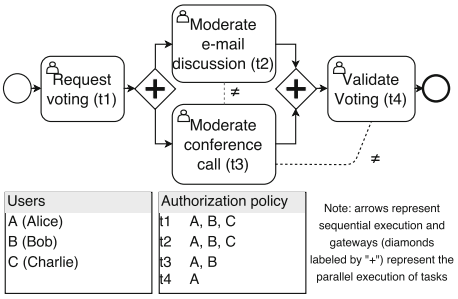

Fig. 1. Voting process

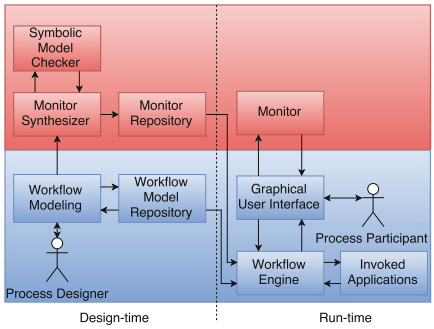

Fig. 2. Tool architecture (Color figure online)

and directs the execution to the *Invoked Applications*, in the case of system and script tasks, or to a *Graphical User Interface* (GUI), in the case of user tasks, which are performed by *Process Participants*.

On top of the WFM components, we add the CERBERUS components shown in red in Fig. 2. The *Monitor Synthesizer* is responsible for interpreting the workflow model and translating it into a transition system format accepted by a *Symbolic Model Checker* capable of computing a reachability graph whose paths are all possible executions of the workflow. Note that only the workflow model (representing the execution constraints) extended with authorization constraints is input to the monitor synthesis. This allows the synthesized monitor to support different authorization policies at run-time. The reachability graph is translated into a language such as Datalog or SQL and stored in the *Monitor Repository*. The *Monitor* itself sits between the GUI and the workflow engine and grants or denies user requests to execute tasks (users only access tasks through the GUI and automatic tasks are not part of the authorization policy or constraints[3]).

The main goals in the design of CERBERUS are usability, scalability and minimal interference with pre-existing functionalities. Usability is achieved because the tool is fully automated and all the formal details are hidden from the process designer, who only has to input the workflow model with a set of constraints that he/she wishes to be enforced (which can be done graphically). Scalability is ensured by the use of modular monitor synthesis (decomposing workflows into components, synthesizing monitors for them and combining the results [4]) and minimal interference is guaranteed by using the tool as a plug-in, so that both monitor synthesis and enforcement can be easily activated or deactivated.

The CERBERUS implementation is built on top of OpInt to synthesize, store, combine and retrieve run-time monitors for security-sensitive workflows therein modeled and enacted. HANA Studio is the IDE that acts as the *Workflow Modeling* component, while the HANA Repository implements both the *Workflow Model Repository* and the *Monitor Repository*. We added the constraint specification and monitor synthesis capabilities in the IDE and used MCMT [6] as the

[3] This is a limitation of the current implementation. Nonetheless the approach is able to monitor any task subject to an authorization policy.

Symbolic Model Checker. The *Monitor Synthesizer* is written in Python (core algorithms) and JavaScript (IDE and repository integration). The monitors are output in SQL as a view that is queried by the execution engine. The *Workflow Engine* differs from traditional WFM systems because OpInt does not directly execute the BPMN models, but instead translates them to executable artifacts (JavaScript and SQL code) that manage and perform the tasks in the workflows. The invoked applications are handled by SQL procedure calls and the GUI for user tasks is integrated in a web task management dashboard.

Since we build on top of a reference architecture, other possible implementations of CERBERUS could use open-source versions of the WFM components. The advantage of OpInt is to have all the components in the same platform.

3 Using Cerberus

The usage of the tool involves four steps: design-time specification, monitor synthesis, deployment, and run-time enforcement. SAP HANA is an in-memory relational database, so the BPMN artifacts and the monitors are translated to SQL. There is a long tradition of works using relational languages, such as Datalog and SQL, to express role-based access control and other authorization policies [7]. Moreover, we use database tables to store the users (`USERS`), authorization policy (`AP`) and execution history (`HST`).

At **design-time**, a process designer uses the HANA Studio IDE to model the control-flow and authorization constraints of a workflow. Authorization constraints are not part of standard BPMN, and there are many proposed extensions to accommodate them, but we simply use task documentation to input the constraints in textual form. This can be changed in the future so that constraints are specified as graphical elements. Authorization policies are specified by linking each task to an assignment table in the database, which is only populated at deployment-time. When design is complete, the model is translated to SQL by pressing a button in the IDE.

To model the example of Fig. 1, a process designer uses the IDE to create a new BPMN file and graphically drags, drops and connects the required elements: start and end events (the circles in the figure), user tasks (rounded rectangles), sequence flows (solid arrows) and parallel gateways (diamonds labeled by +). The authorization constraints are input in the documentation of the second and third tasks, the authorization policy is linked to the `AP` database table (which is empty at the moment) and the task UIs are linked to web pages.

The **monitor synthesis** runs in parallel with the BPMN-to-SQL compiler and is completely transparent to end users. When the monitor synthesizer receives a request to generate a monitor, the BPMN model file (in XML) is read from the repository and translated to a symbolic transition system that is fed to the SMT-based MCMT model checker. The model checker applies a backward reachability procedure and returns a reachability graph that represents all possible executions of the workflow by symbolic users, which are introduced by the model checker itself to represent placeholders for concrete users that are specified at deployment time in the `USERS` table. The reachability graph is composed

of nodes labeled by first-order formulae representing sets of states and edges representing the execution of tasks by users, with each path representing a possible terminating execution of the system. The formulae encode the conditions that must be met for a user to execute a task and they use an interface to the authorization policy and history of execution that will be realized at run-time as the database tables `AP` and `HST`, respectively. The monitors are thus parametric in the authorization policy, which means that the same monitor needs to be generated only once for each workflow model, regardless of the run-time policy that is deployed with it. In [3], a procedure is described which takes a symbolic transition system and returns a Datalog program whose clauses are a conjunction of literals built out of the state variables in the transition system such that user u can execute task t and the workflow can successfully terminate iff $can\_do(u, t)$ is a logical consequence of the Datalog program with a specific authorization policy and history. The Datalog monitors are then further translated to SQL views that can be queried by the execution engine (aggregation-free SQL and non-recursive Datalog with negation are equivalent and the translation is straightforward [8]). The resulting SQL view, using the database tables representing users, authorization policy and history of execution, is stored in the repository and queried at run-time by the execution engine. The synthesized monitors are modular and can be composed to form more complex monitors, as described in [4]. This allows us to alleviate the state space explosion problem and handle large workflows by decomposing them into smaller modules.

In the example of Fig. 1, the monitor consists of an SQL view defined by a procedure containing, among others, the following query for $t2$ (simplified for the sake of clarity):

```
SELECT U2.ID FROM USERS AS U1, USERS AS U2 WHERE HST.dt1 = 1 AND HST.dt2 = 0
    AND HST.dt3 = 1 AND HST.dt4 = 0 AND (U1.ID <> U2.ID) AND NOT HST.t3by =
    U1.ID AND NOT HST.t3by = U2.ID AND U2.ID IN (SELECT ID2 FROM AP) AND U1.
    ID IN (SELECT ID4 FROM AP)
```

which encodes the fact that, to execute $t2$, the system must be in a state where $t1$ and $t3$ have been executed, but neither $t2$ nor $t4$ (`dt1 = 1 AND dt2 = 0 AND dt3 = 1 AND dt4 = 0`), there must be a user $u1$ who can execute $t2$ (`SELECT ID2 FROM AP`), and a different user $u2$ (`U1.ID <> U2.ID`) who can execute $t4$ (`SELECT ID4 FROM AP`) and neither user should have executed $t3$ because of the SoDs between $t2$ and $t3$ and between $t3$ and $t4$ (`NOT t3by = U1.ID AND NOT t3by = U2.ID`). Other queries for $t2$ and all queries for other tasks have been omitted for the lack of space.

For the **deployment** of a workflow it is necessary to specify the concrete authorization policy by populating the linked database tables. End users manage workflows using a generated API.

At **run-time**, there is a running job responsible for calling the next tasks based on tokens stored in the database, whose flow is specified by the control-flow in the BPMN model. When a human task is executed, the monitor associated to the workflow is called into action by the automatic invoking of a procedure from the task UI. To grant or deny a request, the monitor queries the `USERS`, `AP` and `HST` tables described above to ensure that the requesting user is entitled

to perform the task, the user has or has not performed another conflicting task, and the execution of this task will not prevent the satisfaction of the workflow (as shown in the example query above).

Examples of valid execution scenarios and run-time enforcement for the process in Fig. 1 are given in the Introduction.

4 Discussion

CERBERUS is under development and it has been validated with real-world and synthetic examples [3,4]. Currently, the tool is not available for public use, but business units at SAP showed interest in the OpInt integration and discussion about pilot projects with customers is going on. It is possible to use the CERBERUS architecture with other components, as described in Sect. 2, and we already have implementations of the *Monitor Synthesizer* for Prolog, pyDatalog and MySQL, but there is no integration with other WFM systems. This work is related to runtime verification [5] and tools that address the WSP. The closest related work is [2], which presents a workflow monitor that considers policies and constraints and uses pre-existing IBM components; it does not, however, solve the WSP. As future work, we intend to encourage and study the use of the tool in more real-world scenarios and, leveraging the ideas in [4], build a repository of components with synthesized monitors that can be reused by business designers.

References

1. Armando, A., Ponta, S.E.: Model checking of security-sensitive business processes. In: Degano, P., Guttman, J.D. (eds.) FAST 2009. LNCS, vol. 5983, pp. 66–80. Springer, Heidelberg (2010)
2. Basin, D., Burri, S.J., Karjoth, G.: Dynamic enforcement of abstract separation of duty constraints. ACM TISSeC **15**(3), 13:1–13:30 (2012)
3. Bertolissi, C., dos Santos, D.R., Ranise, S.: Automated synthesis of run-time monitors to enforce authorization policies in business processes. In: ASIACCS (2015)
4. dos Santos, D.R., Ranise, S., Ponta, S.E.: Modularity for security-sensitive workflows. In arXiv (2015)
5. Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. Eng. Dependable Softw. Syst. **34**, 141–175 (2012)
6. Ghilardi, S., Ranise, S.: MCMT: a model checker modulo theories. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 22–29. Springer, Heidelberg (2010)
7. Samarati, P., de Vimercati, S.C.: Access control: policies, models, and mechanisms. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 137–196. Springer, Heidelberg (2001)
8. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. Theory Pract. Log. Program. **8**(2), 129–165 (2008)
9. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. TISSeC **13**, 40:1–40:35 (2010)
10. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer-Verlag New York Inc., Secaucus (2007)