Technical Report

Nr. TUD-CS-2011-0149
June 6th, 2011

Defining Access Control Policies as Tracematches

Authors
Eric Bodden

Defining Access Control Policies as Tracematches

CASED Technical Report TUD-CS-2011-0149

Eric Bodden
Software Technology Group, Technische Universität Darmstadt
Center for Advanced Security Research Darmstadt (CASED)
bodden@acm.org

ABSTRACT

Tracematches are a programming language for runtime monitoring of Java programs. A tracematch declares a set of program events to observe, along with a regular expression. When the program events occur in the order defined by the expression, the tracematch "triggers", executing a user-defined piece of code.

In this work we show how tracematches can be used to define history-based access control policies.

1. INTRODUCTION

Tracematches [1] are a programming language for runtime monitoring of Java programs. In this work we show how tracematches can be used to define history-based access control policies. We argue that defining access control policies using tracematches allows for modular and expressive poilicy definitions.

1.1 Mutually Exclusive Roles

We use examples by Turkmen et al. [3]. As an example for "Mutually Exclusive Roles" (MER), the authors write:

> "A transaction comprises of three steps: creation, validation and checkout. No employee can perform all three operations required to complete a transaction"

In Figure 1 we show a tracematch for this policy. The tracematch defines three events (or "symbols") of interest: create, validate and checkout. Tracematches use AspectJ [2] pointcuts to give abstract names to concrete program events. In line 12, the tracematch defines a regular expression, stating that the subsequent body of code should execute whenever a checkout event follows a create event and then at least one validate event. In the example, the body of code just logs the MER violation to a file.

The code in Figure 2 expands on this example. In lines 15–20, the code asks a supervisor whether it is ok to override the check in this case. If permission is granted, then the tracematch proceeds with the original checkout action as originally planned.

```
1  tracematch(Transaction t, Employee e) {
2      sym create after returning:
3          call(* Transaction.new(..)) && target(t) &&
4          let(e, Session.current().getUser());
5      sym validate after returning:
6          call(* Transaction.validate(..)) && target(t) &&
7          let(e, Session.current().getUser());
8      sym checkout before:
9          call(* Transaction.checkout(..)) && target(t) &&
10         let(e, Session.current().getUser());
11
12     create validate+ checkout {
13         log("User "+e.getName()+" tried to create,"+
14             " validate and check out transaction "+t.getID());
15     }
16 }
```

Figure 1: Tracematch for Mutually Exclusive Roles

```
1  tracematch(Transaction t, Employee e) {
2      sym create after returning:
3          call(* Transaction.new(..)) && target(t) &&
4          let(e, Session.current().getUser());
5      sym validate after returning:
6          call(* Transaction.validate(..)) && target(t) &&
7          let(e, Session.current().getUser());
8      sym checkout around:
9          call(* Transaction.checkout(..)) && target(t) &&
10         let(e, Session.current().getUser());
11
12     create validate+ checkout {
13         log("User "+e.getName()+" tried to create,"+
14             " validate and check out transaction "+t.getID());
15         if(Supervisor.getSupervisor().ask(
16             "Grant "+e.getName()+" permission to check out"+
17             " although same user already created and"+
18             " validated transaction "+t.getID()+"?")) {
19             proceed(t,e);
20         }   else {
21             throw new PolicyViolationException();
22         }
23     }
24 }
```

Figure 2: Tracematch for Mutually Exclusive Roles with enforcement

1.2 Conflict of Interest

In a second example, Turkmen et al. state a conflict-of-interest property:

> Supervisors are responsible to audit transactions and write audit reports. A teller can act as a supervisor only for transactions that are not created, validated or checked out by herself.

Figure 3 shows an appropriate tracematch: if a user performs an audit on a transaction for which he/she has raised either a create, validate or checkout event before, then the tracematch triggers. In this case, the tracematch body calls method `findSupervisorNotInConflictWith` to find an appropriate supervisor to dispatch the audit to instead. The tracematch then automatically delegates the audit by calling `proceed` with that new supervisor.

2. CONCLUSION

We have shown that tracematches can capture access control policies in a declarative and modular way. Traceamtches can not only validate access control policies at runtime, they can even enforce them and can use additional logic to revolve policy violations gracefully and just in time.

3. REFERENCES

[1] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding Trace Matching with Free Variables to AspectJ. In *OOPSLA*, pages 345–364. ACM Press, October 2005.

[2] The AspectJ home page, 2003.

[3] Fatih Turkmen, Eunjin Jung, and Bruno Crispo. Towards run-time verification in access control. In *IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE, 2011.


```
1  tracematch(Transaction t, Employee e) {
2      sym create after returning:
3          call(* Transaction.new(..)) && target(t) &&
4          let(e, Session.current().getUser());
5      sym validate after returning:
6          call(* Transaction.validate(..)) && target(t) &&
7          let(e, Session.current().getUser());
8      sym checkout around:
9          call(* Transaction.checkout(..)) && target(t) &&
10         let(e, Session.current().getUser());
11     sym audit around:
12        call(* AuditSystem.audit(Transaction)) && args(t) &&
13        let(e, Session.current().getUser());
14
15     (create|validate|checkout)+ audit {
16         log("User "+e.getName()+" tried to audit "+
17             " her own transaction "+t.getID());
18
19         Supervisor s =
20             AuditSystem.findSupervisorNotInConflictWith(e);
21
22         log("Delegating audit to "+s.getName()+ " instead.");
23
24         proceed(t,s);
25     }
26 }
```

Figure 3: Tracematch to resolve a Conflict of Interest situation