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

Do Duc Anh Nguyen, Fabien Autrel, Ahmed Bouabdallah and Guillaume Doyen
†OCIF - IRISA (UMR CNRS 6074), IMT Atlantique
Rennes, France
firstname.lastname@imt-atlantique.fr

***Abstract*—In order to maintain a sufficient protection level of their infrastructure, automating security management is at the core of current operators issues. The Interface to Network Security Function (I2NSF) is a framework that takes part of the Intent-Based Networking (IBN) paradigm. It consists of automating the translation of high-level policies into low-level configurations of Network Security Functions (NSF) and appears as a promising way to overcome the complexity of this challenging task. However, if the I2NSF framework provides a comprehensive architectural and data model for such an automation, it provides neither detection nor prevention mechanisms against conflicting security requirements. In this paper, we assess to what extent state-of-the-art mechanisms can shift the initial I2NSF proposal toward a robust framework. As such, we extend (1) the reference architecture to integrate some checking components and (2) the consumer-facing data model to enforce separation constraints and partial ordering relationships. By considering a large set of rules and conflicting situations, we evaluate the performance of our solution within an early implementation of I2NSF achieved in an IETF Hackathon.**

***Index Terms*—Security management, I2NSF, Policy, Conflict detection and mitigation**

I. Introduction

In order to ensure the security of today's high-complexity infrastructures and ecosystems, the need for self-protection has been identified early in autonomic computing, and network softwarization has been the cornerstone, which, for a decade, made the concept of autonomy, or to a lesser extent, automation, a reality. Following this idea, Intent-Based Networking (IBN) [1] was introduced with the aim of facilitating human operators against configuration management and monitoring by using $intents$. The Interface to Network Security Function (I2NSF) [2], a framework from the Internet Engineering Task Force (IETF), aims to facilitate security management by providing standard interfaces that consider high-level requirements issued by human operators to dynamically enforce security configurations on Network Security Functions (NSFs).

However, the translation of security policies into NSF configurations lacks consistency and compliance verification, leading to conflicts and unpredictable domain state. In order to address this issue, we strive to answer the following research questions: *(1) How can we identify and detect conflicting I2NSF high-level rules? (2) Can state-of-the-art conflict detection methods fit with the I2NSF features? (3) What performance is required for such an approach to be acknowledged?*

This paper proposes a set of mechanisms to detect and resolve potential conflicting rules in I2NSF at the consumer-facing level. Firstly, given the expression of high level requirements in I2NSF, we exploit its closeness to the Attribute-Based Access Control (ABAC) policy formalism to assess to what extent an acknowledged detection solution from the literature [3] targeting ABAC conflicts may fit with I2NSF features. Secondly, we extend the consumer-facing data model to express separation constraints [4] and partial ordering relationships, which mitigate the potential conflicting rule sets identified. Finally, given the systematic conflict checking strategy we adopt, we evaluate its scalability in terms of execution time and memory usage with a large set of 10,000 rules, considering exhaustive conflicting situations.

The rest of this paper is organized as follows: Section II presents the background and related work of IBN security, I2NSF as well as conflict detection and resolution in security policies. Section III motivates our formal reformulation of rule conflicts by remaining faithful to the ABAC spirit from which we borrow and adapt the central concepts. Section IV depicts an extended I2NSF architecture with conflict checkers, the conflicting rule detection mechanism associated with separation constraints, and partial ordering relationships. Section V details the testbed we deployed, our implementation of conflict detection and resolution, and its evaluation. Finally, Section VI concludes the paper and introduces future work.

II. Background and Related Work

In this section, we highlight revelant contributions of the IBN paradigm with an emphasis on security as well as the I2NSF framework. Finally, we review the conflict detection and mitigation in access control policy contributions fitting with the I2NSF use case.

A. Intent-Based Networking

In areas where device configuration is time-consuming and error-prone, early developments with IBN have proved to be very promising. In the context of connected vehicles, [5] proposes *LocJury*, a framework allowing users to request the location of vehicles via IBN and detect potential threats of location privacy violation. In [6], an IBN system is proposed in which users can express their data access requests to check their validity against regulations. Besides privacy and data

protection, IBN is also employed in network security: automatic DDoS response system [7] [8], intent-based multilayer secure service [9], and multi-nation military communication and information systems [10].

B. The I2NSF framework

I2NSF aims to provide software interfaces and data models to facilitate users in controlling and monitoring the behavior of NSF. Such functions conduct network security tasks such as detecting abnormal traffic or providing data protection. As depicted in Figure 1a, which shows the global architecture of I2NSF, this framework is globally organized around two main layers: The Service Layer allows users to express their security policies, and the Capability Layer depicts mechanisms for managing NSF's operations at the implementation level. I2NSF Consumer-Facing Interface (CFI) introduces policies, which are then translated and provisioned by the Security Controller. The Developer's Management System (DMS) manages NSFs registered by vendors. I2NSF targets Network Functions Virtualization (NFV) as a main scenario in which it proposes to manage virtual machines implementing the NSFs through the NSF-facing interface, thus standing for a relevant framework for automatic security deployment [11]. Especially, a first use-case and testbed implementation of I2NSF's main operations is given in [12]. However, I2NSF's scope goes beyond this use case (e.g., IPSec automation [13]).

```
<I2NSF>
   <event-time>09:00-18:00</event-time>
   <condition>
      <src>employee</src>
      <dest>sns-websites</dest>
   </condition>
   <action>drop</action>
</I2NSF>
```

Listing 1: High-level policy example in I2NSF (simplified version), from [12]

Listing 1 shows an example of a simplified high-level policy expressing a security requirement: employees cannot access SNS (Social Network Site) during working time. According to the translation proposed in [2], [15], and depicted in Figure 1b, the I2NSF processing requires three ordered transformations applied by the components in the Security Controller: (1) The rule's attributes are extracted by the Extractor (i.e., "09:00-18:00", "employee", "sns-websites"); (2) The Converter converts the extracted data into concrete ones like IP addresses and websites' names (i.e., "10.0.0.3-10.0.0.15", "Facebook, Instagram"); (3) The Generator generates low-level policies for selected NSFs (e.g., Firewall and Web-filter NSF).

However, I2NSF can suffer from conflicts in high-level policies, leading to inconsistency in NSFs. Some basic resolution strategies are proposed, such as the First Matching Rule [16], which, however, prevents later matching rules from being applied.

C. Policy Conflict Detection and Mitigation

In access control models, if two rules of a policy involve for the same subject and object both permission and prohibition of its access then a conflict arises and the policy is unsound. This issue has been tackled in several formalisms, such as XACML [17], network rules [18], [19], and other access control models with a high degree of abstraction [20], [21], among others. The general solution to this issue consists to establish a partial ordering relationship between the rules. The undelying formalism provides a concept to abstract the subjects of a rule into a set, such as the notion of role in Role-based Access Control (RBAC) [22]. Conflicts can then be solved by introducing constraints that forbid a subject from belonging to two separate sets specified by two conflicting rules. Besides, in order to detect them, approaches such as [3], [23] use pair-wise rule checking to avoid missed conflicts.

III. Leveraging ABAC conflict detection and mitigation methods for I2NSF

The proximity between the Yang data model of the CFI [24] and ABAC motivates our adoption of the ABAC framework to express our solution to detect and mitigate conflicts. As such, in this section, we reformulate the definitions initially proposed in [3], [23] to make them fit with the I2NSF context.

A. Explicit/potential conflicting rules

A conflict between two rules occurs when a user request is applied by both but their decisions are different. For a rule R, we define $E(R)$, $C(R)$ and $Action(R)$ respectively the set of attribute expressions involved in the events and conditions parts and the set of actions associated. If req stands for a request sent to the system, we also define $|E(R)|_{req}$ and $|C(R)|_{req}$ the attribute expressions respectively associated with $E(R)$ and $C(R)$ that apply for req. We first introduce a general notion of conflict, from which we derive operational definitions of explicit and potential conflicting rules.

Definition III.1 (Conflict)**.** A conflict between rules R_i and R_j $(i \neq j)$ occurs if it exists a user request req such that:

1) The two rules are applicable for the given request: $|E(R_i)|_{req} \bigwedge |C(R_i)|_{req} \equiv |E(R_j)|_{req} \bigwedge |C(R_j)|_{req}$
2) The two rules' actions are different: $Action(R_i) \neq Action(R_j)$

According to Definition III.1, a conflict can occur on-the-fly when a specific request is sent. Let us consider again the example from [12] that I2NSF allows us to specify:

1) R_1: Block/Drop employees from accessing sns-websites during the working time (09:00-18:00)
2) R_2: Allow/Pass employees to access sns-websites at lunch time (12:00-14:00)

If an employee accesses Facebook between 12:00-14:00 (included in sns-websites), R_1 and R_2 will be identified as conflicting rules because they are applied to this request with two different actions ($drop \neq pass$).

Definition III.2 (Explicit conflicting rules)**.** R_i and R_j $(i \neq j)$ are identified as explicit conflicting rules if:

1) One rule shares all attribute identifiers with another rule. Suppose A represents an attribute identifier, then:

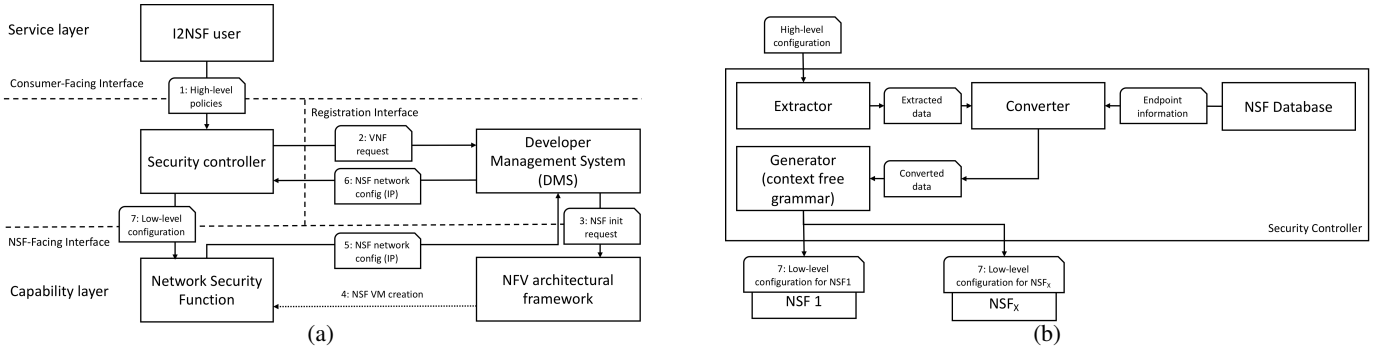

Fig. 1: The I2NSF framework. (a) General architecture and operation [14]; (b) Security controller components [11]

$\forall A \in E(R_i) \bigcup C(R_i), \exists A' \in E(R_j) \bigcup C(R_j)$, such that $A = A'$

2) All the shared attribute identifiers have intersecting values. Suppose A is a shared identifier with $val(A)_{|R_i}$ representing the set of possible values of A when used in rule R_i then: $val(A)_{|R_i} \cap val(A)_{|R_j} \neq \emptyset$
3) The two rules actions are different: $Action(R_i) \neq Action(R_j)$

Besides, I2NSF is also facing some implicit conflicting rule situations since users can express high-level policies with absent attributes. For example:

1) R_1: Block/Drop employees from accessing sns-websites
2) R_2: Allow/Pass employees to access sns-websites at lunch time (12:00-14:00)

R_1 does not specify time constraints, which can be considered as any and then shares the same value with that of R_2 ($Val(ap_{time_{|R_1}}) \cap Val(ap_{time_{|R_2}})$ = 12:00-14:00). It is possible to transform implicit conflicting rules into explicit ones by adding the absent attributes with the value any. Adding an absent attribute to a rule R must follow Law III.1 to create $\overline{R}$, which is equivalent to R.

Law III.1. For a given rule R = (E, C, $Action$) and an attribute A:

- If A is an event attribute not present in E then $\overline{E} = E \wedge (A = any)$ is an extension of E such that $\overline{R}$ = ($\overline{E}$, C, $Action$) is semantically equivalent to R
- If A is a condition attribute not present in C then $\overline{C} = C \wedge (A = any)$ is an extension of C such that $\overline{R}$ = (E, $\overline{C}$, $Action$) is semantically equivalent to R

Completing our previous example, we get:

1) $\overline{R}_1$: Block/Drop employees from accessing sns-websites at any time
2) R_2: Allow/Pass employees to access sns-websites at lunch time (12:00-14:00)

Weakening Definition III.2 by removing second condition creates a superset of explicit conflicting rules.

Definition III.3 (Potential conflicting rules)**.** R_i and R_j ($i \neq j$) are identified as potential conflicting rules if:

1) One rule shares all attribute identifiers with another rule. Suppose A represents an attribute identifier, then: $\forall A \in E(\overline{R_i}) \bigcup C(\overline{R_i}), \exists A' \in E(\overline{R_j}) \bigcup C(\overline{R_j})$, such that $A = A'$
2) The two rules' actions are different: $Action(\overline{R_i}) \neq Action(\overline{R_j})$

B. Separation Constraint and Rule Ordering Definitions

Beyond the conflict detection in policy rules, separation constraints and rule ordering can be used to prevent the conflicts. We formalize them in our context as follows.

Definition III.4 (Separation constraint)**.** A separation constraint is a binary relationship SC defined over the superset S_{ip} of the sets representing the domain values of the attributes used in the high-level policy. A separation constraint forbids an attribute value from belonging to two elements of S_{ip} at the same time:

1) $SC \subseteq S_{ip} \times S_{ip}$
2) $\forall (a, b) \in SC \ . \ a \cap b = \emptyset$

Definition III.5 (Partial ordering relationship over rule set)**.** A partial ordering relationship 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.

IV. A Conflict Detection and Mitigation Framework

Section III provides firm foundations to develop an extension of the I2NSF architecture that implements the ability to cope with conflicting requirements expressed at the CFI. Consequently, the security controller is enriched with several novel components, whose operations are detailed in this section.

A. Global architecture and process

As depicted in Figure 2, our extended I2NSF architecture exhibits two novel components. The real-time conflict checker is utilized to detect potential conflicting rules while the Separation Constraint (SC) and Partial Ordering Relationship (POR) checkers handle the validation of the conflict prevention mechanisms. As such, two feedback loops of operations with the user are defined in this architecture. Firstly, users can express consumer-facing policy rules to the Security Controller, which will be checked against all the existing ones. The outcome of this checker is then sent as a first feedback to the user who can in turn correct its rules specification. Then, the second

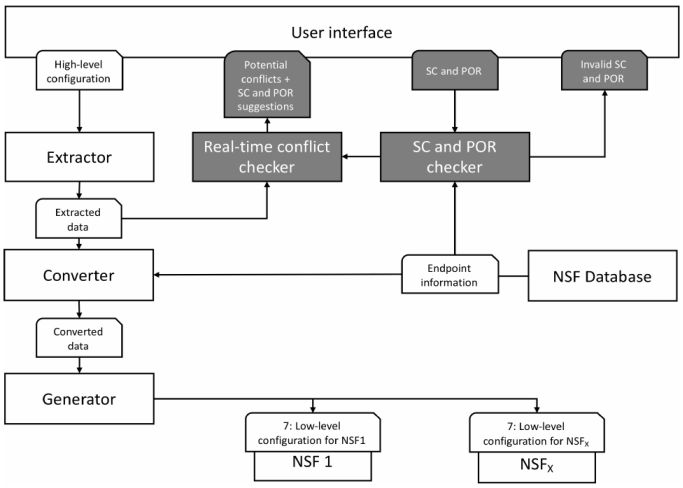

Fig. 2: The architecture of detection and prevention applied to I2NSF. White boxes are those existing in the referred Security Controller, while grey boxes are those we added for conflict detection an prevention.

loop allows the user to subsequently specify SCs and PORs to the system. The SC and POR checker will then determine if the latter are valid according to the infrastructure setup (e.g., endpoints' information) stored in the NSF database. The outcome of this second validation is also sent back to the user to recognize any invalid SC or POR.

B. SC and POR Data Model and checker

In order to integrate SCs and PORs in the I2NSF framework, we have extended the consumer-facing YANG data model with some lightweight statements. For SCs, following our example, users only need to express two elements $employee$ and $manager$ to state that an $employee$ and a $manager$ specified in the attribute src cannot share the same value (i.e., they cannot share a common IP address since an employee and a manager cannot be the same person). In terms of PORs, the policy names of two rules with the order of $name\_R0$, $name\_R1$ are enough to specify that R_0 has a higher priority than R_1. Deploying SCs and rule ordering relationships raise new issues of validation. At this stage, a mechanism, namely the SC and POR checker, is needed to validate the SCs and PORs that have just been deployed by the user. For SC validation, the checker obtains converted data from the two elements (e.g., endpoints' names) mentioned in a given SC. It is considered invalid if their values overlap and vice versa ($\forall(a,b) \in SC\ .\ a \cap b \neq \emptyset$). In terms of POR validation, a POR $R_p(r_1, r_2)$ is deemed invalid if $R'_p(r_2, r_1)$ already exists.

C. Conflicting Rules checker

Given the SC and POR deployment and correctness validation exposed above, valid SCs are used in the real-time conflict checker to identify any rule that has a constraint violation and prune the potential set of conflicting rules. Then, Definition III.3 is implemented to detect potential conflicts with finer granularity, and it results in the final conflicting rule set, which is also the output of the real-time conflict checker. At this stage, the valid PORs can handle the remaining conflicts by suggesting the user choose the rule with the highest precedence. We detail each of these two steps subsequently.

Let $separation\_constraint\_violation\_validate(R0, R1)$ be the function of the SC violation checker, which is used to mitigate potential conflicts by checking two rules $R0$ and $R1$ against a SC. All attribute values in $R0$ are checked against their respective ones in $R1$, in pairs, to detect any specified SC that targets one of those pairs. Then, if a SC is satisfied, these two rules are marked as non-violating SCs or never conflicting rules (returning $false$), otherwise standing for a potential conflict ($true$). For example, $R0$ and $R1$ are never conflicting rules if $R0$ targets a $manager$ and $R1$ targets an $employee$, while they cannot be the same person.

For the remaining rules, the core of the conflicting rules checker is based on Definition III.3, which is implemented in the $detect$ function. According to this definition, $detect(R0, R1)$ states that $R0$ and $R1$ are conflicting rules (returning $true$) if their actions are distinct while known values of shared attributes overlap, and otherwise ($false$). Note that, since our detection processes operate on the CFI, the endpoints' information (e.g., "10.0.0.5", "facebook") of extracted data (e.g., "employee", "sns-websites") is unknown. Consequently, our detection checker must only consider those attributes that do not need to be resolved (i.e., with already known values). Following the example of Listing 1, src and $dest$ are deemed value-convertible attributes or overlapping attribute expressions, while $event\text{-}time$ is inconvertible or non-overlapping.

Finally, the complete algorithm of the real-time conflict checker is shown in Algorithm 1, which gathers the $separation\_constraint\_violation\_validate$ and $detect$. Its complexity is $\mathcal{O}(M \cdot N)$, with N representing the quantity of rules and M the number of attributes, when a new rule is introduced and compared against a set of existing rules.

Algorithm 1: $checkConflict$

```
Data: new rule R
Result: a set of conflicting rules with R
set_conflicting_rules = {}
for R_i in existing_rules do
    if
     separation_constraint_violation_validate(R, R_i)
     then
        if detect(R, R_i) then
            set_conflicting_rules += {R, R_i}
        end
    end
end
return set_conflicting_rules
```

V. Validation

In this section, we discuss the evaluation results of our proposal. The context and the testbed are first introduced,

followed by a description of the metrics we employ. Finally, the evaluation's results are analyzed.

A. Testbed, Implementation and Scenario

In order to evaluate our approach in a realistic context, we have considered the I2NSF implementation presented at IETF Hackathon #113[1]. This code implements early versions of I2NSF components and demonstrates a scenario in the NFV infrastructure where a tenant company operates two NSFs (Firewall and Web filter) to filter employee traffic on SNS websites during working hours. The policy, shown in Listing 1 of Section II-B, is processed by I2NSF to initialize the Firewall and Web-filter NSFs. As such, this scenario provides a realistic foundation of attribute definition to identify potential conflicts. One can notice that particular situations such as stateful firewall rules are beyond the CFI scope and as such, we do not handle them subsequently. The experiments we present below were conducted on an Ubuntu 20.04.4 LTS, Intel i7-10750H 2.6GHz, 16 GB RAM machine and our implementation, written in Python, is about 200 lines long[2].

B. Evaluation Conditions and Performance Metrics

1) Rule set Generation: In order to challenge our conflict detection and mitigation solution, we consider the set of conflict types depicted in Table I.

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

TABLE I: Basic types of evaluated conflicts

Herein, rule $R1$ presents a concise version of the one provided in Listing 1, and it acts as a reference that will be conflicted by other rules. $R2$ is an explicit conflict situation in which, from 12:00 to 14:00, both $R1$ and $R2$ apply while providing divergent actions, namely $drop$ for the first and $pass$ for the second. Then, $R3$ to $R5$ explore implicit conflict situations in which some attributes are missing and may lead to conflicts. To evaluate the impact of different parameters on our detection engine, we consider the case where $event\text{-}time$, dst, and src are missing in $R3$, $R4$, and $R5$, respectively. This substrate of conflicting situations enables the creation of datasets where we randomly choose (1) the type of conflict, from $R2$ to $R5$, and (2) the value of explicit attributes. We especially consider a set of 300 src and 15,000 dst attribute values, each dst containing from 1 to 5 URLs. That way, we can scale the dataset up to 10,000 rules, which eventually stands for a realistic situation. Also, we can study how different parameters, such as the number of rules, deployed SCs, conflicts, and $pass$ and $drop$ actions, affect the performance of I2NSF with our two checkers.

[1]Available at: github.com/jaehoonpaul/i2nsf-framework

[2]Available at: gitlab.imt-atlantique.fr/d22nguye/i2nsf-conflict-detection

2) Performance Metrics: By design, our algorithm detects all conflicts correctly. Consequently, measuring the average number of conflicting rules and conflicted requests is useless. However, its complexity in time and space must be evaluated for real-world use. Therefore, following acknowledged metrics from the literature [3], we evaluate in what follows, under different conditions, the time required to check a full set of rules and the related memory usage. As an early test, we assess the repeatability of the global framework on a scenario containing 10,000 random rules with 10 repetitions. Over all the experiments, the performance exhibited very stable memory usage with a mean of 286.3 MB and a standard deviation of 0.854 MB, as well as the execution time with a mean of 160.79 seconds and a standard deviation of 7.393 seconds, thus demonstrating that throughout our evaluation, there is no need to repeat the scenarios due to the randomness of the rule set.

C. Results

Given the evaluation scenarios and performance metrics we select, the results we collect are represented in Figure 3, and we describe them subsequently.

1) Impact of the number of rules: Figure 3a depicts how the number of rules impacts the memory usage and execution time when we select randomly a scenario with up to 10,000 rules with no conflict, no SC, and no rule priorities. One can see that, due to the systematic and pair-wise evaluation of all rules, as one could expect, both the execution time and memory usage follow a polynomial complexity according to the number of rules. However, although this grow may be an issue for extremely large scale rule sets, up to 10,000 rules, the execution time does not exceed a few dozen of seconds while memory does not exceeds 300MB.

2) Impact of the number and type of conflicts: In this experiment, we generate multiple scenarios such that each of them contains 10,000 random rules, 50% of which are $pass$/$drop$ rules, and no SC. In the first case, represented in Figure 3b, we vary the number of conflicts and fix the ratio between explicit and implicit conflicts at 50%. In the second case, represented in Figure 3c, we fixed the number of conflicts to 10%, but this time, we vary the ratio between explicit and implicit ones. We clearly observe that the performance of our checkers, both in terms of execution time and memory usage, is independent of these two factors which assesses their proper operation whatever the nature of conflicts in the rule set.

3) Impact of the number of drop/pass ratio: Figure 3d shows the impact of the rule actions on our checkers' performance when considering different ratios of $pass$ and $drop$ rules in a rule set. It shows that the execution time is strictly affected by the number of $pass$ rules ($number\_pass\_rule$) and $drop$ rules ($number\_drop\_rule$). More specifically, the smaller the value of $|number\_drop\_rule - number\_pass\_rule|$, the longer the execution time. This is explained by the fact that for the two given rules, if they have the same action, the $detect$ function can stop its execution right at the action checking

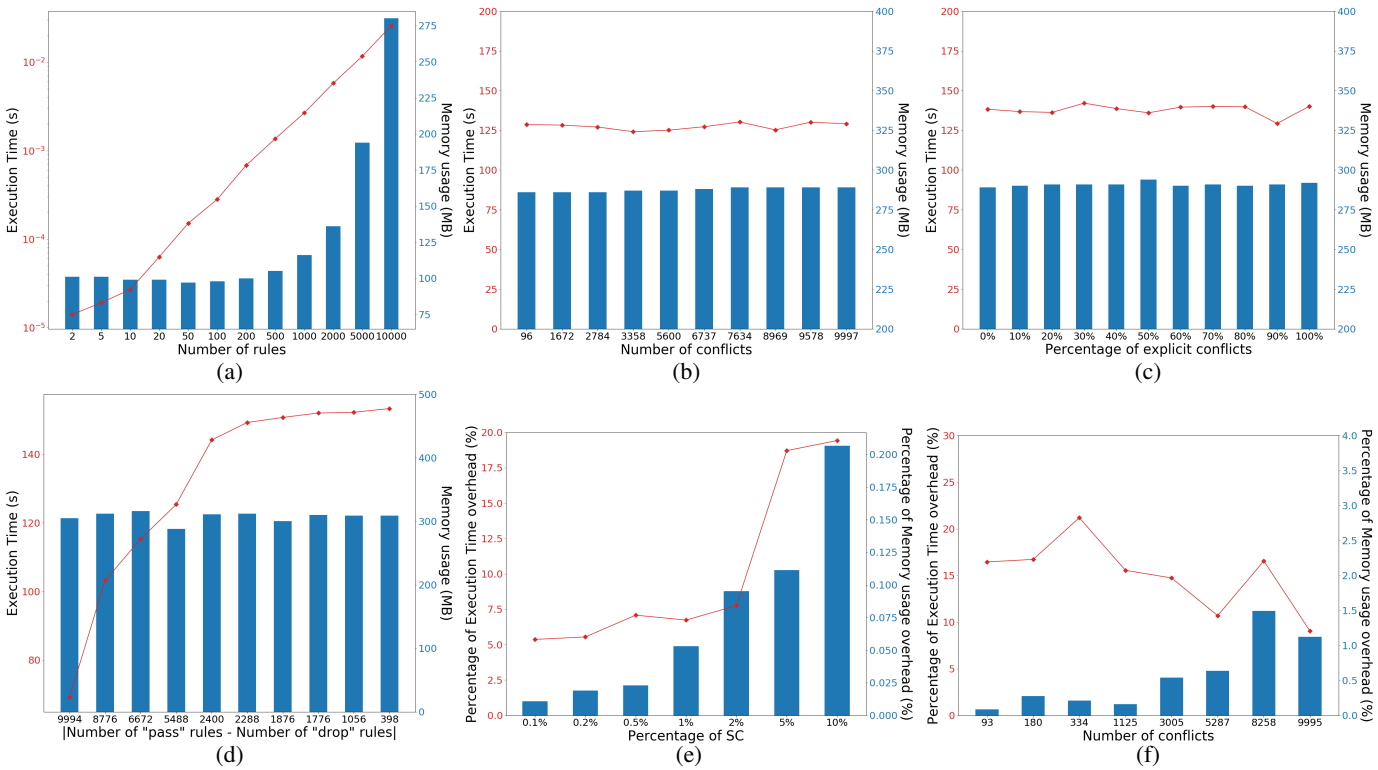

Fig. 3: Execution time of the checkers under different scenarios. (a) A scenario with 10,000 random rules; (b) Several scenarios with varying numbers of conflicts and no SC specification; (c) Several scenarios with varying percentages of explicit conflicts; (d) Several scenarios with 10,000 rules and varying numbers of *pass*/*drop* rules; (e) A scenario with varying numbers of specified SC; (f) Several scenarios with varying numbers of conflicts and SC specification.

step. Otherwise, it needs to process the attribute-overlapping check. In other words, scenarios that have a large number of rules with different actions will take significantly longer to be checked than those with similar actions.

4) Impact of separation constraints: Figure 3e depicts the overhead due to SCs over a basic rule set conflict detection. Here, we consider a rule set similar to the baseline scenario exposed in Figure 3a, without any conflict, and we evaluate to what extent adding SCs induces a performance overhead. It clearly appears that SCs do not impact the memory usage but rather the execution time, which, however, reasonably suffers from a 5% overhead in the case of 0.1% of SCs, up to 20% when 10% of rules have some SCs.

5) Impact of the number of conflicts with separation constraints: As a last evaluation benchmark, we measure to what extent the detection of conflicts accompanied with SCs induce an overhead. To that aim, we consider a scenario of 10,000 random rules with a ratio of 50% between explicit and implicit conflicts as well as for *pass*/*drop* rules. Figure 3f shows that despite the increase in the absolute number of conflicts, both execution time and memory usage get an acceptable and stable overhead with a maximum value of 15% and 1,5%, respectively, thus assessing the well-support of SCs of our solution in the presence of any amount of conflicts.

VI. Conclusion and Future Work

By providing a framework for translating high-level security requirements into low-level security function configurations, I2NSF represents a significant step towards the automation of network security management. In this context, our contribution aims at strengthening this process by integrating into the I2NSF operations mechanisms which identify the potential conflicting requirements, discard the explicit ones, and prevent the others. By freeing the user from all the issues concerning the soundness of the policy to be deployed, such an approach can help to pave the way for a widespread use of I2NSF in experimental investigations. Besides, similarly to [3], our approach can be adopted and applied in any frameworks using ECA or attribute-based rules.

As a short term perspective, we will consider the case conflict checking in dependant rules such as a stateful firewall. Our long term work will concern the use of AI to analyze and guide the resolution of conflicts through SCs and PORs to help users create safe and robust security requirements.

Acknowledgment

This work has been partially supported by the French National Research Agency under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

REFERENCES

[1] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-Based Networking - Concepts and Definitions," RFC 9315, Oct. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9315

[2] J. P. Jeong, P. Lingga, J. Yang, and J. Kim, "Guidelines for Security Policy Translation in Interface to Network Security Functions," Internet Engineering Task Force, Internet-Draft draft-yang-i2nsf-security-policy-translation-11, Apr. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-yang-i2nsf-security-policy-translation/11/

[3] G. Liu, W. Pei, Y. Tian, C. Liu, and S. Li, "A novel conflict detection method for ABAC security policies," *Journal of Industrial Information Integration*, vol. 22, p. 100200, 2021.

[4] G.-J. Ahn and R. 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.

[5] Y. Wang, Z. Tian, Y. Sun, X. Du, and N. Guizani, "LocJury: an IBN-based location privacy preserving scheme for IoCV," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5028–5037, 2020.

[6] B. E. Ujcich and W. H. Sanders, "Data Protection Intents for Software-Defined Networking," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 271–275.

[7] M. F. Hyder and T. Fatima, "Towards Crossfire Distributed Denial of Service Attack Protection Using Intent-Based Moving Target Defense Over Software-Defined Networking," *IEEE Access*, vol. 9, pp. 112 792–112 804, 2021.

[8] M. F. Hyder and M. A. Ismail, "INMTD: Intent-based moving target defense framework using software defined networks," *Engineering, Technology & Applied Science Research*, vol. 10, no. 1, pp. 5142–5147, 2020.

[9] T. Szyrkowiec, M. Santuari, M. Chamania, D. Siracusa, A. Autenrieth, V. Lopez, J. Cho, and W. Kellerer, "Automatic intent-based secure service creation through a multilayer sdn network orchestration," *Journal of Optical Communications and Networking*, vol. 10, no. 4, pp. 289–297, 2018.

[10] R. R. F. Lopes, C. Bildsten, K. Wrona, S. Huopio, D. Eidenskog, and O. L. Worthington, "Cyber security in virtualized communication networks: Open challenges for NATO," in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*. IEEE, 2021, pp. 1–8.

[11] J. P. Jeong, P. Lingga, P. Jung-Soo, D. Lopez, and S. Hares, "Security Management Automation of Cloud-Based Security Services in I2NSF Framework," Internet Engineering Task Force, Internet-Draft draft-jeong-i2nsf-security-management-automation-04, Jul. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-jeong-i2nsf-security-management-automation/04/

[12] J. Kim, E. Kim, J. Yang, J. Jeong, H. Kim, S. Hyun, H. Yang, J. Oh, Y. Kim, S. Hares, and L. Dunbar, "IBCS: Intent-Based Cloud Services for Security Applications," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 45–51, 2020.

[13] R. Marin-Lopez, G. Lopez-Millan, and F. Pereniguez-Garcia, "A YANG Data Model for IPsec Flow Protection Based on Software-Defined Networking (SDN)," RFC 9061, Jul. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9061

[14] D. Lopez, E. Lopez, L. Dunbar, J. Strassner, and R. Kumar, "Framework for Interface to Network Security Functions," RFC 8329, Feb. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8329

[15] J. Yang and J. P. Jeong, "An automata-based security policy translation for network security functions," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2018, pp. 268–272.

[16] S. Hares, J. P. Jeong, J. T. Kim, R. Moskowitz, and Q. Lin, "I2NSF Capability YANG Data Model," Internet Engineering Task Force, Internet-Draft draft-ietf-i2nsf-capability-data-model-32, May 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-i2nsf-capability-data-model/32/

[17] A. H. Anderson, "A Comparison of Two Privacy Policy Languages: EPAL and XACML," in *Proceedings of the 3rd ACM Workshop on Secure Web Services*, ser. SWS '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 53–60. [Online]. Available: https://doi.org/10.1145/1180367.1180378

[18] V. Capretta, B. Stepien, A. Felty, and S. Matwin, "Formal correctness of conflict detection for firewalls," in *Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering*, ser. FMSE '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 22–30. [Online]. Available: https://doi.org/10.1145/1314436.1314440

[19] G. Stone, B. Lundy, and G. Xie, "Network policy languages: a survey and a new approach," *IEEE Network*, vol. 15, no. 1, pp. 10–21, 2001.

[20] S. Benferhat and R. El Baida, "A prioritized-based approach to handling conflicts in access control," in *16th IEEE International Conference on Tools with Artificial Intelligence*, 2004, pp. 286–293.

[21] M. Koch, L. V. Mancini, and F. Parisi-Presicce, "Conflict detection and resolution in access control policy specifications," in *Foundations of Software Science and Computation Structures*, M. Nielsen and U. Engberg, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 223–238.

[22] S. Osborn, R. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 2, pp. 85–106, 2000.

[23] C.-c. Shu, E. Y. Yang, and A. E. Arenas, "Detecting conflicts in ABAC policies with rule-reduction and binary-search techniques," in *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE, 2009, pp. 182–185.

[24] J. P. Jeong, C. Chung, T.-J. Ahn, R. Kumar, and S. Hares, "I2NSF Consumer-Facing Interface YANG Data Model," Internet Engineering Task Force, Internet-Draft draft-ietf-i2nsf-consumer-facing-interface-dm-23, Aug. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-i2nsf-consumer-facing-interface-dm/23/