Verification of Cloud Security Policies

Loïc Miller, Pascal Mérindol, Antoine Gallais and Cristel Pelsser

May 25, 2021

University of Strasbourg, France

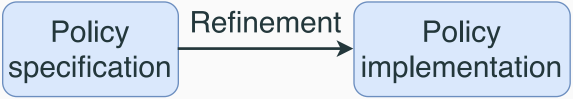
Université de Strasbourg

Attacks enabled by an erroneous policy

- Razer (2017) [10].
  - Improper permissions allowing public viewing of .bash_history, eventually leaking database credentials.
- Facebook (2018) [7].
  - 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 [1, 4, 6].

Objective: Policy verification

Verify the implementation matches the specification

Pinpoint errors

Why metagraphs?

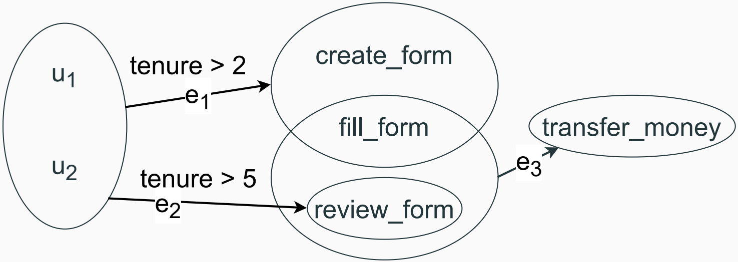
- Existing works dealing with policy verification use SAT solvers [3], decision diagrams [5] or graphs [9].
- Not one of these has all of the following features:
  - **Natural policy modeling**
  - **Visual representation**
  - **Formal foundations**

Metagraphs check all boxes.

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

[1] Dinesha Ranathunga, Matthew Roughan, and Hung Nguyen. "Verifiable Policy-Defined Networking using Metagraphs". In: *IEEE Transactions on Dependable and Secure Computing* (2020).

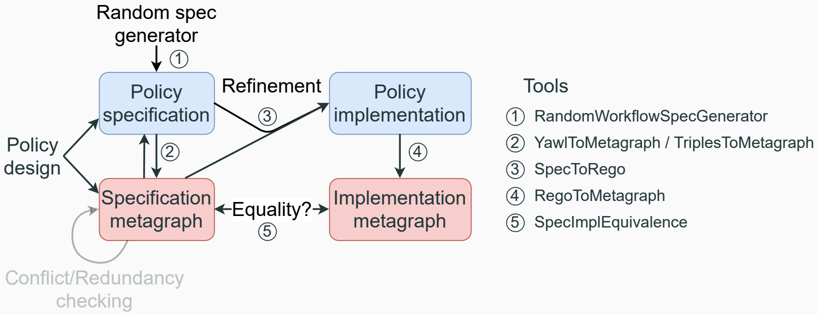
The metagraph: a collection of directed set-to-set mappings[2]



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] Amit Basu and Robert W Blanning. *Metagraphs and their applications*. Vol. 15. Springer Science & Business Media, 2007.

Policy verification procedure



Policy specification: YAWL, or metagraph-like format.
Policy implementation: Rego.

We can pinpoint errors in the policy.

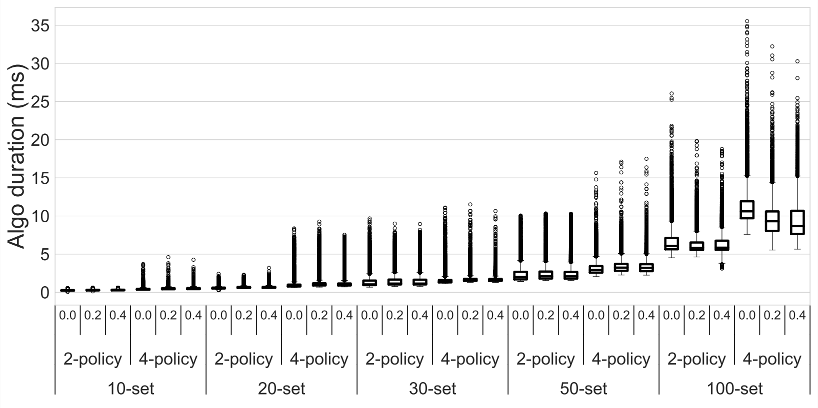
Performance analysis (5)

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.

[3] 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.

[3] 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[3]:
    - RandomPolicySpecGenerator
    - YawlToMetagraph / SpecToRego
    - RegoToMetagraph
    - SpecImplEquivalence

[3]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[3]:
    - RandomPolicySpecGenerator
    - YawlToMetagraph / SpecToRego
    - RegoToMetagraph
    - SpecImplEquivalence
- Evaluated our method: verification times between 0 and 12 ms on average.

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

Thank you!

[1] Amazon. *AWS Policy Generator*. 2020. URL: `%5Curl%7Bhttps://awspolicygen.s3.amazonaws.com/policygen.html%7D` (visited on 11/11/2020).

[2] Amit Basu and Robert W Blanning. *Metagraphs and their applications*. Vol. 15. Springer Science & Business Media, 2007.

[3] Padmalochan Bera, Soumya Kanti Ghosh, and Pallab Dasgupta. “Policy based security analysis in enterprise networks: A formal approach”. In: *IEEE Transactions on Network and Service Management* 7.4 (2010), pp. 231–243.

[4] Oliver Dohndorf et al. “Tool-supported refinement of high-level requirements and constraints into low-level policies”. In: *2011 IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE. 2011, pp. 97–104.

[5] Mohamed G Gouda and Alex X Liu. “Structured firewall design”. In: *Computer networks* 51.4 (2007), pp. 1106–1120.

[6] Kitti Klinbua and Wiwat Vatanawood. “Translating tosca into docker-compose yaml file using antlr”. In: *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. IEEE. 2017, pp. 145–148.

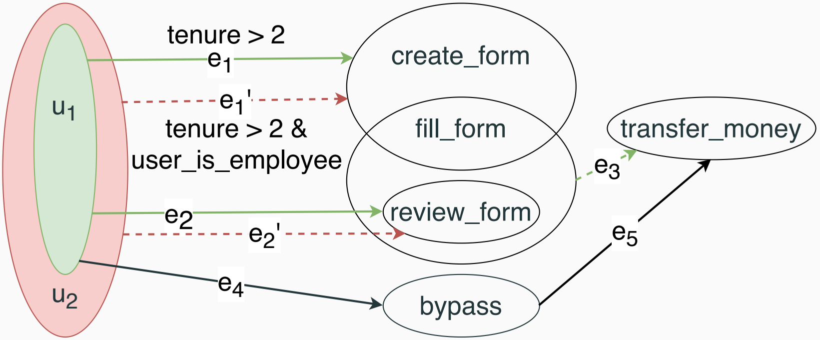
[7] Laxman Muthiyah. *Hacking Facebook Pages*. 2018. URL: `https://thezerohack.com/hacking-facebook-pages` (visited on 12/20/2020).

[8] Dinesha Ranathunga, Matthew Roughan, and Hung Nguyen. “Verifiable Policy-Defined Networking using Metagraphs”. In: *IEEE Transactions on Dependable and Secure Computing* (2020).

[9] Dinesha Ranathunga et al. “Malachite: Firewall policy comparison”. In: *2016 IEEE Symposium on Computers and Communication (ISCC)*. IEEE. 2016, pp. 310–317.

[10] vulners. *Razer US: Database credentials lea*. 2017. URL: `%5Curl%7Bhttps://vulners.com/hackerone/H1:293470%7D` (visited on 12/20/2020).

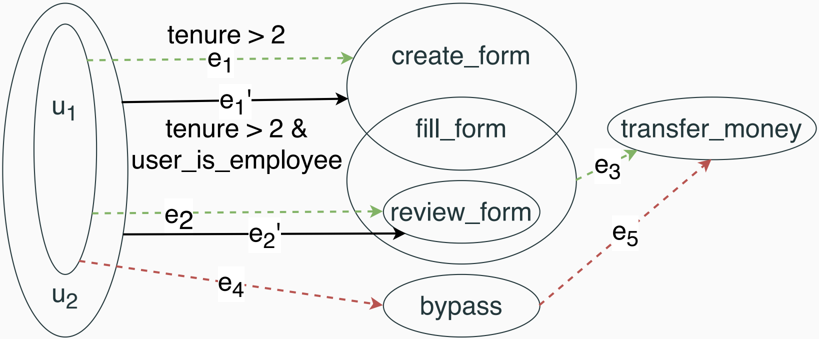
Input dominance



$M_1(\{u_1, u_2\}, \{transfer\_money\}) = \{e_1', e_2', e_3\}$ 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.