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

Peter Amthor

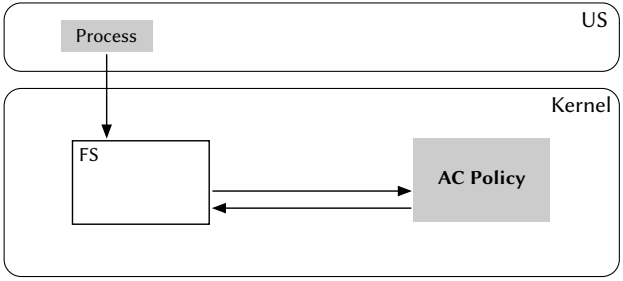
Distributed and Operating Systems Group
Technische Universität Ilmenau
peter.amthor@tu-ilmenau.de

2023–09–28

Herbsttreffen FGBS 2023, Bamberg

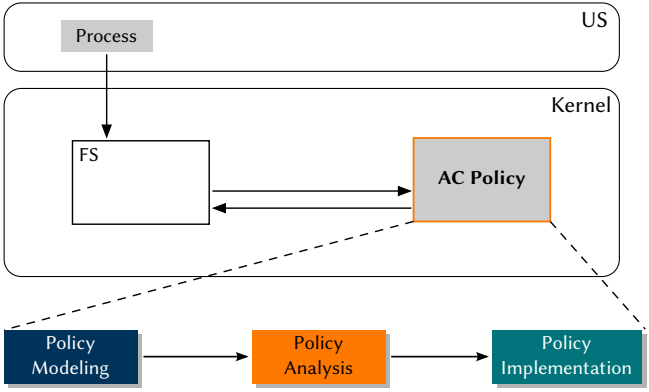
A Bit of Context

- Security as a mission-critical non-functional property of (system) software
- ☞ Reference monitor architecture
- ☞ *Access control (AC) policy*



A Bit of Context

- Security as a mission-critical non-functional property of (system) software
- ☞ Reference monitor architecture
- ☞ *Access control (AC) 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 vulnerabilites
- e. g. automata, flow graphs, FOL, PDL, CSP, ...

Implementation ☞ Programming Languages

- C (and others ... 😇)

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 vulnerabilites
- e. g. automata, flow graphs, FOL, PDL, CSP, ...

Implementation ☞ Programming Languages

- C (and others ... 😇)

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 vulnerabilites
- e. g. automata, flow graphs, FOL, PDL, CSP, ...

Implementation ☞ Programming Languages

- C (and others ... 😇)

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 vulnerabilites
- e. g. automata, flow graphs, FOL, PDL, CSP, ...

Implementation ☞ Programming Languages

- C (and others ... 😇)

Their virtues:

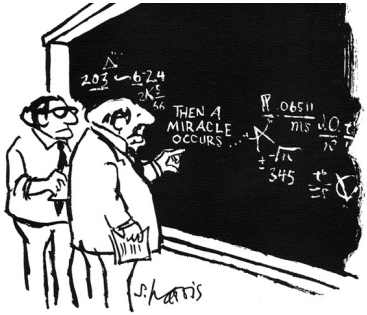
1. Adequate policy abstractions,
   e. g. file attributes, user roles, etc.
2. Verifiability
   of policy correctness
3. Ergonomics
   to constructively avoid errors
4. Reliable enforcement,
   i. e. tamper-proof storage and non-circumventable interfaces of a policy

The Problem

Observations:

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

→ Translations are *costly*
→ Translations are *error-prone*!

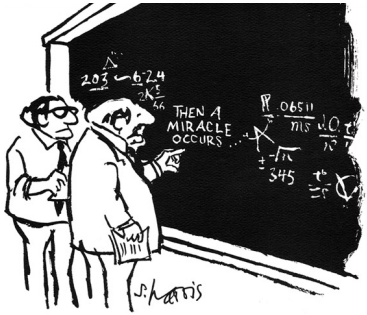


The Problem

Observations:

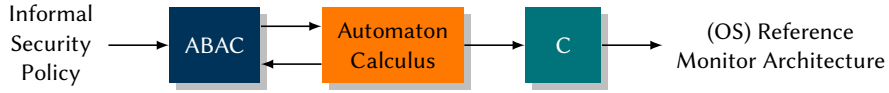
- Different levels of abstraction
- Different expressiveness
- Different syntax and semantics

→ Translations are *costly*
→ Translations are *error-prone*!



The Problem

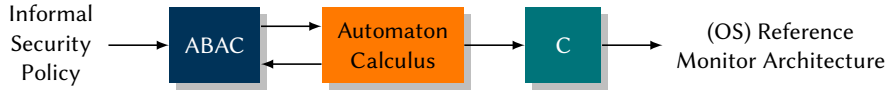
What we should do:



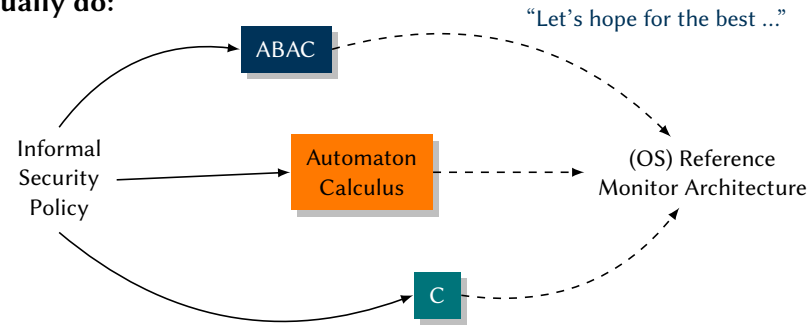
What we actually do:

The Problem

What we should do:



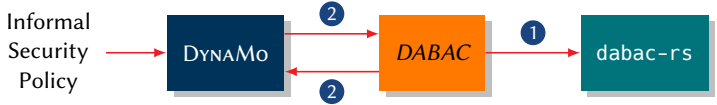
What we actually do:



Our Goal

Make Language Translations ...

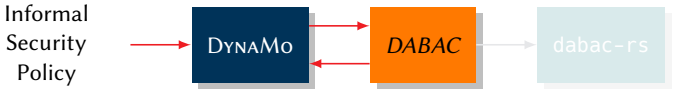
1. automatic (tool-based) whenever possible,
2. semantically small, whenever manually inevitable



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

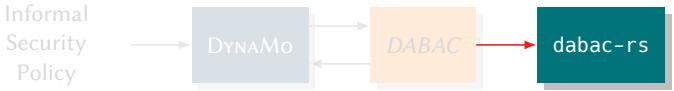
VAR: $r_u = att_{UR_\gamma}(u)$, $I_u = att_{UI_\gamma}(u)$, $I_o = att_{OI_\gamma}(o)$, $w_u = att_{UW_\gamma}(u)$,
$I' = \bigcup_{u' \in U_\gamma \setminus \{u\},\, att_{UW_\gamma}(u') = w_u} att_{UI_\gamma}(u')$

PRE: $auth_{\text{read}}(r_u) \lor \left(auth_{\text{group}}(I_u, I_o) \land auth_{\text{group}}(I', I_o)\right)$

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' */ ... }
```

Conclusion

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 DynaMoand *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

Conclusion

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 DynaMoand *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