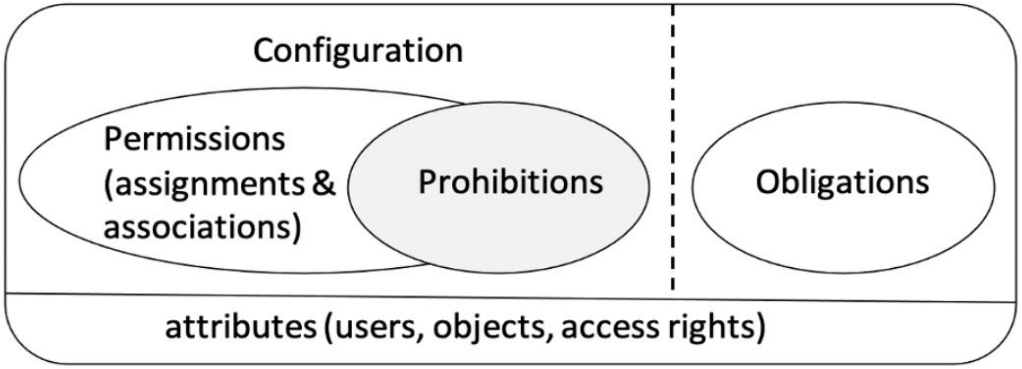
Coverage-Based Testing of Obligations in NGAC Systems

Erzhuo Chen, Vladislav Dubrovenski, Dianxiang Xu
University of Missouri – Kansas City
{vadpb7, lcrnd, dxu} @umsystem.edu

UMKC

Background

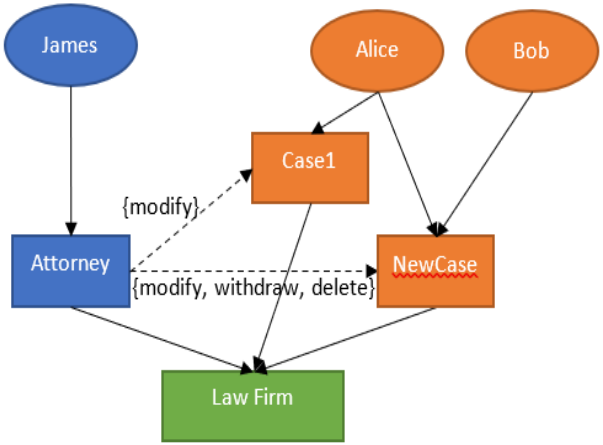
- NGAC (Next Generation Access Control), a new **access control** standard
- Proposed by NIST (National Institute of Standards and Technology)
- Designed to **address limitations**: limited flexibility, difficulty in managing policies, and limited interoperability



Configuration

Configuration ***C*** = <***U,UA,O,OA,AR,PC,ASSIGN,ASSOCIATION,PROHIBITION***>:

- *U*: set of users
- *UA*: set of user attributes
- *O*: set of objects
- *OA*: set of object attributes
- *AR*: set of all access rights
- *PC*: set of policy classes
- *ASSIGN & ASSOCIATION & PROHIBITION*: three sets of relations defined on policy elements



Obligation

<event pattern>::=[<user spec>] [<pc sepc>] <op spec> [<pe spec>]
<response>::=<response condition> <conditional action>{,<conditional action>}
<response condition>::=[if <condition> then]
<conditional action>::=[if <condition> then]<action>{,<action>}
<condition>::=<factor>{and <factor>}

SAMPLE

Obligation ϕ
Event: $\langle supervisor, delete, alex \rangle$
Response:
If *alex* exists **then**
  Actions:
  If $\neg alex.Loan \wedge supervisor.OfficeHour$ **then**
    delete $\langle alex, accounts \rangle$
  If $\neg \langle alex, accounts \rangle$ **then** delete object *alex*

Problem Statement

- NGAC is highly expressive and flexible, enabling creating complex access control policies. Additionally, it allows for dynamic changes to polices.

- However, there is a lack of work on quality assurance of NGAC policies. Meanwhile, the dynamic privilege changes through obligations come with potential concerns about errors and harm to the authorization state, leading to **unauthorized access**, **privilege escalation**, and **denial of service**.

- My research aims to investigate methods for ensuring the quality of obligations.

Approach

Coverage-based test generation method:

1. Define a family of coverage criteria
2. Generate constraints for satisfying coverage criterion
3. Solve constraints by a SMT-based solver

   “SMT-Based Verification of NGAC Policies”. V. Dubrovenski, E. Chen, and D. Xu. 2023.
4. Translate the solution into tests

Obligation Test

Format of an obligation test:

- Test input:

  A sequence of access requests, $\{q_1,q_2,\ldots,q_n\}$.
- Test oracle:

  Expected configuration changes, $\{O_1,O_2,\ldots,O_n\}$.

$t=\{\{q_1, O_1\}, \{q_2, O_2\}, ..., \{q_n, O_n\}\}$, where q_i represents access request and O_i represents the expected configuration changes after the permitted access q_n occurs.

Coverage Criteria

- Obligation Coverage (OC):
  - each obligation is triggered once
- Action Coverage (AC):
  - each action applies once
- Decision Coverage (DC):
  - each outcome (true/false) of decision is covered
- Factor/Decision Coverage (FDC):
  - each outcome (true/false) of factor combinations is covered
  - each factor independently affects the outcome

Obligation ϕ
Event: $\langle supervisor, delete, alex \rangle$
Response:
If *alex* exists **then**
  Actions:
  If $\neg alex.Loan \wedge supervisor.OfficeHour$ **then**
    delete $\langle alex, accounts \rangle$
  If $\neg \langle alex, accounts \rangle$ **then** delete object *alex*

Algorithm for generating OC tests

Obligation ϕ

Event: $\langle supervisor, delete, alex \rangle$

Response:

If *alex* exists **then**

Actions:

If $\neg alex.Loan \wedge supervisor.OfficeHour$ **then** delete $\langle alex, accounts \rangle$

If $\neg\langle alex, accounts \rangle$ **then** delete object *alex*

Function name: GenerateTestForOC

Input: Policy $P = (C_0, \Phi)$, C_0 is initial configuration, Φ is initial obligation

Output: *KnownSequences* is a set of distinct event sequences

```
1  foreach φ in Φ do
2      foreach sequence in KnownSequences do
3          if event(φ) in sequence then
4              continue;
5      newSeq ← seqFinder(P,φ);
6      if newSeq = null then
7          continue;
8      update ← true;
9      foreach sequence in KnownSequences do
10         if isSequenceCovered(sequence, newSeq) then
11             update ← false;
12             break;
13         else
14             if isSequenceCovered(newSeq, sequence) then
15                 remove sequence from KnownSequences;
16     if update then
17         add newSeq to KnownSequences;
18 return KnownSequences;
```

Algorithm for generating OC tests

Obligation ϕ

Event: $\langle supervisor, delete, alex \rangle$

Response:

If *alex* exists **then**

Actions:

If $\neg alex.Loan \wedge supervisor.OfficeHour$ **then** delete $\langle alex, accounts \rangle$

If $\neg\langle alex, accounts \rangle$ **then** delete object *alex*

newSeq={q1,q2,q3,q4} and

an old seq={q1,q3}

=> only keep {q1,q2,q3,q4}

Function name: GenerateTestForOC

Input: Policy $P = (C_0, \Phi)$, C_0 is initial configuration, Φ is initial obligation

Output: *KnownSequences* is a set of distinct event sequences

```
1  foreach φ in Φ do
2      foreach sequence in KnownSequences do
3          if event(φ) in sequence then
4              continue;
5      newSeq ← seqFinder(P,φ);
6      if newSeq = null then
7          continue;
8      update ← true;
9      foreach sequence in KnownSequences do
10         if isSequenceCovered(sequence, newSeq) then
11             update ← false;
12             break;
13         else
14             if isSequenceCovered(newSeq, sequence) then
15                 remove sequence from KnownSequences;
16     if update then
17         add newSeq to KnownSequences;
18 return KnownSequences;
```

UMKC

Algorithm for generating AC tests (part)

Obligation ϕ

Event: $\langle supervisor, delete, alex \rangle$

Response:

If *alex* exists **then**

Actions:

If $\neg alex.Loan \wedge supervisor.OfficeHour$ **then**

delete $\langle alex, accounts \rangle$

If $\neg\langle alex, accounts \rangle$ **then** delete object *alex*

```
1  foreach action in obligation.response do
2      if action.covered then
3          continue;
4      if currentConstraints = null then
5          action.covered ← true;
6          coveredCount++;
7          currentConstraints ← obC ∧ reC ∧ action.conC;
8          solution ← action.selfSolution;
9          solution.involvedActions ← solution.involvedActions ∪ action.index;
10     else
11         tmpConstraints ← currentConstraints ∧ action.conC;
12         tmpSolution ← solver(P,tmpConstraints);
13         if tmpSolution = null then
14             continue;
15         else
16             action.covered ← true;
17             coveredCount++;
18             currentConstraints ← tmpConstraints;
19             solution ← tmpSolution;
20             solution.involvedActions ← solution.involvedActions ∪ action.index;
```

Evaluation: subject policies

	#PC	#UA	#OA	#ASM	#ASC	#PRO	#OBL
Bank	2	6	10	33	6	-	-
GPMS	4	34	27	91	8	-	19

Bank:

An access control system of the management structure of a bank system.

GPMS:

A web-based application that aims to automate the grant proposal approval workflow at an academic institution.

Evaluation: obligation mutation operators

No	Fault Type		Mutation Operator
1	Extra obligation	ROB	Remove one OBligation
2	Wrong subject	CES	Change Event Subject
3	Extra subject	RES	Remove Event Subject
4	Wrong operation	CEO	Change Event Operation
5	Missing operation	AEO	Add Event Operation
6	Extra operation	REO	Remove Event Operation
7	Wrong target	CET	Change Event Target
8	Extra target	RET	Remove Event Target
9	Extra condition	ROC	Remove One Condition
10	Negated condition	NOC	Negate One Condition
11	Extra condition	ROF	Remove One Factor
12	Negated condition	NOF	Negate One Factor
13	Extra action	ROA	Remove One Action
14	Wrong action	COA	Change One Action
15	Wrong ascendant in create	CAC	Change Ascendant in Create
16	Wrong descendant in create	CDC	Change Descendant in Create
17	Wrong direction in create	RDC	Reverse Direction of Create
18	Wrong assignment ascendant	CASA	Change AScendant in Assign
19	Wrong assignment descendant	CDA	Change Descendant in Assign
20	Wrong assignment direction	RDA	Reverse Direction of Assign
21	Wrong grant subject	CSG	Change Subject in Grant
22	Wrong grant target	CTG	Change Target in Grant
23	Wrong access right in grant	CARG	Change Access Right in Grant
24	Missing access right in grant	AARG	Add Access Right in Grant
25	Extra access right in grant	RARG	Remove Access Right in Grant
26	Wrong subject in deny	CSD	Change Subject in Deny
27	Wrong target in deny	CTD	Change Target in Deny
28	Wrong access right in deny	CARD	Change Access Right in Deny
29	Missing access right in deny	AARD	Add Access Right in Deny
30	Add access right in deny	RARD	Remove Access Right in Deny

Evaluation

Mutation Scores (%) for GPMS-NGAC

Mutant Group	# Mutants	OC	AC	DC	FDC
event mutants	1548	87.3	87.5	87.5	87.5
action mutants	5650	4.7	73.2	73.2	73.2
condition mutants	168	0	38.1	85.1	91.7
overall	7366	22.0	75.5	76.6	76.8

Mutation Scores (%) for Bank

Mutant Group	# Mutants	OC	AC	DC	FDC
event mutants	188	70.7	72.9	73.9	73.9
action mutants	78	0	82.8	83.9	83.9
condition mutants	18	0	38.5	69.2	69.2
overall	306	43.5	73.0	76.5	76.5

MKPR Scores

Subject	OC	AC	DC	FDC
GPMS-NGAC	34.4	62.4	43.3	32.8
Bank	11.1	6.1	3.7	2.8

MS(**M**utation **S**core) = #KM(**K**illed **M**utants) / #NEM(**N**on-**E**quivalent **M**utants)

MKPR(**M**utants **K**illed **P**er **R**equest) = #KM/ #Test

Conclusions

- Presented the test coverage criteria for NGAC obligations
- Presented efficient methods for generating tests to satisfy each coverage criterion
- Conducted empirical studies to evaluate the fault detection capabilities and cost-effectiveness of these coverage-based test generation methods
- FDC test suites, can provide a high level of confidence in the correct enforcement of access control