IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

A Robust Approach for the Detection and Prevention of Conflicts in I2NSF Security Policies

SOTERN TEAM - IRISA, IMT ATLANTIQUE

- Do Duc Anh NGUYEN (Presenter) - do-duc-anh.nguyen@imt-atlantique.fr
- Fabien AUTREL - fabien.autrel@imt-atlantique.fr
- Ahmed BOUABDALLAH - ahmed.bouabdallah@imt-atlantique.fr
- Guillaume DOYEN - guillaume.doyen@imt-atlantique.fr

Outline

1. Introduction
   - Context
   - Intents for Security: the I2NSF Framework
2. Conflict Problem and Resolution
   - Problem of Conflict
   - Conflict Detection and Resolution
3. Evaluation
   - Conditions and Performance Metrics
   - Results
4. Conclusion and Future Works

Outline

1. Introduction
   - Context
   - Intents for Security: the I2NSF Framework
2. Conflict Problem and Resolution
   - Problem of Conflict
   - Conflict Detection and Resolution
3. Evaluation
   - Conditions and Performance Metrics
   - Results
4. Conclusion and Future Works

Context

Ensuring adequate protection in complex infrastructures requires the automation of security management

Intent-Based Networking (IBN)

IBN allows the user to specify the intent, which is the desired outcome, without the need for detailed operations to automate configuration orchestration

Interface to Network Security Function (I2NSF) [1]

The I2NSF framework, which supports the implementation of IBN components, aims to provide users with software interfaces and data models to specify policy rule sets and automate configuration management

[1] Framework for Interface to Network Security Functions, RFC 8329, 2018

Intents for Security: the I2NSF Framework

Reference architecture contains

- I2NSF User
- Network Operator Management System (Security Controller)
- Developer Management System (DMS)
- Network Security Functions (NSFs)

[2] proposed the implementation of I2NSF

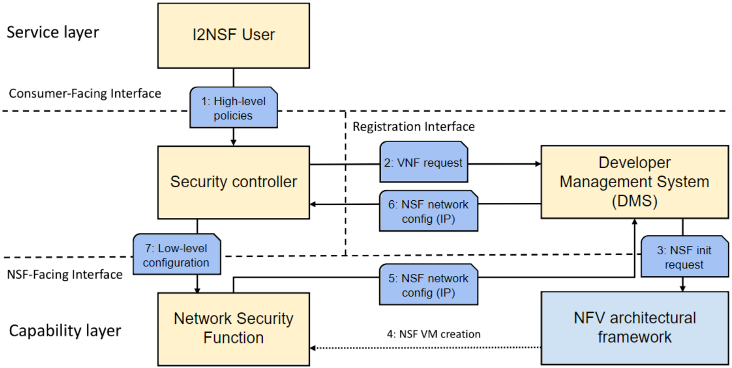

Service layer
I2NSF User
Consumer-Facing Interface
1: High-level policies
Registration Interface
Security controller
2: VNF request
Developer Management System (DMS)
6: NSF network config (IP)
7: Low-level configuration
3: NSF init request
NSF-Facing Interface
5: NSF network config (IP)
Capability layer
Network Security Function
4: NSF VM creation
NFV architectural framework

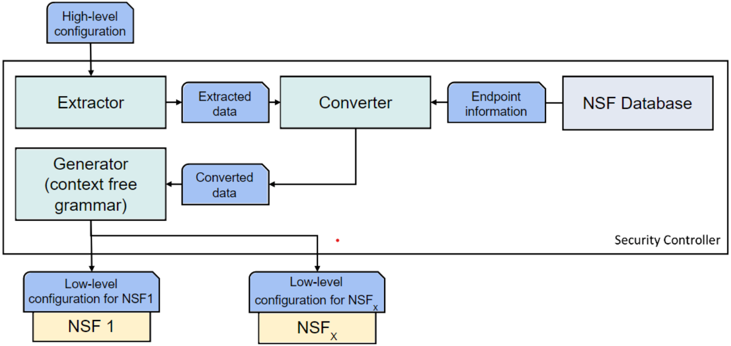
Figure: I2NSF architecture [2]

[2] IBCS: Intent-Based Cloud Services for Security Applications, IEEE Communications Magazine, 2020

I2NSF Security Controller

Security Controller [2] contains

- Extractor: Extract data
- Converter: Convert data into low-level data
- Generator: Provision required NSFs and generate corresponding configurations



Example: a policy rule ”Prohibit employees from accessing social websites during 9:00-18:00”

- Extractor: ”employees”, ”social website”, ”9:00-18:00”, ”Prohibit”
- Converter: ”10.0.0.3”, ”facebook, instagram”, ”9:00-18:00”, ”Prohibit”
- Generator: Concrete configurations for required NSFs

Research Questions

I2NSF high-level policy rules are specified based on the YANG data model

```
<I2NSF>
    <policy-name>block_web_employee</policy-name>
    <event>
        <time-information>
            <begin-time>09:00</begin-time>
            <end-time>18:00</end-time>
        </time-information>
    </event>
    <condition>
        <src>employees</src>
        <dst>sns-websites</dst>
    </condition>
    <action>drop</action>
</I2NSF>
```

```
<I2NSF>
    <policy-name>allow_web_employee</policy-name>
    <event>
        <time-information>
            <begin-time>12:00</begin-time>
            <end-time>14:00</end-time>
        </time-information>
    </event>
    <condition>
        <src>employees</src>
        <dst>sns-websites</dst>
    </condition>
    <action>pass</action>
</I2NSF>
```

Conflict occurs and makes I2NSF unreliable. Therefore, research questions are raised:

- How can we identify and detect conflicting I2NSF high-level rules?
- Can state-of-the-art conflict detection methods fit with the I2NSF features?
- What performance is required for such an approach to be acknowledged?

Outline

1 Introduction

- Context
- Intents for Security: the I2NSF Framework

2 Conflict Problem and Resolution

- Problem of Conflict
- Conflict Detection and Resolution

3 Evaluation

- Conditions and Performance Metrics
- Results

4 Conclusion and Future Works

Problem of Conflict

Conflicting Rules [3] in Attribute-Based Access Control (ABAC)

A conflict occurs when a user request is applied by two rules, but their decisions are different

Adapted definition from [3], two I2NSF policy rules are conflicting rules if

- Their actions are contradictory
- One rule shares all attribute identifiers of Event and Condition with another rule
- Values of shared attributes must intersect

Rule	Event	Condition	Action
R1	time=[09:00, 18:00]	src=employee, dest=sns-websites	Drop
R2	time=[12:00, 14:00]	src=employee, dest=sns-websites	Pass

→ Simply detect this conflict by comparing their attribute values and actions

⇒ Explicit conflicting rules

[3] A novel conflict detection method for ABAC security policies, Journal of Industrial Information Integration, 2021

Problem of Conflict

Rule	Event	Condition	Action
R1	time=[09:00, 18:00]	src=employee, dest=sns-websites	Drop
R2		src=employee, dest=sns-websites	Pass

→ Cannot compare due to an existing absent attribute

⇒ Implicit conflicting rules

Rule	Event	Condition	Action
R1	time=[09:00, 18:00]	src=employee, dest=sns-websites	Drop
R2	time = any	src=employee, dest=sns-websites	Pass

→ Consider value of absent attributes as ”any” [3] ⇒ Become explicit conflicting rules

[3] A novel conflict detection method for ABAC security policies, Journal of Industrial Information Integration, 2021

Conflict Detection

Examine high-level policies

- Value-inconvertible attributes can be compared (e.g., no converted infor for ”time” value)
- Value-convertible attributes cannot be precisely compared (e.g. the value of ”src” can be converted to a set of IP addresses)

The conflict identified between high-level policies is called a potential conflicting rule

Potential conflicting rules

Two high-level policy rules are potential conflicting rules if

- Their actions are contradictory
- One rule shares all attribute identifiers of Event and Condition with another rule
- ~~Values of shared attributes must intersect~~

→ The real-time conflict checker uses this definition to identify potential conflicts by checking a new rule against the existing rule set

Conflict Resolution

To mitigate the conflicting rule set, the I2NSF data model is extended to allow users to express

Partial ordering relationship (POR)

A POR R_p is defined over the abstract rule set R_r . Let r_1 and r_2 be two abstract rules belonging to R_r . If $R_p(r_1, r_2)$ holds, then r_1 has a higher priority than r_2

→ If $R'_p(r_2, r_1)$ exists, $R_p(r_1, r_2)$ will be considered as invalid POR; otherwise, it is valid

Separation Constraint (SC) [4]

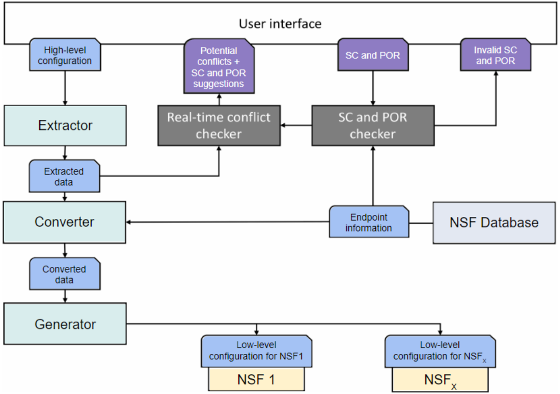
A *SC* is defined to forbid a value from belonging to two sets at the same time

Example: The $SC(doctor, nurse)$ prohibits the IP address sets associated with the *doctor* and *nurse* in the "src" attribute from sharing common IP addresses

→ If *doctor* and *nurse* share the common IP addresses, this *SC* is invalid; otherwise, it is valid

[4] The RSL99 language for role-based separation of duty constraints, Proceedings of the fourth ACM workshop on Role-based access control, 1999

Global architecture



We propose

- Two novel components are added to the I2NSF Security Controller
  - The real-time conflict checker
  - The Separation Constraint and Partial Ordering Relationship checker (SC and POR checker)
- Two closed loops interacting with the user

Outline

1. Introduction
   - Context
   - Intents for Security: the I2NSF Framework
2. Conflict Problem and Resolution
   - Problem of Conflict
   - Conflict Detection and Resolution
3. Evaluation
   - Conditions and Performance Metrics
   - Results
4. Conclusion and Future Works

Conditions and Performance Metrics

Setup

- The testbed was presented at the IETF Hackathon
- Machine: Ubuntu 20.04.4 LTS, Intel i7-10750H 2.6GHz, 16 GB RAM
- Multiple sets of 10,000 random rules

Rule	Src	Dst	Start time	End time	Action
R1	employees	sns-websites	09:00	18:00	drop
R2	employees	sns-websites	12:00	14:00	pass
R3	employees	sns-websites			pass
R4	employees		15:00	16:00	pass
R5		sns-websites	17:00	19:00	pass

Performance Metrics:

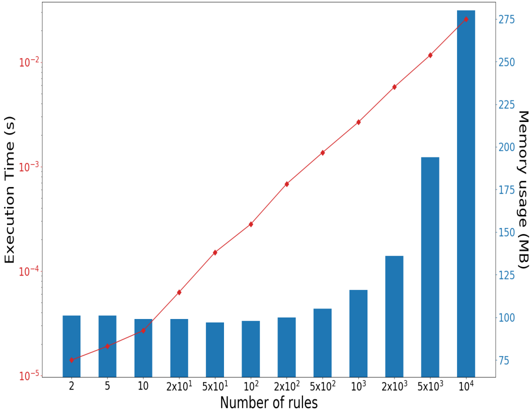
- Memory usage
- Execution time

Early test results after 10 repetitions

- Accuracy: Detect all conflicts correctly
- Performance
  - Memory usage: 286.3 ± 0.8 (MB)
  - Execution time: 160.79 ± 7.39 (s)

Results (1)

Impact of the number of rules

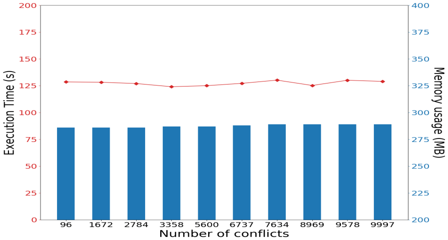


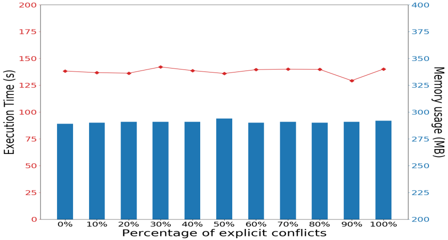
- Preparation: A random rule set with no conflicts, no SC
- Observation:
  - Both the execution time and memory usage follow a polynomial complexity according to the number of rules
  - The execution time does not exceed a few dozen of seconds while the memory does not exceed 300MB

→ This growth may be an issue for extremely large-scale rule sets, containing up to 10,000 rules

Results (2)

Impact of the number and type of conflicts

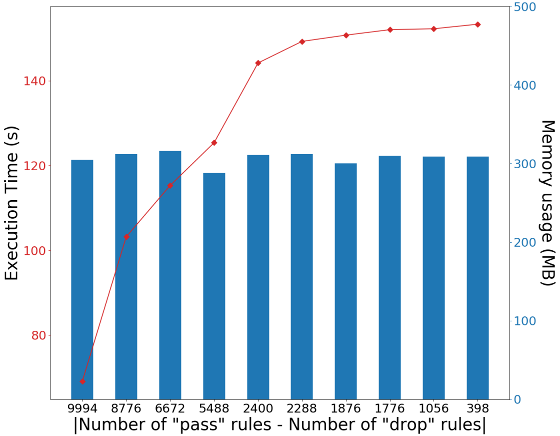




- Preparation: Multiple random rule sets with 50% pass/drop rules, and no SC
  - First scenario: Vary the number of conflicts and set the ratio of explicit and implicit conflicts to 50%.
  - Second scenario: Fix the number of conflicts at 10% and vary the ratio of explicit and implicit conflicts.
- Observation: Both execution time and memory usage are independent of these two factors
  → The real-time conflict checker works properly regardless of the type and number of conflicts

Results (3)

Impact of the number of drop/pass ratio

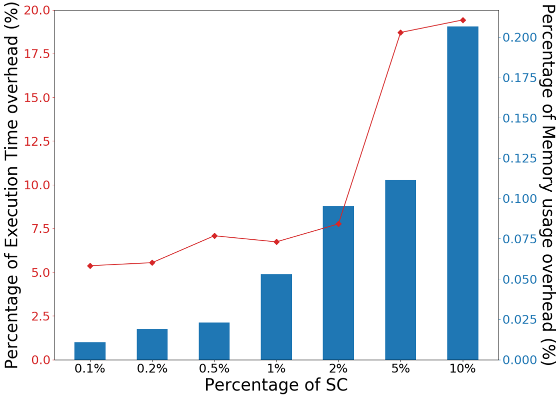


- Preparation: Multiple random rule sets with different ratios of pass and drop rules
- Observation:
  - The smaller the value of $|number\_drop\_rule - number\_pass\_rule|$, the longer the execution time
  - Memory usage is independent

Explain: If two rules have the same action, the *detect* can halt at the action check

Results (4)

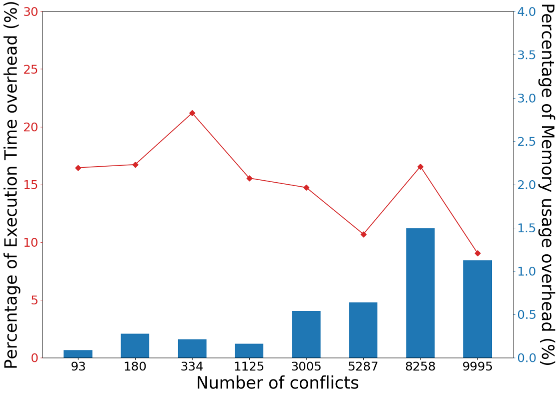
Impact of separation constraints: Measure induced performance overhead of SCs



- Preparation: a random rule set without any conflict
- Observation:
  - SCs do not impact the memory usage
  - SCs can cause execution time overhead, up to 5% with 0.1% of SCs and 20% with 10% of rules having SCs

Results (5)

Impact of the number of conflicts with separation constraints: Measure the performance overhead caused by the number of conflicts when SCs are present



- Preparation: multiple random rule sets with 50% pass/drop rules, 50% explicit/implicit conflicts
- Observation: Both execution time and memory usage have an acceptable and stable overhead with a maximum value of 15% and 1,5%, respectively

→ Assess the well support of SCs of our solution in the presence of any amount of conflicts

Outline

1. Introduction
   - Context
   - Intents for Security: the I2NSF Framework
2. Conflict Problem and Resolution
   - Problem of Conflict
   - Conflict Detection and Resolution
3. Evaluation
   - Conditions and Performance Metrics
   - Results
4. Conclusion and Future Works

Conclusion and Future Works

Propose the detection and prevention mechanisms for I2NSF at the high-level domain, our approach

- Identifies potential conflicting requirements and prevent them at the high level domain
- Frees users from all the issues concerning the soundness of the policy to be deployed
- Can be adapted and applied in any framework using the attribute-based formalism

⇒ This approach can promote I2NSF adaptation in experimental investigations

Future works

- Short-term perspective: Consider checking dependent rules, such as in a stateful firewall
- Long-term work: Concern AI to analyze and guide conflict resolution by suggesting SCs and PORs to help users create secure and robust security requirements

Question

Thank you for listening. Any question?

References I

[1] Diego Lopez et al. Framework for Interface to Network Security Functions. RFC 8329. Feb. 2018. DOI: 10.17487/RFC8329. URL: https://www.rfc-editor.org/info/rfc8329.

[2] Jinyong Kim et al. „IBCS: Intent-Based Cloud Services for Security Applications“. In: IEEE Communications Magazine 58.4 (2020), pp. 45–51. DOI: 10.1109/MCOM.001.1900476.

[3] Gang Liu et al. „A novel conflict detection method for ABAC security policies“. In: Journal of Industrial Information Integration 22 (2021), p. 100200.

[4] Gail-Joon Ahn and Ravi Sandhu. „The RSL99 language for role-based separation of duty constraints“. In: Proceedings of the fourth ACM workshop on Role-based access control. 1999, pp. 43–54.

Detection

```
Input: $R0$, $R1$
Output: True if $R0$ conflicts with $R1$ and False otherwise
if $is\_different\_action(R0, R1)$ then
    if not $exist\_nonoverlapped\_ap(R0, R1)$ then
        return True
    end if
end if
return False
```

Algorithm 1: *detect*

Function *detect* follows ABAC proposal [3]

- $is\_different\_action(R0, R1)$: compare actions
- $exist\_nonoverlapped\_ap(R0, R1)$: compare all attributes

→ The complexity: $O(A)$ where
A: the attribute number defined in the DM

⇒ The real-time conflict checker uses *detect* to check a new rule against the installed rule set

[3] A novel conflict detection method for ABAC security policies, Journal of Industrial Information Integration, 2021

Real-time Conflict Checker

```
Input: new rule $R$
Output: a potential conflicting rule set with $R$
set_conflicting_rules = {}
for $R_i$ in existing_rules do
    if SC_violation_validate($R$, $R_i$) then
        if detect($R$, $R_i$) then
            set_conflicting_rules += {$R$, $R_i$}
        end if
    end if
end for
return set_conflicting_rules
```

Algorithm 2: Real-time conflict checker

The real-time conflict checker

- *SC_violation_validate*(R, R_i): check SC violation
- *detect*$(R0, R1)$: check conflict

→ The complexity: $O(N * A)$ where
N: the number of installed rules
A: the attribute number defined in the DM

Partital Ordering Relationship

Example

```
<I2NSF>
  <policy-name>POR_R1_R2</policy-name>
  <priority-order-rules>
    <rule-identity>R1</rule-identity>
    <rule-identity>R2</rule-identity>
  </priority-order-rules>
</I2NSF>
```

Separation Constraint

Given two rules

- ”Forbiding doctors from accessing social websites”
- ”Allow nurses to access social websites”

```
Example     <I2NSF>
                <policy-name>SC_doctor_nurse</policy-name>
                <separation-constraint><not-share-common-value>
                        <attribute-id>src-target</attribute-id>
                        <value>doctor</value>
                        <value>nurse</value>
                </separation-constraint></not-share-common-value>
            </I2NSF>
```

[4] The RSL99 language for role-based separation of duty constraints, Proceedings of the fourth ACM workshop on Role-based access control, 1999

Deployment of an I2NSF testbed

A ground architecture to allow the deployment of any subsequent contribution

- Selection of a testbed implemented and presented at IETF Hackathon (#104 to #113)
- Installation and setup of an underlying Devstack distribution
- Reproduction of the standard scenario considered in [2]

Several bugs and issues which made the testbed setup and standard test scenario difficult to implement

- Installation errors in inconsistent version between Devstack plugins
- NSF database of Security Controller is inconsistent in capabilities compared to their instruction
- NSFs do not send IP address to DMS after being initiated
- Service chaining failed because NSFs do not process the incoming packet

[2] IBCS: Intent-Based Cloud Services for Security Applications, IEEE Communications Magazine, 2020