Automatic Security Analysis of Business Processes

Daniel Ricardo dos Santos\textsuperscript{1,2}

Advisors:
Silvio Ranise\textsuperscript{1}
Luca Compagna\textsuperscript{2}
Serena Ponta\textsuperscript{2}

\textsuperscript{1}Security and Trust - FBK
\textsuperscript{2}SAP Labs France

October 21st, 2014
Outline

1. Introduction
2. State of the art
3. Preliminary Results
4. Future Work
This work is part of the SECENTIS project and aims to apply the resulting tools on the SAP HANA database and cloud platform.
Context

- Business processes and process-aware applications need to enforce security policies in the form of complex authorization constraints.

- Separation/Binding of Duty and others related to the execution history or contextual information (e.g., location/time).

- Termination (WSP), authorization delegation, and resiliency.
Problem

- Developers may directly implement a policy in the application or use run-time enforcement monitors provided by the execution platform.

- We must verify that the policy enforced by the application and the intended policy, specified by the business rules, are compatible.

- We work on methods for synthesizing run-time monitors and analyzing database-backed web applications that realize workflows.
Research Goals

- Given a workflow specification and a set of authorization constraints (policy), generate a run-time monitor that enforces the policy.

- Given a process-aware application implemented in JavaScript+SQL and a set of authorization constraints (policy), detect and correct vulnerabilities in policy enforcement.
State of the art

- Workflow Satisfiability has been extensively studied, but not the synthesis of a full monitor for causality and authorization constraints [1, 4]

- Deutsch et al. [6, 7, 5] worked on the specification and verification of data-driven web applications and business processes with correctness properties specified in temporal logic, but no special attention to security

- Policy-weaving problem: taking as input a program, a high-level policy and a description of how system calls affect privilege; automatically rewrite the program in a way that it satisfies the policy [9, 8, 10]
Automated Synthesis of Run-time Monitors

- New methodology to automatically synthesize run-time monitors capable of ensuring the successful termination of workflows while enforcing authorization policies and SoD constraints.

- Divided in two parts: (i) specification and (ii) verification of security-aware workflows.

- Specification starts with Petri nets for the control-flow and security requirements, then derives a symbolic representation to be used by a model checker, considering a finite but unknown number of users.
Automated Synthesis of Run-time Monitors

- The verification part has an off-line and an on-line phase, in the off-line phase we compute all possible terminating executions of the workflow and in the on-line phase we use this information to synthesize a run-time monitor, that can be implemented in Datalog or SQL.

- Control-flow is DAG (no loops)

- Data-flow is completely abstracted
Architecture

Enforcement
(on-line)

Authorization module

All possible states leading to success

BR

Symbolic Model Checker

Workflow state s

Grant/deny

User u can execute task t?

Formula characterizing final states

Constrained workflow specification

Workflow engine with users

Security and Trust Retreat - FBK - October 21, 2014
Example - BPMN
Example - Petri net
### Example - Transition System

<table>
<thead>
<tr>
<th>event</th>
<th>enabled</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CF</td>
<td>Auth</td>
</tr>
<tr>
<td>$t_1(u)$</td>
<td>$p_0 \wedge \neg d_{t_1}$</td>
<td>$a_{t_1}(u)$</td>
</tr>
<tr>
<td>$t_2(u)$</td>
<td>$p_1 \wedge \neg d_{t_2}$</td>
<td>$a_{t_2}(u) \wedge \neg h_{t_3}(u) \wedge \neg h_{t_1}(u)$</td>
</tr>
<tr>
<td>$t_3(u)$</td>
<td>$p_2 \wedge \neg d_{t_3}$</td>
<td>$a_{t_3}(u) \wedge \neg h_{t_2}(u)$</td>
</tr>
<tr>
<td>$t_4(u)$</td>
<td>$p_3 \wedge \neg d_{t_4}$</td>
<td>$a_{t_4}(u) \wedge \neg h_{t_1}(u)$</td>
</tr>
<tr>
<td>$t_5(u)$</td>
<td>$p_4 \wedge p_5 \wedge p_6 \wedge \neg d_{t_5}$</td>
<td>$a_{t_5}(u) \wedge \neg h_{t_3}(u) \wedge \neg h_{t_2}(u)$</td>
</tr>
</tbody>
</table>
Example - State Space
Example - Monitor

- $U = \{a, b, c\}, R = \{r_1, r_2, r_3\}$
- $UA = \{(a, r_1), (a, r_2), (a, r_3), (b, r_2), (b, r_3), (c, r_2)\}$
- $TA = \{(r_3, t_1), (r_2, t_2), (r_2, t_3), (r_1, t_4), (r_2, t_5)\}$

<table>
<thead>
<tr>
<th>#</th>
<th>Token in</th>
<th>$h_{t_1}$</th>
<th>$h_{t_2}$</th>
<th>$h_{t_3}$</th>
<th>$h_{t_4}$</th>
<th>$h_{t_5}$</th>
<th>$(u, t)$</th>
<th>Resp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_0$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$(a, t_1)$</td>
<td>deny</td>
</tr>
<tr>
<td>1</td>
<td>$p_0$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$(b, t_1)$</td>
<td>grant</td>
</tr>
<tr>
<td>2</td>
<td>$p_1, p_2, p_3$</td>
<td>$b$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$(b, t_2)$</td>
<td>deny</td>
</tr>
<tr>
<td>3</td>
<td>$p_1, p_2, p_3$</td>
<td>$b$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$(a, t_2)$</td>
<td>grant</td>
</tr>
<tr>
<td>4</td>
<td>$p_4, p_2, p_3$</td>
<td>$b$</td>
<td>$a$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$(c, t_3)$</td>
<td>grant</td>
</tr>
<tr>
<td>5</td>
<td>$p_4, p_5, p_3$</td>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
<td>-</td>
<td>-</td>
<td>$(a, t_4)$</td>
<td>grant</td>
</tr>
<tr>
<td>6</td>
<td>$p_4, p_5, p_6$</td>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
<td>$a$</td>
<td>-</td>
<td>$(b, t_5)$</td>
<td>grant</td>
</tr>
<tr>
<td>7</td>
<td>$p_7$</td>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Results
Results
TestREx: a testbed for repeatable exploits

- A framework for packing and running applications with their environments; injecting exploits and monitoring their success; and generating security reports

- Provided with a corpus of example vulnerabilities

- Goal: A benchmark on which we can test the effectiveness of our techniques

- Developed in collaboration with Stanislav Dashevskyi
Future Work

- Overcome the limitations of our current monitor approach: control- and data-flow
- Test our results in SAP HANA, using workflows provided by them and their execution engine
- Work on policy analysis and policy-weaving for JavaScript
- Integrate TestREx with policy analysis and testing
Future Work - other ideas to be considered

- User-role assignment ensuring least privilege in workflows
- Purpose-based access control for workflows
Thank you!

dossantos@fbk.eu
TestREx: a testbed for repeatable exploits


[4] Jason Crampton, Michael Huth, and JimHuan-Pu Kuo. Authorized workflow schemas: deciding realizability through ltl(f) model checking. *International Journal on*


[10] Richard Joiner, Thomas Reps, Somesh Jha, Mohan