

Language-Support for Correct and Reliable Enforcement of Access Control Policies

Peter Amthor

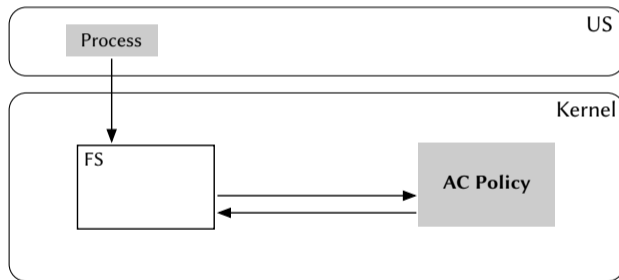
Distributed and Operating Systems Group
Technische Universität Ilmenau
peter.amthor@tu-ilmenau.de

2023-09-28



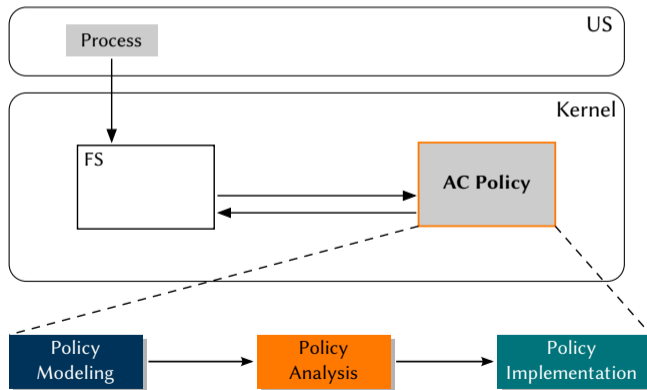
A Bit of Context

- Security as a mission-critical non-functional property of (system) software
- 📖 Reference monitor architecture
- 📖 *Mandatory access control (MAC) policy*



A Bit of Context

- Security as a mission-critical non-functional property of (system) software
- 📖 Reference monitor architecture
- 📖 *Mandatory access control (MAC) policy*



Typical Languages (and why they are used)

Languages

Modeling Specification Languages

- ... for attribute-based AC models (ABAC)
- e. g. XACML, *Polar*, SELinux policy language

Analysis Formal Languages

- ... to find e. g. privilege escalation or information flow vulnerabilities
- e. g. automata, flow graphs, FOL, PDL, CSP, ...

Implementation Programming Languages

- C

Their virtues:

- ① Adequate policy abstractions, e. g. file attributes, user roles, etc.
- ② Verifiability of policy correctness
- ③ Ergonomics to constructively avoid errors
- ④ Reliable enforcement, i. e. tamper-proof storage and non-circumventable interfaces of a policy

Typical Languages (and why they are used)

Languages

Modeling Specification Languages

- ... for attribute-based AC models (ABAC)
- e. g. XACML, *Polar*, SELinux policy language

Analysis Formal Languages

- ... to find e. g. privilege escalation or information flow vulnerabilities
- e. g. automata, flow graphs, FOL, PDL, CSP, ...

Implementation Programming Languages

- C

Their virtues:

- ① Adequate policy abstractions, e. g. file attributes, user roles, etc.
- ② Verifiability of policy correctness
- ③ Ergonomics to constructively avoid errors
- ④ Reliable enforcement, i. e. tamper-proof storage and non-circumventable interfaces of a policy

Typical Languages (and why they are used)

Languages

Modeling Specification Languages

- ... for attribute-based AC models (ABAC)
- e. g. XACML, *Polar*, SELinux policy language

Analysis Formal Languages

- ... to find e. g. privilege escalation or information flow vulnerabilities
- e. g. automata, flow graphs, FOL, PDL, CSP, ...

Implementation Programming Languages

- C

Their virtues:

- ① Adequate policy abstractions, e. g. file attributes, user roles, etc.
- ② Verifiability of policy correctness
- ③ Ergonomics to constructively avoid errors
- ④ Reliable enforcement, i. e. tamper-proof storage and non-circumventable interfaces of a policy

Typical Languages (and why they are used)

Languages

Modeling Specification Languages

- ... for attribute-based AC models (ABAC)
- e. g. XACML, *Polar*, SELinux policy language

Analysis Formal Languages

- ... to find e. g. privilege escalation or information flow vulnerabilities
- e. g. automata, flow graphs, FOL, PDL, CSP, ...

Implementation Programming Languages

- C

Their virtues:

- ① Adequate policy abstractions, e. g. file attributes, user roles, etc.
- ② Verifiability of policy correctness
- ③ Ergonomics to constructively avoid errors
- ④ Reliable enforcement, i. e. tamper-proof storage and non-circumventable interfaces of a policy

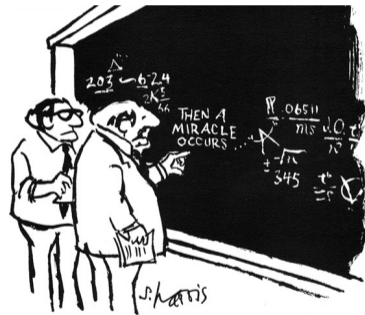
The Problem

Observations:

- Different levels of *abstraction*
- Different *expressiveness*
- Different *syntax and semantics*

→ Translations are *costly*

→ Translations are *error-prone!*



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

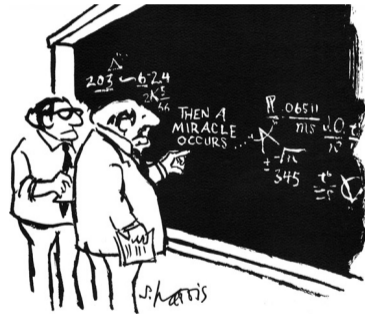
The Problem

Observations:

- Different levels of abstraction
- Different expressiveness
- Different syntax and semantics

→ Translations are *costly*

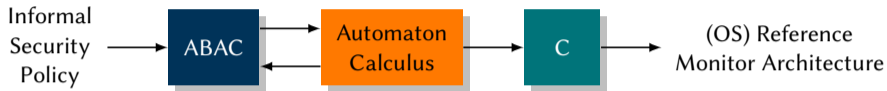
→ Translations are *error-prone!*



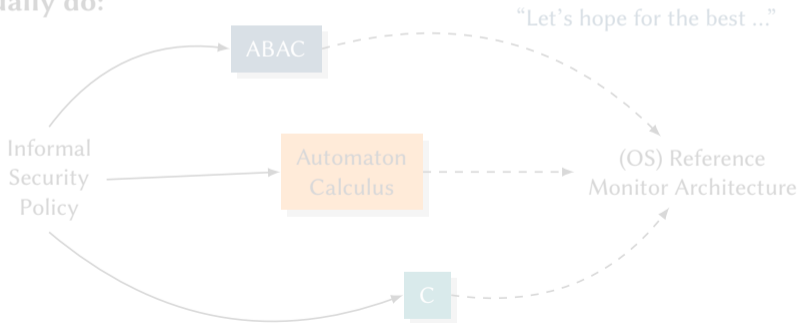
"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

The Problem

What we should do:

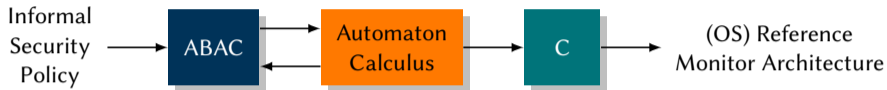


What we actually do:

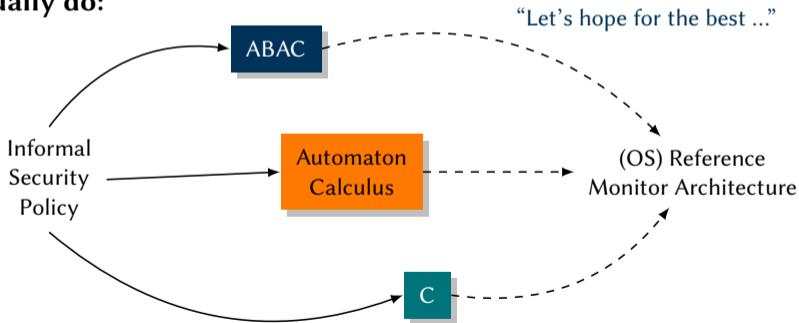


The Problem

What we should do:

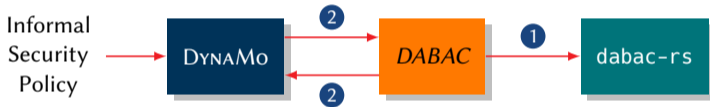


What we actually do:



Make Language Translations ...

- 1 automatic (tool-based) whenever possible,
- 2 semantically *small*, whenever manually inevitable



Contributions:

- *DYNAMO*: ergonomic, machine-readable ABAC specification language
- *DABAC*: flexible formal calculus for analyzing dynamic properties
- *dabac-rs*: reference monitor runtime library in Rust
- *dmo2rs*: transpiler from *DYNAMO* to Rust (prototype w. i. p)

Policy Specification → Policy Analysis



DYNAMo

```
begin state-transition-scheme:
```

```
read_obj(S s_caller, O o_record):
```

```
  pre: check_acf('read' s_caller, get_att(s_caller, 'roles_active')) or
```

```
  (
```

```
    check_bool('joint_groups', s_caller, o_record) and
```

```
    check_bool('joint_groups', get_att(s_caller, 'wards').any.s, o_record)
```

```
  );
```

```
  begin post:
```

```
    set_att(o_record, 'last_access', query(UTC_NOW));
```

```
    ...
```

```
  end post;
```

Policy Specification → Policy Analysis



DABAC Model

► **readObj**($u \in U_\gamma, o \in O_\gamma$) ::=

VAR: $r_u = att_{UR_\gamma}(u), l_u = att_{Ul_\gamma}(u), l_o = att_{Ol_\gamma}(o), w_u = att_{UW_\gamma}(u),$

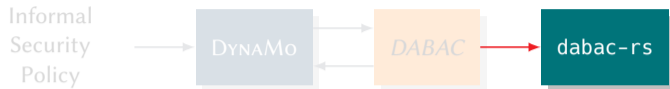
$l' = \bigcup_{u' \in U_\gamma \setminus \{u\}, att_{UW_\gamma}(u')=w_u} att_{Ul_\gamma}(u')$

PRE: $auth_{read}(r_u) \vee (auth_{group}(l_u, l_o) \wedge auth_{group}(l', l_o))$

POST: $att_{OT_\gamma}(o) \leftarrow att_t(tPhysClock)$

...

Policy Implementation



dabac-rs

```
let ra_comp = DABACComp {  
  name: "roles_active".to_owned(),  
  features: [None, None, Some(Aa(0)), None, None, Some(Dyn)],  
  inner: &USER_ROLES_ACT,  
};  
let dabac_model = DABACBuilder::new()  
  .with_component(&user_comp).with_component(&ra_comp). ...  
  .with_policy("path/to/policy.dmo") // also native Rust code possible  
  .build(); // initialize model  
...  
// peter = User(42) requests 'read' access on myrec = Object(8):  
if dabac_model.op("read_obj", (peter, myrec)) { /* actual application logic for 'read' */ ... }
```

Problem Statement

- ① AC policy engineering: a problem (not just) of system software
 - ② Correctness of a policy is crucial
 - ③ Correctness of enforcement as a reference monitor is crucial
- ☞ ... adequate language support is crucial

Current Work ... is to put the pieces together:

- Validate modeling and analysis capabilities of DYNAMO and DABAC
→ real-world scenarios, both local and distributed
- Extend tooling for automatic implementation of AC policies (dmo2rs)
- Optimize heuristic analysis methods for dynamic state machine properties (privilege escalation)
→ separate work

Problem Statement

- ① AC policy engineering: a problem (not just) of system software
 - ② Correctness of a policy is crucial
 - ③ Correctness of enforcement as a reference monitor is crucial
- 👉 ... adequate language support is crucial

Current Work ... is to put the pieces together:

- Validate modeling and analysis capabilities of DYNAMO and DABAC
→ real-world scenarios, both local and distributed
- Extend tooling for automatic implementation of AC policies (dmo2rs)
- Optimize heuristic analysis methods for dynamic state machine properties (privilege escalation)
→ separate work

Language-Support for Correct and Reliable Enforcement of Access Control Policies

Peter Amthor

Distributed and Operating Systems Group
Technische Universität Ilmenau
peter.amthor@tu-ilmenau.de

2023-09-28

