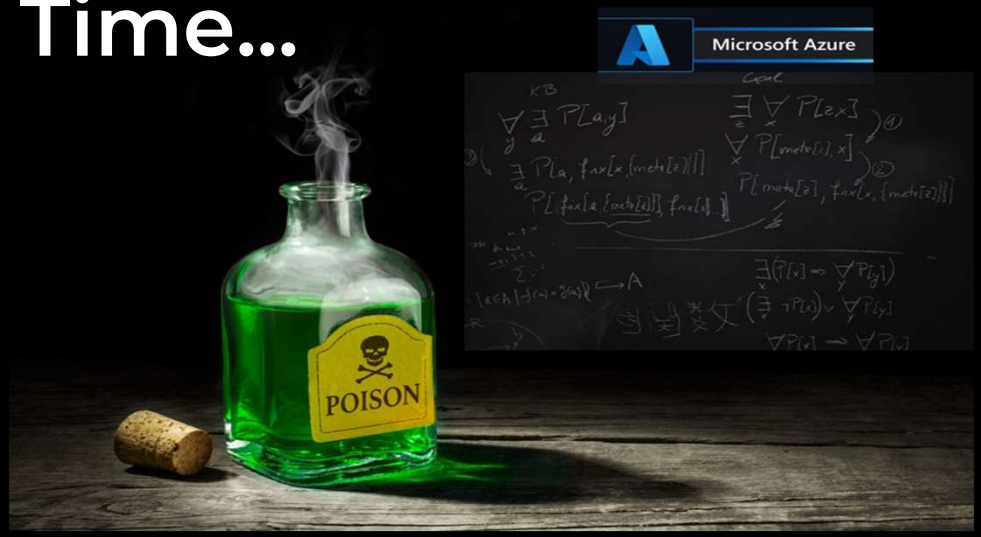
Just In Time...



...Access?

Author: Christophe Parisel
linkedin.com/in/parisel

Permalink:
https://github.com/labyrinthinesecurity/automatedReasoning/blob/main/3_ToxLocks.pdf

How to oversee Just In Time (JIT) access to Azure PaaS control plane when:

- ❑ PIM doesn’t cover your service
- ❑ You don’t want to pay for PIM
- ❑ PIM doesn’t fit into your corporate IAM landscape

Azure IAM product owner requirements

In a given subscription,

- A Blue team must make occasional, emergency changes to some sensitive assets
- A Green team is gatekeeping access to sensitive assets
- Green grants access to Blue after each request review
- Access is temporary, it expires after “H” hours
- Green has no other modification rights in the subscription
- Blue cannot self-approve its requests

Note that ensuring only legitimate principals belong to their respective teams (Green, Blue or other) is not the same IAM control as a JIT control.

Proposal: meet “Tox - Locks”, two controls into one

“Toxic” control: Prevent toxic combinations

In AAD RBAC roles, Green and Blue principals must have mutually exclusive role assignments scoped to the subscription.

This control is point-in-time, because it takes a long time to compute given the current state of the AAD Graph API.

“Locks” control: Prevent dangling locks

- Sensitive assets are grouped into dedicated “sensitive resource groups”
- Sensitive resource groups must be locked at all times
- If not, the deviation must last no longer than “H” hours

This control is continuous: it’s based on real-time data pulled from Azure Resource Graph API and the Authorization Resource Provider API.

Implementation

⇒ Leverage automated reasoning with a SMT solver to prove both parts of the JIT control work as expected.

“Toxic” control:

- Define two objects LOCKMASTER and CONTRIBUTOR
- Add LOCKMASTER != CONTRIBUTOR as a SMT constraint
- When a principal is granted actions Authorizations/{Write|Delete|*}, equate her UUID to LOCKMASTER
- When a principal is granted actions */{Write|Delete|*} or RP/*/{Write|Delete|*} equate her UUID to CONTRIBUTOR
- Check if the property is SATISFIABLE
- If not, raise an anomaly

“Locks” control:

- Define two objects SENSITIVE and NOTSENSITIVE
- Add SENSITIVE != NOTSENSITIVE as a SMT constraint
- Choose some frequency F < H
- At frequency F, enumerate sensitive assets and pick their Resource Group UUID
  - Equate these RG UUIDs to SENSITIVE
- At frequency F, enumerate all Resource Group UUIDs
  - Equate each UUID not equal to the above to NOTSENTIVE
  - Equate each UUID which is not locked to NOTSENSITIVE
  - Check if the property is SATISFIABLE

Implementation (continued)

“Locks” control (continued):

- If the property is UNSATISFIABLE, it means a lock was removed:
  - Pull the time of modification from the Authorization RP API
  - If time is > H, raise an anomaly and disclose the SMT solution
  - Else, ignore and carry on

Note:
The dangling lock control must execute every F > H or it will miss positives.
To ensure control continuity, we implement the SMT continuity algorithm[*]:

- define certified intervals over a compliance period (larger than F)
- build formula φ made of all excluded certified intervals I_n over the period. φ := And(φ, Not(I_n))
- check if φ == True is UNSATISFIABLE
- If not, raise an anomaly and disclose all SMT solutions

[*]: https://github.com/labyrinthinesecurity/automatedReasoning/blob/main/2_ProvableCloudControls.pdf

Tox-Locks

Examples

Example #1: proving toxic combinations

Given a subscription under JIT policy:

- John is Owner
- Julia is authorizations Contributor

John=CONTRIBUTOR
John=LOCKMASTER
Julia=LOCKMASTER
} CONTRIBUTOR == LOCKMASTER

Hence,
CONTRIBUTOR != LOCKMASTER is UNSATISFIABLE.

Example #2: proving dangling locks

Say we want a JIT policy on Azure Route Tables and we have only one with ID
/sub/MySub/RGs/MyRG/provider/Microsoft.Network/RouteTables/MyTable
Suppose that MyRG is unlocked.

At frequency F,

1. enumerate route tables: /sub/MySub/RGs/MyRG/provider/Microsoft.Network/RouteTables/MyTable
2. pick their Resource Group: /sub/MySub/RGs/MyRG
3. equate these RGs to SENSITIVE: SENSITIVE == /sub/MySub/RGs/MyRG
4. Enumerate all Resource Groups: /sub/MySub/RGs/MyRG, /sub/MySub/RGs/AnotherRG
5. Equate each RG not equal to SENSITIVE RGs to NOTSENSITIVE:
   NOTSENSITIVE == /sub/MySub/RGs/AnotherRG
6. Equate each RG which is not locked to NOTSENSITIVE:
   NOTSENSITIVE == /sub/MySub/RGs/MyRG
   NOTSENSITIVE == /sub/MySub/RGs/AnotherRG

SENSITIVE != NOTSENSITIVE is UNSATISFIABLE.

Depending whether the lock has been removed for longer than “H”, we may raise an anomaly.