Policy Verification Using Metagraphs

Loïc Miller, Pascal Mérindol, Antoine Gallais and Cristel Pelsser

November 7, 2021

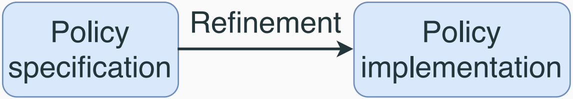
University of Strasbourg, France

Attacks enabled by an erroneous policy

- Razer (2017) [razer].
    - Improper permissions allowing public viewing of .bash_history, eventually leaking database credentials.
- Facebook (2018) [facebook].
    - Improper policy allowing third-party applications to become admin of a page and remove the actual owner permanently.

Motivation

Access Control is an essential building block of security.
Generally managed by a policy administrator.



Translating a policy specification to its implementation is **prone to errors**, even with the available semi-automatic or automatic tools [awstool, dohndorf2011tool, klinbua2017translating].

Objective: Policy verification

Verify the implementation matches the specification

Pinpoint errors

Why metagraphs?

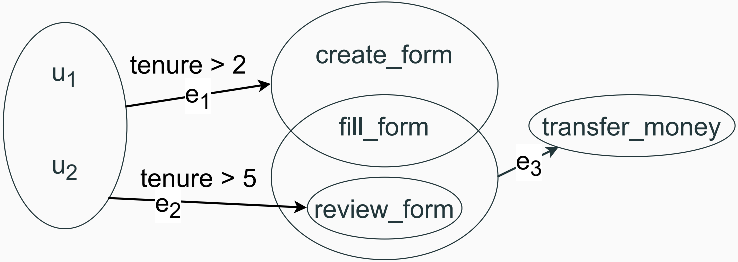
- Existing works dealing with policy verification use SAT solvers [**bera2010policy**], decision diagrams [**gouda2007structured**] or graphs [**ranathunga2016malachite**].

	SAT solvers	Decision diagrams	Graphs	Metagraphs
Natural policy modeling	■	◪	◪	■
Visual representation	□	◪	■	■
Formal foundations	■	■	◪	■

- Properties **specific to metagraphs** for detecting conflicts and redundancies[1].

[1] **ranathunga2020verifiable**.

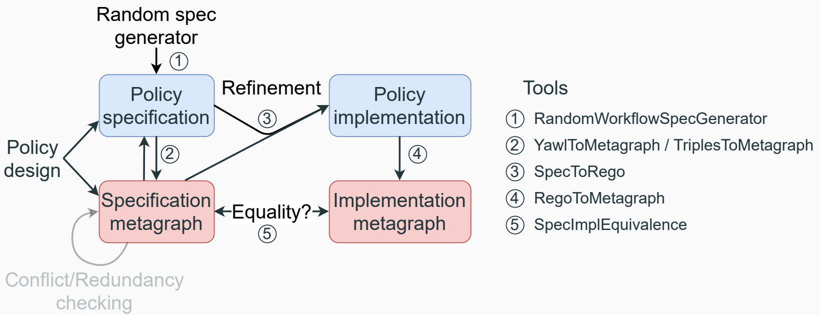
The metagraph: a collection of directed set-to-set mappings[2]



Employees (u_1, u_2) and tasks (*create_form*, *fill_form*, *review_form*, *transfer_money*) are put into relation by the edges (e_1, e_2, e_3) between sets of elements.

[2] basu2007metagraphs.

Policy verification procedure[3]



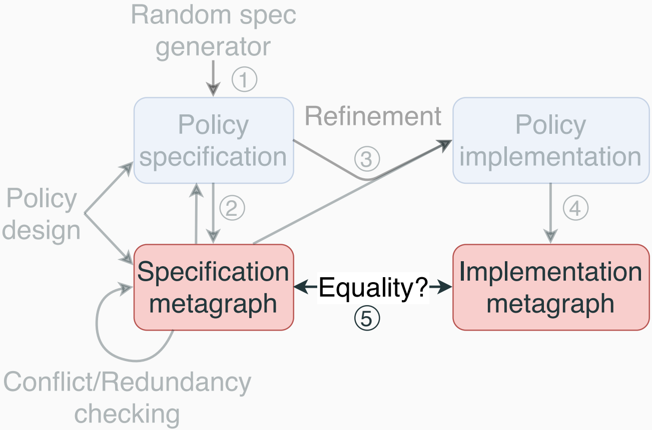
Policy specification: YAWL, or metagraph-like format.

Policy implementation: Rego.

We can pinpoint errors in the policy.

[3] Data, code, and results publicly available. See https://zenodo.org/record/4912289.

Performance analysis ⑤



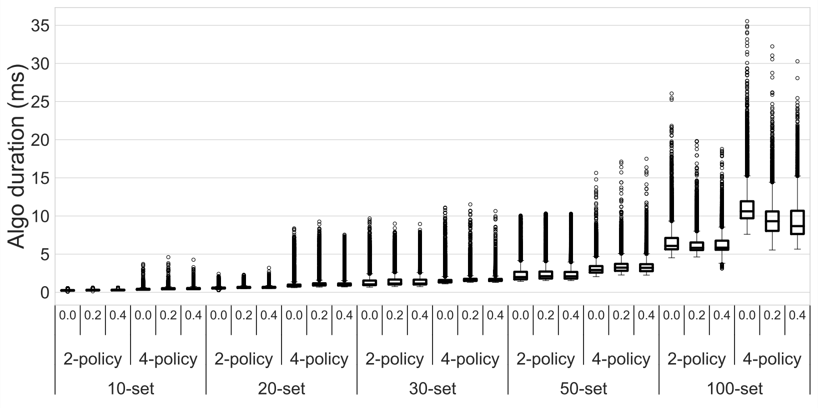
Performance analysis ⑤

We measure the time required to compare two metagraphs.

- **Random** policies to get more robust results.
- **Number of elements in the policy**: 10, 20, 30, 50 or 100.
- **Policy size**: 2 or 4 propositions per edge.
  $\rightarrow$ 300 policy specifications ($5 \times 2 \times 30$)
- **Translation error rate**: 0.0, 0.2 and 0.4.
  $\rightarrow$ 27,000 policy implementations ($300 \times 3 \times 30$)
- 30 measures per implementation.
  $\rightarrow$ 810,000 measures (27000×30)

Rego policy files between 305 and 24729 lines of code, **in line** with observed policies.

Time increases with number of elements and policy size



- Verification times between 0 and 12 ms on average.
- Error rate has a negligible effect (correlation of 0.01).

Conclusion

- New policy verification method using metagraphs.

[4] Code, data and guidance at https://github.com/loicmiller/policy-verification

Conclusion

- New policy verification method using metagraphs.
- Motivated the use of metagraphs to represent and verify policies.

[4] Code, data and guidance at https://github.com/loicmiller/policy-verification

Conclusion

- New policy verification method using metagraphs.
- Motivated the use of metagraphs to represent and verify policies.
- Developed suite of tools[4]:
    - RandomPolicySpecGenerator
    - YawlToMetagraph / SpecToRego
    - RegoToMetagraph
    - SpecImplEquivalence

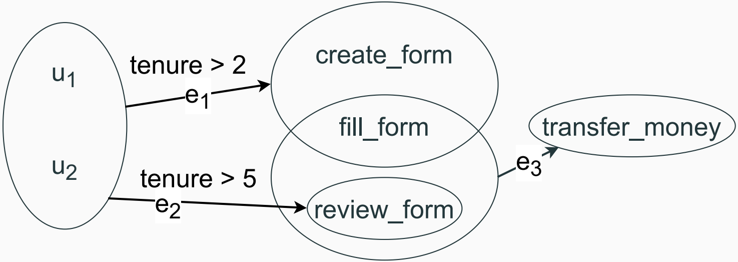
[4] Code, data and guidance at https://github.com/loicmiller/policy-verification

Conclusion

- New policy verification method using metagraphs.
- Motivated the use of metagraphs to represent and verify policies.
- Developed suite of tools[4]:
    - RandomPolicySpecGenerator
    - YawlToMetagraph / SpecToRego
    - RegoToMetagraph
    - SpecImplEquivalence
- Evaluated our method: verification times between 0 and 12 ms on average.

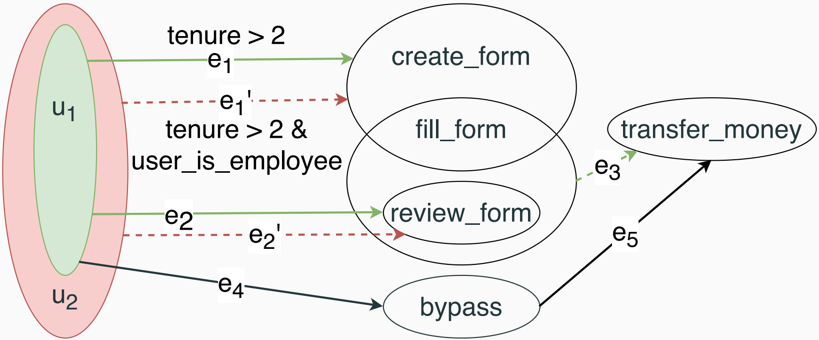
[4] Code, data and guidance at https://github.com/loicmiller/policy-verification

Goal: Identify redundancies/conflicts/incompleteness in the policy.



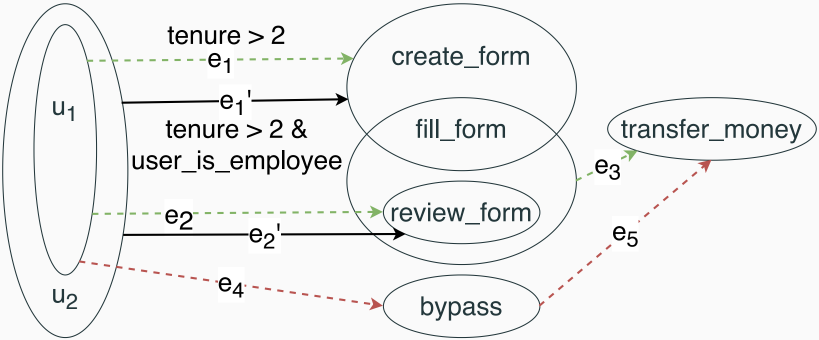
$M_1(\{u_1, u_2\}, \{transfer\_money\}) = \{e_1, e_2, e_3\}$ is not a simple path, its a **metapath**.

Input dominance



$M_1(\{u_1, u_2\}, \{transfer\_money\}) = \{e'_1, e'_2, e_3\}$ is not input-dominant because $M_2(\{u_1\}, \{transfer\_money\}) = \{e_1, e_2, e_3\}$ is a metapath.

Edge dominance



$M_1(\{u_1\}, \{transfer\_money\}) = \{e_1, e_2, e_3, e_4, e_5\}$ is not edge-dominant because $M_2(\{u_1\}, \{transfer\_money\}) = \{e_1, e_2, e_3\}$ is a metapath.

Usage of dominant metapaths

- Dominant metapaths identify necessary elements.
- Elements not on any dominant metapath are redundant.

- Computationally expensive solution (A^*).

Thank you!