Cyber Security Body of Knowledge

Formal Methods for Security
3.09.2021

David Basin
ETH Zurich

The CyBOK project would like to understand how the CyBOK is being used and its uptake. The project would like organisations using, or intending to use, CyBOK for the purposes of education, training, course development, professional development etc. to contact it at contact@cybok.org to let the project know how they are using CyBOK.

bristol.ac.uk

About the Presenter

Biography sketch:

- **Ph.D. in Computer Science** 1989, Cornell University
- **Postdoctoral researcher**, 1990-1996 at U. Edinburgh and MPI Saarbrücken
- **Professor of Computer Science**, 1997-2002, University of Freiburg Germany
- **Professor of Computer Science**, ETH Zurich, 2003 – present

Research group: Information Security Group

Founder: Anapaya Systems

ETH zürich

Formal Methods for Security – Overview

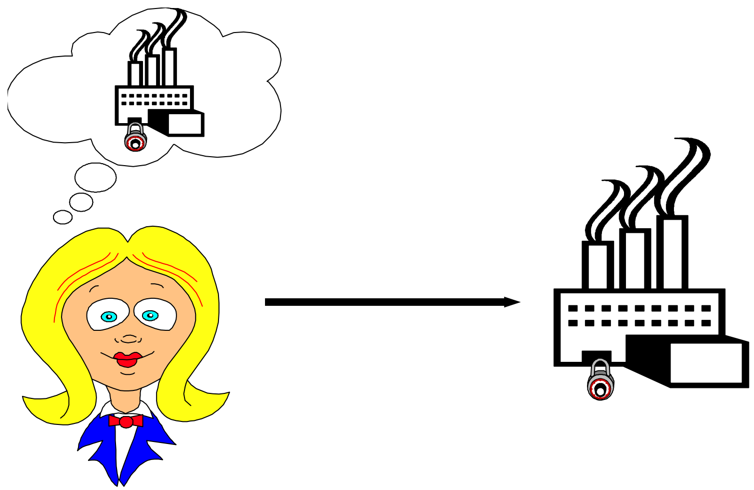
- Introduction and Motivation
- Foundations, Methods, and Tools
- Hardware
- Cryptographic Protocols
- Software and Large-Scale Systems
- Configurations

Formal Methods for Security – Overview

- **Introduction and Motivation**
- Foundations, Methods, and Tools
- Hardware
- Cryptographic Protocols
- Software and Large-Scale Systems
- Configurations

What are Formal Methods?

- Foundations, methods, and tools for rigorously developing and reasoning about systems and their components
- Emphasis on **firm mathematical basis**: predict, calculate, and prove!



- Particularly attractive for **critical systems**

  $\Longrightarrow$ **Security** is critical!

Focus on Modelling and Proof

- Prove system satisfies its specification in an **adversarial environment**
  Requires precise specification of:
  - **System** at some appropriate level of abstraction
  - **Adversarial environment** that the system operates in
  - **Properties** e.g., security properties that system should satisfy

- **Example**: information on a disk may be secure against a network adversary, but not one with physical access to the disk

- Adversary or properties sometimes left implicit
  Example: in static analysis the properties may simply be absence of certain bug classes like buffer overflows or injection attacks

Scope is Wide

- **Systems**: hardware, software, modules, protocols, …
- **Abstraction**: design versus code
- **Kinds of properties/thoroughness**: "shallow properties" like type correctness versus "deeper properties" like functional correctness
  $f(x) = x + 1$: function from *N* to *N* versus successor function
- **Approaches**: interactive versus automatic

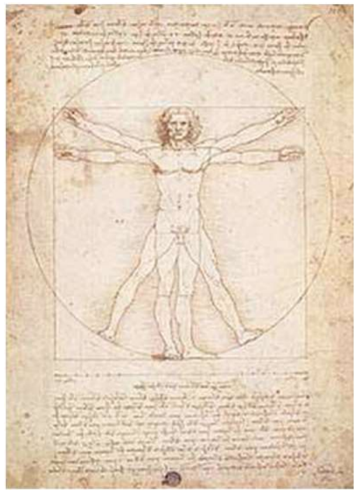
Substantial overlap with formal methods for correctness

– But also new challenges for security
– Differences in system detail, properties, and environment

Why Bother?

- Inadequacy of conventional development methods
  - Test and fix $\Longrightarrow$ penetrate and patch
  - Adversaries are not typical users.
    Highly skilled at finding obscure bugs
  - Conventional development methods not up to task

- Quest for more scientific development methods
  - Programs are mathematical objects
  - Place security on a firm mathematical footing
  - Progress from an **Art** to a **Science**



Limitations

- Models of systems & adversaries versus the real thing
  - –Does system model accurately capture system's behaviours?
  - –Could adversary do more in practice?
- Are properties appropriate for the given usages?
- Complexity: most security questions are undecidable. So:
  - –approximate behaviours
  - –use effective semi-decision procedures
  - –humans provide input, like invariants, proof steps, etc.

Formal Methods for Security – Overview

- Introduction and Motivation
- **Foundations, Methods, and Tools**
- Hardware
- Cryptographic Protocols
- Software and Large-Scale Systems
- Configurations

No Canonical Best Method

- Specification options
  - Code or executable specifications
  - Variants of transition systems / automata
  - Logics like FOL, HOL, temporal logic, …
- Verification options: algorithms and tools
  - **Automatic**: BDDs, SMT, model checkers, …
  - **Interactive**: higher-order logics, type theories, also tools for weaker logics that benefit from lemmas or hints
- Mature tools exist for many relevant analysis problems

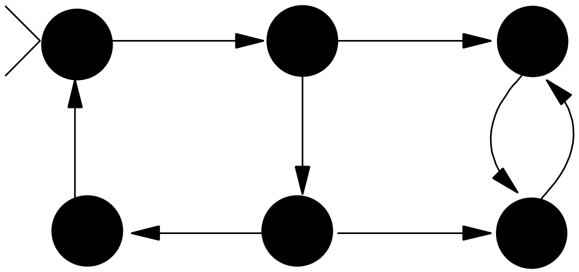
Foundations: Trace Properties

- Abstract view: semantics given by behaviours

$$\square\ ,\quad s_0\, s_1\, s_2\ \cdots$$

- s_i may be states, actions, state/action pairs, …
- Set of traces define system semantics

  Given by automaton or program with a transition-system semantics



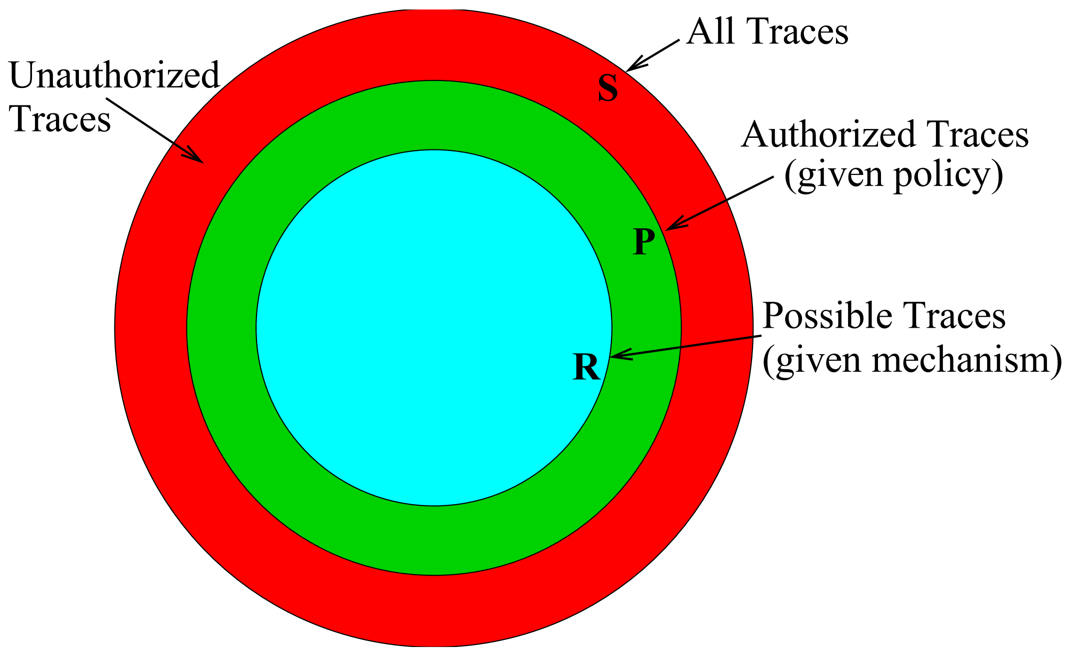
```
for(i = 1; i <= number - 1; i++)
{
    for(j = i; j > 0 && a[j - 1] > a[j]; j--)
    {
            temp = a[j];
            a[j] = a[j - 1];
            a[j - 1] = temp;
    }
}
```

Foundations: Trace Properties

- Also define system properties, e.g., using temporal logics

$$\Box(\text{FundsWithdraw} \rightarrow \blacklozenge \text{EnterPIN})$$

- Correctness then reduces to language containment

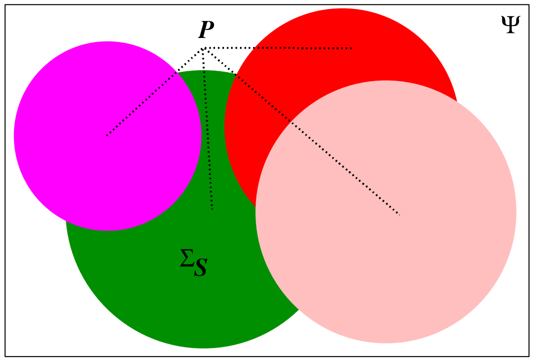


Foundations: Hyperproperties

CyBOK

- Properties of ***sets*** of traces.
  - Membership not determined by considering individual system traces
  - One must examine the ***entire*** set of traces.

- Let Ψ denote universe of all possible finite/infinite sequences.
- A **security policy** P is specified as a predicate on **sets** of executions, i.e., it characterizes a **subset of** $\mathcal{P}(\Psi)$.
- A system S defines a set $\Sigma_S \subseteq \Psi$ of actual executions.
- S **satisfies** P iff $\Sigma_S \in P$.



Hyperproperties
Example: Timing Side-Channel Analysis

- Adversaries observe system I/O + time taken for function execution
- Modelled using ***timed traces***: events modelling function computation augmented with computation time
- If a function has no timing side-channel, then its computation time should be independent of any secret input.
  ⟹ The time taken to execute on any secret is the same as the time taken to execute on any other secret.
- Analysing any individual trace is insufficient.
  One must examine the **set** of all of the system's traces.
- In this example, it would suffice to examine all ***pairs*** of system traces (a ***2-safety hyperproperty***).

Foundations: other Options

- Focus on processes and process interactions
  - Numerous relationships between processes exist capturing notions like ”interchangeable”, “observationally equivalent” or “refines”
  - Some come with decision procedures, e.g., FDR2/3/4 for CSP
- Richer semantics that incorporates time or probabilities
- Use of general purpose logics to formalize semantics
  - Weak logics that are easy to automate, like propositional logic
  - Expressive logics like HOL

Property Checking

- Interactive Theorem Proving
  - E.g., Isabelle/HOL, Coq
- Decision Procedures
  - E.g., Chaff or Grasp (PL) or Z3, CVC4, or Yices (SMT)
  - Model checkers for LTL and CTL, like NuSMV
- Static analysis
  - Automated procedures for particular classes of properties
  - May approximate behaviours
- Dynamic analysis
  - Check property on execution trace arising at runtime

CyBOK

Property Checking

Model Checking

Input: system model
Input: formal specification
Output: counterexample?

Guarantee:
any modeled system behavior satisfies the specification

Runtime Monitoring

Input: system trace
Input: formal specification
Output: verdict

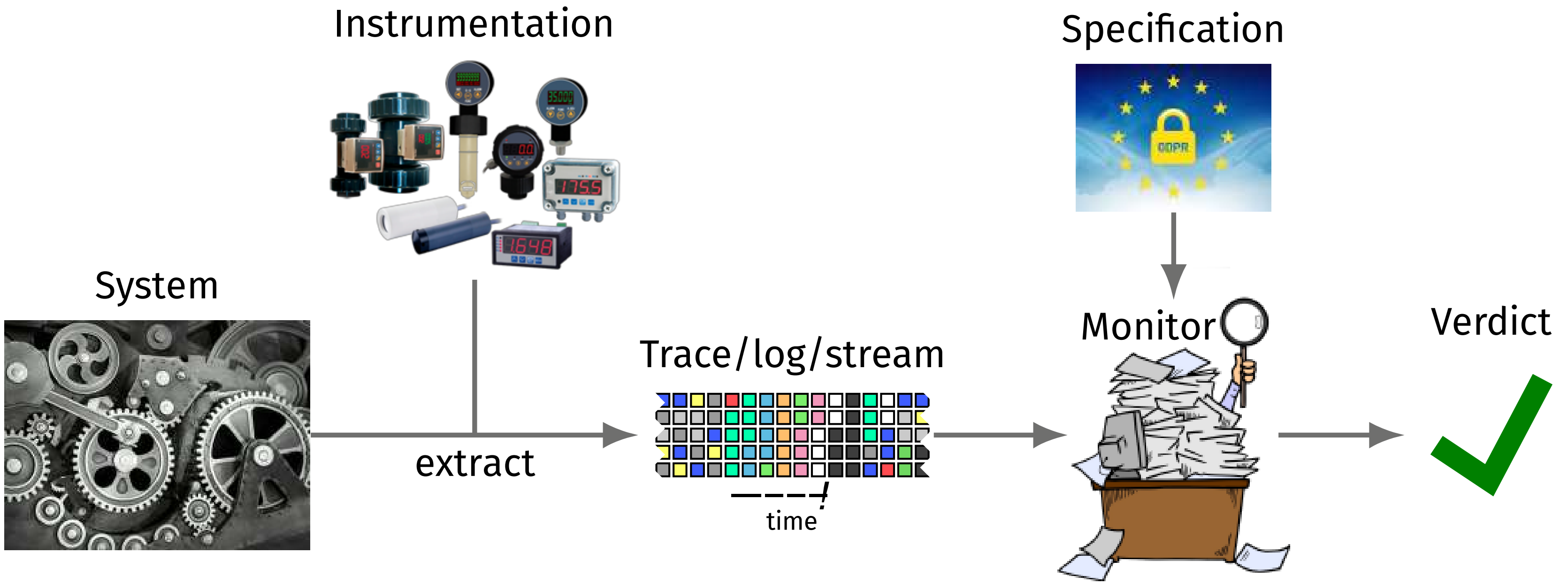
Guarantee:
on the **real inputs** the **real system** behavior satisfies the specification

Software Testing

Input: system
Input: test cases
Output: failed assertions?

Guarantee:
on the **mock inputs** the **real system** behavior satisfies the specification

Example: Dynamic Analysis (Runtime Verification)



Example tools: Java PathExplorer, MonPoly, QEA, …

Formal Methods for Security – Overview

- Introduction and Motivation
- Foundations, Methods, and Tools
- **Hardware**
- Cryptographic Protocols
- Software and Large-Scale Systems
- Configurations

Hardware

- Great success for Formal Methods, e.g. model checking
  - Development since 1980s: core algorithms, BDDs, SAT-based
  - Successful use by semiconductor and design automation companies
  - Industrial temporal logics standardized and widely used

- Security-specific applications
  - Common Criteria certification of hardware or microcode
  - Verified stacks: OS, compiler, assembler, machine code, hardware, …
  - Side channel analysis, e.g., show branchings' timing behaviour does not leak information about secrets
  - API attacks on security tokens

Formal Methods for Security – Overview

- Introduction and Motivation
- Foundations, Methods, and Tools
- Hardware
- **Cryptographic Protocols**
- Software and Large-Scale Systems
- Configurations

FM Success Story for Security

Dramatic Change in How We Think About Security Protocols

A Typical Protocol

IKE, Phase 1, Main Mode, Digital Signatures, Simplified

$$
\begin{array}{lll}
(1) & I \rightarrow R: & C_I,\ ISA_I \\
(2) & R \rightarrow I: & C_I,\ C_R,\ ISA_R \\
(3) & I \rightarrow R: & C_I,\ C_R,\ g^x,\ N_I \\
(4) & R \rightarrow I: & C_I,\ C_R,\ g^y,\ N_R \\
(5) & I \rightarrow R: & C_I,\ C_R,\ \{ID_I,\ SIG_I\}_{SKEYID_e} \\
(6) & R \rightarrow I: & C_I,\ C_R,\ \{ID_R,\ SIG_R\}_{SKEYID_e}
\end{array}
$$

$$
\begin{array}{lll}
SKEYID & = & h(\{N_I, N_R\}, g^{xy}) \\
SKEYID_d & = & h(SKEYID, \{g^{xy}, C_I, C_R, 0\}) \\
SKEYID_a & = & h(SKEYID, \{SKEYID_d, g^{xy}, C_I, C_R, 1\}) \\
SKEYID_e & = & h(SKEYID, \{SKEYID_a, g^{xy}, C_I, C_R, 2\}) \\
HASH_I & = & h(SKEYID_a, \{g^x, g^y, C_I, C_R, ISA_I, ID_I\}) \\
HASH_R & = & h(SKEYID_a, \{g^y, g^x, C_R, C_I, ISA_R, ID_R\}) \\
SIG_I & = & \{HASH_I\}_{K_I^{-1}} \\
SIG_R & = & \{HASH_R\}_{K_R^{-1}}
\end{array}
$$

Does argument order matter?

Why all the nested keyed hashes?

Model Checkers and Theorem Provers

- Provide formal specifications (important itself!)
  – Clarify protocol, environment, properties
- Tool support to debug, verify, and explore alternatives
- Substantial progress made for many protocols that matter
  – ISO/IEC 9798, EMV, 5G, TLS 1.3, HSMs, …
- Companies are slowly coming on board as tool users

In following, I discuss symbolic methods. **Enc(m,k)**

For computational approaches, see chapter. **0100101…**

Example: Symbolic Analysis
Interleaving Trace Models

- Modeling idea: model possible communication events.

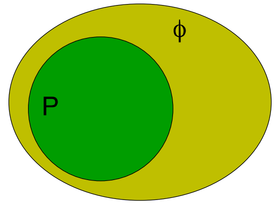
$$
\begin{aligned}
A &\to B : M_1 \\
C &\to D : P_1 \\
Spy &\to A : M_2 \\
C &\to D : P_2 \\
&\vdots
\end{aligned}
$$

- A **trace** is a sequence of events.
- Trace-based interleaving semantics: **protocol** denotes a trace set.

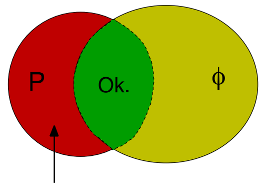
  Interleavings of (partial) protocol runs and attacker messages.
- Attacker model (Dolev-Yao): the attacker controls the network.

  He can **read**, **intercept**, and **create** messages.

Symbolic Analysis (cont.)



Ok, no attacks.



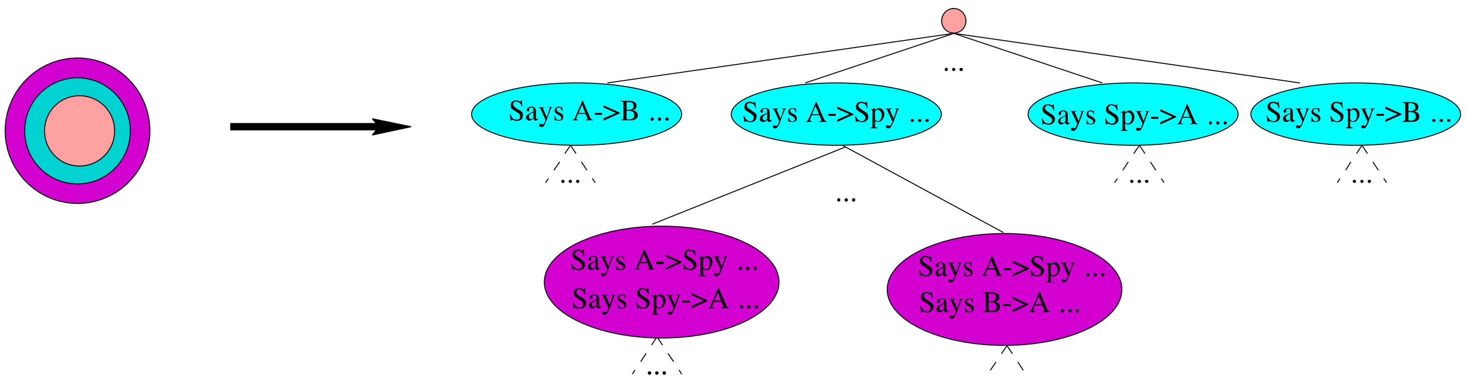
Attacks.

- Verification: define set of interleavings inductively
  - Protocol semantics corresponds to a set of traces
  - So do properties
  - So correctness well defined
- Induction used to establish set containment
  - Key idea behind “Paulson’s Inductive Method”
  - Proofs in Isabelle/HOL
- Many protocols analyzed: TLS, SET, Kerberos IV, …
  - Typically takes a few days of work
  - Flaws come out in terms of unprovable goals, suggesting attacks

Symbolic Analysis (cont.)

- Alternative: algorithmic verification

  Recast inductive definition as search tree



- Attacks: traces falsifying desired property
- If no attacks: protocol is secure (undecidable problem!)
- Efficient Model-checking tools exist: Tamarin, ProVerif, …

  E.g., Tamarin does backword search from set of attack states, constructing symbolic traces with constraints to finitely represent infinite sets of ground instances

Formal Methods for Security – Overview

- Introduction and Motivation
- Foundations, Methods, and Tools
- Hardware
- Cryptographic Protocols
- **Software and Large-Scale Systems**
- Configurations

Information Flow Control

- Enforcement of confidentiality and integrity guarantees during system execution.
  - Confidentiality: no information flow from high to low
  - Integrity: dually, no flow from low to high

- Example of (indirect) information flow
  - Observing **l** reveals parity of **h**
  - Security relevant, e.g., **h** is a secret password

```
h := h mod 2
l := 0
if (h = 1)
  then l := 1
  else skip
```

Information Flow Control (cont.)

CyBOK

- Variety of techniques designed to prevent such leaks

- **Static**: via type systems, static analysis, …
  E.g., Jif, Flow Caml, SPARK, JOANA

```
e : low     [low] |- b      e : high     [high] |- b
---------------------      ----------------------
[low] |- if e { b }        [high] |- if e { b }
```

- **Dynamic**: e.g., tracking “taint” at runtime

Application: Cryptographic Libraries

- Involves many challenging problems
  - **Freedom from side channels** due to assignment, branching, memory access patterns, cache behaviour, power consumption
  - **Memory safety:** only valid memory locations written and read
  - **Cryptographic security:** code implements a function secure WRT standard security notion, possibly under assumptions on its building blocks
- Variety of approaches
  - High-level strongly typed languages like F*, which support verification of both functional and security properties
  - Lower-level assembly-like languages, e.g., VALE
  - In both cases, SMT solvers help automate proofs about Pre-conditions, post-conditions, and invariants

Application: Kernel Components

- OS critical for security of overall systems
  - **Data separation**: processes cannot read each other's data
  - **Temporal separation**: processes use resources sequentially, and these resources are properly sanitized before being passed on
  - **Damage limitation**: effects of compromises limited
- SeL4 microkernel verification
  - 8,700 LoC (C) + 600 LoC assembler
  - Fully verified from abstract specification down to implementation Uses two large refinement steps between functional and C specs.
  - Safety properties: kernel doesn't crash, perform unsafe operations, …
  - 20 person/years. A showcase for formal methods

Application: Web

- Web Programming with JavaScript
  - Use FM to provide a semantics
  - Develop compilers from languages with more easily provable properties, like F*, to Javascript
  - Explore alternatives, like WebAssembly, w/ formal semantics and associated verification tools

- Components and their interaction
  - Semantics for browsers, web servers, HTML, …
  - Proofs about mechanisms preventing injection, scripting, other attacks
  - Verification of properties of different web protocols, e.g., for SSO

Formal Methods for Security – Overview

- Introduction and Motivation
- Foundations, Methods, and Tools
- Hardware
- Cryptographic Protocols
- Software and Large-Scale Systems
- **Configurations**

Configurations

- Relevant for security when systems are deployed and used
- Security analysis of configurations
  - Does my (RBAC / ABAC / …) configuration satisfy some high-level policy or have some desired properties?
  - Change-impact analysis
  - Such problems can be reduced to logical inference problem in appropriate logical fragments (FOL or an SMT fragment)
- Configuration Synthesis
  - Translate policy to a configuration or even a runtime monitor
  - Methods based on logical inference and program synthesis techniques

Contact:

Professor David Basin
Information Security Group
basin@inf.ethz.ch

ETH Zurich
Universitätstrasse 6
8092 Zurich, Swizterland

https://infsec.ethz.ch