Detecting Multi-Step IAM Attacks in AWS Environments via Model Checking

Ilia Shevrin¹, Oded Margalit²

¹Citi
²Ben-Gurion University
Background

Cloud adoption is on the rise, more data is stored in the cloud

Security posture of cloud applications is a growing concern

AWS introduced the shared responsibility model:

AWS is responsible for **infrastructure and hardware**

The customer is responsible for **application, data and IAM**

But IAM is notoriously hard to master due to its complexity

---

McGraw Hill's S3 buckets exposed 100,000 students' grades and personal info

Educator gets an F for security
IAM Misconfiguration Example

1. CreateRole Carol
1. PutRolePolicy Carol
   ```json
   {"effect":"allow",
   "action":"*",
   "resource": "Alice"
   }
   ```
1. AssumeRole Carol
1. GetObject bobs-bucket/*
Existing AWS IAM Security Tools

Rhino Security Labs identified 20+ AWS IAM privilege escalation techniques and released Pacu - an open source tool that scans policies and detects potential usage of these techniques.

AWS developed Zelkova that mathematically verifies properties in IAM policies - for example checking if a bucket is public.
A Model Checking Approach

Known technique to check whether a model of a system satisfies certain properties

System
AWS IAM

Formal Model

Model Checking

Traces
Multi-Step
Attack Vectors

Property
Attack Target

Was already suggested in the context of network vulnerabilities by Ritchey & Ammann in 2000
AWS IAM Model

Attacker

Chooses the request to perform next

Impacts what actions the attacker can perform next

Policy Evaluation Logic

Affects the outcome of the policy evaluation

Actions Semantics

Affects AWS organization on the next step

Organization
Actions Semantics Encoding

Recognize only *means* actions - affect IAM directly or indirectly (in total around 60 AWS actions)


AWS documentation at https://docs.aws.amazon.com/IAM/latest/APIReference/API_PutRolePolicy.html

---

Recognize a privilege escalation prone action

**PutRolePolicy**

*Adds or updates an inline policy document.*

- When you embed an inline policy in a role, policy is created at the same time as the role.
- For more information about inline policies, see [IAM Permissions Management](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html).
- A role can also have a managed policy attached to a managed policy, use [CreatePolicy](https://docs.aws.amazon.com/IAM/latest/APIReference/API_CreatePolicy.html). For information about the maximum number of policies per user, see the [User Guide](https://docs.aws.amazon.com/IAM/latest/UserGuide/limits.html).

---

Understand AWS documentation and translate into an action semantics formula

```
action = PutRolePolicy implies
for all account in accounts:
  for all role in account.IAMRoles:
    ((resourceAccount = account.id and
      resourceName = role.name) implies
     role.MaximumPermissions' = true) and
    ((resourceAccount != account.id or
      resourceName != role.name) implies
     role.MaximumPermissions' = role.MaximumPermissions)
```
Policy Evaluation Logic Encoding

"Effect": "allow",
"Action": "*Role*",
"Resource": "*

"Effect": "deny",
"Action": "*Role*",
"Resource": "Alice"

Allow statements:
action in "*Role*" and
resourceType = Role

deny statements:
action in "*Role*" and
resourceName = "Alice" and
resourceType = Role

Encode formulas similarly to Zelkova

ServiceControlPoliciesAllow and
(ResourceBasedPoliciesAllow or
(IdentityBasedPoliciesAllow and
PermissionsBoundariesAllow)) and

not ServiceControlPoliciesDeny and
not ResourceBasedPoliciesDeny and
not PermissionsBoundariesDeny and
not IdentityBasedPoliciesDeny
Implementation and Evaluation

We implemented the model checking process using Java + Z3 SAT Solver API. We used a bounded model checking algorithm (BMC) + an exhaustive version.

We use a large pre-production AWS organization with ~100 accounts, with an average of ~200 IAM resources in each account.

We ask how does the model checking process manage to detect existing misconfigurations in real world AWS environments.

Gathered 141 different requests from security engineers, testing who can get access to data resources such as S3 buckets or SQS queues.
Evaluation Results

- Security engineers were satisfied with the results and fixed a lot of sneaky over-permissive policies.
- Additional performance evaluation showed that the approach detects IAM attacks of up to 5 steps, in accounts with hundreds of resources, in under a minute.
Thank you!
Questions?
iliashevrin@mail.tau.ac.il