Security Policy Tool User Manual

InfoBeyond Technology LLC

August 15, 2017

A Comprehensive Implementation of NIST SP 800-192 and XACML 2.0/3.0
for Access Control Policy Composition, Analysis, Tests, Leak Inspection, and Verification

Dedicated to Policy Authors, A.K.A., Policy Composer, and Cybersecurity Specialist

Web: www.SecurityPolicyTool.com

*E-mail:*Info@InfoBeyondtech.com

Version 1.0.1

Contents

1 Preface

1.1 Access Control Flaws

Access control (AC) protects the secret financial, enterprise, organization, healthcare, defense, and various IT resources/services in an online system. In order to protect the classified resources, the security specialist needs to compose a set of AC policies (e.g., in XACML policies) to prevent unintended access. However, the current AC policies are composed and deployed into an AC system without comprehensive security tests and verifications. This results in many AC flaws (e.g., information or service leaks) in the systems and these AC flaws are normally hidden from us until observable damages (e.g., secret data leakage) are caused.

More specifically in a complex system, it is really a challenge to compose a multitude of policies with a number of rules to prevent AC flaws. These AC flaws [1] could be unintentionally opened to external cybersecurity attackers as well as the insiders. Particularly, misconfigured and fault policies, as well as error policy combining algorithms, could result in unexpected AC leaks that again cause serious economic and political consequence. In the last decade, we have been witnessing many cybersecurity incidents (e.g., large-scale data breaches, WikiLeaks), due to the misconfiguration of AC policies instead of the failure of cryptographic primitives or protocols.

1.2 NIST's Specification

NIST has released several specifications in order to help government and enterprises to enhance the nation's critical access control security. Some of these specifications are:

- NIST SP 800-192: Verification and Test Methods for Access Control Policies/Models [2],
- NIST IR 800-7874: Guidelines for Access Control System Evaluation Metrics [3], and
- NIST SP 800-162: Guide to Attribute Based Access Control (ABAC) Definition and Considerations [4].
- NIST IR 7316: Assessment of Access Control Systems [5].

As stated by NIST, many of the access control incidents (e.g., data breaches, insiders) are caused by misconfigured access control policies. These specifications describe the AC security requirements to avoid these incidents and recommend to thoroughly and automatically check the syntactic and semantic faults of AC policies before deploying them for operation [6].

1.3 About the Security Policy Tool

Since 2007, professors, scientists, and national security experts have been pursuing a method to detect the AC flaws from the AC policies. Security Policy Tool ($\mathcal{SPT}$) is such a tool to meet the need to compose, test, and validate the AC policies in an attempt to ensure there are no AC leaks when the policies are deployed in a system. By $\mathcal{SPT}$ tests, the AC policies can be effectively analyzed by a policy author to find unintended accessibility. With the identification of the fault and unintended policies, the policy author can fix the policies or rules to exclude the AC vulnerabilities. For such a purpose, $\mathcal{SPT}$ has many analyzing functions for the policy author to find the correlations among the rules and the AC accessibility. In addition, $\mathcal{SPT}$ offers the functions to conveniently compose AC models. It also contains a state-of-art XACML editor for graphical and text integrated policy editing.

$\mathcal{SPT}$ incorporates all the functions in the NIST's ACPT (Access Control Policy Tool) [7] with significant enhancements and advanced extension in terms of usability and functionalities. It is an comprehensive implementation of the NIST specifications [2-5], especially the NIST's SP 800-192 - Verification and Test Methods for Access Control Policies/Models. It also is compatible with XACML 3.0 policy models in a framework of PEP (Policy Enforcement Point), PDP (Policy Decision Point), PIP (Policy Information Point), and PAP (Policy Administration Point). Due to these security policy compliances, $\mathcal{SPT}$ satisfies the policy testing and analyzing requirements for the state-of-art AC systems as well as the legacy AC systems.

1.4 The Use of the Manual

$\mathcal{SPT}$ for AC policy test and analysis is targeted for the AC policy authors, as known as policy developer or policy composer, AC software developers, IT AC security managers, cybersecurity specialists, or other professionals in the performance of AC systems. It can be used for enhancing the AC security of (i) IT, IoT, Cloud, Telecom, Data Center, Telecom infrastructure, Device/Service Control Systems, (ii) Government, Defense, National Cybersecurity Systems, (iii) Transportation & Logistics Resource Access Systems, (iv) BFSI (Banking, Financial Services and Insurance) Systems, (v) Healthcare, Chemical/Pharma, Manufacturing and Utilities Access Control Systems, (vi) Organization, University, Education Control Systems, Retail, Oil, Gas & Energy Cybersecurity Control Systems, (vii) Hospitality & Residential Access Control Systems, (viii) Computer Networks and Remote Controls, and other Access Control Systems.

2 Summary

Security Policy Tool ($\mathcal{SPT}$) enables powerful AC testing and analyzing functions such that the policy authors can validate and fix the faulty, unintended, misconfigured policies. This ensures there are no security flaws (e.g., AC leaks) when the policies are deployed in a system. It has the following key functionalities:

- **Security Model and Policy Editing**: $\mathcal{SPT}$ has user-friendly GUI (Graphic User Interface) for you to conveniently compose/edit AC Attributes (Subject, Resource, Action, and Environment), Conditions, Rules, Policies, and algorithms. AC model templates, e.g., ABAC (Attributed-based Access Control), MLS (Multilevel Security), and Workflows, are provided for systematically policy editing, modification, and updating. Policy inheritance allows the policy authors to effectively compose and manage the policies for a large hierarchical organization.
- **Policy Testing**: $\mathcal{SPT}$ presents rich functions for comprehensive policy tests to verify the AC policies against your desirable security requirements. Giving the security requirements, one or a set of policies can be tested for policy leakage discovery:
  - Merged Policy Verification,
  - Combined Policy Verification,
  - Merged Policy Separation of Duty, and
  - Combined Policy Separation of Duty

  These tests support all rule combination algorithms such as First Applicable. Combinatorial tests allow you to automatically generate the test suite to achieve a testing coverage, corresponding to an approximate flaw detection percentage. User-friendly GUI presents the testing results by tables and identifies the internal correlations to reflect the testing results in connection with the rules or policies.
- **Policy Analyzing and Verification**: $\mathcal{SPT}$ enables a policy author to analyze the rules and policies of their AC authentication consequences in responses to the various AC requests. Potential security vulnerabilities in the policies could be detected in order to prevent AC flaws before these policies are deployed. AC Separation of Duty/Conflicts of Interest can be identified by analyzing the testing results. From the analysis, the policy author can fix the problematic policies with new tests and analysis till the intended AC security goal is achieved.
- **XACML (eXtensible Access Control Markup Language) Converter and Editor**: $\mathcal{SPT}$ can automatically input and convert XACML 2.0/3.0 documents into AC models in $\mathcal{SPT}$ data format. It further has an XACML 3.0 editor to reduce the mistakes caused by policy manually editing. It is able to automatically convert the $\mathcal{SPT}$ data into XACML 3.0 policy format and output it for portability.

In addition to the above functions, typical policy AC errors are described and showcase these errors from the policy tests and analysis. In addition to find the AC leaks, $\mathcal{SPT}$ reduces the policy deployment and maintenance cost. It is convenience to process a large number of rules (e.g., hundreds of rules or more) and policies.

3 ACPT vs. $\mathcal{SPT}$

Thanks NIST for the ACPT [7] research and development. Please visit NIST website for more information. ACPT is a prototype implementation. It has attracted a number of users and meanwhile received numerous feedbacks from these users. Upon the NIST's support, $\mathcal{SPT}$ is developed as a commercial tool that addresses the functional limitations in the ACPT. It meanwhile considers the users' feedbacks for new functions. $\mathcal{SPT}$ is highly superior to ACPT in the following aspects:

- **Efficiency and Capabilities**: $\mathcal{SPT}$ refines the AC models (e.g., ABAC) to offer AC policy verification efficiency and capabilities. For example, ACPT has a limited model checking engine and $\mathcal{SPT}$ enhances this engine for improving the model checking capability.
- **Security and Compatibility**: $\mathcal{SPT}$ offers comprehensive AC security verification features. It includes all XACML 3.0 policy/rule combining algorithms. $\mathcal{SPT}$ enables powerful policy analysis to detect the policy AC flaws as specified in the NIST SP 800 192. All these functions are limited in ACPT.
- **XACML, Flexibility, and Usability**: $\mathcal{SPT}$ should enable XACML import and export in support of mandatory and optimal XACML features. $\mathcal{SPT}$ is flexible and convenient for policy authors to perform the test and review the results flexibly. It has the functions for policy author to easily inspect the AC flaw and fix them.

$\mathcal{SPT}$ evolves as a powerful tool in order to signify the user's values as a consequence of policy verifications. At first, $\mathcal{SPT}$ makes it really different for the AC security with and without policy verification. Secondly, $\mathcal{SPT}$ significantly reduces the cost and time efforts for AC policy development, deployment, managements, and maintenance. Furthermore, it is robust and powerful for complicate AC systems with a number of AC rules and policies. Due to rich and powerful functions, $\mathcal{SPT}$ is a dispensable tool for policy authors to develop, test, and verify AC policies for worry-free AC flaws.

4 Acknowledgement

Thanks are expressed to the NIST (National Institute of Standards and Technology), Department of Commerce, and the National Small Business Administration. The R&D of the Security Policy Tool is sponsored by NIST through a SBIR (Small Business Innovation Research) program (2015-2018). From the SBIR Phase I and II program, Security Policy Tool has been successfully developed as a commercial version of the NIST's ACPT (Access Control Policy Tool). More specifically, ACPT is a prototype of Proof of Concept for rich policy testing functions. With our efforts, Security Policy Tool substantially enhances and expands the NISTs ACPT design with many advanced and user-friendly features for achieving high-security confidence AC levels. For example, Security Policy Tool provides many policy analyzing functions that allow a policy author to inspect the potential security leaks over a large number of policies and rules. Furthermore, Security Policy Tool is a user-friendly, robust, efficient, professional, and functionally powerful AC security policy tool.

We would like to thank Dr.Vincent Hu who gave us tremendous technical consultant for the Security Policy Tool technology. Appreciation should be also presented to many NIST's Computer Security Resource Center experts of this matter. We would like to extend our sincere gratitude to many other SBIR staffs for their efforts to ensure the success of the tool.

5 Concept

$\mathcal{SPT}$ follows the general AC concept defined in NIST 7316 Specification [1] and XACML 3.0 [2]. According to NIST 7316, Access Control Policy (i.e., *Policy* for simplicity) carries the AC requirements that specify how the access is managed and who may access information under what circumstances. It may pertain to resource and service protection within or across organizational units or may be based on need-to-know, competence, authority, obligation, or conflict-of-interest factors. More specifically, XACML 3.0 defines a policy as a set of rules with rule-combining algorithms to describe the AC requirements. Terms used in $\mathcal{SPT}$ can be categorized into:

- **XACML**: $\mathcal{SPT}$ has the same terms defined in XACML 3.0. These terms are Access Control, Action, Attribute, Condition, Decision, Environment, Policy, Policy Combining Algorithm, Predicate, Resource, Rule, Rule-combining Algorithm, Subject, etc. Please refer XACML 3.0 for these terms if they are not covered in this manual.
- **Inheritance**: $\mathcal{SPT}$ defines several inheritance terms to model the AC relationships such as originator, beneficiary, subject inheritance, and object inheritance.
- **AC Model**: $\mathcal{SPT}$ considers three types of AC models that are ABAC (Attribute-based Access Control), Multilevel Security Model, and Workflow.
- **Policy Test**: For policy testing and analysis, $\mathcal{SPT}$ defines Security Requirement, Separation of Duty, Conflict of Interest, Merged Policy Verification, and Combined Policy Verification.

This section describes the fundamental concept and terms related to the AC policy test and analysis in $\mathcal{SPT}$ software tool. Figure 1 shows an example of a partial Emergency Hospital Organization Chart and we will use it for AC policy demonstration.

5.1 Policy and Attribute

A policy is a statement to guide the decision of an access request in attempt to achieve a rational decision, i.e., *Permit* and *Deny*. Decisions determined from policies could be intended (i.e., expected) or unintended (i.e., AC flaw) in terms of AC security requirements. If an intended *Deny* is granted with *Permit* unintentionally, an AC leak of the corresponding resource occurs. A policy consists of rule(s) described by a collection of attributes in a logical way. As defined in XACML 3.0, an attribute is the characteristics of a Subject, Resource, Action, or Environment:

- **Subject Attributes**: Subject attributes describe the actor (e.g., a user or a software agent) in an attempt to access, characterized by age, clearance, department, role, job title, etc.
- **Resource Attributes**: Resource attributes describe the object being accessed e.g., data and resources (e.g., medical record, and bank account), services provided in the AC system, and other system components, e.g., a device.
- **Action Attributes**: Action attributes describe the action being attempted, e.g., read, delete, view, approve, etc.

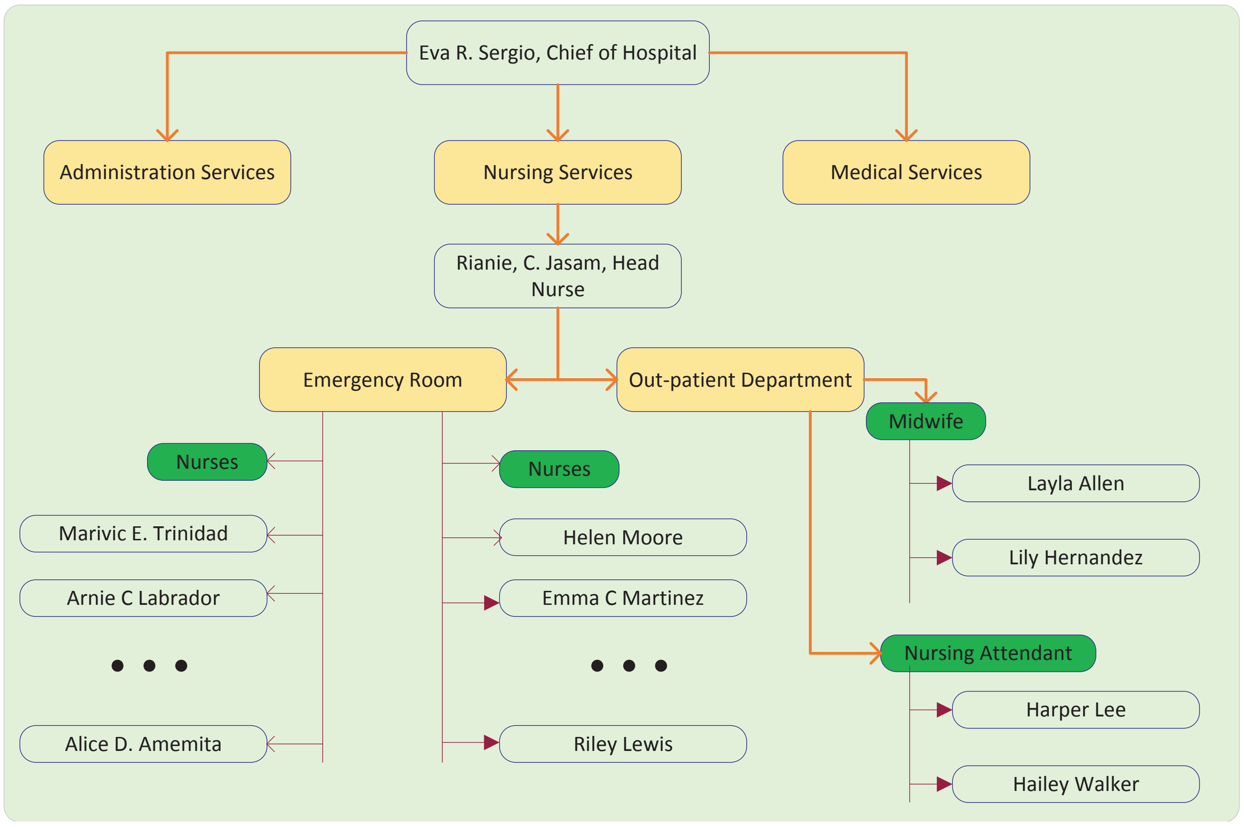


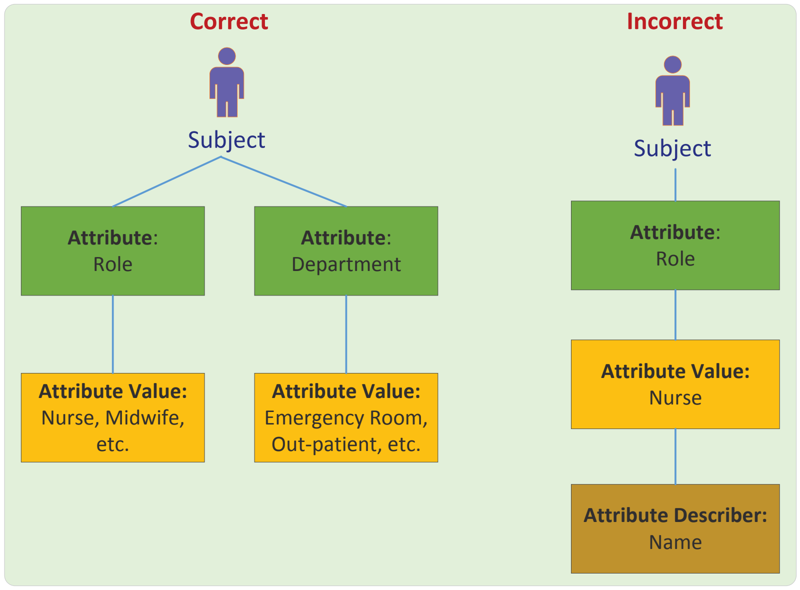
Figure 1: An Example of Organization Chart - Nursing Service

- **Environment Attributes**: Environment attributes describes the environment relevant to an authorization decision and they are independent of a particular subject, resource, or action. The environment attributes are generally used to specify the time, location, system status, or other dynamic aspects of the AC scenarios.

All attributes have a Name, Data type, and one or more attribute Values. For the example in Figure 1, *Role* can be defined as a subject attribute name with a *string* data type that has the attribute values: *Chief of Hospital*, *Nurse*, *Midwife*, and *Nursing Attendant*. Further, Resource, such as *Patient Record* (Data type - *string*, Values - *Prescription*, *Medical Record*, *Personal Information*) can be defined for the hospital resource attributes. Similarly, Action, such as CRUD (e.g., *Create*, *Read*, *Update*, *Delete*), can be defined as an Action attribute for the hospital example in Figure 1. One can define as many attributes as desired for a real AC system. However, the definition only supports up to one level of classification for the attributes. An attribute is the root and it can only have one level of children for its attribute values as shown in Figure 2. Figure 2 (a) show the correct attribute definition while Figure 2 (b) is incorrect. Figure 2 (b) shows two-level of attribute values, which is not allowable in $\mathcal{SPT}$ and XACMAL standard.

5.2 Condition

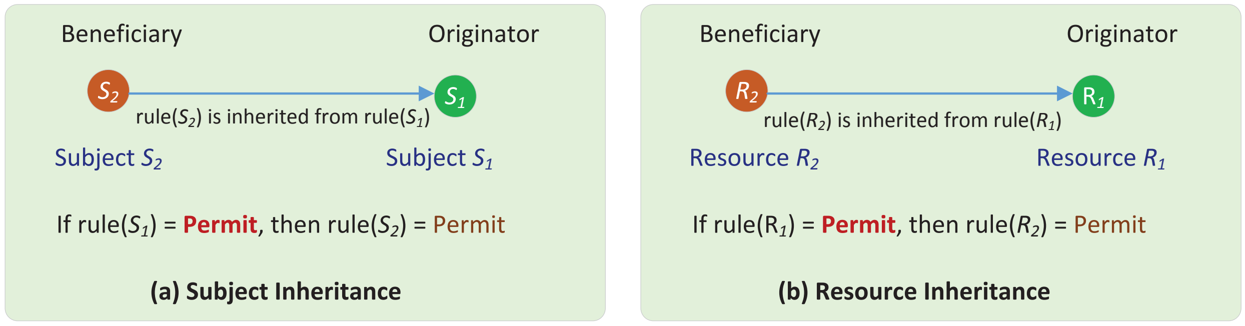
XACML 3.0 defines the condition as a function (i.e., an expression of a predicate) that evaluates to *True* or *False* or *Indeterminate*. Conditions only exist in rules. Conditions are essen-



Correct
Subject
Attribute: Role
Attribute Value: Nurse, Midwife, etc.
Attribute: Department
Attribute Value: Emergency Room, Out-patient, etc.
Incorrect
Subject
Attribute: Role
Attribute Value: Nurse
Attribute Describer: Name

Figure 2: Correct Way to Use Attribute

tially an advanced form of a target which can use a broader range of functions and more importantly can be used to compare two or more attributes together. For example, we can define $Is\_login\ permit\ from\ 09:00:00\ am\ to\ 05:00:00\ pm$ as a condition with the possible evaluation of *True* or *False* or *Indeterminate*. *Indeterminate* means the condition is unable to be evaluated, e.g., an error occurred or some required values are missing. With the condition, it is possible to implement segregation of duty checks, e.g, the time period, or Relationship-based AC (ReBAC). Conditions are usually used to check if a certain requirement on the subject's attribute is met or not. For example, an employee can access a classified document only if the access request is from the company facilities (e.g., a company laptop). In $\mathcal{SPT}$, *Indeterminate* is not considered in the policy test. The reason is that $\mathcal{SPT}$ always believe the system can evaluate the condition variable whichever the system is. It is not meaningful to test a policy with an *Indeterminate* condition.



Beneficiary
Originator
S_2
S_1
rule(S_2) is inherited from rule(S_1)
Subject S_2
Subject S_1
If rule(S_1) = Permit, then rule(S_2) = Permit
(a) Subject Inheritance
Beneficiary
Originator
R_2
R_1
rule(R_2) is inherited from rule(R_1)
Resource R_2
Resource R_1
If rule(R_1) = Permit, then rule(R_2) = Permit
(b) Resource Inheritance

Figure 3: Subject and Object Inheritance

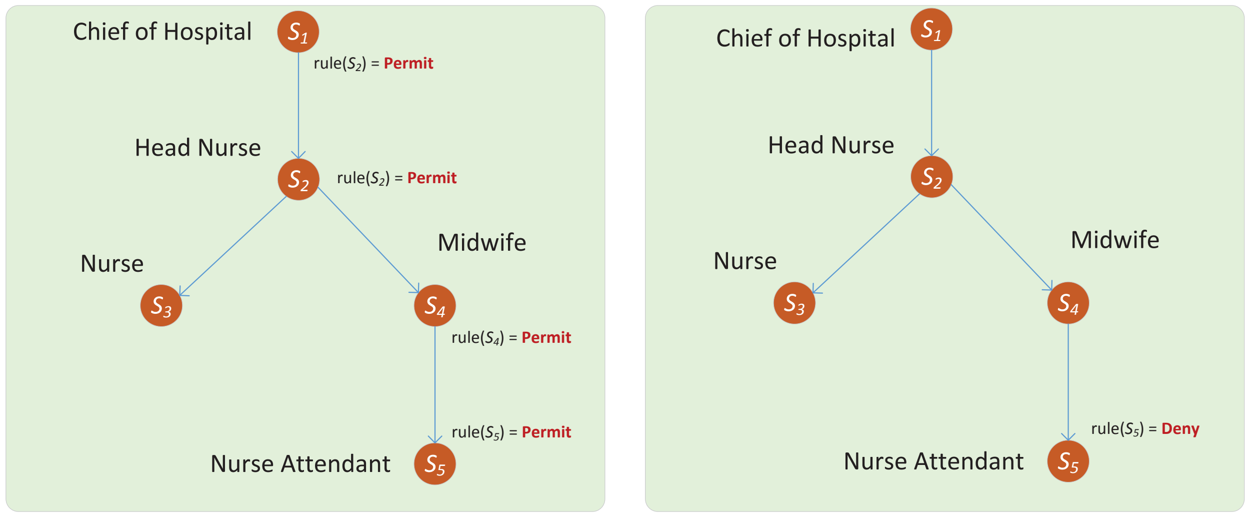
5.3 Inheritance

AC inheritance helps a policy an author to better compose polices, especially for a large organization that has many subjects or resources. Meanwhile, it helps for policy tests and analysis. Inheritance specifically defines a set of hierarchical attribute relations. Figure 3 shows an example of the subject inheritance. Figure 3 (a) shows that Subject S_2 is the beneficiary of Subject S_1 which could be a rule originator. By following the inheritance, the beneficiary, e.g., S_2, could enjoy the accessing right of the originator S_1. The inheritance logic can be intuitively stated by:

$$If\ Rule(S_1) = Permit,\ then\ Rule(S_2) = Permit \tag{1}$$

Figure 3 (b) shows the resource inheritance where Resource R_2 is the beneficiary of Resource R_1. Similarly, R_2 could enjoy the accessing privilege of the originator R_1, stated by:

$$If\ Rule(R_1) = Permit,\ then\ Rule(R_2) = Permit \tag{2}$$



(a): If rule(S_5) = **Permit**, then rule(S_4), rule(S_2), rule(S_1) = Permit

(b): If rule(S_5) = **Deny**, the rule is **NOT** propagated over the inheritance

Figure 4: Subject Inheritance Architecture

$\mathcal{SPT}$ implements the inheritance in a hierarchical structure. This enales a policy author to specify a set of policy rules without duplicating the composition. Meanwhile, this keeps the rule consistence to avoid rule errors while editing multiple rules. Figure 4 shows the hierarchical inheritance that can be inferred from the nursing service in Figure 1, which simplifies the rule definition as the beneficiary subject can inherit the rule from the originator. As shown in Figure 4 (a), there is an inheritance relations as the following inheriting chain:

$$Chief\ of\ Hospital \rightarrow Head\ Nurse \rightarrow Midwife \rightarrow Nurse\ Attendant \tag{3}$$

According to the inheriting chain, a rule (e.g., $rule(s_5)$) at the Nurse Attendant will be correspondingly prorogated to Midwife (e.g., $rule(s_4)$), Head Nurse (e.g., $rule(s_2)$), and Chief of Hospital (e.g., $rule(s_1)$).

However, the rule prorogation over the inheritance relations is **only** effective for the rule that has a decision of **Permit**, which represents the accessing permit to a specific resource. If a rule has a decision of Deny, the rule will not be prorogated as the negative accessing right would not be inherited. This is because a subject or resource at a higher hierarchical level would not inherit the negative accessing privilege of a subject or subject in a lower level. Consider a *Folder* and its *Subfolder* as two inherited resource. The Permit of accessing the *Subfolder* indicates the Permit of the accessing the *Folder*. Otherwise, the *Subfolder* isn't accessible. On the other hand, the Deny of the access of the *Subfolder* doesn't mean the Deny of the *Folder*. The decision of the *Folder* will be not affected by the Deny of the access of the *Subfolder*. Figure 1 (b) shows the example that $rule(s_5) = Deny$ at $NurseAttendant$ will not be inherited to $Midwife\ S_4$ as the $Midwife$ may have higher authority on the resource specified in $rule(s_5)$. Therefore, the following relation is incorrect:

$$If\ rule(s_5) = Deny,\ then\ rule(s_4) = Deny \tag{4}$$

As indicated Figure 1 (a), there can be multiple levels of inheritance but there cannot be a loop of inheritance. The reason is that loop don't permitted in the hierarchical structure. $\mathcal{SPT}$ has the function of automatical loop detection to avoid you to define any loop of inheritance.

Further, a subject/resource attribute can inherit multiple subject/subject attributes. Figure 1 shows that *Head Nurse* is a beneficiary of all subject attributes of the subtree rooted at *Head Nurse*, i.e., a beneficiary of *Nurse*, *Midwife*, and *Nurse Attendant*.

5.4 AC Algorithm

XACML defines policy and rule algorithms to handle special cases such as the policy having conflicted or insufficient rules for decision making. There are two types of AC algorithms: rule/policy combining algorithm and policy enforcement algorithm.

5.4.1 Combining Algorithm

A policyset could have one or more policies and meanwhile, a policy could consist of a number of individual rules. For multiple policies and rules, a decision in response to a security requirement is the result of the consideration of all the policies and rules via different combining algorithms. However, these policies and rules could have the overlapped scope such that conflict of decisions could emerge when multiple policies or rules are applied. Combining algorithms is to specify the way to handle duplicate policies or rules that integrate the different decisions. Decisions could be different while applying different rule or policy combining algorithms. Policy test is to verify if the decision of applying these combining algorithms is the intended one or not. Meanwhile, the sequence of the rules during the combining could affect the decision and it needs to verify if the policies and rules are ordered properly to achieve our intended decision.

Let's image an example that the first rule says that managers can view the documents in a system while the second rule regulates that none can work before $9:00:00$ am. What if the request is about the manager Alice in an attempt to view a document at $8:00:00$ am? Which rule wins? This is what the rule combination algorithms tell us. They help resolve conflicts of the decision.

$\mathcal{SPT}$ currently supports XACML policy and rule combination algorithms as the following:

- **First Applicable**: First applicable takes the permission decision of the first occurrence of a particular variable if the same variable was in the policy twice with a different Permission. For example if two policies were merged, it would take the permission decision of the first one.
- **Deny Override**: Deny Override scans through the entire model to find all the duplicate variables. Once a deny permission of duplicate variable is found it will deny the access to that particular rule in the model.
- **Permit Override**: Permit Override scans through the entire model to find all the duplicate variables. Once a grant permission of duplicate variable is found it will permit the access to that particular rule in the model.
- **Only One Applicable**: Only One Applicable is only applied for policy sets to combine policy sets and policies. It cannot be used to combine rules. This algorithm is that only one of the policy produces a valid decision whether Deny or Permit.
- **Deny Unless Permit**: This algorithm only allows two decisions: Permit or Deny. It is intended for those cases where a permit decision should have priority over a deny decision, and an Indeterminate or NotApplicable must never be the result.
- **Permit Unless Deny**: This algorithm only allows two decisions: Permit or Deny. It is intended for those cases where a deny decision should have priority over a permit decision, and an Indeterminate or NotApplicable must never be the result.

$\mathcal{SPT}$ additionally supports optional combining algorithms:

- **Weak-Consensus**: Weak-consensus requires that policies should not conflict with each other. It denies an access request if some policies deny the request, and no policy permits it. It permits an access request if some policies permit the request, and no policy denies it. It outputs conflict if some policies permit and some deny.
- **Strong-Consensus**: This algorithm requires that all policies must agree on a decision. It denies an access request if all policies deny the request. It permits an access request if all policies permit the request. Conflict is output otherwise. Note that this algorithm is different from weak-consensus since a policy may neither permit nor deny a request (e.g., it might not be applicable to the request). When some policies deny a request and others are not applicable to it, weak-consensus denies the request but strong consensus outputs conflict.
- **Weak-Majority**: When different policies make conflicting decisions (permit and deny) about a request, the request is permitted (denied, resp.) if the number of policies permitting (denying, resp.) it is greater than the number of policies denying (permitting, resp.) it.
- **Strong-Majority**: Strong-majority permits (denies, resp.) a request if more than half of all policies, i.e., $1/2$, permit (deny, resp.) it.
- **Super-Majority-Permit**: Super-majority-permit permits an access request if more than $2/3$ of all policies permit it, and denies it otherwise.

5.4.2 Policy Enforcement Algorithm

Policy enforcement algorithm is used to make decision on Not Applicable requests. When the incoming request does not match any rule in the AC policy, we say the request is Not Applicable. For a Not Applicable request, the rules in the AC policy are not sufficient to make the decision. Policy enforcement algorithm is added to each policy to resolve this problem. There are two kinds of policy enforcement algorithms: Deny Based and Permit Based.

- **Deny Based**: When the incoming request does not match any rule in the AC policy, it will be denied.
- **Permit Based**: When the incoming request does not match any rule in the AC policy, it will be permitted.

5.5 AC Models

An AC model defines the relationships among Subjects, Resources, Actions, Environments, Conditions, and their AC effectiveness of Decision. It is a framework that dictates how subjects access objects. It uses AC technologies and security mechanisms to enforce the rules and objectives of the model. $\mathcal{SPT}$ supports three AC models namely, ABAC, MultiLevel, and Workflow. In $\mathcal{SPT}$, the legacy AC models, including Discretionary Access Control (DAC), Identity-Based Access Control (IBAC), Mandatory Access Control (MAC), Rule-Based Access Control (RAC), Role-Based Access Control (RBAC), Organization-Based Access control (OrBAC), History-Based Access Control (HBAC), can be generally evolved into the ABAC model. Therefore, $\mathcal{SPT}$ ignores these legacy AC models.

5.5.1 ABAC

ABAC is also named as Policy-Based Access Control (PBAC) as the access permissions are granted to a request through a set of policies. $\mathcal{SPT}$ has the ABAC model template in favor of a policy author to continently define many AC rules based on their environment variables (Subject, Resource, and Action). Unlike MLS and Workflow, ABAC template consists rules without any embedded AC model. Giving the example in Figure 1, the following attributes can be defined:

1. **Subject - Role**: Midwife, Nursing Attendant, Patient
2. **Resource - Patient Info.**: Prescription, Patient Personal Information
3. **Action**: Read, Read and add note, and write
4. **Environment**: Any Environment

In the above definitions, Role is a Subject attribute of three attribute values, named as Midwife, Nursing Attendant, and Patient. Similarly, Patient Info. is a Resource attribute of two values that are Prescription, Patient Personal Information. Meanwhile, three attribute values are defined as shown in the Action definition. Using these attributes, the following ABAC policies with rules can be specified as examples:

Midwife's Policy with the following AC Rules:

1. Subject: Midwife; Resource: Prescription; Action: Read; Permit
2. Subject: Midwife; Resource: Prescription; Action: Read and add note; Permit
3. Subject: Midwife; Resource: Prescription; Action: Write; Permit
4. Subject: Midwife; Resource: Patient Personal Information; Action: Read; Deny
5. Subject: Midwife; Resource: Patient information; Action: Read and add note; Permit
6. Subject: Midwife; Resource: Patient Personal Information; Action: write; Permit

Nursing Attendant's Policy with the following AC Rules:

1. Subject: Nursing Attendant; Resource: Prescription; Action: Read; Permit
2. Subject: Nursing Attendant; Resource: Prescription; Action: Read and add note; Permit
3. Subject: Nursing Attendant; Resource: Prescription; Action: Write; Deny
4. Subject: Nursing Attendant; Resource: Patient Personal Information; Action: Read; Permit
5. Subject: Nursing Attendant; Resource: Patient Personal Information; Action: Read and add note; Deny
6. Subject: Nursing Attendant; Resource: Patient Personal Information; Action: Write; Deny

Patient's Policy with the following AC Rules:

1. Subject: Patient; Resource: Prescription; Action: Read; Permit
2. Subject: Patient; Resource: Prescription; Action: Read and add note; Deny
3. Subject: Patient; Resource: Prescription; Action: Write; Deny
4. Subject: Patient; Resource: Patient Personal Information; Action: Read; Permit
5. Subject: Patient; Resource: Patient Personal Information; Action: Read and add note; Permit
6. Subject: Patient; Resource: Patient Personal Information; Action: Write; Permit

ABAC GUI-based template in $\mathcal{SPT}$ allows policy authors to user-friendly define these polices and rules. XACML 3.0 presents a recommended ABAC architecture. Giving an AC Request, PEP inspects the request and generates an authorization request to the PDP. PDP evaluates the policies and returns a Permit/Deny decision to the PEP.

5.5.2 MultiLevel Security Model

The MultiLevel Security Model (MLS) enforces Bell-LaPadula and Biba models, which protects the resources from being accessed from unauthorized ranking members. This particular model uses two rule properties: No read up, No write down. These properties ensure no Subject cannot read a Resource above their access level or write to a Resource lower than their current rank.

Rank is a numerical value attribute that can only be mapped to any Subject or Resource attributes for the MultiLevel Model. The higher the value of the integer placed on the rank attribute is more classified (in terms of Multilevel Security Level model). When creating a Rank Attribute, the Attribute name, for example, could be any text, for example "`Rank_1`" and its attribute type is an

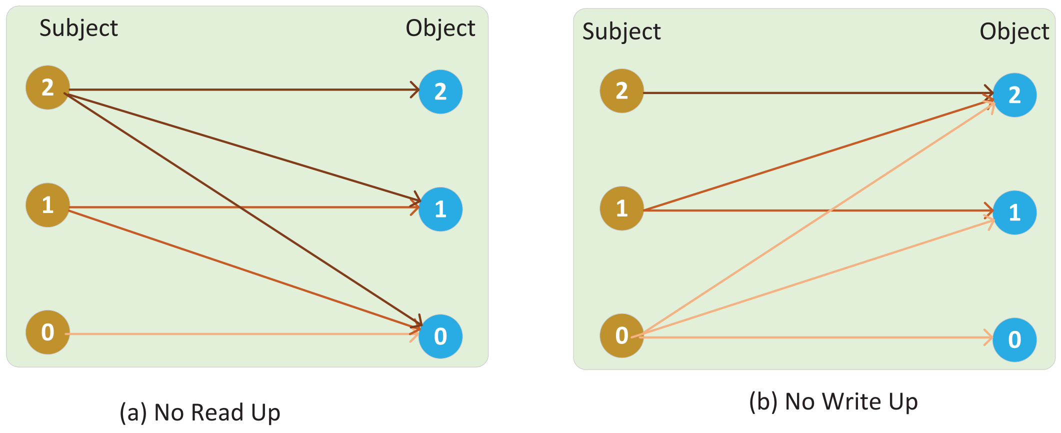


Figure 5: Multilevel Security Models

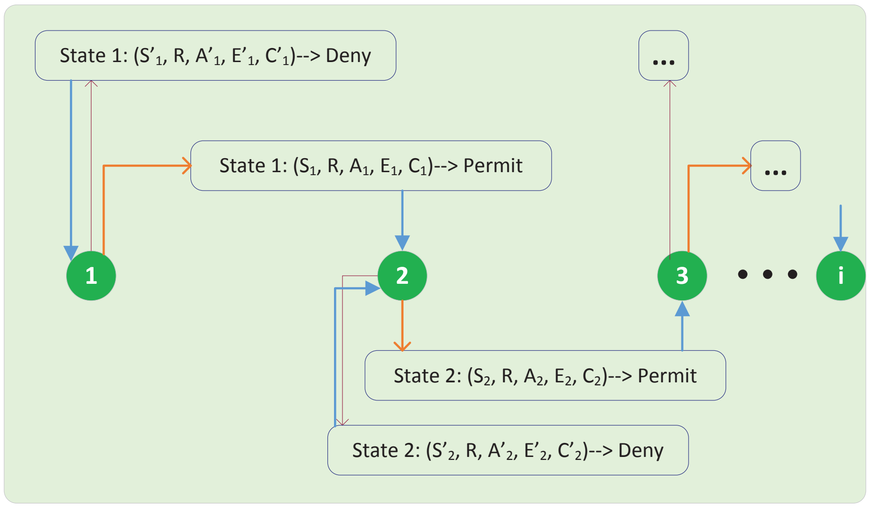
Integer with the attribute value "2" for example. Note that when applying rank attributes it can only enter 1 value per rank attribute.

No Read Up (Bell-LaPadula Model): This property handles which subjects can have access to read the resource. A subject can read its current ranked resource and any resource below its current rank. It is forbidden to read any resource above its rank. Figure 5 (a) is an example of the No Read Up model where three users ranked $0, 1, 2$ respectively and there resources ranked $0, 1, 2$ respectively. Higher the rank of the Subject or Resource higher the security level it has. Figure 5 (a) shows that the user in Rank 2 can read the Resource $0, 1, 2$, the user in Rank 1 can read the Resource $0, 1$, and the user in Rank 0 can only read the Resource 0.

No Write Down (Biba Model): This property protects the information/resource from being changed from unauthorized subjects. A subject can write to a resource as long as they are the same rank level or above their current rank. However, a higher ranked subject cannot write to a lower ranked resource. Higher ranked subjects have access to more classified information so if they are not allowed to write down this will help prevent information leaking to unauthorized subjects. Figure 5 (b) is an example of No Write Down model where three users ranked $0, 1, 2$ respectively and there resources ranked $0, 1, 2$ respectively. Higher the rank of the Subject or Resource higher the security level it has. Figure 5 (a) shows that the user in Rank 2 can only write the Resource 2, the user in Rank 1 can write the Resource $1, 2$, and the user in Rank 0 can write the Resource $0, 1, 2$.

5.5.3 Workflow Model

Workflow model enforces the sequential access privileges State by State in a given sequel which represents the processing flow. Figure 5 shows the Workflow model where a State is associated with a rule that have to be accomplished in this state such that it can be transitioned to the next State. In other words, each State has to process the rule before moving to the next processing state. On the other hands, all other requests other than the specific rule have to be denied. For example, an Invoice can only be paid (e.g., Action in State 2) only after the Department Manager has proved it (e.g., Action in State 1). In this AC example, the request of any Action in State 2 will be always Denied if the Action in State 1 is not accomplished. Let 's consider the following workflow that defines:



State 1: (S'1, R, A'1, E'1, C'1)--> Deny
...
State 1: (S1, R, A1, E1, C1)--> Permit
...
1
2
3
i
State 2: (S2, R, A2, E2, C2)--> Permit
State 2: (S'2, R, A'2, E'2, C'2)--> Deny

Figure 6: Workflow AC Model

1. **Subject - Role**: Client, Engineer, Builder
2. **Resource - Design**: Blueprint
3. **Action - Workflow Actions**: Review, Review and Add Note, and Write

This example is considered as a workflow where the *Blueprint* design has to be review (e.g., Action: Read and Note) by *Client*, the Engineer can then revise (e.g., Action: Review) the Blueprint design based on the Client's note. After the revision the design (e.g., Action Write) by the Engineer, the Builder can then review the Blueprint design for further works. For such an example, the following states can be defined:

State 1: Rule 1 (Client, Blueprint, Review and Add Note) → Permit
State 2: Rule 2 (Engineer, Blueprint, Review) → Permit
State 3: Rule 3 (Engineer, Blueprint, Write) → Permit
State 4: Rule 4 (Builder, Blueprint, Review) → Permit

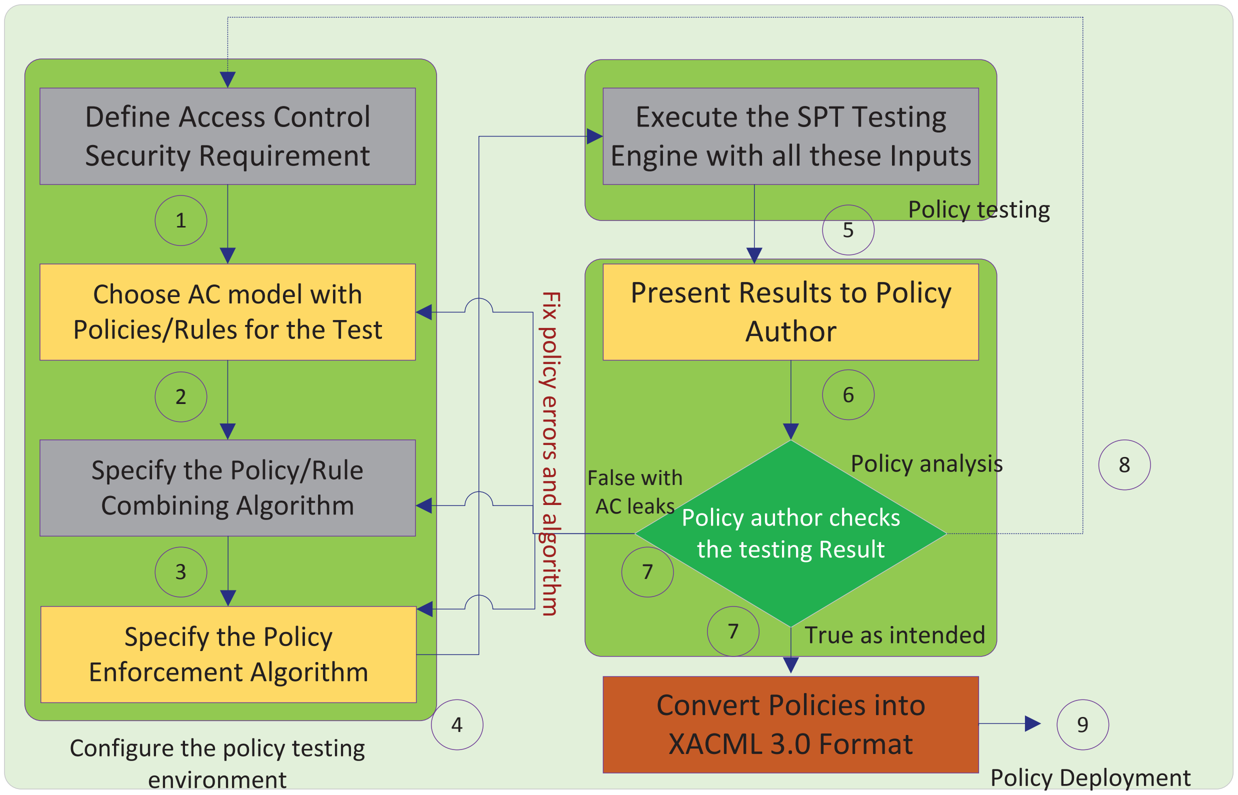
Giving any state in the example, all access requests are denied for the Subject Blueprint unless specified in the above defined states.

5.6 XACML

XACML defines a structure of Extensible Markup Language (XML). It defines an architecture and a processing model describing how to evaluate access requests according to the rules defined in policies. XACML 3.0 is a new version which has new features. For example, XACML 3.0 defines new attribute functions and datatypes. It also enhances XACML 2.0's existing combination algorithms. $\mathcal{SPT}$ is compatible to XACML 3.0. It provides XACML functions, such as importing XACML into a $\mathcal{SPT}$ project, converting XACML 2.0 format to XACML 3.0 format, converting AC model policy into XACML 3.0 format and exporting them, and XACML 3.0 editor.

6 Policy Tests and Analysis

$\mathcal{SPT}$ policy test and analysis run the AC models in a testing engine and a policy author can then analyze the results to verify if the AC decisions from the tests are the intended ones or not. If all the results are matched with the intended results, the policies including the rules and the algorithms in the AC models are correctly composed and they can be deployed into the actual AC systems. Otherwise, there are AC flaws such as AC leaks, unexpectedly deny an access request, etc., in the policies. In this case, the policy author needs to revise the policies or the algorithms and runs for new results until the results are all the expectation. This section illustrates the concept in regard to the policy test and analysis.



Define Access Control Security Requirement
1
Choose AC model with Policies/Rules for the Test
2
Specify the Policy/Rule Combining Algorithm
3
Specify the Policy Enforcement Algorithm
4
Configure the policy testing environment
Fix policy errors and algorithm
Execute the SPT Testing Engine with all these Inputs
Policy testing
5
Present Results to Policy Author
6
Policy analysis
False with AC leaks
Policy author checks the testing Result
7
7
True as intended
8
Convert Policies into XACML 3.0 Format
9
Policy Deployment

Figure 7: Policy Tests and Analysis

6.1 General Testing Procedure

Figure 7 shows $\mathcal{SPT}$ general steps that illustrate the basic policy test and analysis process. At first, the policy author configures the policy testing environment. It includes the specification of one or more AC Security Requirements that act as access requests, e.g., Step 1 in Figure 7. The environment should be further chosen an AC model with the policies that the AC Security Requirement will be applied to, e.g., Step 2 in Figure 7. Meanwhile, the combining and policy enforcement algorithms are specified for the testing environment, e.g., Step 3 in Figure 7.

After that, the $\mathcal{SPT}$ testing engine is executed with the inputting testing environmental settings, e.g., Step 4 in Figure 7. Then, the verification results will be presented to the policy author via GUI, Step 5 in Figure 7. With these results, the policy author can check if the testing results are expected or not, e.g., Step 6 in Figure 7.

Suppose the *TRUE* indicates the intention of the result (i.e., Permit or Deny) which says the AC Security Requirement is matched with the expected AC effectiveness. Otherwise, *FALSE* indicates AC security issues (e.g., AC leaks). In this case, the policy author needs to revise the policy rules or the rule combining and policy enforcement algorithms, e.g., fix the policy errors or algorithms in Step 7 in Figure 7. This manual classifies the general AC flaws into eight types of AC errors.

The above process should be repeated with different AC Security Requirements to achieve comprehensive tests. In the end, the policies after the tests can be converted into XACML 3.0 format and deployed into a real AC system. $\mathcal{SPT}$ provides various verification methods to verify the policies in different ways. These methods will be further discussed in the following subsections.

6.2 AC Security Requirement

Before describing policy tests, we clarify the term of AC Security Requirement. An AC Security Requirement (i.e., Security Requirement for a short term) is a statement of an AC requirement with a Decision in a form of:

$$\{Subject(s),\ Action(s),\ Resource(s),\ Environment(s),\ Condition(s)\} \rightarrow Decision : Permit/Deny \qquad (5)$$

which states an AC request with an AC Decision of $Permit$ or $Deny$.

Consider the nursing service in Figure 1. A Security Requirement example could be:

$$\{Nursing\ Attendant,\ Delete,\ Presription,\ any\ Environment,\ any\ Condition\} \rightarrow Deny \qquad (6)$$

to state that a Nursing Attendant is not permitted to delete the patient Prescription in any Environment and any Condition.

Indicated in Express 5, a Security Requirement has the same format as a rule. The difference is that the Security Requirement is utilized for the purpose of testing if the intended AC security is met or not. Differently, an AC rule is a component of a policy that controls the access of the Resource. If a Security Requirement is proven to be *TRUE*, the AC effectiveness expressed by the Security Requirement is achieved, such as:

$$\{Nursing\ Attendant,\ Delete,\ Presription,\ any\ Environment,\ any\ Condition\} \rightarrow Deny\ \ TRUE \qquad (7)$$

Moreover, if the Security Requirement is matched with the intended AC effectiveness, the AC security is achieved. Otherwise, there are AC flaw(s) caused by the polices or the associated algorithms. On the contrary, if a Security Requirement is proven to be *FALSE*, the AC effectiveness expressed by the Security Requirement is not achieved, such as:

$$\{Nursing\ Attendant,\ Delete,\ Presription,\ any\ Environment,\ any\ Condition\} \rightarrow Deny\ \ FALSE \qquad (8)$$

In this case, if the Security Requirement is matched with the intended AC effectiveness (i.e., Deny), the $FALSE$ results indicates AC flaw(s) that are caused by the polices or the associated algorithms. Otherwise, the verification result of *FALSE* indicates the satisfaction of AC effectiveness (e.g., Permit). $\mathcal{SPT}$ allows three types of Security Requirements that are defined for different AC testing scenarios:

1. **Individual Security Requirement**: This is to test each Security Requirement to verify its AC effectiveness. Multiple individual security requirements can be defined and tested together in the $\mathcal{SPT}$ and however they are tested individually and the results are analyzed separately.
2. **Separation of Duty Security Requirement**: This is to define at least two or more Security Requirements in order to verify the correlated AC effectiveness to verify if there is any Conflict of Interest among these Security Requirements. Conflict of interest is common in the government, legal, financial, market, healthcare, and many real commercial business AC systems. Dual role relationship could cause Conflicts of Interest, such as Mutually-Exclusive Roles. It needs to prevent a person who has a role of authority that conflicts with other role of different access permission of the resource. Consider the roles in a bank. Conflict of interest arises that a loan manager to change his/her client grade (e.g., from silver to premium) in order to lower his/her mortgage account interest rate. In some AC systems, a person can either access Resource R_1 or Resource R_2, but not both. Similarly, Separation of Duty may allow a person to take Action A_1, or Action A_2, but not both. Separation of Duty Security Requirement is for the purpose of detecting the Conflict of Interest.
3. **Combinatorial Security Requirement**: $\mathcal{SPT}$ can automatically generate a number of Individual Security Requirements by pairwise or all-pairs attribute combinatorial algorithms. The collection of these requirements is called as a testing suite. This facilitates the test of the Individual Security Requirements as it avoids a policy author to manually compose a large number of Individual Security Requirements, which is tedious and time-consuming. Similar to software testing, testing suite allows a policy author to achieve a certain testing coverage. Specifically, $\mathcal{SPT}$ can automatically generate $t-$ (i.e., $t = 2, 3, 4, 5, 6$) way combinatorial testing suites. $6-$ way combinatorial testing suite achieves 100% testing coverage of all possible combinations, e.g., 6 variables of Subject, Resource, Action, Condition, Environment, and Permission. However, the generation and testing of $6-$ way combinatorial testing suite have the highest testing and analyzing complexity, as the number of possible combination could be big. Pairwise testing (e.g., $2-$ way combinatorial testing) is commonly suggested as its testing coverage can find $50\% - 90\%$ AC flaws. $4-$ way combinatorial testing can mostly discover most complex AC flaws and its testing coverage can discover closely 100% AC flaws.

Upon the composition of Security Requirements, policy tests and analysis are conducted to verify the AC effectiveness of the security policies.

6.3 Policy Tests

$\mathcal{SPT}$ allows a policy author to design a variety of access cases to verify its AC results (i.e., Permit or Deny) against the AC policies and rules composed through ABAC, MLS, and Workflow models. For such a purpose, a policy author first generates Security Requirements in any of three types,

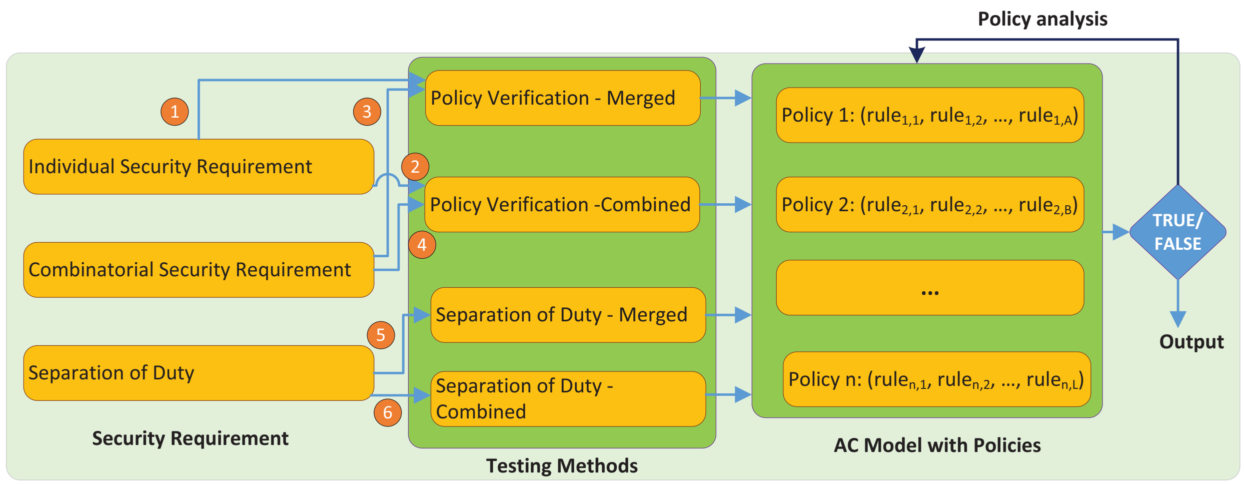

Policy analysis
1
3
2
4
5
6
Individual Security Requirement
Combinatorial Security Requirement
Separation of Duty
Security Requirement
Policy Verification - Merged
Policy Verification -Combined
Separation of Duty - Merged
Separation of Duty - Combined
Testing Methods
Policy 1: (rule1,1, rule1,2, ..., rule1,A)
Policy 2: (rule2,1, rule2,2, ..., rule2,B)
...
Policy n: (rulen,1, rulen,2, ..., rulen,L)
AC Model with Policies
TRUE/ FALSE
Output

Figure 8: Multiple Policy Testing Methods

i.e., Individual Security Requirement, Separation of Duty Security Requirement, or Combinatorial Security Requirement. The policy author then can test these Security Requirements against the composed AC policies to see if the testing results are matched to the Decision of Security Requirements. Particularly, the policy analysis admits the following principle for AC flow detection:

Theory (AC Flaw Detection (ACFD)): The ACFD theory is that the policy testing result (i.e., Permit or Deny) is compared with the Decision of the Security Requirement to yield a TRUE or FALSE:

- If the Decision of the Security Requirement is the intended result, TRUE represents the AC satisfaction and otherwise FALSE indicates AC flaw (s);
- If the Decision of the Security Requirement is the unintended result, TRUE indicates AC flaws and FALSE represents the AC satisfaction.

where the intended result is judged by the policy author according to the access control security requirements.

Applying the AC flaw detection theory with various Security Requirements, $\mathcal{SPT}$ allows the policy author to comprehensively conduct the policy verification, analyze the testing results to detect the potential AC flaws, revise the AC model including the rules, policies, and algorithms, and furthermore retest it till all the results are satisfied.

6.3.1 Single vs. Multiple Policy Verification

Policy test can be generally categorized into Single Policy Verification and Multiple Policy Verification. Single Policy Verification evaluates the Security Requirements against each role in one policy. A rule combination algorithm to resolve the conflicting results among rules. Further, a policy enforcement algorithm is used to achieve a Decision when all rule testing results are Not Applicable. The testing result is applied to ACFD theory for flaw detection. Regardless multiple policies, merged or combined testing method has to chosen to specify how these policies can be aggregated in the test. It is worth mentioning that merged and combined testing methods are equal to Single Policy Verification in the case of one policy. Figure 8 shows different methods of multiple policy tests to verify the Security Requirements: (i) Merged Policy Verification, (ii) Combined Policy Verification, (iii) Merged Policy Separation of Duty, and (iv) Combined Policy Separation of Duty. As shown in Figure 8, Individual Security Requirement and Combinatorial Security Requirement can perform the merged or combined policy verification. Similarly, Separation of Duty Security Requirement can perform the merged or combined policy Separation of Duty tests. Merged and combined tests represent two different PDP processes how the rule and policy are evaluated to reach a decision. The following subsections give the detailed description of the these testing methods.

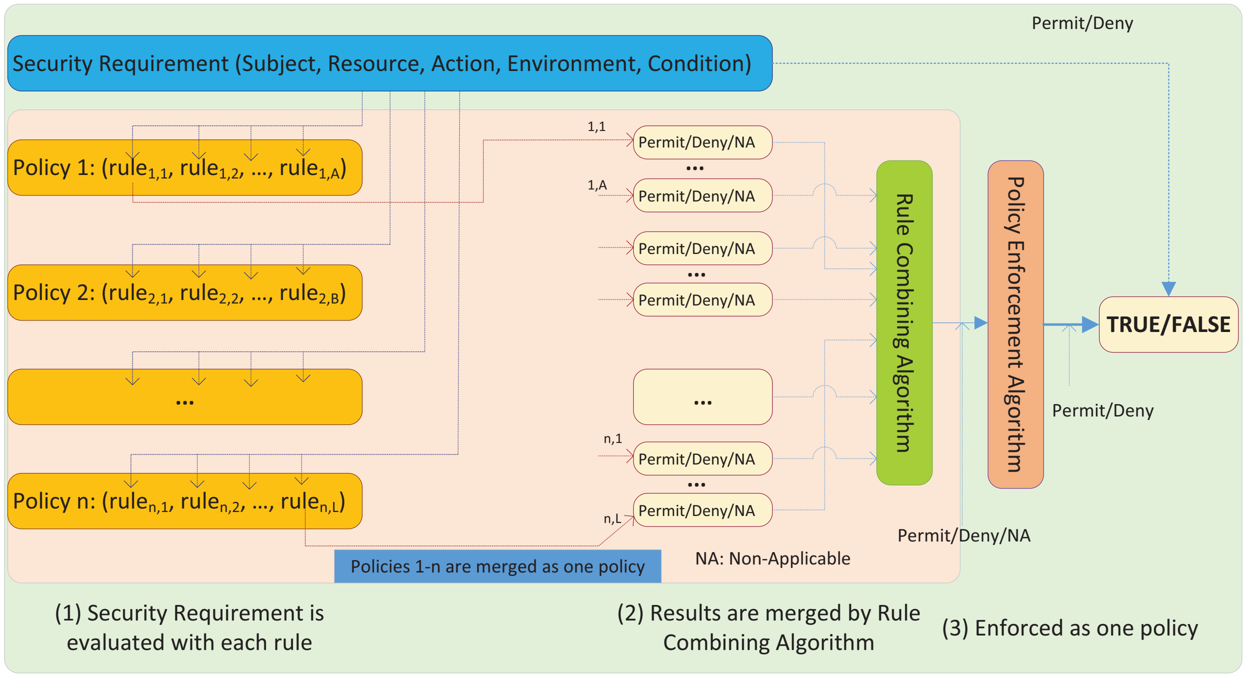

Permit/Deny
Security Requirement (Subject, Resource, Action, Environment, Condition)
Policy 1: (rule$_{1,1}$, rule$_{1,2}$, …, rule$_{1,A}$)
Policy 2: (rule$_{2,1}$, rule$_{2,2}$, …, rule$_{2,B}$)
…
Policy n: (rule$_{n,1}$, rule$_{n,2}$, …, rule$_{n,L}$)
1,1
Permit/Deny/NA
1,A
n,1
n,L
Rule Combining Algorithm
Policy Enforcement Algorithm
TRUE/FALSE
Permit/Deny
Permit/Deny/NA
Policies 1-n are merged as one policy
NA: Non-Applicable
(1) Security Requirement is evaluated with each rule
(2) Results are merged by Rule Combining Algorithm
(3) Enforced as one policy

Figure 9: Merged Policy Verification

6.3.2 Merged Policy Verification

Merged Policy Verifications tests all rules of policies as a merged policy.

Figure 9 shows the principle of the merged policy verification. In this test, the Security Requirement is evaluated with all the rules of the chosen policies. Suppose there are n policies. Policy 1 has A rules, policy 2 has B rules, and so on. Merged Policy Verification is that the Security Requirement is individually tested against each rule in each policy, i.e., $(1,1), \cdots, (1,A), \cdots, (n,1), \cdots, (n,L))$. For the rule tests, it then totally has $A + B+, \cdots, +L$ testing results, i.e., Permit/Deny/NA (Non-Applicable) as shown in Figure 9. All these results are applied to the rule combining algorithm (e.g., first applicable) together. Hereafter, a policy enforcement algorithm is considered to the merged policy to resolve the duplicated/conflictive rules. In the end, the combining result (i.e., Permit or Deny) is compared with the result of the Security Requirement to yield a TRUE or FALSE, which is applied to ACFD theory for flaw detection. As we can see from the above described process, multiple policies (e.g., Policy $1-n$) are tested as if they are merged as one policy where the rules in each policy are tested individually against the Security Requirement subsequently.

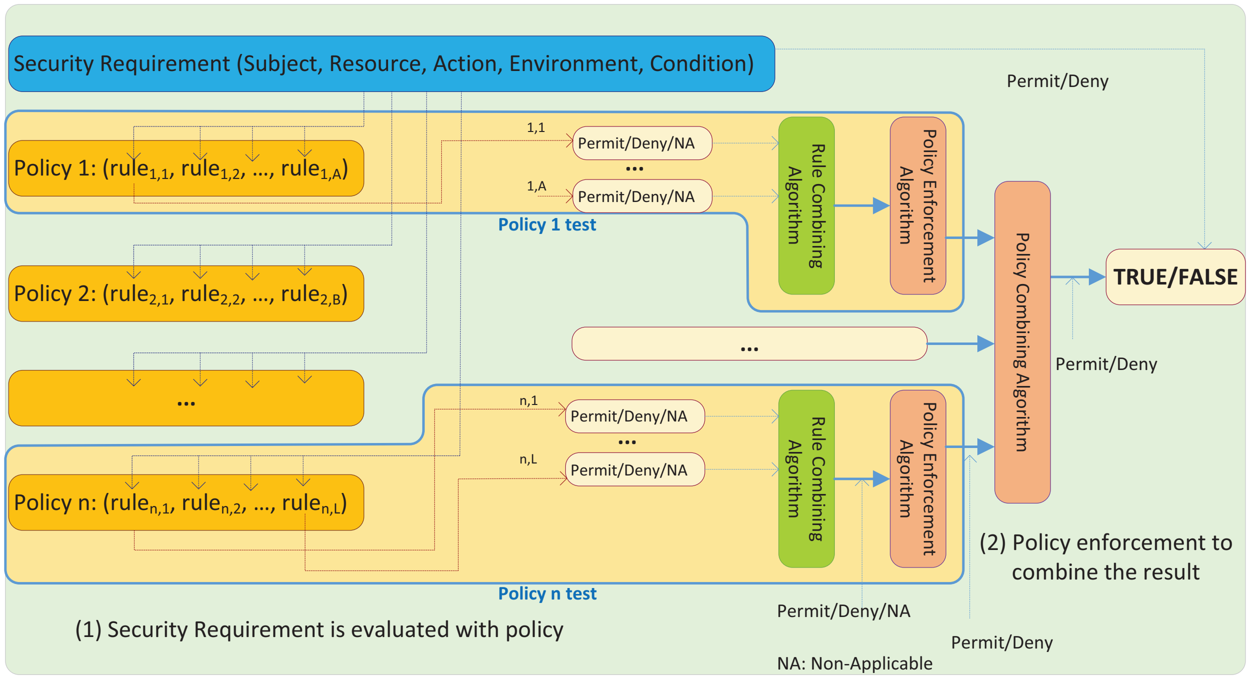

Security Requirement (Subject, Resource, Action, Environment, Condition)
Permit/Deny
Policy 1: (rule1,1, rule1,2, ..., rule1,A)
1,1
Permit/Deny/NA
1,A
Permit/Deny/NA
Policy 1 test
Rule Combining Algorithm
Policy Enforcement Algorithm
Policy Combining Algorithm
TRUE/FALSE
Policy 2: (rule2,1, rule2,2, ..., rule2,B)
...
Permit/Deny
n,1
Permit/Deny/NA
n,L
Permit/Deny/NA
Policy n: (rulen,1, rulen,2, ..., rulen,L)
Policy n test
(2) Policy enforcement to combine the result
Permit/Deny/NA
(1) Security Requirement is evaluated with policy
Permit/Deny
NA: Non-Applicable

Figure 10: Combined Policy Verification

6.3.3 Combined Policy Verification

Combined Policy Verification subsequently tests all policies and then combine the policy testing results into a Decision.

Figure 10 shows the principle of combined policy verification where combines the Decisions of all policies via a policy combining algorithm. In this test, each policy (e.g., Policy $1-n$) is first evaluated independently with Security Requirement. The decision of each policy is made by a rule combining algorithm and results could be Permit, Deny, or NA (Non-Applicable). After the policy enforcement algorithm, the decision of each policy becomes only Permit or Deny. It is noted that the rule combining algorithms and policy enforcements for different policies could be differently specified. As shown in Figure 10, Policy 1 is tested with A times of rule evaluation and A results are first combined via a rule combining algorithm of the policy. In a similar way, other policies (e.g., Policy 2 to Policy n) are tested with their rule combining algorithms as well as their policy enforcement algorithms. The evaluation of the Security Requirement against n policies will result in n Decision either Permit or Deny. These n Decisions are combined with a policy combining algorithm and the result (Permit or Deny) is compared with the expectation of the Security Requirement, which yields TRUE or FALSE. Finally, the TRUE/FALSE result is applied to ACFD theory for flaw detection. $\mathcal{SPT}$ associates the Security Requirement with each policy and gives an insight of the rule evaluation. This allows the policy author to find the problematic policy as well as the rule, showing contradict Decision.

As we can see from the above-described process, multiple policies (e.g., Policy $1-n$) are first tested separately against the Security Requirement. All the policy testing results are combined via a policy combining algorithm and finally compared the Decision in the Security Requirement. The difference between the Merged Policy Verification and the Combined Policy Verification can be understood as:

- **Rule Level Combination**: Merged Policy Verification represents a Rule Level Combination testing approach that combines all the rules of all policies together as a merged policy.
- **Policy Level Combination**: Combined Policy Verification represents a Policy Level Combination testing approach that integrates the decisions at the policy level.

Both the approaches can be implemented in a PDP in an AC system.

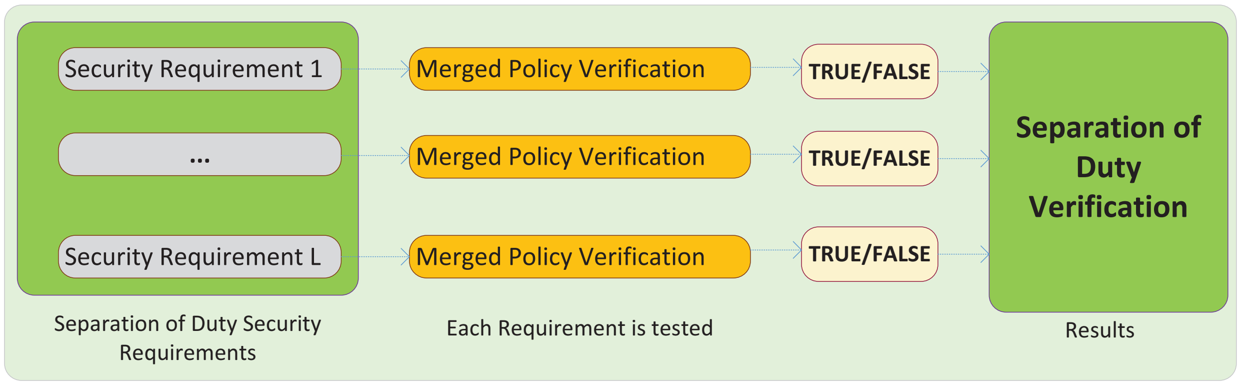

Security Requirement 1
...
Security Requirement L
Separation of Duty Security Requirements
Merged Policy Verification
Merged Policy Verification
Merged Policy Verification
Each Requirement is tested
TRUE/FALSE
TRUE/FALSE
TRUE/FALSE
Separation of Duty Verification
Results

Figure 11: Merged Policy Separation of Duty

6.3.4 Merged Policy Separation of Duty

Merged Policy Separation of Duty enables Merged Policy Verification for multiple Security Requirements to detect Conflict of Interest among Subjects, Resources, or Actions.

Security leaks caused by Separation of Duty is also referred as multiple-duty-related Security Leakage. Consider a number of policy rules for a business transaction. The rule for the duty to initiate a payment may be conflicted with the authorization of this payment by two individuals who have a Conflict of Interests. A simple example is that a single individual may be considered as a Conflict of Interest to execute both transactions of payment initiation and authorization. Figure 11 shows the principle of Merged Policy Separation of Duty verification. It shows that the Merged Policy Separation of Duty involves with two or more Security Requirements and each Security Requirement are tested with Merged Policy Verification. Separation of Duty Verification is to check the results of Merged Policy Verification of each Security Requirement and from the results to check if there are leaks (i.e., conflict) caused by Separation of Duty. Suppose a rule (e.g., $rule(p_1)$) to initiate a payment and another rule (e.g., $rule(p_2)$) to authorize the payment. It is unable to find any AC security flaw if two rules are checked separately. However, when Merged Policy Separation of Duty test is conducted with these rules, the testing results indicate Separation of Duty in the case that a person is granted with permissions to take these two transactions, while these two transactions cause a Conflict of Interest.

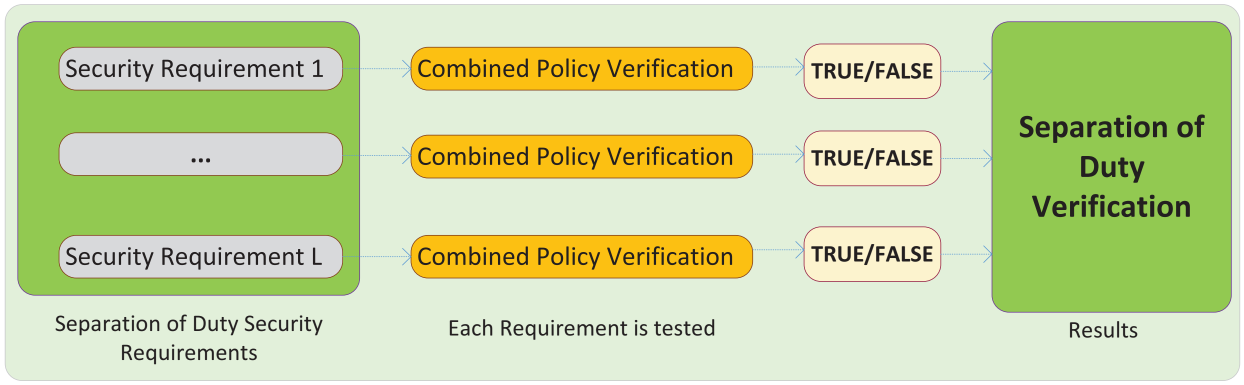

Security Requirement 1
...
Security Requirement L
Combined Policy Verification
Combined Policy Verification
Combined Policy Verification
TRUE/FALSE
TRUE/FALSE
TRUE/FALSE
Separation of Duty Verification
Separation of Duty Security Requirements
Each Requirement is tested
Results

Figure 12: Combined Policy Separation of Duty

The detection of the Separation of Duty error is not straightforward and the policy author needs to first identify the potential Subject (e.g., Role), Resource (e.g., Payment), and Actions (e.g., Create, Approve) and then design appropriate Security Requirements to verify the hyperpiesis. The composition of Separation of Duty Security Requirements should be consistent with the business scenario of Conflicts of Interest.

6.3.5 Combined Policy Separation of Duty

Combined Policy Separation of Duty enables Combined Policy Verification for multiple Security Requirements to detect Conflict of Interest among Subjects, Resources, or Actions. It is a similar function as the Merged Policy Separation of Duty. The difference is that each Security Requirement in the Combined Policy Separation of Duty is tested with Combined Policy Verification as illustrated in Figure 12. This indicates that the Conflict of Interest is considered in the level of the multiple policies. Differently, Merged Policy Separation of Duty considers the Conflict of Interest in the level of rule and the test is based on one merged policy. Consider a policy (e.g., $policy(r_1)$) with a rule to initiate a payment and another policy (e.g., $policy(r_2)$) with a rule to authorize the payment. It may have no AC security flaw if two policies are checked separately. When Combined Policy Separation of Duty test is conducted, a testing result of two $TRUE$s indicates Separation of Duty as a person is granted the permission to take both transactions according to these two policies, while the permission of these transactions is a Conflict of Interest for these two roles in one person.

6.3.6 Access Privilege Preview

In addition to the above illustrated policy tests, $\mathcal{SPT}$ provides Access Privilege Preview functions for policy authors to review:

1. *Giving certain Subject attribute(s), what Resources can be accessed (i.e., Permit) or not accessed (i.e., Deny), e.g., a Nurse can access what information in a system?*

2. *Giving certain Resource attribute(s), who (i.e., Subject) can access them or not access them, e.g., the Prescription can be accessed by who?*

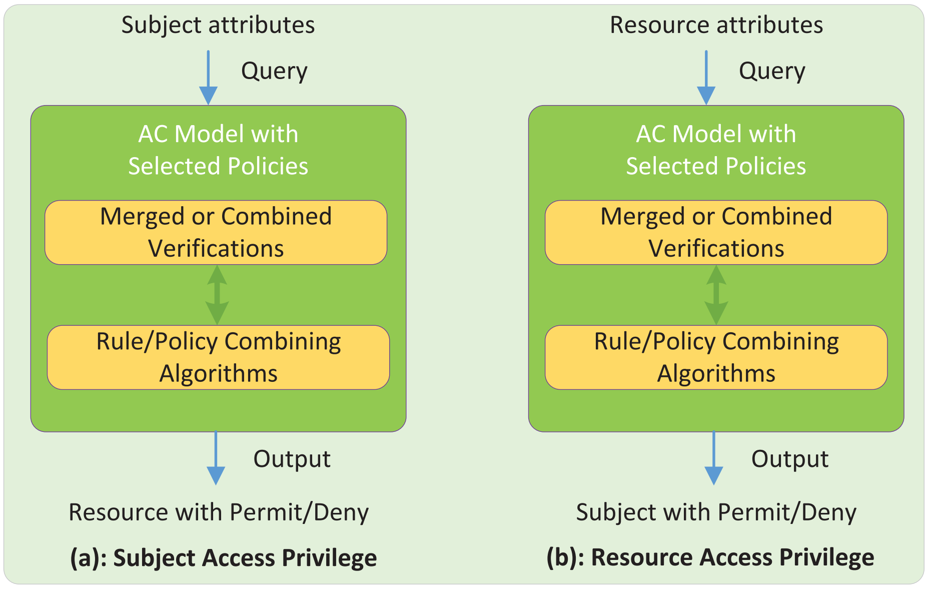

Subject attributes
Query
AC Model with Selected Policies
Merged or Combined Verifications
Rule/Policy Combining Algorithms
Output
Resource with Permit/Deny
(a): Subject Access Privilege
Resource attributes
Query
AC Model with Selected Policies
Merged or Combined Verifications
Rule/Policy Combining Algorithms
Output
Subject with Permit/Deny
(b): Resource Access Privilege

Figure 13: Access Privilege Preview

These access privilege preview functions allow a policy author to query and check if the Resources are properly protected with intended AC permission. Figure 13 shows the principle of the access privilege preview. It first generates a query which is then applied to the selected AC model, merged or combined verification methods, and the combining algorithms, and the testing results show the access privilege of the query. More specifically, these access privilege preview functions can be categorized into:

1. **Subject Privilege Preview**: In this access privilege preview, the policy author specifies a query with the subject attribute of his/her concerns. Consider the example in Figure 1. The policy may want to know the Midwife's access privilege in the nursing service system. He/she can choose the *Role* attribute with a value of *Midwife* and apply a query into the composed AC model. $\mathcal{SPT}$ will perform the tests with the specific algorithms. Upon the tests, $\mathcal{SPT}$ will output all the resources that *Midwife* can access to or not access to. From the testing output, the policy author can then review and analyze the *Midwife*'s access privilege of all resources with permission or decline. If there is any Resource that is permitted/declined unintentionally, an AC flaw is indicated. $\mathcal{SPT}$ gives the details of the resource authorization in connection with each rule.

2. **Resource Privilege Preview**: The resource access privilege preview is similar to subject privilege preview. The difference is that it previews the privileges of all subjects for a giving resource. The policy author specifies a query with the resource attribute of his/her concerns. Similarly, consider the example in Figure 1. The policy may want to know who can access (e.g., update) the *Prescription* in the nursing service system. He/she can choose the *Patient Record* attribute with a value of *Prescription* and apply the query into the composed AC model. $\mathcal{SPT}$ will perform the tests with the specific algorithms and will output all the Subjects that can access to the *Prescription*. From the output, the policy author can then review and analyze the subject's access privilege of the *Prescription* resource. If there is any

Subject that is not intended to access the *Prescription*, an AC flaw is indicated. $\mathcal{SPT}$ gives the details of rule decisions of all subjects which allows the policy author to correct the AC flaw.

Access Privilege Preview presents the summary of Subject or Resource's privilege and it could be very useful to detect the AC flaws. The policy author can review the access privilege and compare it with the intended results in mind. Any contradictions indicate AC flaws and the policy author should modify the policy or algorithms till the desirable results are presented from the test.

6.4 Policy Analysis and Typical AC Flaw

$\mathcal{SPT}$ facilitates policy development and AC flaw detection by providing rich policy relevant analyzing functions. It enables a policy author to easily edit, manage, and review a number of rules and policies. Giving an attribute, $\mathcal{SPT}$ can easily find all the rules engaged with the attribute and its inheritance relations with other attributes. Suppose you update an attribute in the AC model composition. $\mathcal{SPT}$ will automatically change it at all policies with such an attribute. For all the testing functions, $\mathcal{SPT}$ presents the detailed testing analysis as well as the decision associated with the testing environments. $\mathcal{SPT}$ not only gives the testing result, but also all the rules and polices with their decisions which allows the policy author to find: (i) what is the AC flaw, (ii) where to fix the flaw, (iii) how to correct the flaw, (iv) what is the AC effectiveness after the modification. This is because the policy author can easily analyze the results by reviewing the rule or policy decision-making in the test process. These functions will be described in the next section. This subsection gives several typical AC errors which may be useful for policy analysis.

We define an AC flaw as an unintended decision that could be caused by rules, policies, as well as their algorithms. Some of AC flaws are hidden from detection as the decision involves the effectiveness of multiple rules and policies. Some typical AC flaws are classified as below.

Error Type 1 (Block Privilege): Suppose you know A should access B. However, the B's accessibility by A is false by following the policy. This type of error is referred as Privilege Blocking in the specification [2]. It blocks a legitimate access to rightful resources. It can also occur when the properties of an AC policy cannot render a grant or deny decision, or there is no available logic in the AC policy algorithm for evaluating the access request. It can also be a result of the deadlock of access rules where a rule has a dependency on other rule(s), which eventually depend back on the rule itself, so that a subjects request will never reach a decision because of the cyclic referencing.

Error Type 2 (Leak Privilege): Suppose you know A shouldn't have the accessibility of B. However, A is able to B by following the policy. NIST refers this type of error as Privilege Leakage [2]. It is the situations in which a subject is able to access resources that are prohibited by the safety requirements. Such leakage may cause either the privilege escalation from one resource domain or class to prohibited ones such as leakage from lower to higher ranks of an MLS policy, or privilege leak such as from one role to other prohibited ones of an RBAC policy. Leak Privilege (i.e. action and resource pair) can be caused by mistaken privilege assignment directly or careless privilege inheritance indirectly.

Error Type 3 (Not Protected Resource): This is an error that a Resource (e.g., B) is not protected by any rules and policies. For example, a document d is not covered by any policy. The unprotected resource can cause an error type of Block Privilege.

Error Type 4 (Rule Conflict): This is an error that two or more rules are conflicted in a policy, e.g., a rule allows the access while the other declines. NIST refers this type of error as Privilege conflict and stated as the following. Unlike regular programming logic that a later value assignment of a variable overwrites the previous assigned value of the same variable, the rules of an AC policy normally have no precedence consideration in permission evaluation. In other words, AC rules will not be overwritten by other rules unless specifically allowed to. Thus, privilege conflicts appear when the specifications of two or more access rules result in the conflicting decisions of permitting subjects access requests by either direct or indirect (inherit) access assignments. In addition, when multiple policies are evoked for permission, conflicting decisions between policies may occur.

Error Type 5 (Inconsistent Assignment): Suppose a policy author edits a number of XACML policy documents separately. Attributes, conditions, rule or other policy variables/values could be inconstantly assigned in different policies. For example, $Attribute\ Nurse$ could be inconstantly termed as $nursy$, $murse$ in different policy documents. $\mathcal{SPT}$ prevents this error with integrity verification.

Error Type 6 (AC Inheritance Loop): A policy author may specify an error inheritance relation. An AC inheritance loop is caused by recursive and subsequent inheritance, e.g., $Attribute\ A \rightarrow Attribute\ B \rightarrow Attribute\ C \rightarrow Attribute\ A$. $\mathcal{SPT}$ is able to automatically detect and prevent an inheritance loop. This type of error is referred as Cyclic Inheritance at the NIST's specification [2]. It is the problem of privileges inheritance from other users (groups), which also in a chain of inheritance relation inherit back to the user (group)'s privilege. AC Inheritance Loop leads to undecidable or infinite access evaluation process.

Error Type 7 (Undecided Rules): Undecided rule error is that a Resource is unassigned with necessary actions in the workflow AC model. For example, a purchase order is assigned for person to create but there are no rule is assigned to approve the order. The workflow is unable to move forward on the working flows due to this error.

Error Type 8 (Separation of Duty Error): This is an error that two or more rules cause the competing interests among subjects, resources, or actions. For example, a person with Role A and Role B is conflict for accessing a resource.

The above Error Types includes all the AC flows currently identified by NIST SP 800-192 specification [2].

6.5 Consideration of Multiple Policies

As discussed in NIST specification [2], multiple policies could be involved in a distributed enterprise environment, such as cloud, distributed networks or systems, IoTs, or other distributed systems. It has the case that AC policies are independently developed by different collaborative or networked systems. In such a case, an inter-system access request (e.g., cross domain access request) may be evaluated by more than one policies that the requesting subject is governed under.

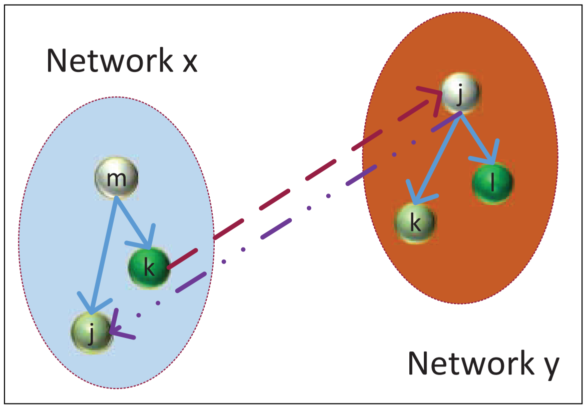


Figure 14: A Flaw Error in a Distributed Network [2]

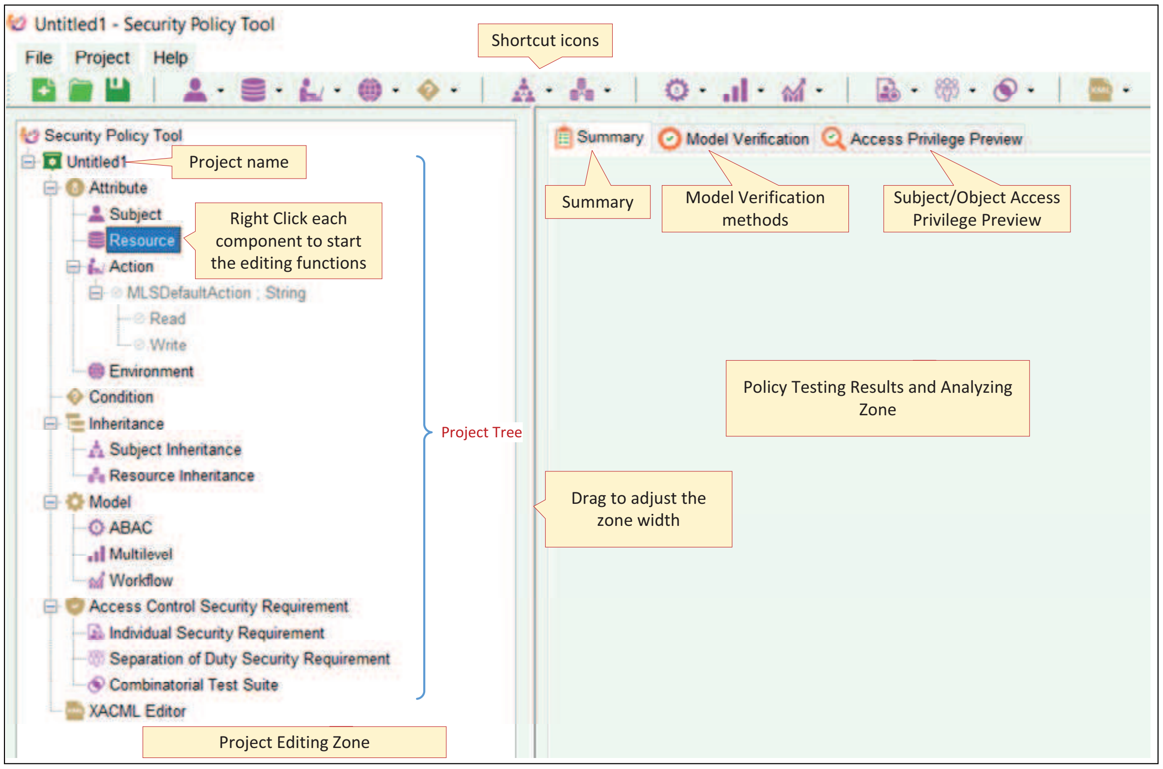
Therefore, AC policy autonomy should also be preserved for secure inter-system access. Maintaining the autonomy of all collaborative systems is a key requirement of the policy for inter-operation. The principle of autonomy states that if an access is permitted by an individual system, it must also be permitted under secure inter-system access. The principle of security states that if an access is denied by an individual system, it must also be denied under secure inter-system access. In such a collaborative system, violations of secure inter-system access can be caused by adding intersystem privilege inheritance relations or other correlations among policies. Figure 1 [2] shows an example that privilege k inherits privilege j through legal inter-system privilege inheritance (because both have the same privilege level j), which is granted in network x but denied in network y. These types of violations can be detected by checking for cyclic inheritance, leak privilege, and Separation of Duty errors. Thus, both security and autonomy can be characterized as safety requirements of a multi-policies AC policy, which should be preserved during collaborations. A meta policy is a policy that is usually applied for reconciling policy conflicts or to handle priorities of access decisions rendered from more than one policy. Thus, in addition to autonomy requirements, AC safety requirement may include a priority model within the meta policy.

7 Operational Guide

$\mathcal{SPT}$ provides a series of functions to compose an AC security model, policy test, policy analysis, AC flaw inspection, AC flaw correction, and finally transfer all the policies in the security model into XACML formatted files such that they can be deployed into an AC system with high-security confidence. This section demonstrates these functions in details. The illustration of the $\mathcal{SPT}$ functions may incorporate the example in Figure 1 for understanding purpose.

7.1 $\mathcal{SPT}$ Installation

$\mathcal{SPT}$ software is generally recommended to be installed on a server or computer with Intel Core $i7-7700K-16GB$ Memory, equal, or higher configuration. This is because the policy verification of combinatorial testing suites could involve a number of security requirements (e.g., hundreds or thousands of combinatorial security requirements) is computationally complex with a high volume of memory consumption. It also recommends a 34″ or bigger monitor with a minimal resolution of 1920×1080 or higher. The testing results are displayed by tables with rich texts. $\mathcal{SPT}$ can run in Microsoft Windows 8 and 10 OS systems.



Untitled1 - Security Policy Tool
File
Project
Help
Shortcut icons
Security Policy Tool
Untitled1
Project name
Attribute
Subject
Resource
Right Click each component to start the editing functions
Action
MLSDefaultAction : String
Read
Write
Environment
Condition
Inheritance
Subject Inheritance
Resource Inheritance
Model
ABAC
Multilevel
Workflow
Access Control Security Requirement
Individual Security Requirement
Separation of Duty Security Requirement
Combinatorial Test Suite
XACML Editor
Project Tree
Project Editing Zone
Summary
Model Verification
Access Privilege Preview
Summary
Model Verification methods
Subject/Object Access Privilege Preview
Policy Testing Results and Analyzing Zone
Drag to adjust the zone width

Figure 15: $\mathcal{SPT}$ Project Main Interface

7.2 Project Main Interface

Figure 15 shows a blank project interface when the $\mathcal{SPT}$ program is started. Without any doubt, a project has a project name. Roughly, $\mathcal{SPT}$ project interface includes menus, shortcut icons, policy editing zone, testing and analyzing zones. Figure 15 shows the major functional components:

- **Project Tree**: In the policy editing zone, the project tree is utilized to organize the AC security model and policy components. The project tree specifically consists of “Attribute”, “Condition”, “Inheritance”, “Model”, “Access Control Security Requirement”, and a “XACML editor”. Each tab (i.e., component) of the project tree may have sub-tabs to provide a multitude of AC composing and updating functions. These functions appear upon the right click of the tab, e.g., right click “Attribute”.
- **Summary**: The “Summary” tab in the Policy Testing Results and Analyzing Zone provides a global view of the defined AC model, AC resources, and the policy testing and analyzing results.
- **Model Verification**: “Model Verification” allows the policy author to step-by-step configure the policy testing schema and algorithms, perform comprehensive model checking functions, present the model testing results, conduct AC flaw inspection, and perform AC flaw correction and retesting. It also has the functions to output the testing and analyzing results by table or other formats.
- **Access Privilege Preview**: “Access Privilege Preview” enables a policy author to query the accessible resource giving a subject in its AC circumstance, e.g., querying what are the resources that can be accessed by a $Nursing\ Attendent$. It also allows you to preview the subject accessibility of given resources, e.g., query who can access the patient $Prescription$.

It is noted that $\mathcal{SPT}$ allows a policy author to open multiple projects and each project will have separate interfaces.

7.3 New or Open an Project

There are several ways to start a project with the “File” menus as shown in Figure 15:

- **New a Project**: This is to create a new project from a “New” submenus under “File”,
- **Open a Project**: An exiting project can be opened from a “Open” submenus under “File”. “Open” submenus allows to open a recent file,
- **Import XACML**: A project can be established from an existing XACML policy by imputing an external XACML file from “Import” submenus under “File”.

It is noted that multiple AC policy projects can be created and they are opened in separate project interfaces.

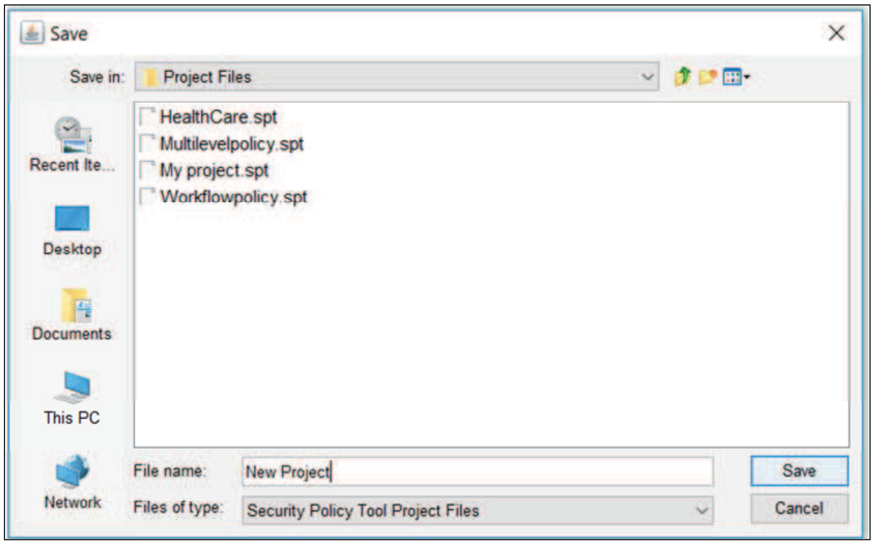


Figure 16: Selecting "Save As" to Save the Project

7.4 Saving a Project

"Save As" submenus under "File" saves the project in the $.spt$ format. Figure 16 shows a saving example. As shown in Figure 16, $\mathcal{SPT}$ saves the project information as a spt file type. This format can be only properly interpreted by $\mathcal{SPT}$ software. Meanwhile, the XACML policy can be exported as XACMAL files. In addition, the policy editing and testing results can be saved to Microsoft Excel (e.g., xlsx, xls) files and they can be printed out, which will be shown later.

7.5 Add/Update/Delete Attributes and Attribute Values

The composition of attributes is the first step to create an AC model. The defined attributes will be used to compose policies with rules. More specifically, attributes and their values can be added, updated (e.g., modified or revised), and deleted.

7.5.1 Attribute Composition

It is noted that $\mathcal{SPT}$ defines a $MLSDefaultAction$ attribute by default, which is exclusively used Multileve Security Model. $MLSDefaultAction$ has two actions, namely Read and Write, as shown in Figure 17. Due to this, Multilevel security is prevented from defining any other Action attributes and values. We use Subject to illustrate the Add, Update, and Delete attribute composition operations. It is similar to add/update/delete Resource, Action, and Environment attributes.

Add an Attribute: Let's take Subject Attribute as an example to show how to define a new attribute. The steps are:

- Right click on "Subject", then left click "Add a New Subject Attribute" as shown in Figure 17. An "Add Subject Attribute" box will pop up as shown in Figure 18 (a).

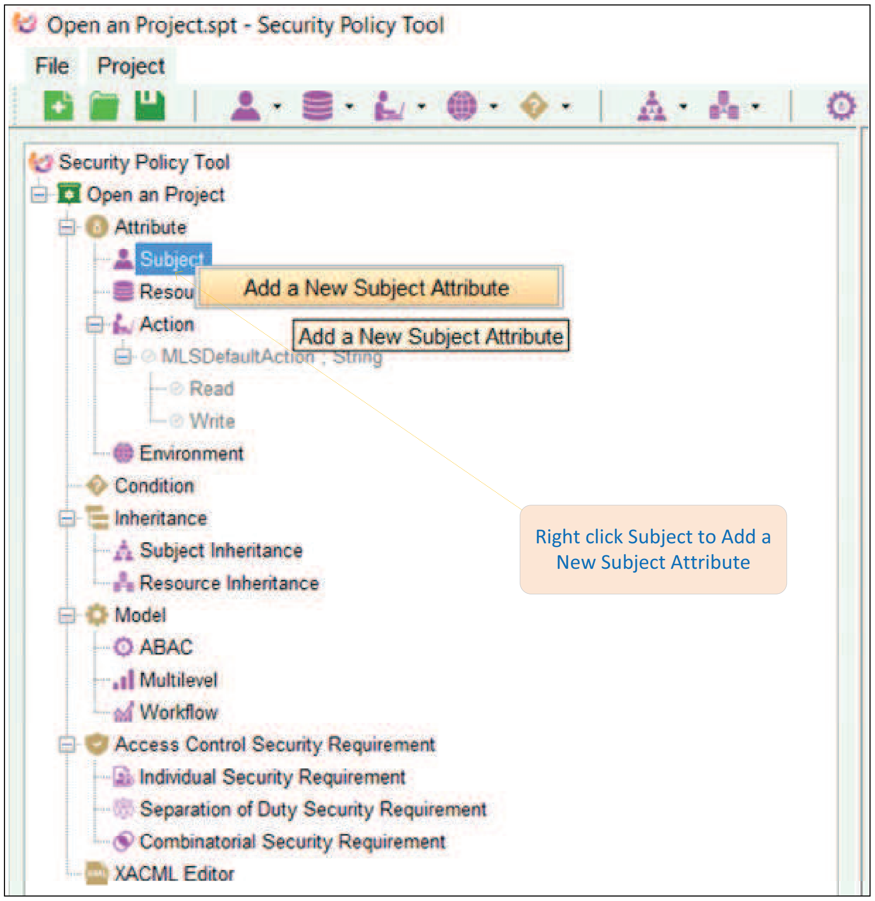
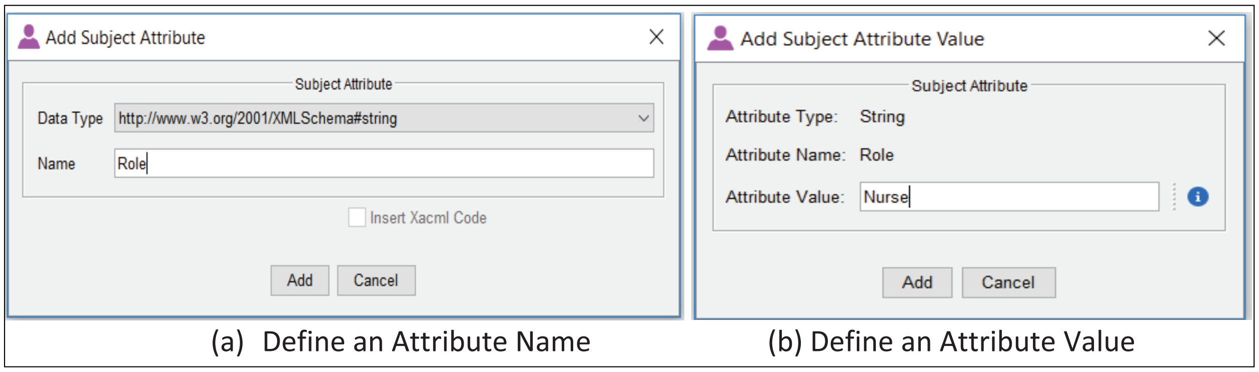

Open an Project.spt - Security Policy Tool
File
Project
Security Policy Tool
Open an Project
Attribute
Subject
Resou
Action
MLSDefaultAction : String
Read
Write
Environment
Condition
Inheritance
Subject Inheritance
Resource Inheritance
Model
ABAC
Multilevel
Workflow
Access Control Security Requirement
Individual Security Requirement
Separation of Duty Security Requirement
Combinatorial Security Requirement
XACML Editor
Add a New Subject Attribute
Add a New Subject Attribute
Right click Subject to Add a New Subject Attribute

Figure 17: Right click on “Subject” to Add a New Subject Attribute

- Select an attribute “Data Type” from the drop down menu, such as *string*, *boolean*, *double*, *time*, *date*, *dateTime*, *anyURI*, *ipAddress*, *dnsName*, etc. $\mathcal{SPT}$ supports all “Data Type” defined in XACML.
- “Name” the attribute, such as *Role*.



Data Type: http://www.w3.org/2001/XMLSchema#string
Name: Role

Attribute Type: String
Attribute Name: Role
Attribute Value: Nurse

Figure 18: Windows for Adding an Attribute

- Click “Add” and then the composed attribute will show in the Subject list and meanwhile a popup “Add Subject Attribute Value” window from which you define the *Attribute Value*, such as *Nurse*, *Midwife*, etc. The "Add Subject Attribute Value" window as shown in Figure 18 (b) allows you to define an attribute value, e.g., *Nurse*. Meanwhile, it allows the definition of multiple attribute values while these values are separated by comma ",", e.g., *Chief of Hospital, Head Nurse, Nurse, Midwife, Nursing Attendant* are defined as five values of the attribute role in Figure 1.

It is worth mentioning that $\mathcal{SPT}$ performs the integrity check such that the composed attribute value is correctly matched to the "Data Type". Some example attribute values are provided for a policy author to easily edit the attribute value in a correct format. Similar to the definition of Subject attributes and their values, a policy author can compose a set of Subject, Resource, Action, and Environment attributes, according to the specific AC system.

Update an Attribute: An composed attribute can be modified anytime, even during the policy composition and test. Consider a subject attribute, e.g., *role*:

- Right click on the subject attribute (e.g., role) that needs to be modified and select "Update Access Control Attribute", as shown in Figure 19. A box of "Update Subject Attribute" will pop up with the subject information.
- Change the attribute name to a new one, e.g., *Updated Role*, and then
- Click on "Update" to finish the updating.

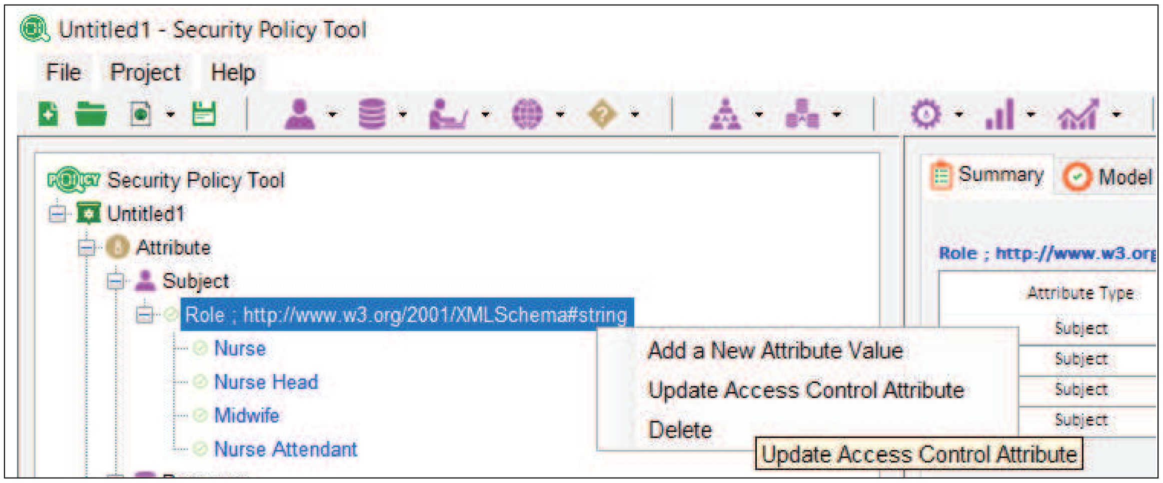


Figure 19: Right Click a Subject Attribute and Select "Update Attribute"

It is noted that you can only change the "Name" of the selected attribute from the "Update Subject Attribute" function. The modified attribute will be automatically updated in the rules, policies, and the security requirements if the attribute has been already used in the AC model. This maintains the attribute consistency in the entire scope of the project. If you think you have wrongly defined the attribute "Data Type", you'd like to use the "Delete" function to remove it and redefine a correct one.

Delete an Attribute: This allows you to delete an attribute:

- Right click the attribute that needs to be deleted in the Project tree and select "Delete", as shown in Figure 20.

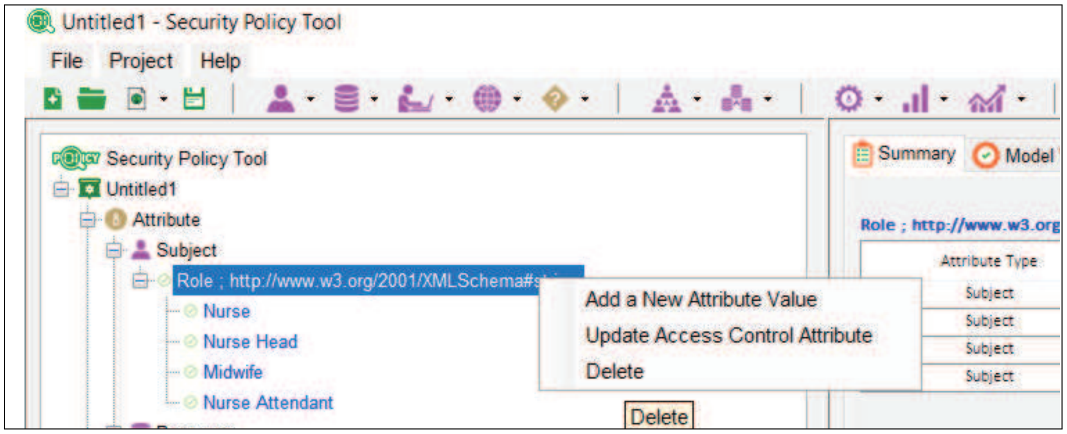


Figure 20: Right Click the Attribute that Needs to be Deleted and Select "Delete"

- Click "Yes" in the popup box to make sure the deletion.

Note: If an attribute is employed in any rule, the attribute cannot be deleted because that policy with a non-existing attribute is incomplete. The attribute can only be deleted if you first remove all the rules and security requirements that are engaged with the attribute.

Attribute Summary: When an attribute is composed, the "Attribute Summary" tab (under the "Summary" tab) will be automatically updated with the defined attribute. Clicking on the "Subject", the "Subject Summary" tab displays all the subject attributes with details of the definition, e.g., Data Type, Name, Values, and Time Created, and Time Modified, as shown in Figure 20. By clicking a specific Attribute, e.g., *Nurse*, the Summary tab lists all the Rules and Security Requirements that the Attribute Value is used in. The definition of Security Requirements will be further discussed.

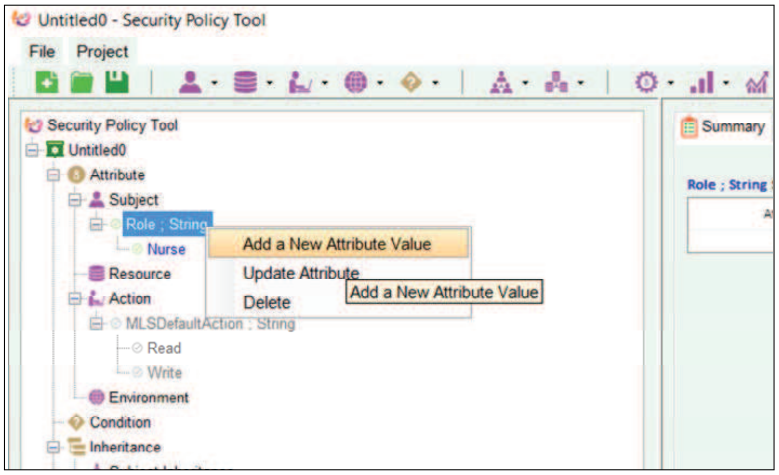


Figure 21: Right click on the Targeted Subject Attribute and Select "Add a New Attribute Value"

7.5.2 Attribute Values

Add Attribute Values: An attribute should have at least one attribute value. $\mathcal{SPT}$ allows one or more attribute values to be added at one time to an attribute after the attribute is created. Adding an attribute value is conducted by:

- Right click the selected attribute and then left click “Add a New Attribute Value”. It will pop up a window such as “Add Subject Attribute Value”, as shown in Figure 21.
- Input the attribute value of the specific attribute “Data Type”. Multiple attribute values can be input by separating them with a comma, such as *Administration, Nursing, Medical* for three values of attribute *Division* in the example of Figure 1.
- Click “Add” button to add the value.

Note: The editing of attribute values doesn’t allow you to change the attribute “Data Type” and “Name”. $\mathcal{SPT}$ prevents duplicated attribute values for a given attribute.

Update an Attribute Value: It allows to modify an attribute value:

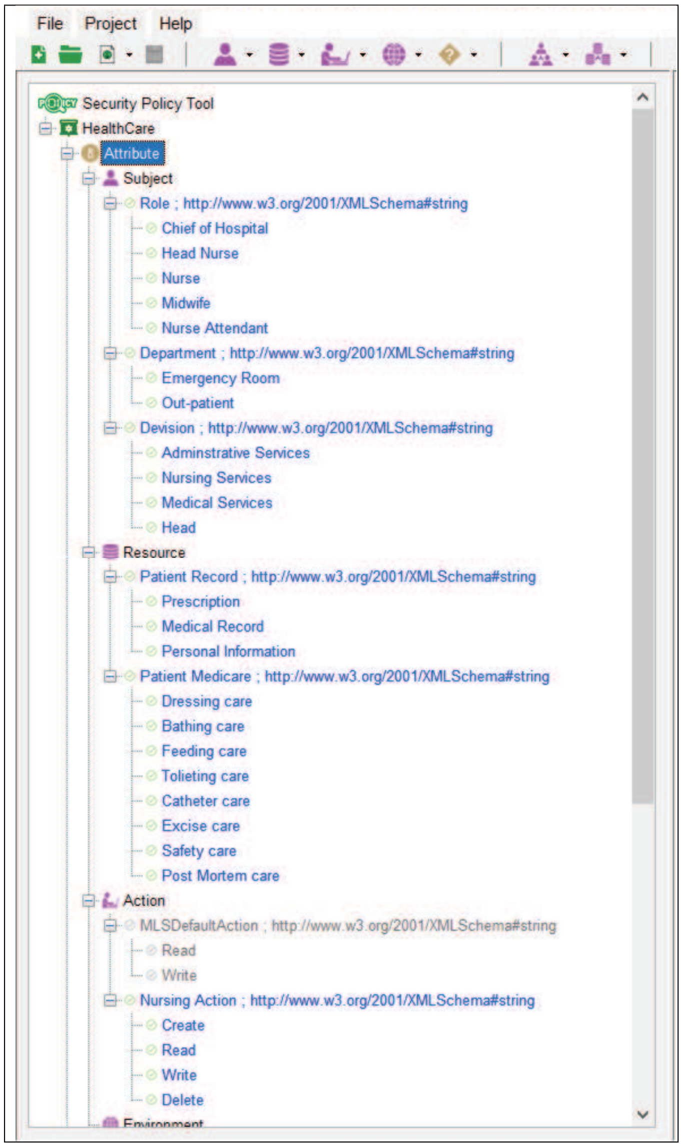
- Right click the attribute value that needs to be updated and select “Update Attribute Value”.
- Change the attribute value in the popup box.
- Click “Update”.

If the attribute value has been utilized in rule and security requirement, the modified one will be automatically updated in these engaged rule and the security requirement to maintain the attribute consistency.

Remove an Attribute Value: An attribute value can be deleted if it is not engaged in any rule and security requirement.

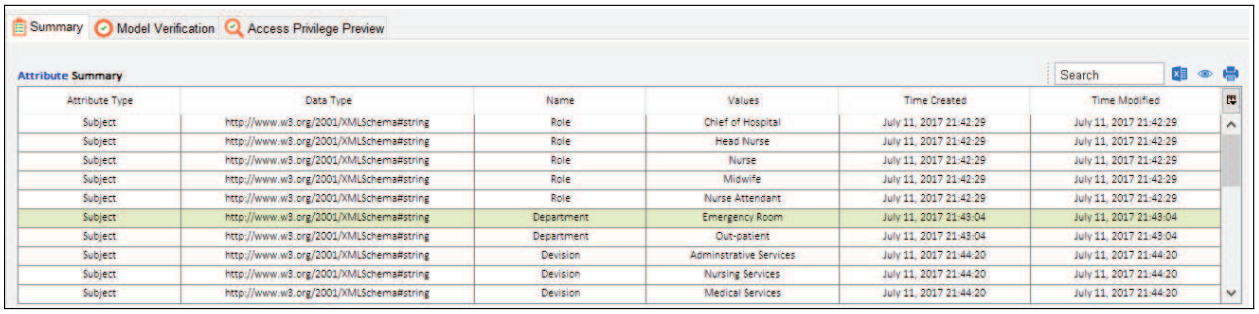
- Right click the attribute value that needs to be deleted in the list and select “Remove Attribute Value”.
- Click “Yes” to make sure the deletion.

If an attribute value is engaged in a policy or a security requirement, the attribute value cannot be deleted immediately because this makes the policy and security requirement incomplete with a undefined attribute. In this case, you should first remove all the rules and security requirements that utilize the attribute value before removing the attribute value.



```
Security Policy Tool
HealthCare
  Attribute
    Subject
      Role ; http://www.w3.org/2001/XMLSchema#string
        Chief of Hospital
        Head Nurse
        Nurse
        Midwife
        Nurse Attendant
      Department ; http://www.w3.org/2001/XMLSchema#string
        Emergency Room
        Out-patient
      Devision ; http://www.w3.org/2001/XMLSchema#string
        Adminstrative Services
        Nursing Services
        Medical Services
        Head
    Resource
      Patient Record ; http://www.w3.org/2001/XMLSchema#string
        Prescription
        Medical Record
        Personal Information
      Patient Medicare ; http://www.w3.org/2001/XMLSchema#string
        Dressing care
        Bathing care
        Feeding care
        Tolieting care
        Catheter care
        Excise care
        Safety care
        Post Mortem care
    Action
      MLSDefaultAction ; http://www.w3.org/2001/XMLSchema#string
        Read
        Write
      Nursing Action ; http://www.w3.org/2001/XMLSchema#string
        Create
        Read
        Write
        Delete
    Environment
```

Figure 22: Attributes Composed for Figure 1



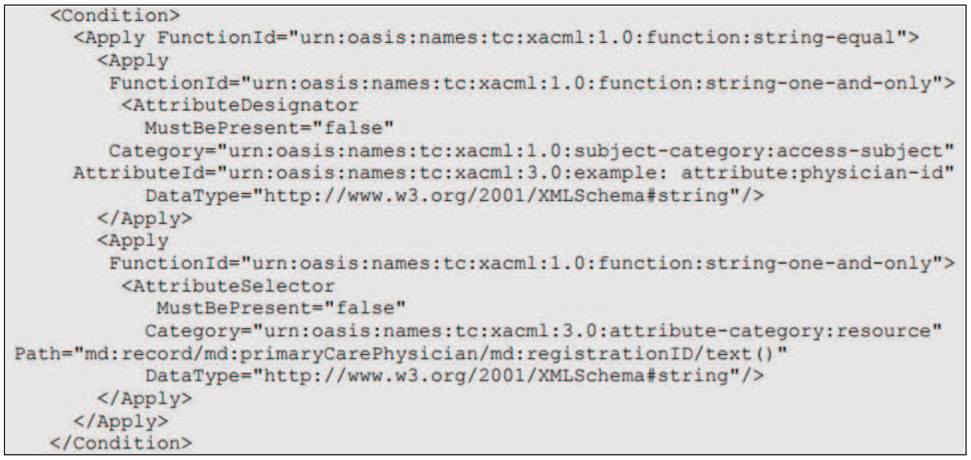
Attribute Summary

Attribute Type	Data Type	Name	Values	Time Created	Time Modified
Subject	http://www.w3.org/2001/XMLSchema#string	Role	Chief of Hospital	July 11, 2017 21:42:29	July 11, 2017 21:42:29
Subject	http://www.w3.org/2001/XMLSchema#string	Role	Head Nurse	July 11, 2017 21:42:29	July 11, 2017 21:42:29
Subject	http://www.w3.org/2001/XMLSchema#string	Role	Nurse	July 11, 2017 21:42:29	July 11, 2017 21:42:29
Subject	http://www.w3.org/2001/XMLSchema#string	Role	Midwife	July 11, 2017 21:42:29	July 11, 2017 21:42:29
Subject	http://www.w3.org/2001/XMLSchema#string	Role	Nurse Attendant	July 11, 2017 21:42:29	July 11, 2017 21:42:29
Subject	http://www.w3.org/2001/XMLSchema#string	Department	Emergency Room	July 11, 2017 21:43:04	July 11, 2017 21:43:04
Subject	http://www.w3.org/2001/XMLSchema#string	Department	Out-patient	July 11, 2017 21:43:04	July 11, 2017 21:43:04
Subject	http://www.w3.org/2001/XMLSchema#string	Devision	Adminstrative Services	July 11, 2017 21:44:20	July 11, 2017 21:44:20
Subject	http://www.w3.org/2001/XMLSchema#string	Devision	Nursing Services	July 11, 2017 21:44:20	July 11, 2017 21:44:20
Subject	http://www.w3.org/2001/XMLSchema#string	Devision	Medical Services	July 11, 2017 21:44:20	July 11, 2017 21:44:20

Figure 23: Attributes in a Table Composed for Figure 1

Figure 22 shows examples of the attributes defined for the Nursing Service described in Figure 1. The attributes are organized by a tree and each node represents an attribute name or an attribute value.

Clicking "Attribute" in the project tree, " Attribute Summary" tab will present the attribute summary as shown in Figure 23. It lists all the attributes with the detailed definition. For each attribute, the table shows the "Attribute Type", "Data Type", "Name", "Value", "Attribute Created", and "Time Modified". The Summary also provides a Search function to find an attribute composed. Meanwhile, it has an Excel icon from which you can save the attributes in an Excel table. An enlarge icon offer you to display the attributes in a separate page. In addition, you can print the summary by clicking on the Print icon.

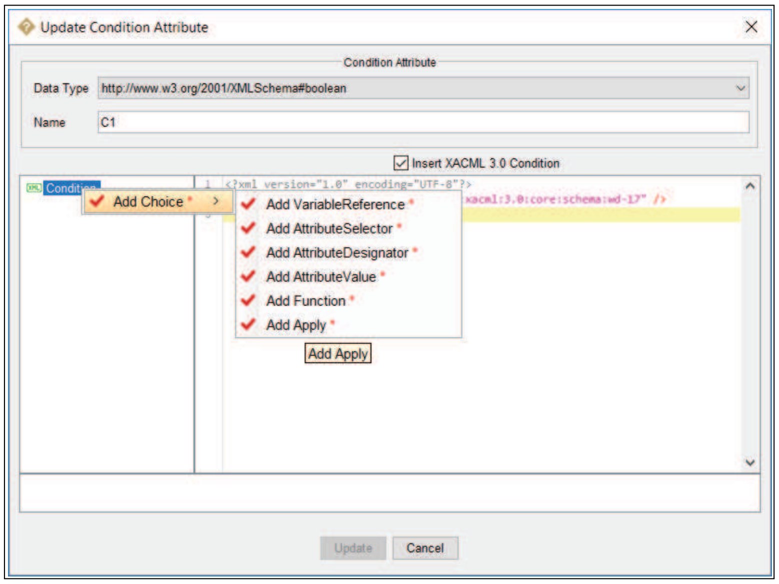


```
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Apply
         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <AttributeDesignator
            MustBePresent="false"
         Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
      AttributeId="urn:oasis:names:tc:xacml:3.0:example: attribute:physician-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
        <Apply
         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <AttributeSelector
             MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
Path="md:record/md:primaryCarePhysician/md:registrationID/text()"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
      </Apply>
    </Condition>
```

Figure 24: Condition - An Example

7.6 Condition

Condition is a boolean decision function to decide if Effect applies. It may be absent. The boolean decision function can be evaluated to "True", "False" or "Indeterminate. If the Condition evaluates to True, then the Rule's Effect (Permit or Deny that is associated with successful evaluation of the Rule) will be returned. Figure 24 shows an XACML 3.0 Condition example. Please noted that the evaluation of the boolean decision function of the Condition comes from the request context in the AC actual system, e.g., the request time, the location, etc. In other words, the context of the actual system is evaluated by real time. $\mathcal{SPT}$ is unable to evaluate the context to yield a boolean decision as it is returned from the evaluation of the actual system. On the other hand, Condition can be incorporated in the policy test if the boolean decision (i.e., $True/False$) is known. The policy author can define a condition with "True" or "False" decision to evaluate the Effect of the condition in policies. This means that $\mathcal{SPT}$ can verify the Effect given the "True" or "False" decision. From $\mathcal{SPT}$, the policy author can test: (i) What is the Effect for a given "True" of a condition, and (ii) What is the Effect for a given "False" of a Condition. The evaluation of "Indeterminate for condition is not considered in $\mathcal{SPT}$ test.



Data Type http://www.w3.org/2001/XMLSchema#boolean

Name C1

[✓] Insert XACML 3.0 Condition

```
<?xml version="1.0" encoding="UTF-8"?>
xacml:3.0:core:schema:wd-17" />
```

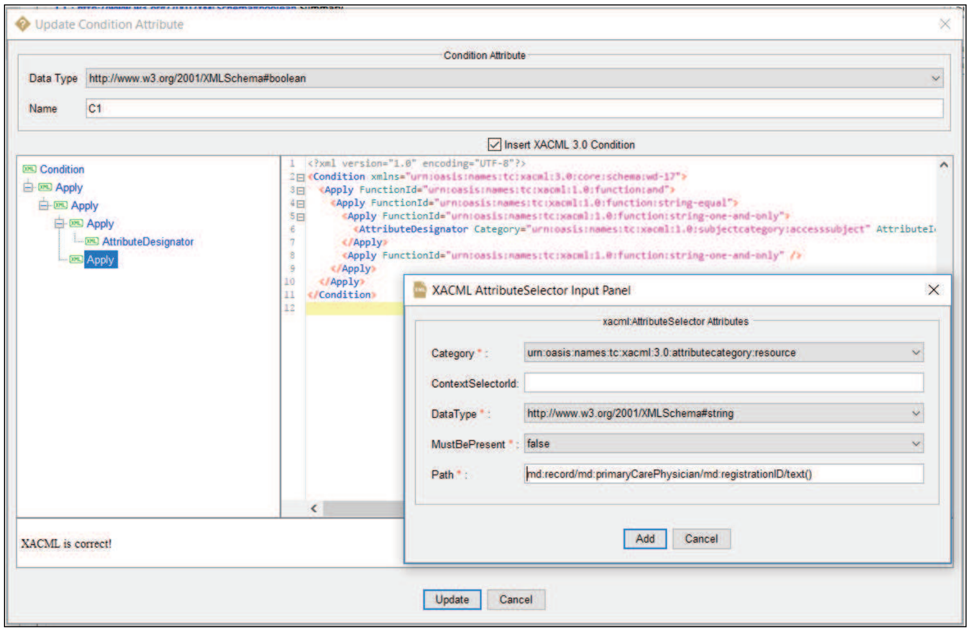
Figure 25: Condition Editing

$\mathcal{SPT}$ provides GUI for a policy author to edit an XACML 3.0 condition which starts by right clicking on "Condition" in the project tree. "Add a new Condition Attribute" shows up and clicking on it to pop up a "Add Condition Attribute" window. By choosing "Insert XACML 3.0 Condition", you will see a window as shown in Figure 25. As shown in this window, the Data Type of the condition is set up as boolean. Then, you can name the condition, such as C1 as shown in Figure 25.

Right clicking on "Condition", it allows you to add "Add Choice" such as "Add VariableReference", "Add AttributeSeclector", "Add AttributeDesignator", and so on as shown in Figure 25. Click on "Add", these condition elements will be translated into XACML 3.0 code. Meanwhile, these Condition elements can be modified. Let's consider the example in Figure 24. The Condition has three $< Apply \cdots Apply >$ elements, an $AttributeDesignator$, and an $AttributeSeletor$. They can be added one by one by following the XACML structure as shown in Figure 24. Figure 26 shows the interface to add an $AttributeSeletor$ that includes the spcifications of a Category, DataType, MustbePresent, Path, and other optional variables. After clicking on the "Add" button the XACML 3.0 codes will be added to the condition box. All the elements are similarly added.

7.7 Subject/Resource Inheritance Composition

Upon the composition of attributes, inheritance can be created to reflect the hierarchical subject and resource attribute structure. Inheritance is an important feature to compose a number of rules with integrity in the entire policy scope. If a rule is defined/updated/removed at a subject or resource, its beneficiaries are automatically defined/updated/removed the corresponding rules. Please refer


Update Condition Attribute

Condition Attribute

Data Type: http://www.w3.org/2001/XMLSchema#boolean

Name: C1

☑ Insert XACML 3.0 Condition

```
<?xml version="1.0" encoding="UTF-8"?>
<Condition xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subjectcategory:accesssubject" AttributeI
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" />
    </Apply>
  </Apply>
</Condition>
```

XACML is correct!

XACML AttributeSelector Input Panel

xacml:AttributeSelector Attributes

Category * : urn:oasis:names:tc:xacml:3.0:attributecategory:resource

ContextSelectorId:

DataType * : http://www.w3.org/2001/XMLSchema#string

MustBePresent * : false

Path * : md:record/md:primaryCarePhysician/md:registrationID/text()

Figure 26: Add an Condition Element

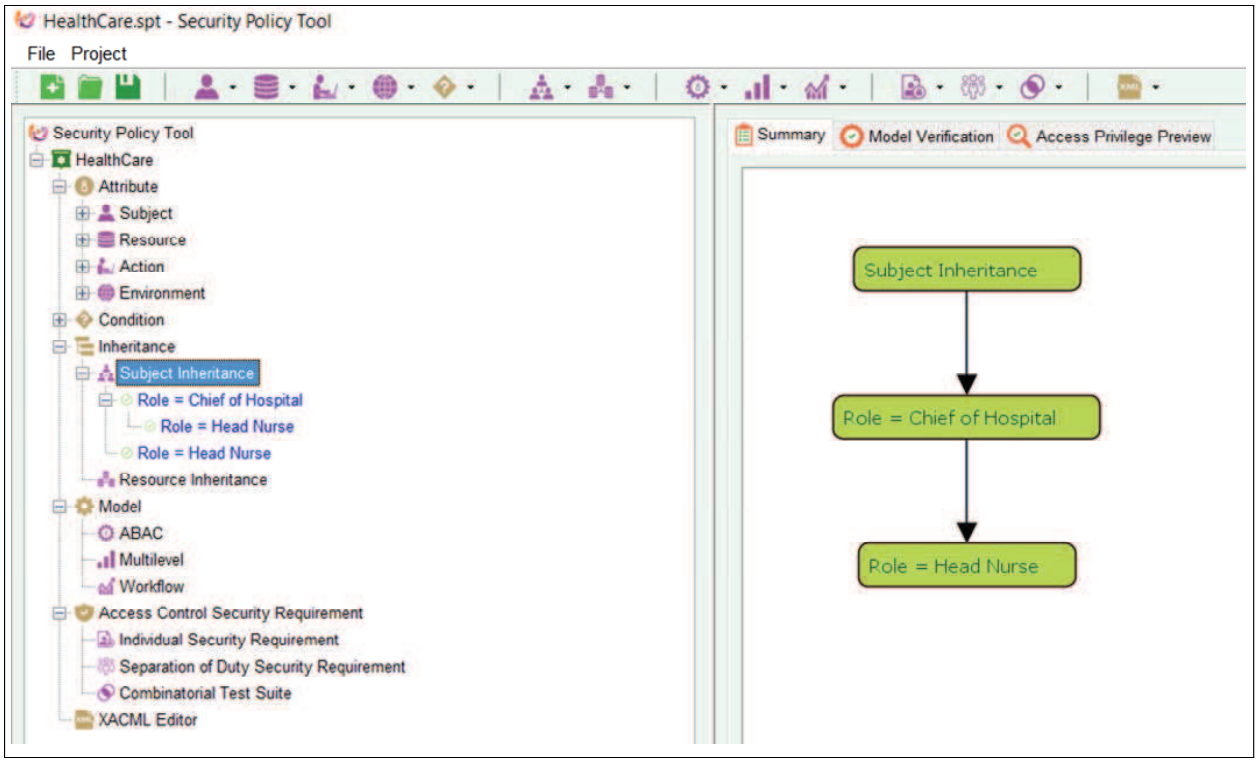
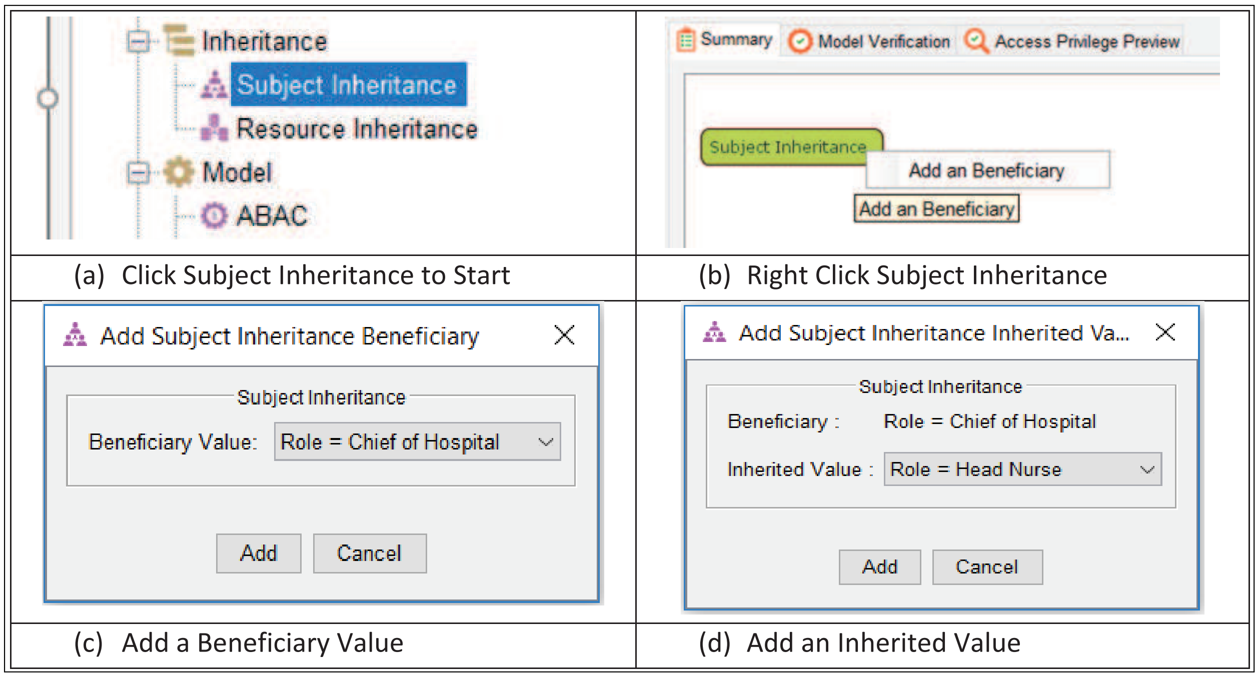


Figure 27: Inheritance - A Simple Example

Section 5.3 for the details. $\mathcal{SPT}$ allows a policy author to graphically create inheritance relations in an easy way. Figure 27 shows a simple subject inheritance graph that only has one beneficiary and one originator, denoted by $Chief\ of\ Hospital \rightarrow Head\ Nurse$.

Inheritance could be the subject inheritance or resource inheritance. The composition of them is similar. The inheritance composition starts by clicking on the "Inheritance" in the project tree which will lead you to the inheritance Summary. Inheritance can be graphically built. Consider the Subject inheritance. Figure 28 shows how the inheritance in Figure 27 is composed and the process is explained below.


Inheritance
Subject Inheritance
Resource Inheritance
Model
ABAC
(a) Click Subject Inheritance to Start
Summary
Model Verification
Access Privilege Preview
Subject Inheritance
Add an Beneficiary
Add an Beneficiary
(b) Right Click Subject Inheritance
Add Subject Inheritance Beneficiary
Add
Cancel
(c) Add a Beneficiary Value
Add Subject Inheritance Inherited Va...
Add
Cancel
(d) Add an Inherited Value

Subject Inheritance

Beneficiary Value: Role = Chief of Hospital

Subject Inheritance

Beneficiary : Role = Chief of Hospital

Inherited Value : Role = Head Nurse

Figure 28: Steps to Create an Inheritance Relation

Create a Subject Inheritance: The addition of an inheritance relation has the following operations:

- Right click the "Subject Inheritance" in the inheritance graph or in the project tree and select "Add a New Beneficiary" as shown in Figure 28 (a) and (b). An "Add Subject Inheritance Beneficiary" window will popup that allows you to choose the "Beneficiary Value", as shown in Figure 28 (c),
- Select a "Beneficiary Value", e.g, $Chief\ of\ Hospital$, from the drop-down menu in an "Add Subject Inheritance Beneficiary" window and click "Add", as shown Figure 28 (c). After this step, an "Add Subject Inheritance Inherited Value" window will pop up, and then
- Select an "Inherited Value", e.g., $Head\ Nurse$, from the drop down menu in the "Add Subject Inheritance Inherited Value" window and click "Add" as shown in Figure 28 (d).

The chosen "Beneficiary Value" will be the Beneficiary and the "Inherited Value" will be the Originator, e.g., $Chief\ of\ Hospital \rightarrow Head\ Nurse$, as described in Figure 3, where the Beneficiary will automatically inherit the rule composed for the Originator.

Add New Inherited Value To A Beneficiary: A new inheritance relation can be added to the hierarchical structure by an operation of "Add an Inherited Value" to a selected beneficiary. The steps are:

- Right click a selected beneficiary e.g., $Role = HeadNurse$, and then select "Add an Inherited Value", and
- Select an "Inherited Value", e.g., $Role = Nurse$, from the dropdown menu in the "Add Subject Inheritance Inherited Value" and click "Add".

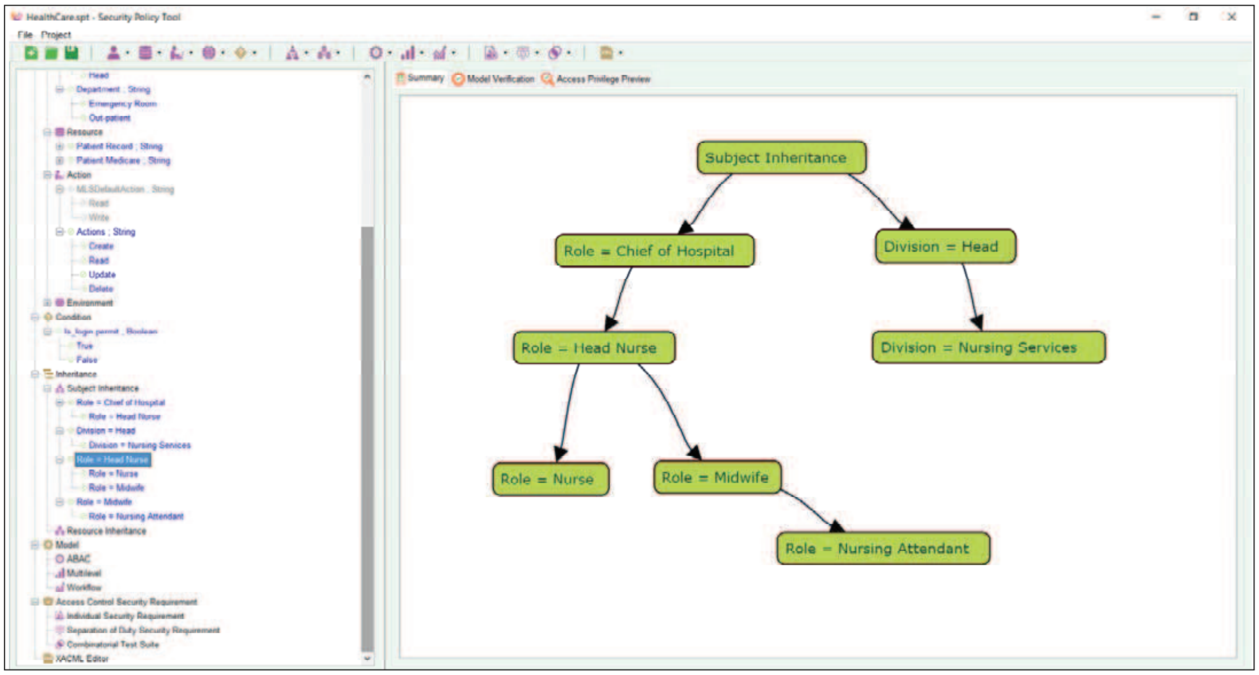


Figure 29: An Example Inheritance Graph

By repeating the above steps, you can add as many as inheritance relations to the composing AC model. A multilevel hierarchical structure can be then composed as a tree. The effort for you to manage the rule is reduced when more inheritance relations are added. This is because the rules are automatically prorogated and updated if the rule in the Originator is composed or modified. This gives a clear view of the AC policies and meanwhile, it reduces the number of rules to compose and maintain. The rule prorogation principle is discussed in Subsection 5.3.

$\mathcal{SPT}$ provides a visual graph to present the inheritance relations. The graph is shown in the "Summary" of the "Subject Inheritance". Clicking "Subject Inheritance", the "Summary" tab then shows the subject inheritance graph. Clicking on the "Inheritance" will lead you to inheritance Summary tab which shows both the Subject Inheritance and Subject Inheritance graphs. The added inheritance relations will show up on the inheritance graph immediately once the relation is composed. The inheritance graph can be zoomed in and out by scrolling the mouse up or down respectively. Meanwhile, the graph is dynamically organized in a way to have a better view. You can also use the mouse to hold on any blank space to move the entire graph and drag a graph component to organize the graph in your way.

Delete a Beneficiary/Inherited Value: Any inheritance relation can be deleted from the graph and the relation is updated to your AC model. Deleting a beneficiary or inherited object has the following operations:

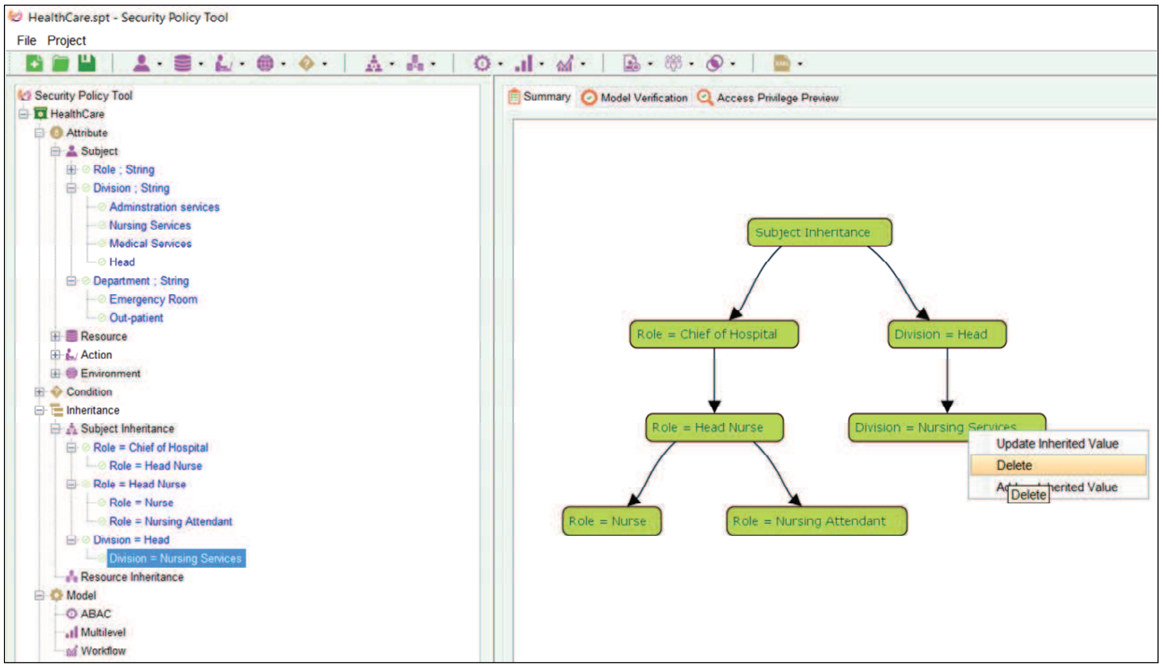


Figure 30: Delete a Beneficiary/Inherited Value

- Right click the beneficiary/inherited value that needs to be deleted, such as $Division = Nursing\ Service$ as shown in Figure 30. Then, select “Delete”, and
- Click “Yes” in the “Selection an Option” popup window to complete the deletion. The beneficiary/inherited value and its inherited relations are both deleted from the graph.

Note: In a similar way, you can update a Beneficiary/Inherited Value. It is worth mentioning that you can edit (i.e., Add, Delete, Update) the inheritance graph according to the actual inheritance relations while the corresponding inheritance relations among rules are automatically updated without additional actions to take.

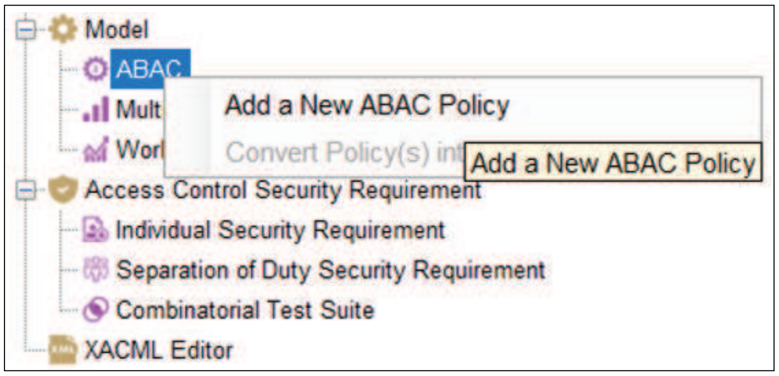
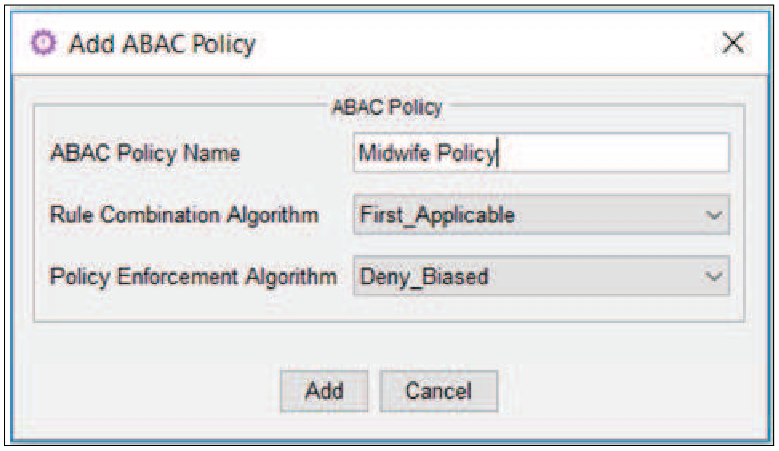


Figure 31: Right Click ABAC to Add a New ABAC Policy

7.8 ABAC Model Composition

After the composition of the attributes and conditions, an AC model can be generated with a set of AC policies. This section demonstrates the composition of an ABAC access control security model that consists of one or more policies with rules to specify the access permission and the correlated environments. For the Nursing Service example in Figure 1, the following description shows how to define the policies in Section 5.5.1.

Create a New ABAC Policy: The composition of a policy includes the operations to define the policy parameters with a set of rules. Navigating to "ABAC Model" in the project tree, the following steps show the creation of a policy:


Add ABAC Policy

ABAC Policy

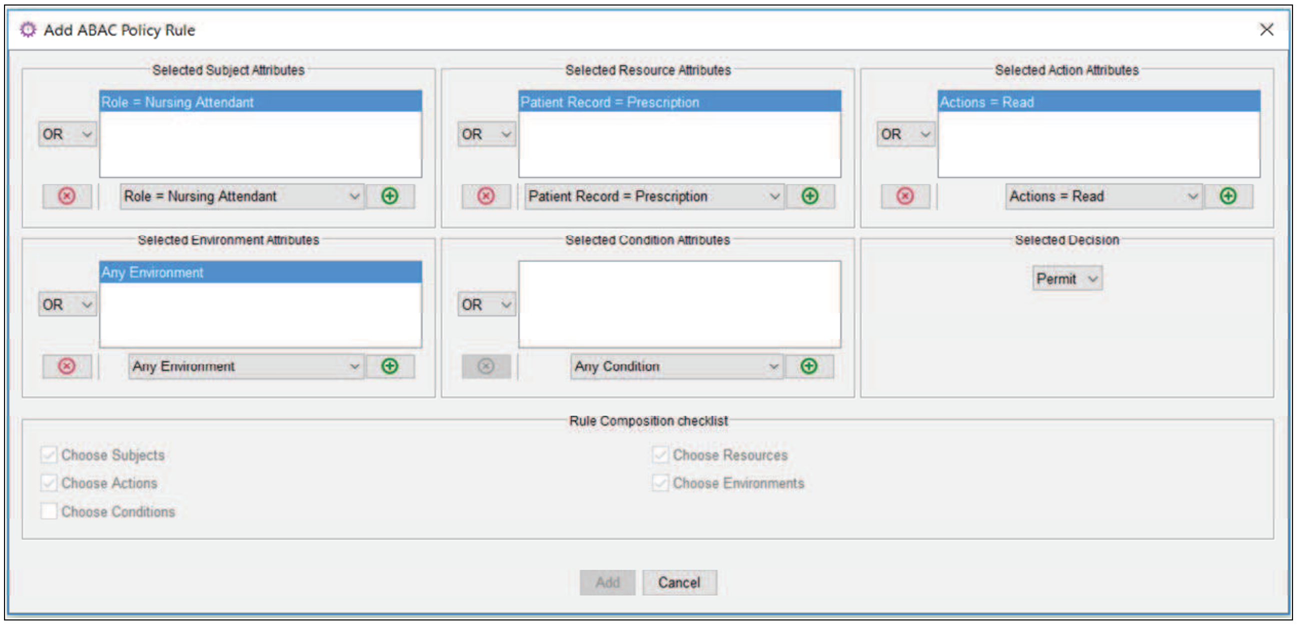
ABAC Policy Name: Midwife Policy

Rule Combination Algorithm: First_Applicable

Policy Enforcement Algorithm: Deny_Biased

Figure 32: Add ABAC Policy - Policy Algorithms

- Right click "ABAC" under "Model" in the project tree, then select "Add a New ABAC Policy", as shown in Figure 31,
- Name the ABAC policy, e.g., *Midwife Policy* as shown in Figure 32,


Add ABAC Policy Rule

Selected Subject Attributes: OR; Role = Nursing Attendant

Selected Resource Attributes: OR; Patient Record = Prescription

Selected Action Attributes: OR; Actions = Read

Selected Environment Attributes: OR; Any Environment

Selected Condition Attributes: OR; Any Condition

Selected Decision: Permit

Rule Composition checklist

- [✓] Choose Subjects
- [✓] Choose Resources
- [✓] Choose Actions
- [✓] Choose Environments
- [ ] Choose Conditions

Figure 33: User Interface to Add an Policy Rule

- Choose a "Rule Combination Algorithm" such as *First Applicable* as shown in Figure 32, and a "Policy Enforcement Algorithm" such as *Deny Based* as shown in Figure 32,

- Click “Add”, then
- An “Add ABAC Policy Rule” window pops out as shown in Figure 33. The composition of policy rule involves:
  - **Add Attributes**: From the GUI-based interface in Figure 33, it first selects a Subject attribute value (s), e.g., $Role = Nursing\ Attendent$, from the $Selected\ Subject\ Attributes$ dropdown menu. The subject dropdown menu lists all early composed Subject attributes and their values. Then, clicking the button $\oplus$ adds the subject attribute with a value. On the other hand, $\otimes$ allows you to remove a selected attribute. In a similar way, Resource attribute, action, environment, and condition values can be chosen for the rule composition. In a similar way, Resource, Action, Environmental attributes, and Condition are composed by choosing the attribute values and adding them into the rule. It is noted that all rule components including Subject, Resource, Action, Environment, Condition, and Decision are required to construct a complete policy rule. As shown in Figure 33, a “Rule Composition Checklist” verifies the integrity and the “And” button will be activated after all the rule elements are selected.
  - **Add Rule Decision**: As stated, a Decision (i.e., Permit or Deny) has to be chosen as the expected rule decision. It is noted that all rule components including Subject, Resource, Action, Environment, Condition, and Decision are required to construct a complete policy rule. As shown in Figure 33, a “Rule Composition Checklist” verifies the integrity and the “And” button will be activated only after all the rule elements are selected. Finally, click “And” to add this rule the policy.

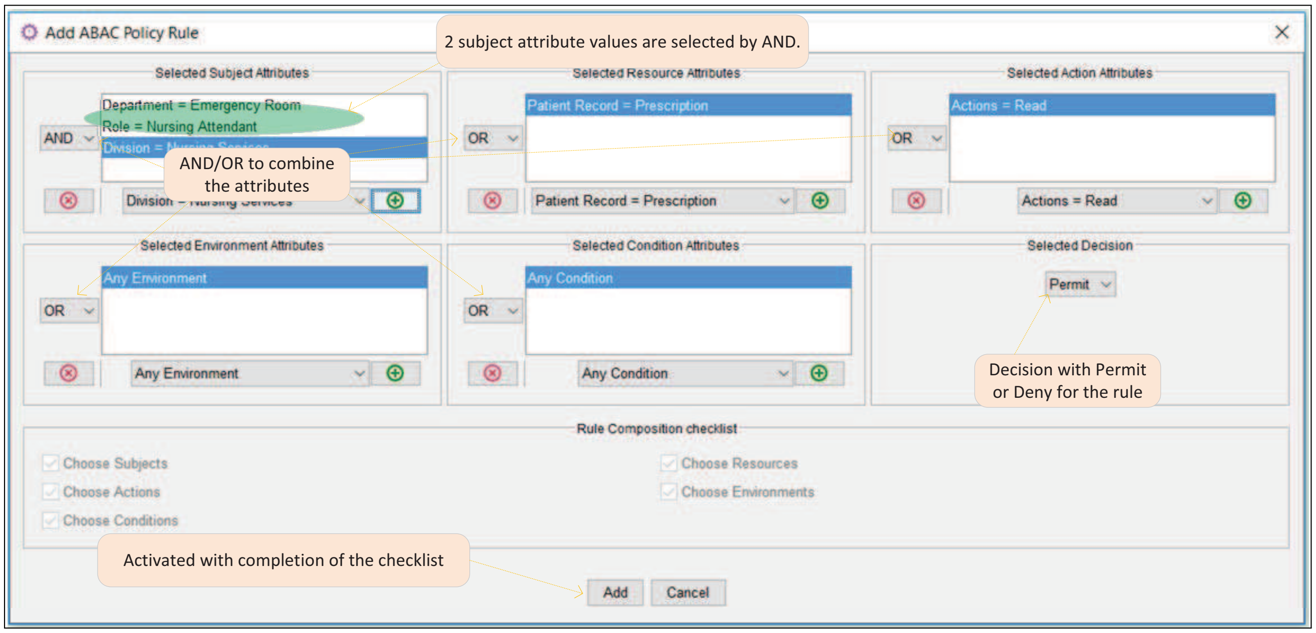
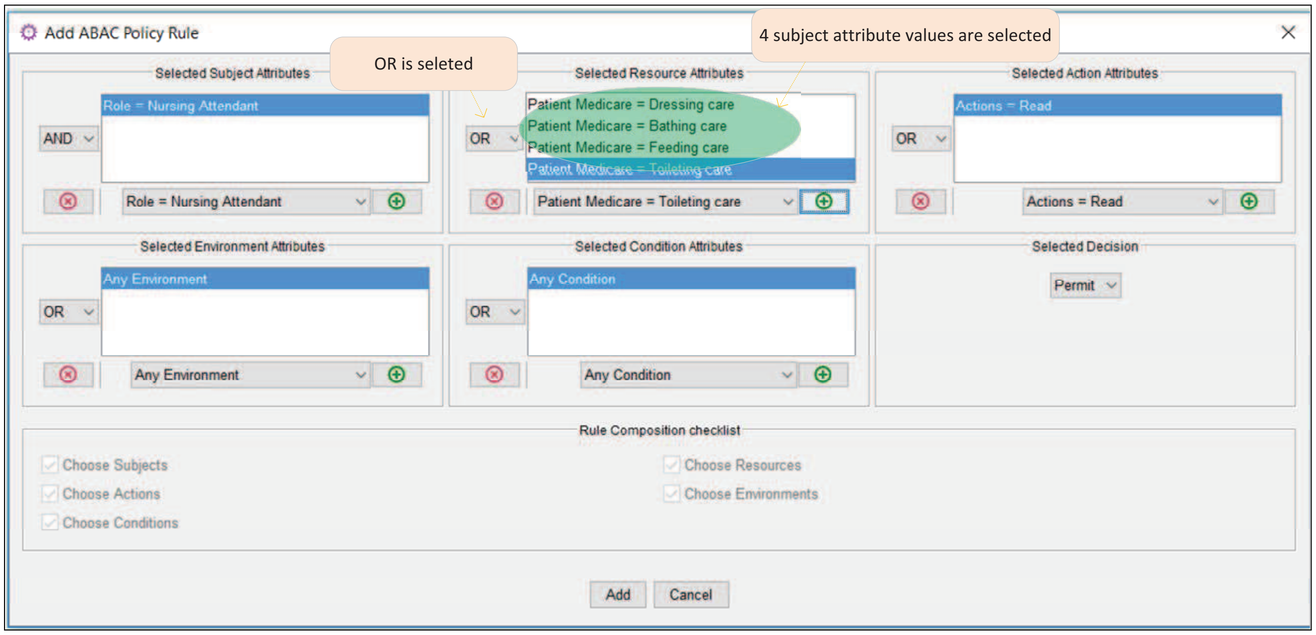

Selected Subject Attributes: AND — Department = Emergency Room; Role = Nursing Attendant; Division = Nursing Services
Selected Resource Attributes: OR — Patient Record = Prescription
Selected Action Attributes: OR — Actions = Read
Selected Environment Attributes: OR — Any Environment
Selected Condition Attributes: OR — Any Condition
Selected Decision: Permit
Add ABAC Policy Rule
2 subject attribute values are selected by AND.
AND/OR to combine the attributes
Decision with Permit or Deny for the rule
Rule Composition checklist
Choose Subjects
Choose Actions
Choose Conditions
Choose Resources
Choose Environments
Activated with completion of the checklist
Add
Cancel

Figure 34: Rule Composition with “AND” Operator

“AND” or “OR” Operators: Multiple attribute values can be selected from the dropdown menu for rule composition. Figure 34 shows two selected subject attribute values, e.g., $Subject$: $Department = Emergencyroom$ & $Role = Nursing\ Attendent$. If multiple attribute values are added to an attribute (e.g., Subject, Resource, or Environment), “AND” or “OR” operators will be applied for those attribute values in the different following ways.

“AND” Operator: “AND” Operator combines all the attribute values as an aggregated attribute value that means all the attribute values have to satisfied in a rule for the intended Decision. For example, when “AND” is applied to two Subject attribute values in Figure 34, $\mathcal{SPT}$ will generate the following rule:

$Subject : Department = Emergency\ Room;\ \&\ Role = Nursing\ Attendent;$
$Resource : Patient\ Record = Prescription;$
$Action : Read;$
$Decision : Permit$
where two subject attribute values are combined together as an aggregated one in <u>one rule</u>, which means that $Nursing\ Attendent$ from $Department = Emergency\ room$ has the $Read$ accessibility for Resource $Patient\ Record = Prescription$. As we can see, “AND” combines attribute values in one rule.



Selected Subject Attributes: AND — Role = Nursing Attendant

Selected Resource Attributes: OR — Patient Medicare = Dressing care; Patient Medicare = Bathing care; Patient Medicare = Feeding care; Patient Medicare = Toileting care

Selected Action Attributes: OR — Actions = Read

Selected Environment Attributes: OR — Any Environment

Selected Condition Attributes: OR — Any Condition

Selected Decision: Permit

Rule Composition checklist: ✓ Choose Subjects; ✓ Choose Resources; ✓ Choose Actions; ✓ Choose Environments; ✓ Choose Conditions

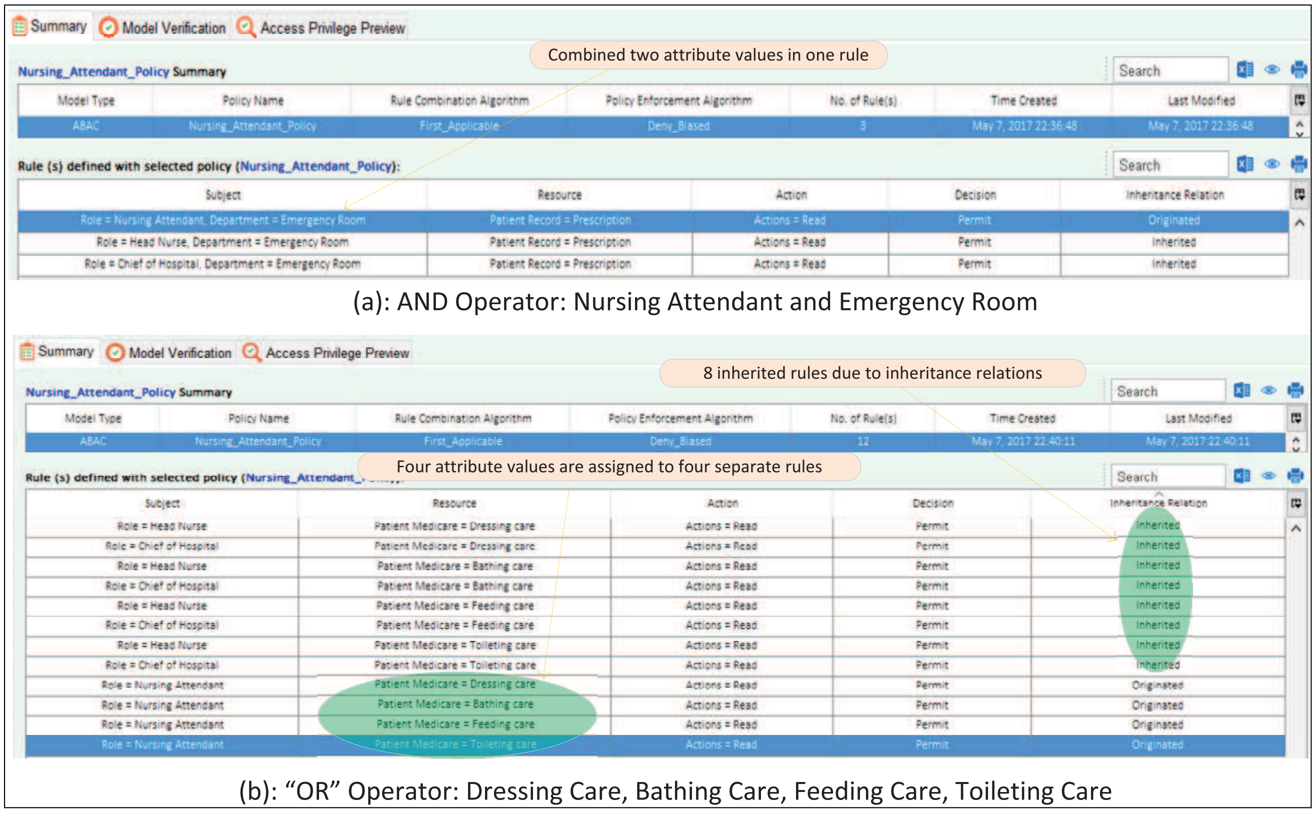
Figure 35: Rule Composition with “OR” Operator

“OR” Operator: “OR” Operator is different than “AND” Operator. Figure 35 shows four selected resource attribute values, e.g., $Patient\ Medicare = Dressing\ care$, $Patient\ Medicare = Bathing\ care$, $Patient\ Medicare = Feeding\ care$, and $PatientMedicare = Toileting\ care$. These resource attribute values are applied to “OR”, and in such a case, they will be individually considered in separate rules. As a result, $\mathcal{SPT}$ will generate multiple rules and these attribute values are separated in these rules. For the example in Figure 35, <u>four rules</u> will be generated as below:

1. $Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicare = Dressing\ care;$
$Action : Read; Decision : Permit$
2. $Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicare = Bathing\ care;$
$Action : Read; Decision : Permit$
3. $Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicare = Feeding\ care;$
$Action : Read; Decision : Permit$
4. $Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicare = Toileting\ care;$

$Action : Read; Decision : Permit$

where four attribute values are separately considered in four different rules, which means $Nursing\ Attendent$ has the $read$ accessibility for these four Resources.



Nursing_Attendant_Policy Summary

Model Type	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	Nursing_Attendant_Policy	First_Applicable	Deny_Biased	3	May 7, 2017 22:36:48	May 7, 2017 22:36:48

Rule (s) defined with selected policy (Nursing_Attendant_Policy):

Subject	Resource	Action	Decision	Inheritance Relation
Role = Nursing Attendant, Department = Emergency Room	Patient Record = Prescription	Actions = Read	Permit	Originated
Role = Head Nurse, Department = Emergency Room	Patient Record = Prescription	Actions = Read	Permit	Inherited
Role = Chief of Hospital, Department = Emergency Room	Patient Record = Prescription	Actions = Read	Permit	Inherited

(a): AND Operator: Nursing Attendant and Emergency Room

Summary
Model Verification
Access Privilege Preview
8 inherited rules due to inheritance relations
Four attribute values are assigned to four separate rules
Search

Nursing_Attendant_Policy Summary

Model Type	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	Nursing_Attendant_Policy	First_Applicable	Deny_Biased	12	May 7, 2017 22:40:11	May 7, 2017 22:40:11

Rule (s) defined with selected policy (Nursing_Attendant_

Subject	Resource	Action	Decision	Inheritance Relation
Role = Head Nurse	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited
Role = Chief of Hospital	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited
Role = Head Nurse	Patient Medicare = Bathing care	Actions = Read	Permit	Inherited
Role = Chief of Hospital	Patient Medicare = Bathing care	Actions = Read	Permit	Inherited
Role = Head Nurse	Patient Medicare = Feeding care	Actions = Read	Permit	Inherited
Role = Chief of Hospital	Patient Medicare = Feeding care	Actions = Read	Permit	Inherited
Role = Head Nurse	Patient Medicare = Toileting care	Actions = Read	Permit	Inherited
Role = Chief of Hospital	Patient Medicare = Toileting care	Actions = Read	Permit	Inherited
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Read	Permit	Originated
Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Read	Permit	Originated
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Read	Permit	Originated
Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Read	Permit	Originated

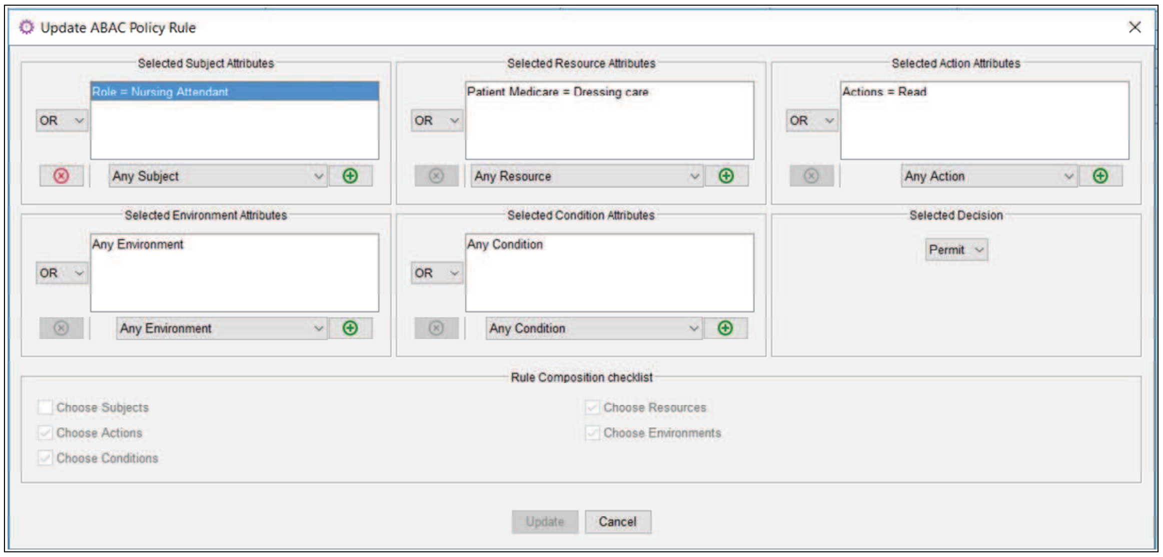
(b): "OR" Operator: Dressing Care, Bathing Care, Feeding Care, Toileting Care

Figure 36: Rules Generated for $Nursing\ Attendant$ Policy

Figure 36 compares the "AND" and "OR" results in the rule composition. Figure 36 (a) shows the composed role (i.e., the highlighted one) from Figure 34 where two attributes values are integrated in one rule. In addition to the edited rule, two inherited rules are automatically generated since the inheritance relation: $Chief\ of\ Hospital \rightarrow Head\ Nurse \rightarrow NursingAttendent$, which is composed in the Inheritance. Figure 36 (b) shows four rules using the "OR" Operator in Figure 35. It shows that the four attribute values are composed in four separate rules. Similarly, eight inherited rules are automatically generated since the inheritance relation: $Chief\ of\ Hospital \rightarrow Head\ Nurse \rightarrow Nursing\ Attendent$. It totally results in 12 rules with four originated rules and eight inherited rules in the example.

Add a New ABAC Rule: A new rule can be added to a policy by right clicking on the policy name in the project tree (e.g., the above $Nursing\ Attendant\ Policy$) while a blank "Add ABAC Policy Rule" window pops out as shown in Figure 33. After filling out the rule information and clicking the "Add" button by following the same steps as shown in Figure 33, the rule (s) will be added to the policy. In this way, a set of rules can be composed for the policy.

Update/Delete an ABAC Rule: A policy rule can be updated or deleted from a policy. Right click on the rule name in the project tree that you want to update or delete, tabs of "Update ABAC Rule" and "Delete" show up. Clicking on "Update ABAC rule" will show an "Update ABAC



Selected Subject Attributes: OR — Role = Nursing Attendant — Any Subject

Selected Resource Attributes: OR — Patient Medicare = Dressing care — Any Resource

Selected Action Attributes: OR — Actions = Read — Any Action

Selected Environment Attributes: OR — Any Environment — Any Environment

Selected Condition Attributes: OR — Any Condition — Any Condition

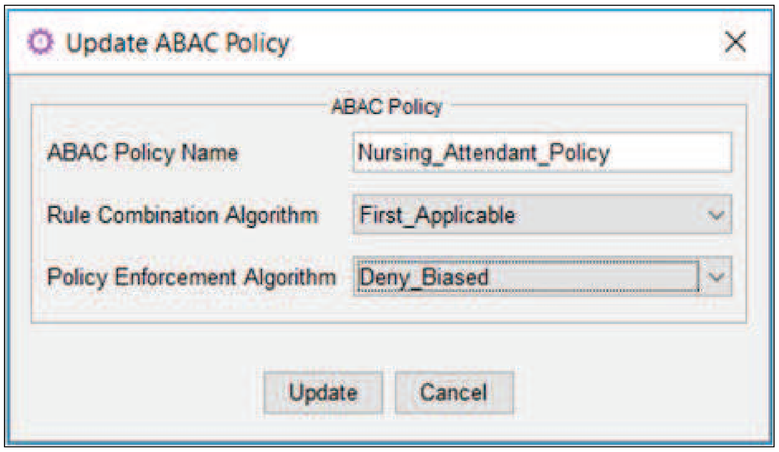
Selected Decision: Permit

Rule Composition checklist

- ☐ Choose Subjects
- ☑ Choose Resources
- ☑ Choose Actions
- ☑ Choose Environments
- ☑ Choose Conditions

Figure 37: Update an ABAC Policy Rule

Policy Rule" from which the policy rule can be modified, as shown in Figure 37. Clicking on the "Update" button to redefined the policy. On the other hand, clicking on "Delete" tab, the selected rule will be removed after clicking on "Yes" to confirm the deletion.



ABAC Policy

ABAC Policy Name	Nursing_Attendant_Policy
Rule Combination Algorithm	First_Applicable
Policy Enforcement Algorithm	Deny_Biased

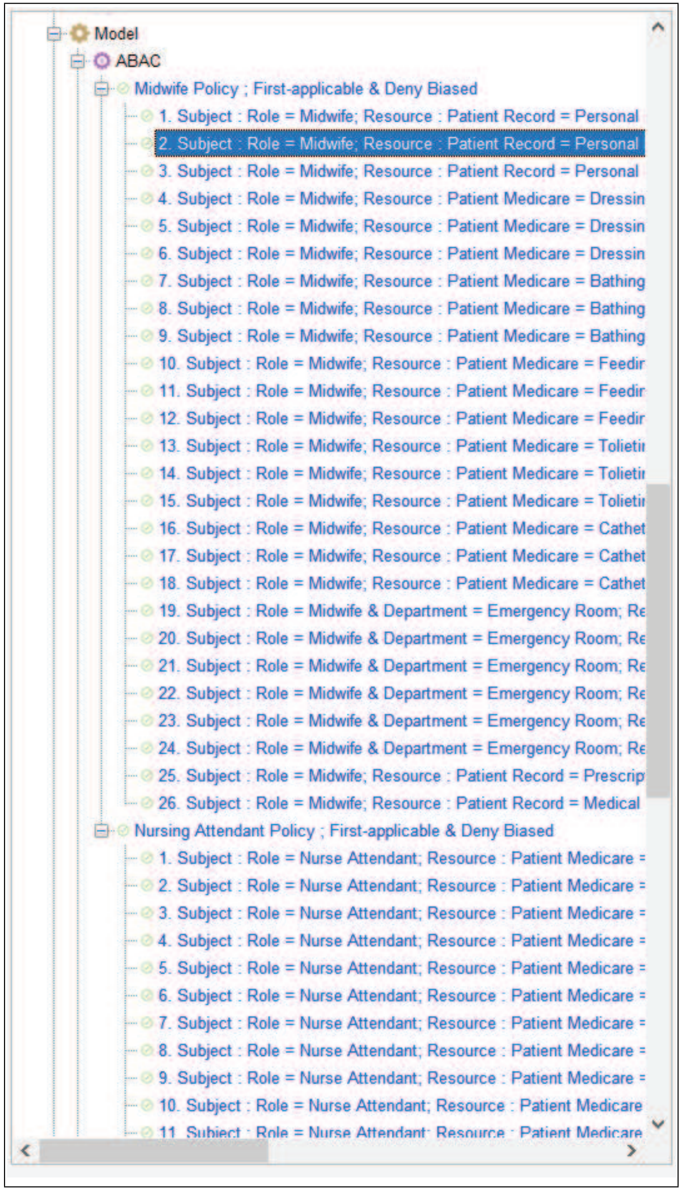
Figure 38: Update ABAC Policy

Add/Update a New ABAC Policy: An ABAC model could have more than one policy. Right click on the "ABAC", a tab of "Add a New ABAC Policy" shows up and click on it. "Add ABAC Policy" window pops up and it leads you to add a new policy with the same steps from Figure 31 to Figure 35. Clicking on an existing policy, a tab of "Update ABAC Policy" will lead you to modify the policy. The information of the Policy Name, Rule Combination Algorithm, and Policy Enforcement Algorithm can be modified and click on "Update" to save the modification, as shown in Figure 38.

Order the ABAC Rules: When a rule is defined, a sequence is assigned as an order of the rule in the policy. Figure 39 shows $Midwife\ Policy$ that has 26 rules and they are ordered in a sequence from 1 to 26. As some rule-combining algorithms, such as Ordered-deny-overrides, takes the order of the rules into account for the policy Decision evaluation. Decision of a policy could be different when it is tested under different rule orders in the case that the rule-combining algorithm is impacted by the rule sequence. For such a reason, $\mathcal{SPT}$ allows the adjustment of the sequence of the rules in a policy. The adjustment is accomplished by a Click to Drag-and-drop operation:

- **Click and Hold on a Policy Rule**: Giving a policy rule for adjustment, you can choose this rule by clicking and holding on it.
- **Drag-and-Drop the Chosen Policy Rule**: While holding the chosen rule, you can drag the policy and then drop it to a new sequence where you want to place.

For example, you can click and hold a rule 3 (i.e., in the sequence 3), drag and drop it to the sequence between 1 and 2. This will exchange the sequence of the rule 2 and rule 3.



```
Model
  ABAC
    Midwife Policy ; First-applicable & Deny Biased
      1. Subject : Role = Midwife; Resource : Patient Record = Personal
      2. Subject : Role = Midwife; Resource : Patient Record = Personal
      3. Subject : Role = Midwife; Resource : Patient Record = Personal
      4. Subject : Role = Midwife; Resource : Patient Medicare = Dressin
      5. Subject : Role = Midwife; Resource : Patient Medicare = Dressin
      6. Subject : Role = Midwife; Resource : Patient Medicare = Dressin
      7. Subject : Role = Midwife; Resource : Patient Medicare = Bathing
      8. Subject : Role = Midwife; Resource : Patient Medicare = Bathing
      9. Subject : Role = Midwife; Resource : Patient Medicare = Bathing
      10. Subject : Role = Midwife; Resource : Patient Medicare = Feedir
      11. Subject : Role = Midwife; Resource : Patient Medicare = Feedir
      12. Subject : Role = Midwife; Resource : Patient Medicare = Feedir
      13. Subject : Role = Midwife; Resource : Patient Medicare = Tolietir
      14. Subject : Role = Midwife; Resource : Patient Medicare = Tolietir
      15. Subject : Role = Midwife; Resource : Patient Medicare = Tolietir
      16. Subject : Role = Midwife; Resource : Patient Medicare = Cathet
      17. Subject : Role = Midwife; Resource : Patient Medicare = Cathet
      18. Subject : Role = Midwife; Resource : Patient Medicare = Cathet
      19. Subject : Role = Midwife & Department = Emergency Room; Re
      20. Subject : Role = Midwife & Department = Emergency Room; Re
      21. Subject : Role = Midwife & Department = Emergency Room; Re
      22. Subject : Role = Midwife & Department = Emergency Room; Re
      23. Subject : Role = Midwife & Department = Emergency Room; Re
      24. Subject : Role = Midwife & Department = Emergency Room; Re
      25. Subject : Role = Midwife; Resource : Patient Record = Prescrip
      26. Subject : Role = Midwife; Resource : Patient Record = Medical
    Nursing Attendant Policy ; First-applicable & Deny Biased
      1. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      2. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      3. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      4. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      5. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      6. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      7. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      8. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      9. Subject : Role = Nurse Attendant; Resource : Patient Medicare =
      10. Subject : Role = Nurse Attendant; Resource : Patient Medicare
      11. Subject : Role = Nurse Attendant; Resource : Patient Medicare
```

Figure 39: Rule Sequence

Note: A policy is not assigned with a sequence number when the policy is created. For applying order-related policy algorithm, the policies are ordered by the sequence of the selection and a Drag-to-reorder function is provided to adjust the sequence.

7.9 Multilevel Security Model Composition

A MultiLevel Security Model as illustrated in Subsection 5.5.2 can be composed by first clicking the "Multilevel" under "Model" in the project tree. For this model, one or more policies can be defined for multilevel access control. Upon the right click of "Multilevel" and the selection of "Add a New Multilevel Policy" from the menu, a rule composition window namely, "Add Multilevel Policy Rule", will pop up as shown in Figure 40. Four steps are shown in Figure 40 for composing

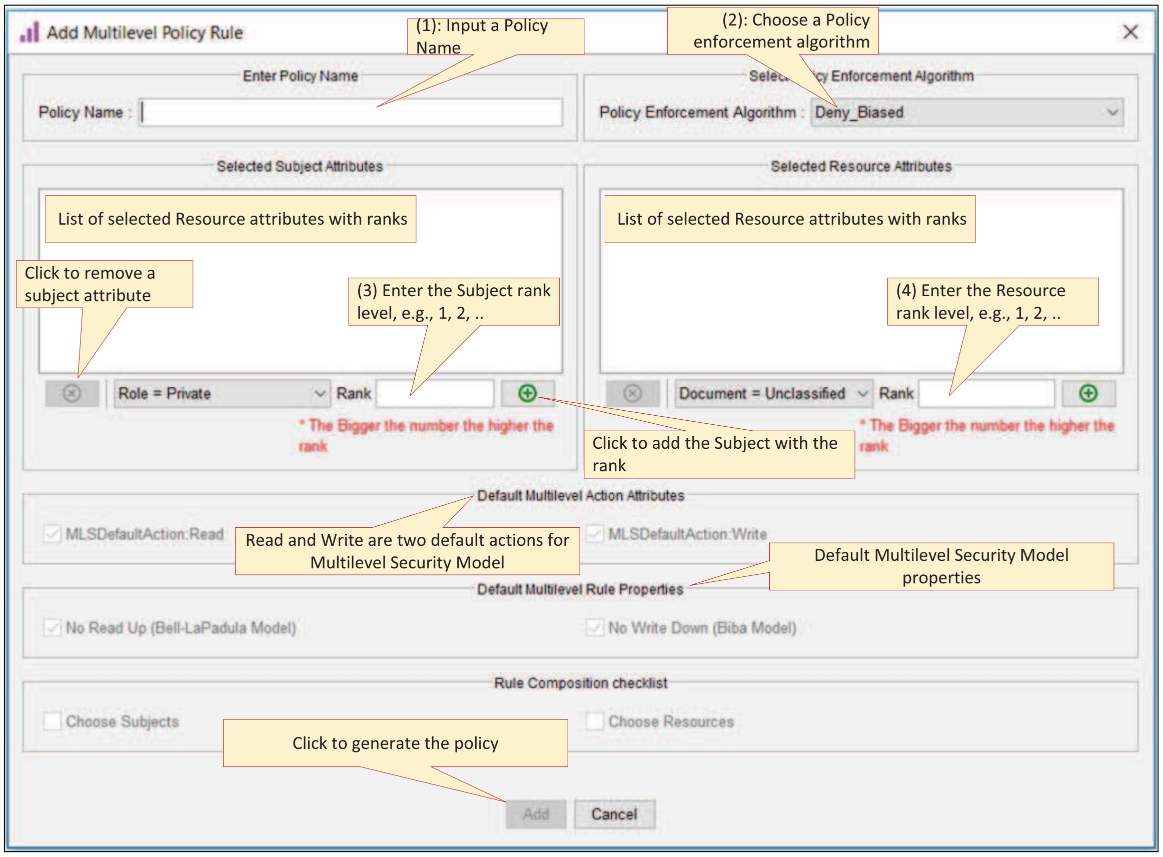
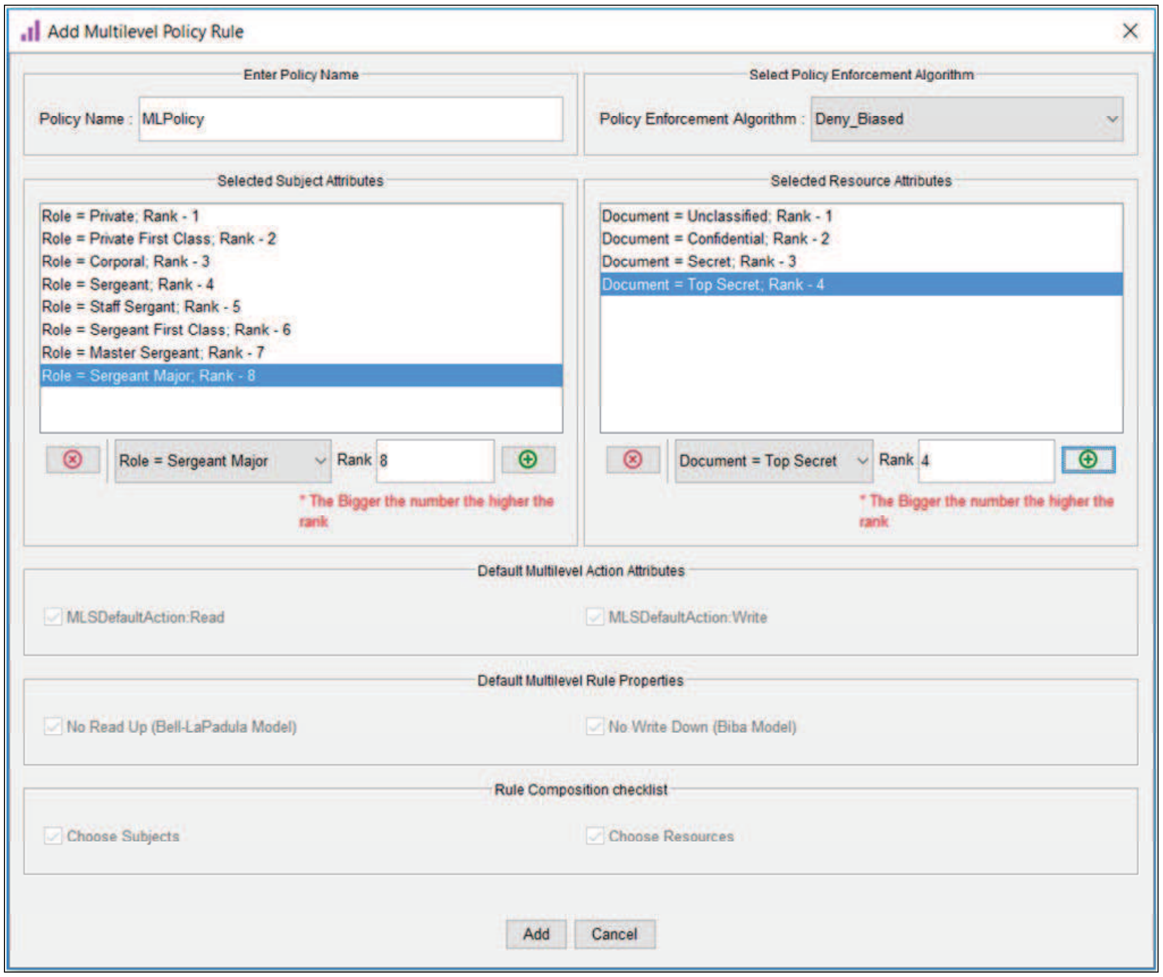

Add Multilevel Policy Rule
(1): Input a Policy Name
(2): Choose a Policy enforcement algorithm
Enter Policy Name
Policy Name :
Select Policy Enforcement Algorithm
Policy Enforcement Algorithm : Deny_Biased
Selected Subject Attributes
List of selected Resource attributes with ranks
Click to remove a subject attribute
(3) Enter the Subject rank level, e.g., 1, 2, ..
Role = Private
Rank
* The Bigger the number the higher the rank
Selected Resource Attributes
List of selected Resource attributes with ranks
(4) Enter the Resource rank level, e.g., 1, 2, ..
Document = Unclassified
Rank
* The Bigger the number the higher the rank
Click to add the Subject with the rank
Default Multilevel Action Attributes
MLSDefaultAction:Read
Read and Write are two default actions for Multilevel Security Model
MLSDefaultAction:Write
Default Multilevel Security Model properties
Default Multilevel Rule Properties
No Read Up (Bell-LaPadula Model)
No Write Down (Biba Model)
Rule Composition checklist
Choose Subjects
Click to generate the policy
Choose Resources
Add
Cancel

Figure 40: "MultiLevel" Security Model Composition

a policy. The first step is to define a policy name in the input box of "Policy Name". The second step is to choose a policy algorithm from the dropdown menu of "Policy Enforcement Algorithm". The third step is to compose the Subject attributes and specify their security rank, i.e., $1, 2, \cdots$, where the bigger the number the higher the rank. The selected subject attribute value with its rank is added to the list of the "Selected Subject Attributes" by clicking on ⊕ button. Meanwhile, you can remove an attribute value from the list using ⊗ button. The next step is similar which is to compose the Resource attributes as well as the rank. In the end, clicking "Add" will add the policy into the Multilevel AC model.

Figure 41 showcases an example for MultiLevel Security Model composition. The policy is named as *MLPolicy* as shown in Figure 41. In this example, eight Subject attribute values are specified with different levels $(1-8)$, e.g., $Private$ is a Role attribute value in rank 1. It is worth mentioning that the *Role* attribute and its values are all composed before the model composition, which uses the method as illustrated in Subsection 7.5.1. Figure 41 shows the Resource attribute values with ranks. As shown in Figure 41, the document is classified into four ranks, i.e., $1-Unclassified$, $2-Classified$, $3-Secret$, $4-Top\ Secret$. The higher rank represents a higher security level. Clicking on the "Add" button generates the multilevel policy.

A multilevel policy is built by admitting the Multilevel Security Model described in Subsection 5.5.2. Decision for each rule will be automatically generated according to Bell-Lapadula model and Biba Model. Due to this, there are no option (e.g., "Selected Decision" as shown in Figure 33) to choose a Decision (i.e., Permit or Deny). Giving the example in Figure 41, clicking on $MLPolicy$


Add Multilevel Policy Rule

Enter Policy Name

Policy Name : MLPolicy

Select Policy Enforcement Algorithm

Policy Enforcement Algorithm : Deny_Biased

Selected Subject Attributes

Role = Private; Rank - 1
Role = Private First Class; Rank - 2
Role = Corporal; Rank - 3
Role = Sergeant; Rank - 4
Role = Staff Sergant; Rank - 5
Role = Sergeant First Class; Rank - 6
Role = Master Sergeant; Rank - 7
Role = Sergeant Major; Rank - 8

Role = Sergeant Major Rank 8

* The Bigger the number the higher the rank

Selected Resource Attributes

Document = Unclassified; Rank - 1
Document = Confidential; Rank - 2
Document = Secret; Rank - 3
Document = Top Secret; Rank - 4

Document = Top Secret Rank 4

* The Bigger the number the higher the rank

Default Multilevel Action Attributes

MLSDefaultAction:Read
MLSDefaultAction:Write

Default Multilevel Rule Properties

No Read Up (Bell-LaPadula Model)
No Write Down (Biba Model)

Rule Composition checklist

Choose Subjects
Choose Resources

Figure 41: A Example of MultiLevel Policy Composition

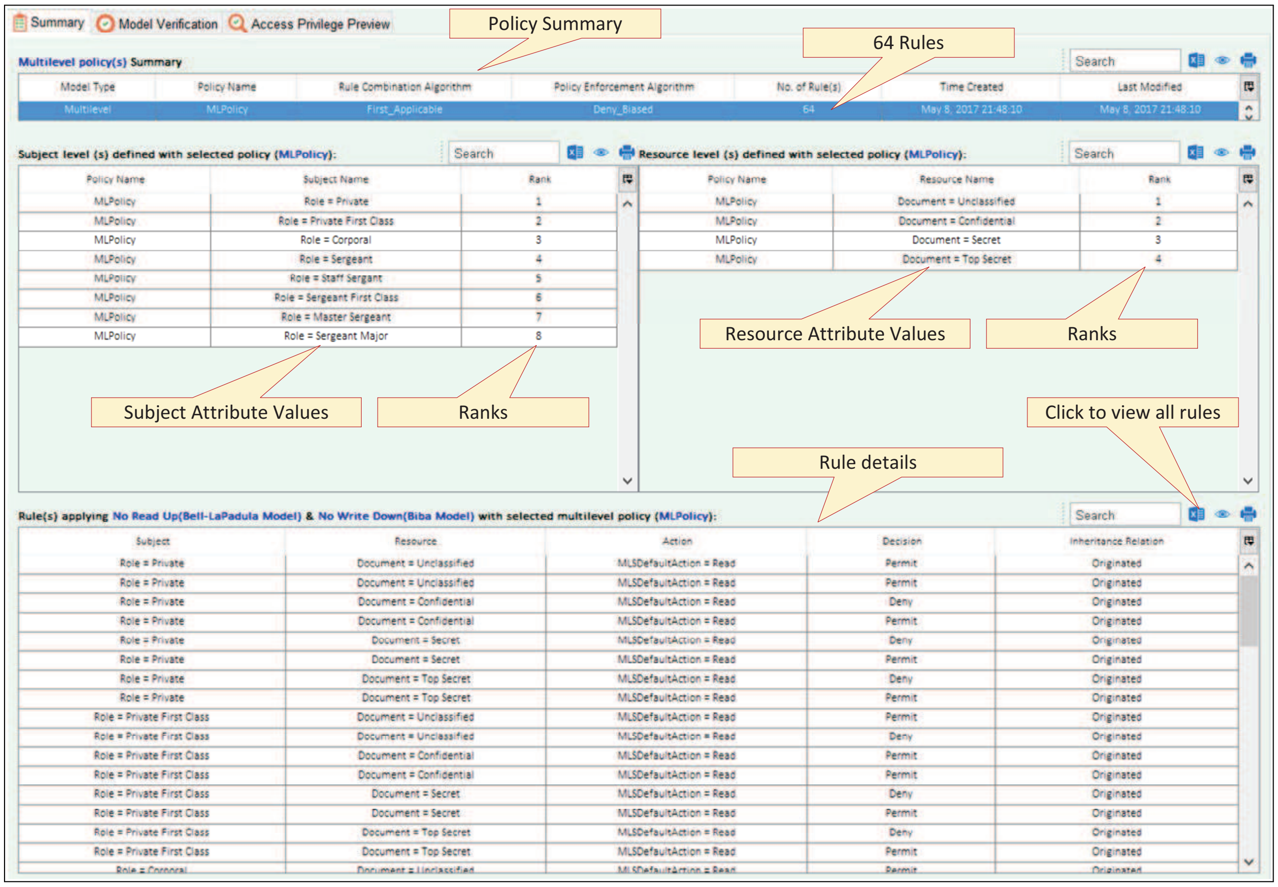
will navigate to the policy summary. Figure 42 shows the $MLPolicy$ summary composed from Figure 41. The summary lists the Subject and Resource attribute values and the ranks. It shows that there are 64 rules that are totally composed. This is because the Subject attribute $Role$ has defined 8 attribute values and the Resource attribute $Document$ has 4 ranked values in Figure 41. Therefore, the total number of multilevel rules is:

$$8 \times 4 \times 2 = 64 \tag{9}$$

where 2 represents $Read$ and $Write$ operations in the Bell-Lapadula and Biba models, respectively.

In addition to multilevel Subject and Resource values, Figure 42 lists the $MLPolicy$ rules with the detailed rule information, i.e., information of the Subject, Resource, Actions, Decisions, and Inheritance Relations. The detailed rules can be viewed from Excel or a separate page. They also could be printed if a printer is installed in the Computer. All the policy rules are also listed under the "MLPolicy" in the policy editing zone. Instead of one policy, a Multilevel Security Model could have multiple policies and each policy has its own attributes with ranks.

Update/Delete Multilevel Policy: It is worth mentioning that it is not allowable to individually update/delete a multilevel policy rule as the editing of a rule may infringe the policy integrity, imposed by the Multilevel Security Model. Due to this, the updating/deleting functions for individual



Multilevel policy(s) Summary

Model Type	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
Multilevel	MLPolicy	First_Applicable	Deny_Biased	64	May 8, 2017 21:48:10	May 8, 2017 21:48:10

Subject level (s) defined with selected policy (MLPolicy):

Policy Name	Subject Name	Rank
MLPolicy	Role = Private	1
MLPolicy	Role = Private First Class	2
MLPolicy	Role = Corporal	3
MLPolicy	Role = Sergeant	4
MLPolicy	Role = Staff Sergant	5
MLPolicy	Role = Sergeant First Class	6
MLPolicy	Role = Master Sergeant	7
MLPolicy	Role = Sergeant Major	8

Resource level (s) defined with selected policy (MLPolicy):

Policy Name	Resource Name	Rank
MLPolicy	Document = Unclassified	1
MLPolicy	Document = Confidential	2
MLPolicy	Document = Secret	3
MLPolicy	Document = Top Secret	4

Rule(s) applying No Read Up(Bell-LaPadula Model) & No Write Down(Biba Model) with selected multilevel policy (MLPolicy):

Subject	Resource	Action	Decision	Inheritance Relation
Role = Private	Document = Unclassified	MLSDefaultAction = Read	Permit	Originated
Role = Private	Document = Unclassified	MLSDefaultAction = Read	Permit	Originated
Role = Private	Document = Confidential	MLSDefaultAction = Read	Deny	Originated
Role = Private	Document = Confidential	MLSDefaultAction = Read	Permit	Originated
Role = Private	Document = Secret	MLSDefaultAction = Read	Deny	Originated
Role = Private	Document = Secret	MLSDefaultAction = Read	Permit	Originated
Role = Private	Document = Top Secret	MLSDefaultAction = Read	Deny	Originated
Role = Private	Document = Top Secret	MLSDefaultAction = Read	Permit	Originated
Role = Private First Class	Document = Unclassified	MLSDefaultAction = Read	Permit	Originated
Role = Private First Class	Document = Unclassified	MLSDefaultAction = Read	Deny	Originated
Role = Private First Class	Document = Confidential	MLSDefaultAction = Read	Permit	Originated
Role = Private First Class	Document = Confidential	MLSDefaultAction = Read	Permit	Originated
Role = Private First Class	Document = Secret	MLSDefaultAction = Read	Deny	Originated
Role = Private First Class	Document = Secret	MLSDefaultAction = Read	Permit	Originated
Role = Private First Class	Document = Top Secret	MLSDefaultAction = Read	Deny	Originated
Role = Private First Class	Document = Top Secret	MLSDefaultAction = Read	Permit	Originated
Role = Corporal	Document = Unclassified	MLSDefaultAction = Read	Permit	Originated

Figure 42: $MLPolicy$ Summary for the Example in Figure 41

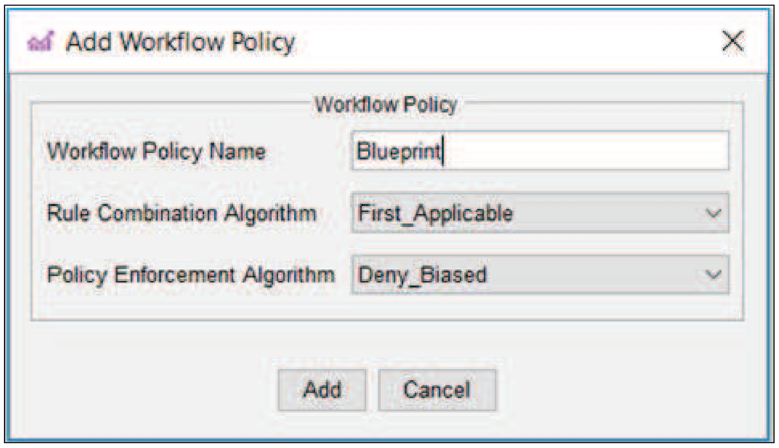
rules are disabled, i.e., no operation by right clicking any rules under "MLPolicy" in the policy tree. This prevents the infringement of the integrity of the Multilevel Security Model. On the other hand, a composed multilevel policy can be updated as a whole and the revision will still admit to the Multilevel Security Model in Subsection 5.5.2. Updating multilevel policy is performed by:

- **Navigating to Multilevel Policy**: Navigate to "Multilevel" and right click the multilevel policy, e.g., $MLPolicy$, that needs to be updated. It then shows an option of "Update Multilevel Policy" and you click it. An "Update Multilevel Policy Rule" window will show up. The "Update Multilevel Policy Rule" window is similar to Figure 41 with the filling of the previously configured policy information.

- **Modify the Multilevel Policy**: "Update Multilevel Policy Rule" window allows you to edit all the policy information, e.g., clicking ⊗ to remove a selected attribute value and ⊕ to add new an attribute value with a rank.

- **Update**: Click "Update" to finish the operation.

Further, a composed MultiLevel Policy can be removed. This can be done by right-clicking the policy (e.g., $MLPolicy$ in the project tree) that needs to be deleted. Click "Delete", and then click "Yes" to confirm the deletion. Along with the deletion of the policy, all the rules will be automatically deleted.


Add Workflow Policy

Workflow Policy

Workflow Policy Name: Blueprint

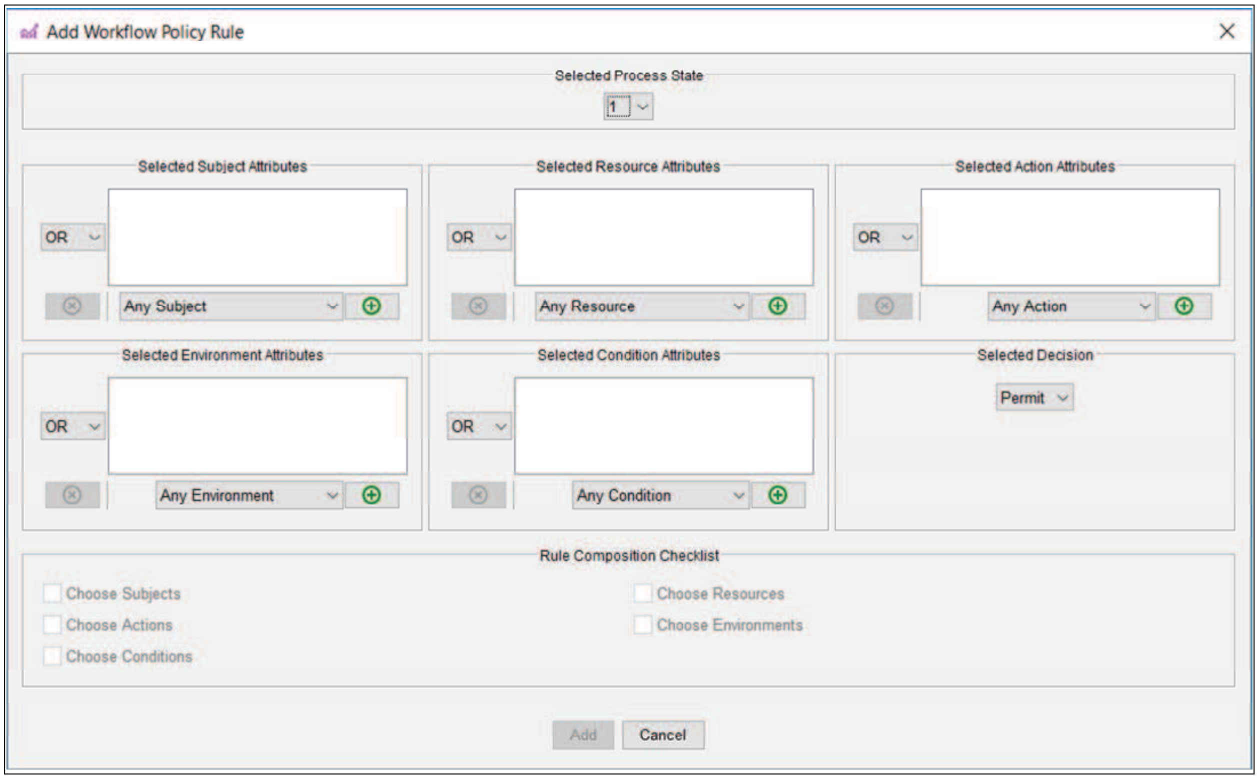
Rule Combination Algorithm: First_Applicable

Policy Enforcement Algorithm: Deny_Biased

Figure 43: Interface to Add a Workflow Policy

7.10 Workflow Model

A Workflow Security Model as illustrated in Subsection 5.5.3 can be composed by first clicking the ”Workflow” under “Model” in the project tree. For this model, each policy is defined as an association of a process state. The accomplishment of the process state acts as an extra condition for the process in the next state.


Add Workflow Policy Rule

Selected Process State: 1

Selected Subject Attributes: OR, Any Subject

Selected Resource Attributes: OR, Any Resource

Selected Action Attributes: OR, Any Action

Selected Environment Attributes: OR, Any Environment

Selected Condition Attributes: OR, Any Condition

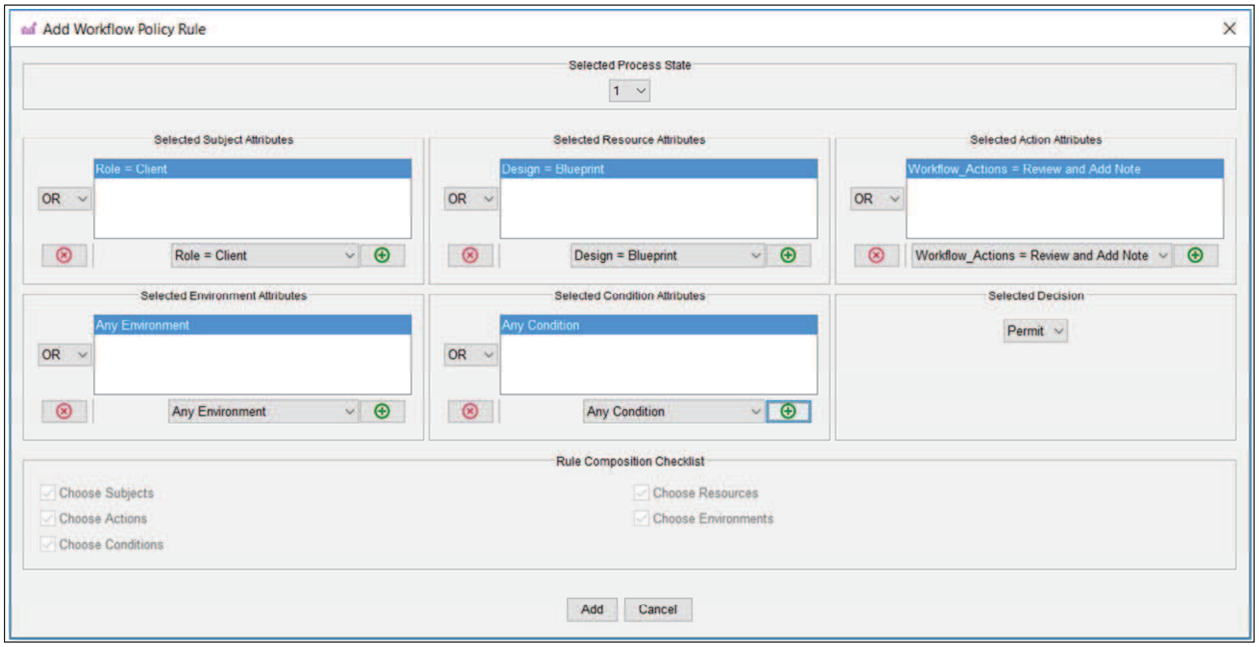
Selected Decision: Permit

Rule Composition Checklist

- Choose Subjects
- Choose Resources
- Choose Actions
- Choose Environments
- Choose Conditions

Figure 44: Interface to Add Rule for a Workflow Policy

Upon the right click of “Workflow”, “Add a New Workflow Policy” will show up and clicking it will result in a window of “Add Workflow Policy” as shown in Figure 43. This window is to give a name for the policy (e.g., Blueprint in Figure 43) and choose a Rule Combination Algorithm and Policy Enforcement algorithms from drop-down menus. The policy will be added after clicking on the “Add” button in the “Add Workflow Policy” window. Hereafter, “Add Workflow Policy Rule” window is popped out for rule composition, as shown in Figure 44. Different than ABAC and Multilevel Security Models, a rule for a Workflow policy has to choose a Process State (i.e., $1, 2, \cdots$) that indicates the rule has to be performed in such a state before moving to the next Process State.


Add Workflow Policy Rule

Selected Process State: 1

Selected Subject Attributes: OR; Role = Client

Selected Resource Attributes: OR; Design = Blueprint

Selected Action Attributes: OR; Workflow_Actions = Review and Add Note

Selected Environment Attributes: OR; Any Environment

Selected Condition Attributes: OR; Any Condition

Selected Decision: Permit

Rule Composition Checklist

✓ Choose Subjects
✓ Choose Resources
✓ Choose Actions
✓ Choose Environments
✓ Choose Conditions

Figure 45: Rule Composition for a Blueprint Policy

Figure 45 shows the rule composition of the Blueprint example that is illustrated in Subsection 5.5.3. It provides $\oplus$ and $\otimes$ buttons to add an attribute value from the dropdown menu or remove a attribute value. It is noted that all the attributes are composed in prior by following the operations as discussed in Subsection 7.5.1 and we ignore the illustration of these steps in this example. The “ADD” and “OR” operators have the same logic combination of two or more attribute value in rule composition. By clicking the “Add” button, Figure 45 will generate the following rule:

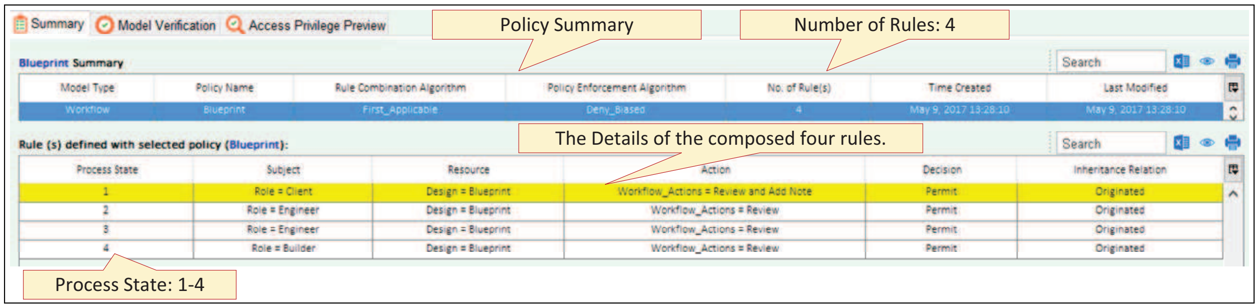
$ProcessState = 1; Subject : Role = Client; Resource; Design = Blueprint;$
$Workflow\ Action = Review\ and\ Add\ Node; Decision : Permit$

After the creation of the first rule, one or more rules can be composed in a similar way. It first right clicks the “Policy Name” under “Workflow” in the project tree and it shows a tab of “Add a New Workflow Rule”. Clicking “Add a New Workflow Rule”, “Add Workflow Policy Rule” as shown in Figure 44 will pop up such that a new rule can be composed. For the $Blueprint$ example in Subsection 5.5.3, the other three rules are:

$ProcessState = 2; Subject : Role = Engineer; Resource : Design = Blueprint;$
$Workflow\ Action = Review\ and\ Add\ Node; Decision : Permit$

$ProcessState = 3; Subject : Role = Engineer; Resource : Design = Blueprint;$
$Workflow\ Action = Review; Decision : Permit$

$ProcessState = 4; Subject : Role = Builder; Resource : Design = Blueprint;$
$Workflow\ Action = Review; Decision : Permit$



Blueprint Summary

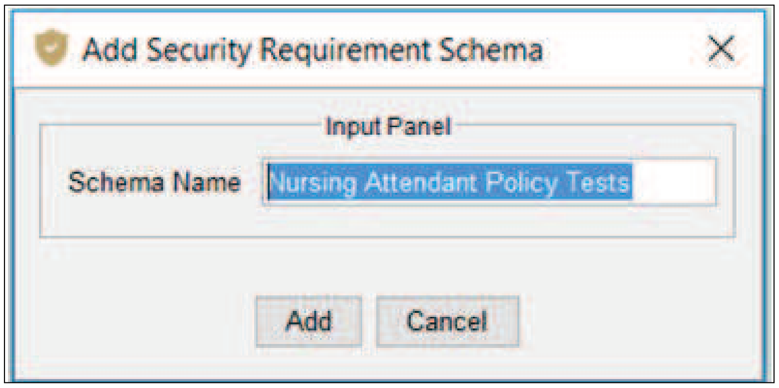
Model Type	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
Workflow	Blueprint	First_Applicable	Deny_Biased	4	May 9, 2017 13:28:10	May 9, 2017 13:28:10

Rule (s) defined with selected policy (Blueprint):

Process State	Subject	Resource	Action	Decision	Inheritance Relation
1	Role = Client	Design = Blueprint	Workflow_Actions = Review and Add Note	Permit	Originated
2	Role = Engineer	Design = Blueprint	Workflow_Actions = Review	Permit	Originated
3	Role = Engineer	Design = Blueprint	Workflow_Actions = Review	Permit	Originated
4	Role = Builder	Design = Blueprint	Workflow_Actions = Review	Permit	Originated

Figure 46: Blueprint Workflow Rules

Figure 44 shows the summary of the composed *Blueprint* policy which has four rules. The policy and rule information is also displayed in the policy tree. A workflow could have more than one policies and the policy composition have the same process as discussed above. Meanwhile, the policy can be updated to revise the policy, such as changing the policy name, rule combination algorithm, or policy enforcement algorithm. The policy rule can be updated with revised process state, attribute names, etc. Meanwhile, a policy rule can be removed by "Delete" from the options by right-clicking the rule under the policy (e.g., "Blueprint") in the policy editing zone. Similarly, a policy can be deleted which will remove all the rules under this policy.



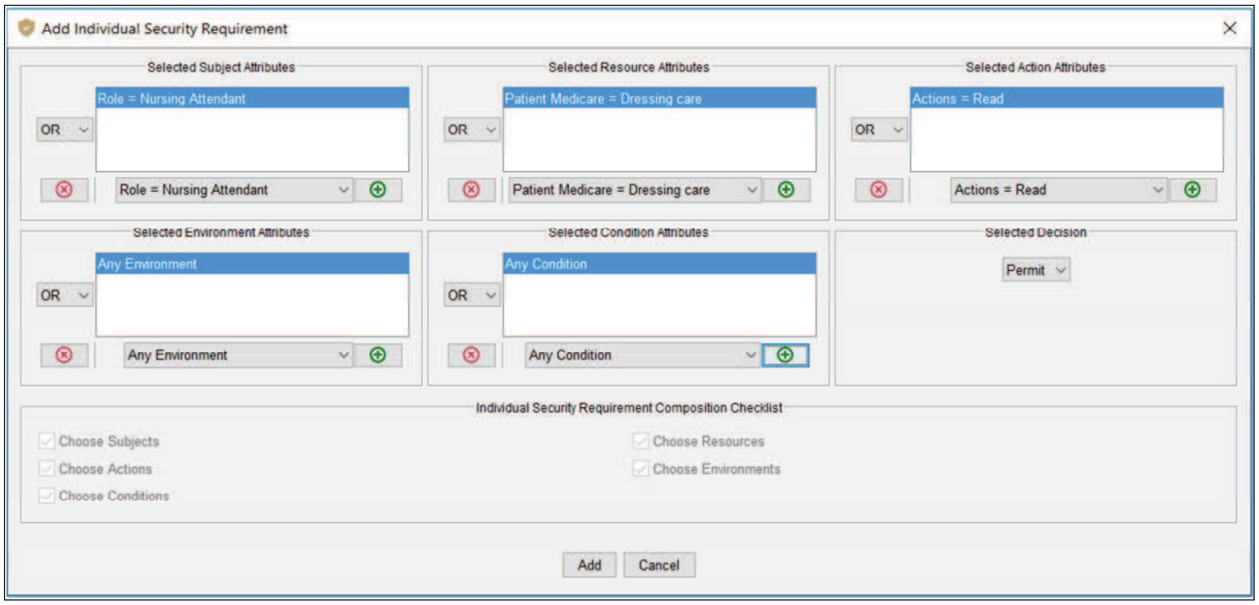
Schema Name: Nursing Attendant Policy Tests

Figure 47: Create a Security Requirement Schema

7.11 Access Control Security Requirement

The composed policies in each AC model are tested and analyzed with the use of AC Security Requirements. AC Security Requirements are the security request cases (i.e., acting as security request in the real AC system) that are used to test the AC effectiveness against the composed AC policy. They specifically verify if the composed AC policies can achieve the intended AC results

(Permit or Deny) or not. This checks whether there are AC flaws in the policies or not. Please refer to Subsection 6.2 for the AC Security Requirement background knowledge. This section illustrates how AC security requirements are composed. $\mathcal{SPT}$ supports the composition of (i) Individual Security Requirement, (ii) Separation of Duty Security Requirement, and (iii) Combinatorial Test Suites, as illustrated in the following subsections.


Add Individual Security Requirement

Selected Subject Attributes: OR; Role = Nursing Attendant

Selected Resource Attributes: OR; Patient Medicare = Dressing care

Selected Action Attributes: OR; Actions = Read

Selected Environment Attributes: OR; Any Environment

Selected Condition Attributes: OR; Any Condition

Selected Decision: Permit

Individual Security Requirement Composition Checklist
- ✓ Choose Subjects
- ✓ Choose Actions
- ✓ Choose Conditions
- ✓ Choose Resources
- ✓ Choose Environments

Figure 48: Create an AC Security Requirement

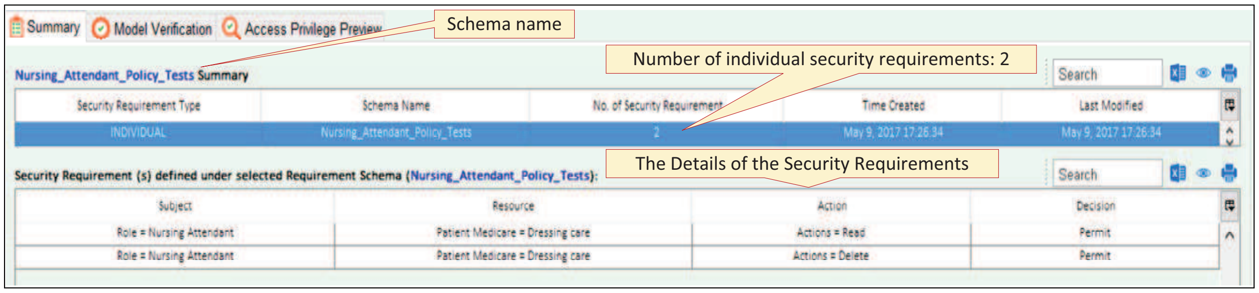
7.11.1 Individual Security Requirement

$\mathcal{SPT}$ uses the concept of Schema to organize the individual security requirements into a group. A schema can be regarded as a name for a group of the AC Security Requirements. A schema is generated by right click the “Individual Security Requirement” under “Access Control Security Requirement” in the project tree and then hitting “Add a New Security Requirement Schema”, which will result in the window in Figure 47. Using the Nursing Service as an example, a schema is named as $Nursing\ Attendant\ Policy\ Tests$, which is shown in Figure 47. After creating the schema, an “Add Individual Security Requirement” window, as shown in in Figure 48, pops out and from this window, an AC security requirement can be composed. Filling out the attributes and selecting other parameters as shown in Figure 48, the following statement of AC Security Requirement is generated:

$Subject : Role = Nursing\ Attendant; Resource : Patient\ Medicare = Dressing\ Care;$
$Action = Read; Decision : Permit$

The above AC Security Requirement is to verify if it is TRUE or FALSE that an $Nursing\ Attendant$ is permitted to $read$ the patient resource of $Dressing\ Care$. The above AC Security Requirement can be tested against all the composed ABAC policy (e.g., a policy with the rules in Figure 36) with a testing method. Similarly, a number of AC Security Requirements can be composed for the $Nursing\ Attendant\ Policy\ Tests$ schema. For example, the following Security Requirement can be generated by selecting a $Delete$ Action:

$Subject : Role = Nursing\ Attendant; Resource : Patient\ Medicare = Dressing\ Care;$
$Action = Delete; Decision : Permit$


Summary
Model Verification
Access Privilege Preview
Schema name
Number of individual security requirements: 2
The Details of the Security Requirements
Search

Nursing_Attendant_Policy_Tests Summary

Security Requirement Type	Schema Name	No. of Security Requirement	Time Created	Last Modified
INDIVIDUAL	Nursing_Attendant_Policy_Tests	2	May 9, 2017 17:26:34	May 9, 2017 17:26:34

Security Requirement (s) defined under selected Requirement Schema (Nursing_Attendant_Policy_Tests):

Subject	Resource	Action	Decision
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Read	Permit
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Permit

Figure 49: Summary of the two Generated AC Security Requirement

Clicking on the $Nursing\ Attendant\ Policy\ Tests$ Schema will be navigated to the schema Security Requirement summary. Figure 50 shows the schema information (e.g., Security Requirement Type, Schema Name, Number of Security Requirements,) and the details of each generated AC security requirements.

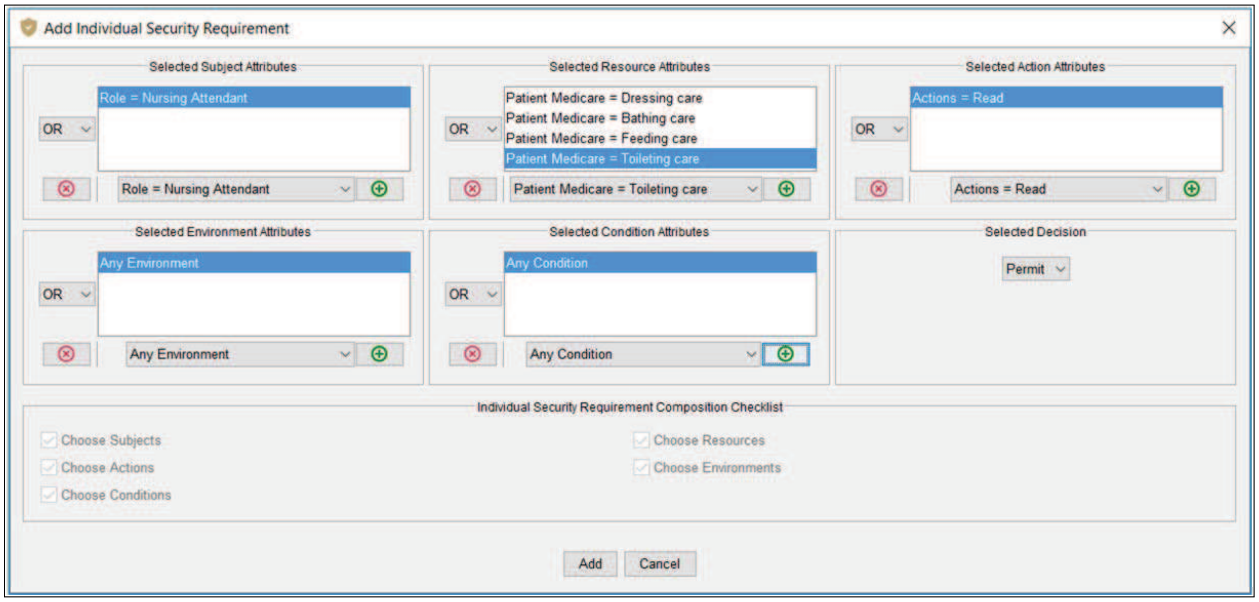
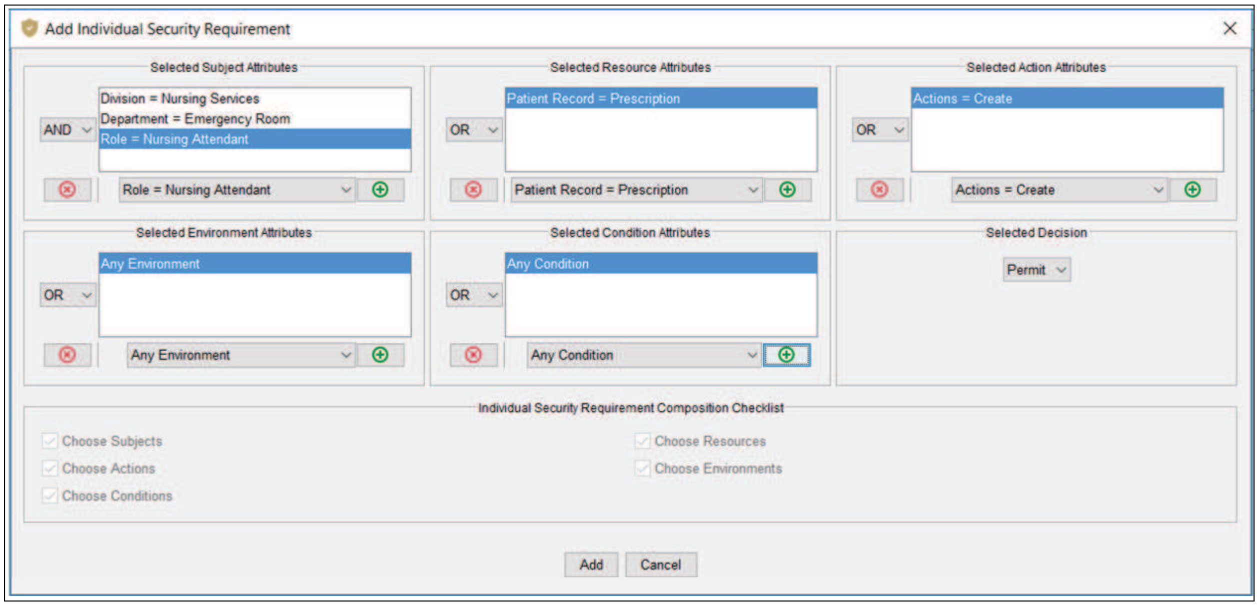

Add Individual Security Requirement
Selected Subject Attributes
OR
Role = Nursing Attendant
Role = Nursing Attendant
Selected Resource Attributes
OR
Patient Medicare = Dressing care
Patient Medicare = Bathing care
Patient Medicare = Feeding care
Patient Medicare = Toileting care
Patient Medicare = Toileting care
Selected Action Attributes
OR
Actions = Read
Actions = Read
Selected Environment Attributes
OR
Any Environment
Any Environment
Selected Condition Attributes
OR
Any Condition
Any Condition
Selected Decision
Permit
Individual Security Requirement Composition Checklist
Choose Subjects
Choose Resources
Choose Actions
Choose Environments
Choose Conditions
Add
Cancel

Figure 50: "OR" to Create Multiple Security Requirement

Multiple Security Requirements can be added at one time from the "Add Individual Security Requirement" window by selecting "OR" to combine multiple attributes. Figure 50 shows that $Patient\ Medicare = Dressing\ care$, $Patient\ Medicare = Bathing\ care$, $Patient\ Medicare = Feeding\ care$, and $PatientMedicare = Toileting\ care$ can all added by "OR" operator in the Resource attributes such that four Security Requirements will be generated. If you further choose two action attributes, e.g., $Read$ and $Update$, by “OR” operator, there will generate eight AC Security Requirements, e.g., $4 \times 2 = 8$, in the schema.


Add Individual Security Requirement

Selected Subject Attributes
AND
Division = Nursing Services
Department = Emergency Room
Role = Nursing Attendant
Role = Nursing Attendant

Selected Resource Attributes
OR
Patient Record = Prescription
Patient Record = Prescription

Selected Action Attributes
OR
Actions = Create
Actions = Create

Selected Environment Attributes
OR
Any Environment
Any Environment

Selected Condition Attributes
OR
Any Condition
Any Condition

Selected Decision
Permit

Individual Security Requirement Composition Checklist
✓ Choose Subjects
✓ Choose Resources
✓ Choose Actions
✓ Choose Environments
✓ Choose Conditions

Add
Cancel

Figure 51: “AND” to Create Multiple Security Requirement

Note: Inheritance is not applied to Security Requirements as they are not rules.

To Add/Update/Delete Individual Security Requirements To an Existing Schema: New Security Requirements can be added to a schema with a right click of the schema under “Individual Security Requirement”, e.g., $Nursing\ Attendant\ Policy\ Tests$. A popup window as in Figure 51 will appear for composing one or more new Security Requirement. Figure 51 shows an example of “AND” three attributes for composing Security Requirement. By using the “Add” operator, the following Security Requirement will be generated:

$Subject : Division = Nursing\ Services\ \&$
$Department = Emergency\ Room\ \&\ Role = Nursing\ Attendant;$
$Resource : Patient\ Record = Prescription\ Care; Action : Action = Create;$
$Decision : Permit$

The Security Requirement says that the $Nursing\ Attendant$ from $Emergency\ Room$ of the Nursing Service division can create a Prescription record for a patient. In the same way, more Security Requirements could be added to the Schema.

To Update an Individual Security Requirement: A Security Requirement in a schema can be updated with modification. This operation can be initiated by right-clicking a selected security requirement and you will see the tabs of “Update Security Requirement”. Clicking the tab will have the “Update Individual Security Requirement” window for you to modify the security requirement by adding and removing attributes, or changing the Decision. In the end, clicking “Update” to confirm the revision.

To Delete an Individual Security Requirement/Schema: This is to right-click the schema or security requirement that needs to be deleted and hit "Delete". In the pop out box, click "Yes" to confirm the deletion. Deleting a schema will remove all the Security Requirements.

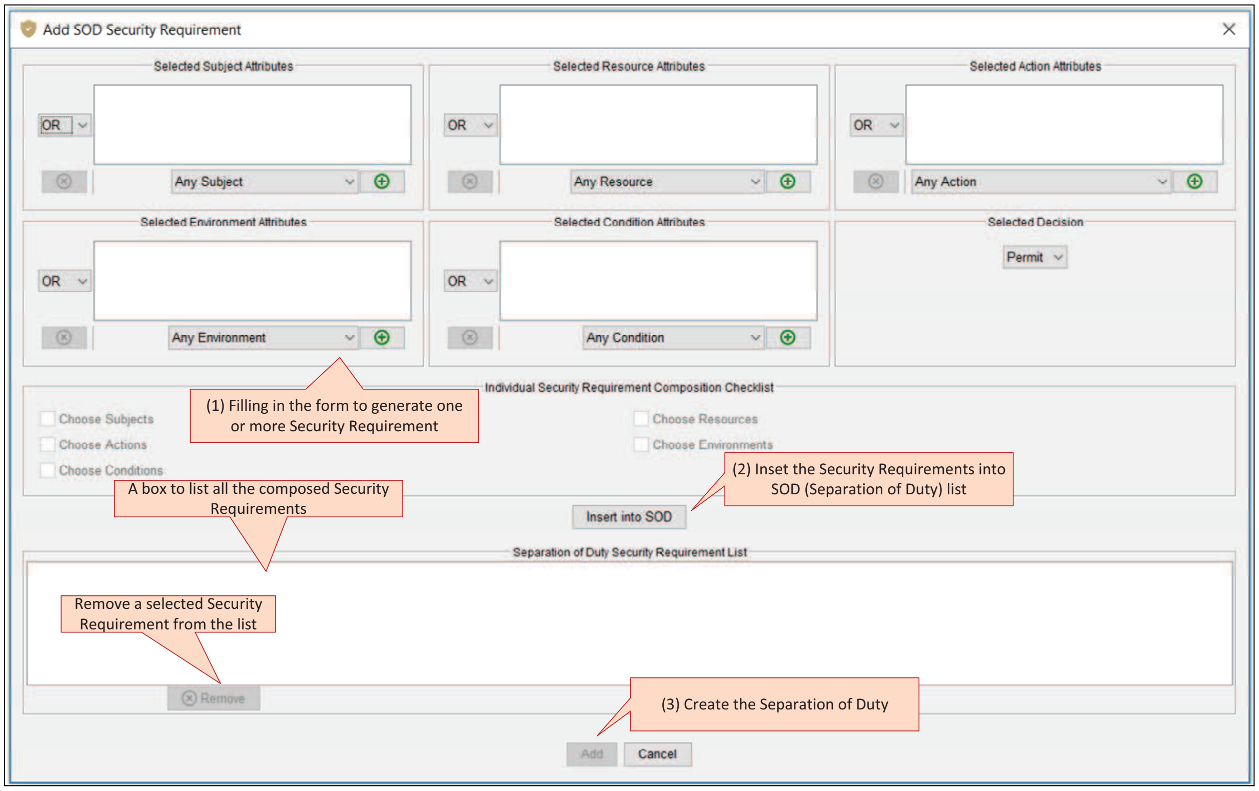

Add SOD Security Requirement
Selected Subject Attributes
Selected Resource Attributes
Selected Action Attributes
OR
Any Subject
Any Resource
Any Action
Selected Environment Attributes
Selected Condition Attributes
Selected Decision
Any Environment
Any Condition
Permit
(1) Filling in the form to generate one or more Security Requirement
Individual Security Requirement Composition Checklist
Choose Subjects
Choose Actions
Choose Conditions
Choose Resources
Choose Environments
(2) Inset the Security Requirements into SOD (Separation of Duty) list
A box to list all the composed Security Requirements
Insert into SOD
Separation of Duty Security Requirement List
Remove a selected Security Requirement from the list
Remove
(3) Create the Separation of Duty
Add
Cancel

Figure 52: Composition of a Separation of Duty Security Requirement

7.11.2 Separation of Duty Security Requirements

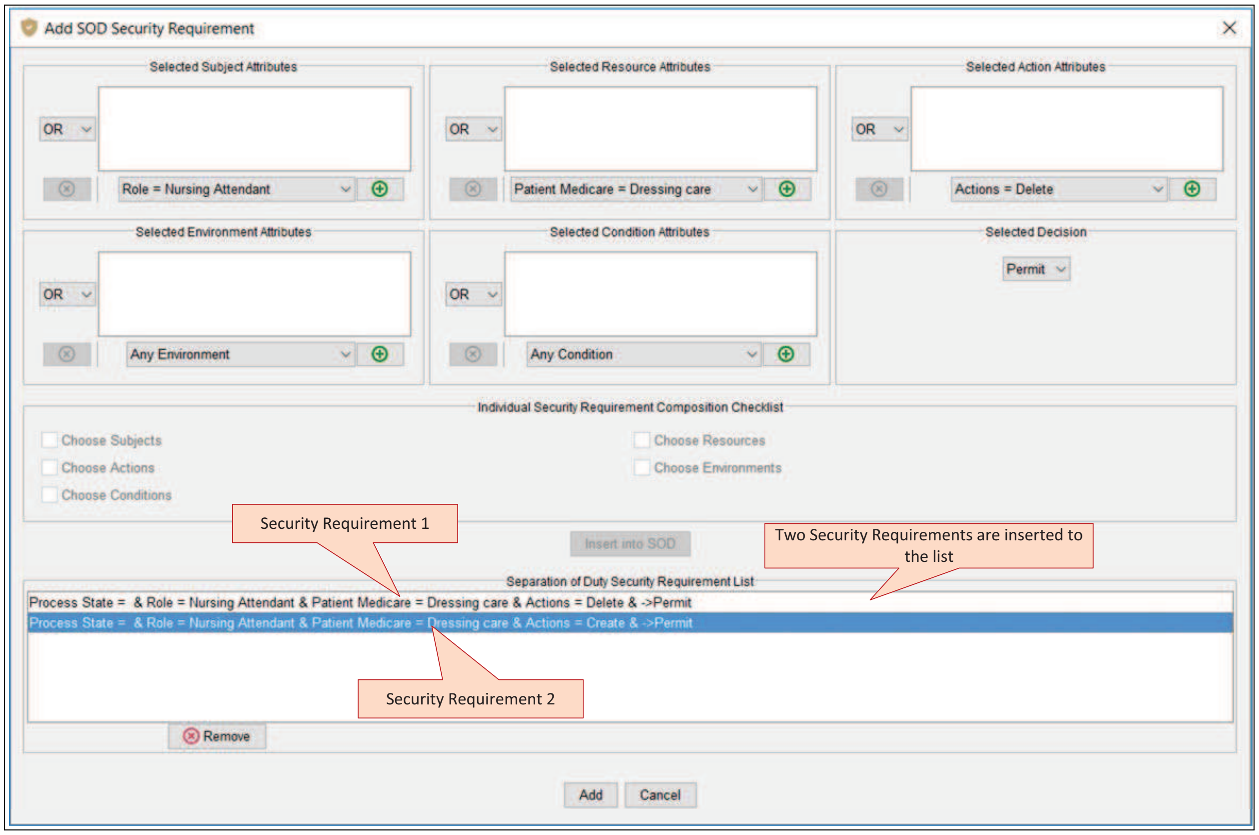
Individual Security Requirement is utilized to test the AC flaw (e.g., Error Type 1 - Block Privilege) caused by single AC request. Differently, Separation of Duty Security Requirements is used to test the AC flaw (e.g., Error Type 8) caused by two or more different correlated security requirements. It evaluates the access right with more than one AC Subjects, Resources, or Actions to detect Conflict of Interest that may cause fraud or information leakage. More specifically, the AC decisions of these requirements are evaluated in a correlated way, e.g., Permit of a Security Requirement A conflicts the Permit of a Security Requirement B.

The composition of Separation of Duty Security Requirement is initiated by right click the "Separation of Duty Security Requirement" in the project tree. It will lead to a tab of "Add a New SOD Security Requirement" and click it will have the window as shown in Figure 52, which shows three steps to add a Separation of Duty Security Requirement:

- **Create a Security Requirement**: The first step is to fill out the form of attributes, condition, and permission in order to compose a Security Requirement. This follows the same operations previously discussed in Figure 48, Figure 50, and Figure 51.
- **Insert the Security Requirement into SoD List**: The second step is to insert the composing Security Requirement into a "Separation of Duty Security Requirement List". If all the

elements in the “Individual Security Requirement Checklist” are checked, the composing Security Requirement is complete. As a result, the “Insert of SoD” button will be activated and clicking it adds the composing Security Requirement into the “Separation of Duty Security Requirement List”. Step 1 and Step 2 is repeated to add multiple Security Requirements into the “Separation of Duty Security Requirement List”. If you have any Security Requirement is wrongly composed, you can first select this Security Requirement in the list and click “Remove” to remove it from the list.

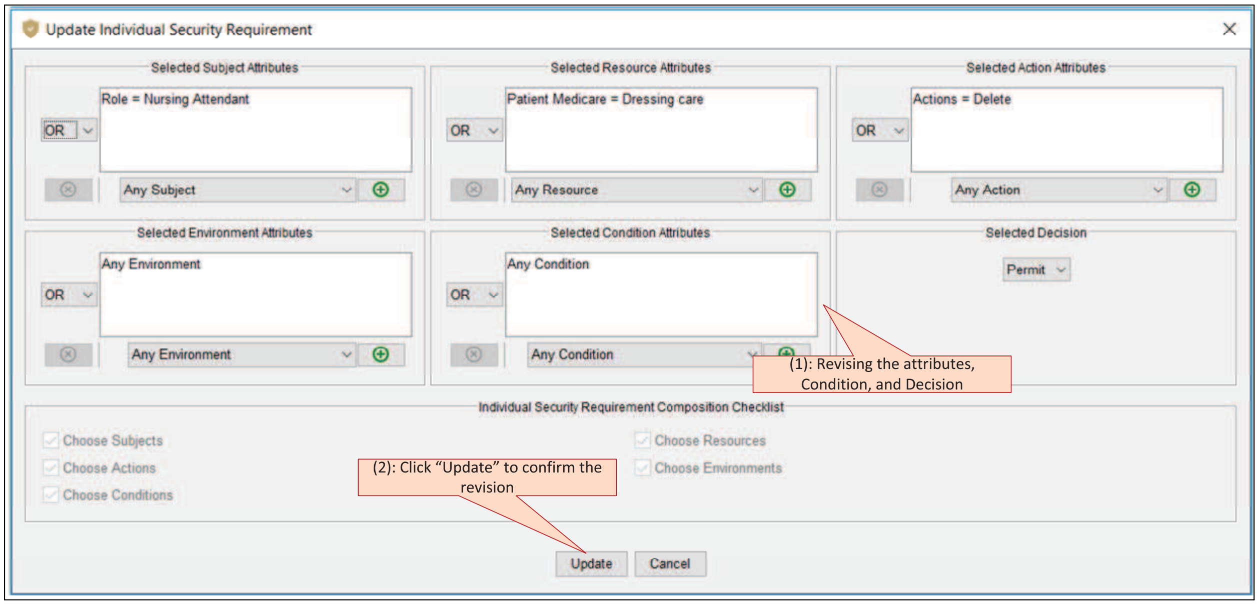
- **Create a Separation of Duty Security Requirement**: When you have two or more Security Requirements are inserted in the “Separation of Duty Security Requirement List” you can click the “Add” button to add the Separation of Duty Security Requirement.


Add SOD Security Requirement
Selected Subject Attributes
OR
Role = Nursing Attendant
Selected Resource Attributes
OR
Patient Medicare = Dressing care
Selected Action Attributes
OR
Actions = Delete
Selected Environment Attributes
OR
Any Environment
Selected Condition Attributes
OR
Any Condition
Selected Decision
Permit
Individual Security Requirement Composition Checklist
Choose Subjects
Choose Resources
Choose Actions
Choose Environments
Choose Conditions
Security Requirement 1
Insert into SOD
Two Security Requirements are inserted to the list
Separation of Duty Security Requirement List
Security Requirement 2
Remove
Add
Cancel

Process State = & Role = Nursing Attendant & Patient Medicare = Dressing care & Actions = Delete & ->Permit
Process State = & Role = Nursing Attendant & Patient Medicare = Dressing care & Actions = Create & ->Permit

Figure 53: An Example of a Separation of Duty Security Requirement

Generally, one Separation of Duty Security Requirement includes two security requirements to state a Conflict of Interest, e.g., two actions of a role are conflicted on each other. Figure 53 shows an example of a Separation of Duty Security Requirement that includes two different Security Requirements. In some case, multiple security requirements are required to express the Conflict of Interest by a Separation of Duty Security Requirement, e.g., multiparty Conflict of Interests. By clicking on the “Add” button, the Separation of Duty Security Requirement will be added under the tab of “Separation of Duty Security Requirement” in the project tree. $\mathcal{SPT}$ will automatically generate a schema name in the form of $SOD\ i$, where i is the generated sequence number. The sequence number indicates that a schema could have a serial of Separation of Duty Security Requirements, such as Conflicts of Interest with colleagues, vendors and others whichever that are appropriate in the practice, e.g., healthcare and financial services.


Update Individual Security Requirement

Selected Subject Attributes: OR — Role = Nursing Attendant — Any Subject

Selected Resource Attributes: OR — Patient Medicare = Dressing care — Any Resource

Selected Action Attributes: OR — Actions = Delete — Any Action

Selected Environment Attributes: OR — Any Environment — Any Environment

Selected Condition Attributes: OR — Any Condition — Any Condition

Selected Decision: Permit

Individual Security Requirement Composition Checklist: Choose Subjects, Choose Actions, Choose Conditions, Choose Resources, Choose Environments

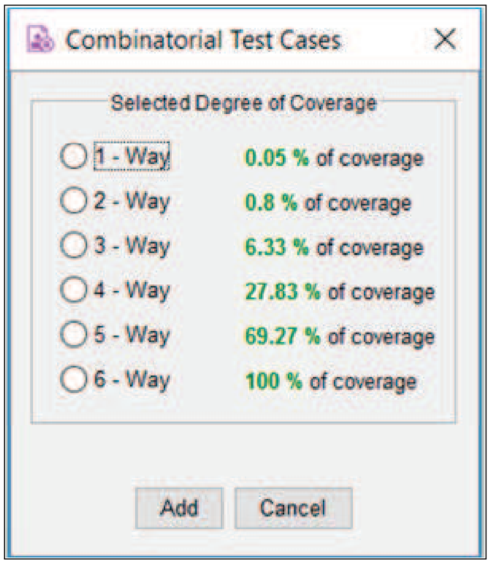
(1): Revising the attributes, Condition, and Decision

(2): Click "Update" to confirm the revision

Figure 54: Update a Security Requirement $SOD\ i$

Update a Separation of Duty Security Requirement: A defined Separation of Duty Security Requirement can be updated by modifying each security requirement. If you right click $SOD\ i$ (e.g., SOD 1), you can add new Security Requirement by selecting "Add New SOD Security Requirement (s)". Under the $SOD\ i$ in the project tree, if you right click on a specific Security Requirement, an option of "Update Security Requirement" appears, as shown in Figure 54. After clicking on this option, an "Update Individual Security Requirement" will pop out for you to edit (e.g., add and delete) the Attributes, Condition, and Decision. Upon the revision of the Security Requirement, click "Update" to confirm the modification.

Add/Delete a Security Requirement for a SOD Schema: A new Security Requirement can be added to a $SOD\ i$ schema by right clicking the selected $SOD\ i$ (e.g., SOD 1) and then hit the "Add New Individual Security Requirement" option. A blank window similar to Figure 54 will pop out which allows you to define a new Security Requirement and add it into $SOD\ i$ schema. In addition, a selected Security Requirement can be deleted by a "Delete" option which appears by right clicking on a specific Security Requirement. Furthermore, a $SOD\ i$ schema can be deleted by right clicking the selected $SOD\ i$ (e.g., SOD 1) and click the "Delete" option.

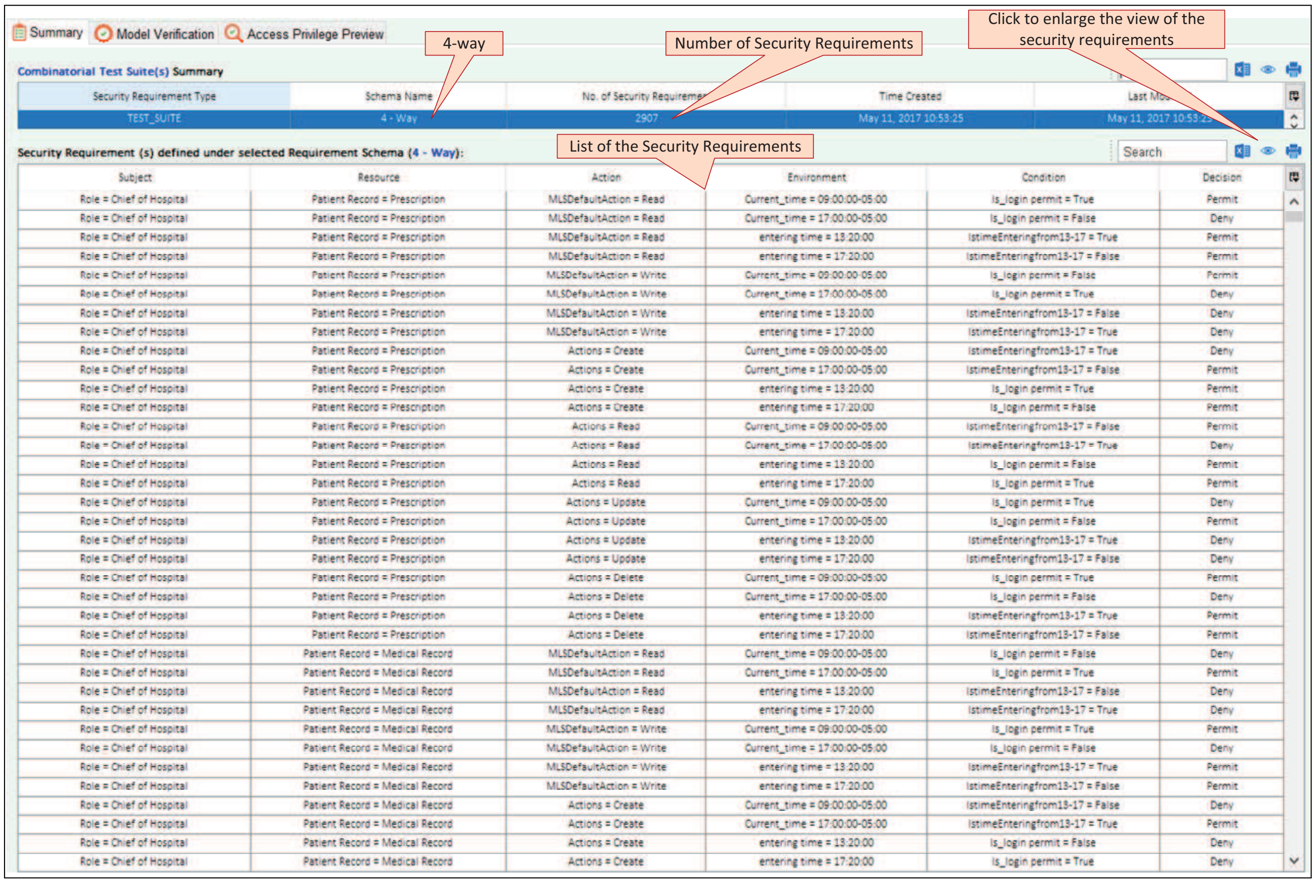


1 - Way 0.05 % of coverage
2 - Way 0.8 % of coverage
3 - Way 6.33 % of coverage
4 - Way 27.83 % of coverage
5 - Way 69.27 % of coverage
6 - Way 100 % of coverage

Figure 55: Combinatorial Test Suite Generation Interface

7.11.3 Combinatorial Test Suite

Individual Security Requirements generate specific security requests manually configured as shown in Figure 48. Combinatorial Test Suite is an exhaustive collection of $t - way$ Individual Security Requirements, where t is the number of interactions of Attributes, Condition, and Decisions for Security Requirement composition. Considering all possible $t - way$ combinations, this approach can provide compressive generation of Security Requirements for policy test. It has two steps to generate a Combinatorial Test Suite. The first step is to right click "Combinatorial Test Suite" in the project tree such that an option of "Add New Combinatorial Test Suite" appears. Clicking this option will give an interface as shown in Figure 55. The second step is to choose an $t - way$ option and then click on "Add" to generate a Combinatorial Test Suite.


Summary
Model Verification
Access Privilege Preview
4-way
Number of Security Requirements
Click to enlarge the view of the security requirements
List of the Security Requirements
Search

Combinatorial Test Suite(s) Summary

Security Requirement Type	Schema Name	No. of Security Requireme	Time Created	Last Mo
TEST_SUITE	4 - Way	2907	May 11, 2017 10:53:25	May 11, 2017 10:53:2

Security Requirement (s) defined under selected Requirement Schema (4 - Way):

Subject	Resource	Action	Environment	Condition	Decision
Role = Chief of Hospital	Patient Record = Prescription	MLSDefaultAction = Read	Current_time = 09:00:00-05:00	Is_login permit = True	Permit
Role = Chief of Hospital	Patient Record = Prescription	MLSDefaultAction = Read	Current_time = 17:00:00-05:00	Is_login permit = False	Deny
Role = Chief of Hospital	Patient Record = Prescription	MLSDefaultAction = Read	entering time = 13:20:00	IstimeEnteringfrom13-17 = True	Permit
Role = Chief of Hospital	Patient Record = Prescription	MLSDefaultAction = Read	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Permit
Role = Chief of Hospital	Patient Record = Prescription	MLSDefaultAction = Write	Current_time = 09:00:00-05:00	Is_login permit = False	Permit
Role = Chief of Hospital	Patient Record = Prescription	MLSDefaultAction = Write	Current_time = 17:00:00-05:00	Is_login permit = True	Deny
Role = Chief of Hospital	Patient Record = Prescription	MLSDefaultAction = Write	entering time = 13:20:00	IstimeEnteringfrom13-17 = False	Deny
Role = Chief of Hospital	Patient Record = Prescription	MLSDefaultAction = Write	entering time = 17:20:00	IstimeEnteringfrom13-17 = True	Deny
Role = Chief of Hospital	Patient Record = Prescription	Actions = Create	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = True	Deny
Role = Chief of Hospital	Patient Record = Prescription	Actions = Create	Current_time = 17:00:00-05:00	IstimeEnteringfrom13-17 = False	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Create	entering time = 13:20:00	Is_login permit = True	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Create	entering time = 17:20:00	Is_login permit = False	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Read	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = False	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Read	Current_time = 17:00:00-05:00	IstimeEnteringfrom13-17 = True	Deny
Role = Chief of Hospital	Patient Record = Prescription	Actions = Read	entering time = 13:20:00	Is_login permit = False	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Read	entering time = 17:20:00	Is_login permit = True	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Update	Current_time = 09:00:00-05:00	Is_login permit = True	Deny
Role = Chief of Hospital	Patient Record = Prescription	Actions = Update	Current_time = 17:00:00-05:00	Is_login permit = False	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Update	entering time = 13:20:00	IstimeEnteringfrom13-17 = True	Deny
Role = Chief of Hospital	Patient Record = Prescription	Actions = Update	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Deny
Role = Chief of Hospital	Patient Record = Prescription	Actions = Delete	Current_time = 09:00:00-05:00	Is_login permit = True	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Delete	Current_time = 17:00:00-05:00	Is_login permit = False	Deny
Role = Chief of Hospital	Patient Record = Prescription	Actions = Delete	entering time = 13:20:00	IstimeEnteringfrom13-17 = True	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Delete	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Permit
Role = Chief of Hospital	Patient Record = Medical Record	MLSDefaultAction = Read	Current_time = 09:00:00-05:00	Is_login permit = False	Deny
Role = Chief of Hospital	Patient Record = Medical Record	MLSDefaultAction = Read	Current_time = 17:00:00-05:00	Is_login permit = True	Permit
Role = Chief of Hospital	Patient Record = Medical Record	MLSDefaultAction = Read	entering time = 13:20:00	IstimeEnteringfrom13-17 = False	Deny
Role = Chief of Hospital	Patient Record = Medical Record	MLSDefaultAction = Read	entering time = 17:20:00	IstimeEnteringfrom13-17 = True	Deny
Role = Chief of Hospital	Patient Record = Medical Record	MLSDefaultAction = Write	Current_time = 09:00:00-05:00	Is_login permit = True	Permit
Role = Chief of Hospital	Patient Record = Medical Record	MLSDefaultAction = Write	Current_time = 17:00:00-05:00	Is_login permit = False	Deny
Role = Chief of Hospital	Patient Record = Medical Record	MLSDefaultAction = Write	entering time = 13:20:00	IstimeEnteringfrom13-17 = True	Permit
Role = Chief of Hospital	Patient Record = Medical Record	MLSDefaultAction = Write	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Permit
Role = Chief of Hospital	Patient Record = Medical Record	Actions = Create	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = False	Deny
Role = Chief of Hospital	Patient Record = Medical Record	Actions = Create	Current_time = 17:00:00-05:00	IstimeEnteringfrom13-17 = True	Permit
Role = Chief of Hospital	Patient Record = Medical Record	Actions = Create	entering time = 13:20:00	Is_login permit = False	Deny
Role = Chief of Hospital	Patient Record = Medical Record	Actions = Create	entering time = 17:20:00	Is_login permit = True	Deny

Figure 56: An Example of Combinatorial Test Suite ($4 - way$)

Figure 56 shows an example of a generated Combinatorial Test Suite with the Nursing Service ABAC model, which is a $4 - way$ Combinatorial Test Suite. It has 2907 Individual Security Requirements and all these requirements are listed in the Summary table. It is noted that the policy author only needs to choose one test suite as they are duplications. The individual Security Requirements in $2 - way$ Combinatorial Test Suite includes all the requirements in a $1 - way$ Combinatorial Test Suite, and so on. $2 - way$, $3 - way$, or $4 - way$ are recommended, which specifically is determined by the Probability of AC flaw detection and the verification time overhead, as explained below.

Figure 55 indicates that $\mathcal{SPT}$ allows a maximal t of six as it could be four variables of attributes (Subject, Resource, Action, and Environment), one Condition, and one Decision:

$$t_{max} = 4 + 1 + 1 = 6$$

where $6 - way$ Combinatorial Test Suite is a full collection of all Security Requirements and the evaluation of them reaches to 100% of flaw detection probability.

For each t, a Degree of Coverage is calculated as shown in Figure 55. The degree of Coverage is the percentage of Individual Security Requirements that will be generated by $t - way$ interactions compared to the total number of Individual Security Requirements possible. Figure 55 shows that the Degree of Coverage is 0.8% in the example for the $2 - way$ Combinatorial Test Suite. It can be seen that a larger t has a higher Degree of Coverage as shown in Figure 55. Meanwhile, a Degree

Table 1: **Probability of AC Flaw Detection**

$t - way$	**Estimated Probability of AC flaw detection**
$1 - way$	$10\% - 40\%$
$2 - way$	$50\% - 90\%$
$3 - way$	$75\% - 99\%$
$4 - way$	$90\% - 100\%$
$5 - way$	$96\% - 100\%$
$6 - way$	100%

of Coverage will result in a Probability of AC flaw detection, which is the probability that an AC flaw can be detected. Table 1 shows the estimated Probability of AC flaw detection corresponding to different $t - way$ suites. The results in Table 1 is an estimation based on the prior experiments and the actual value is hard to provide in the $\mathcal{SPT}$ tool. It shows $3 - way$ could generally achieve a very high flaw detection probability and $4 - way$ achieves an even higher security confidence.

On the other hand, a larger t will cause more time for test suite generation and verification . At first, for a larger t, $\mathcal{SPT}$ will take a longer time to generate the security requirements for Combinatorial Test Suite. As the generation is automatically computed in $\mathcal{SPT}$, such a time may not be significant compared to the time for verifying each testing results using the Combinatorial Test Suite. Upon the test which will be illustrated in the Subsection 7.12, the policy author needs to check the verification results of all the Security Requirements and it could be very time-consuming as the Security Requirements could be thousands or more. In other words, the policy author needs to humanly review the testing results to detect AC flaws. Therefore, the heavy time overhead, e.g., hours to days, should be considered as a key factor in t selection.

7.12 Policy Testing and Analysis

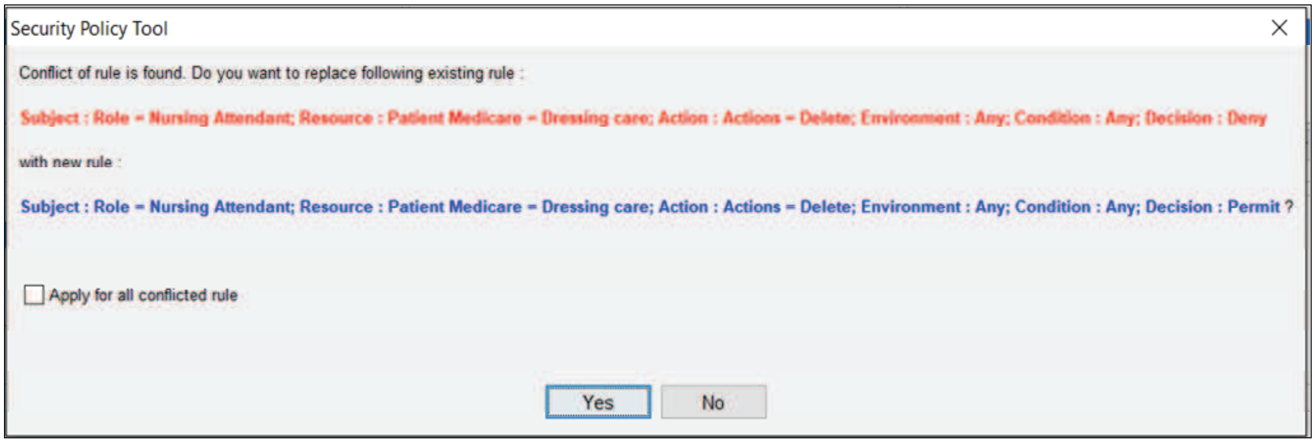
Upon the composition of an AC model with policies, policy testing and analysis can be then conducted to effectively identify the AC flaws and conveniently fix them. These flaws could be any of errors as listed in Subsection 6.4. For the purpose of AC flaw inspection and correction, $\mathcal{SPT}$ provides various policy testing and analyzing approaches and they are discussed in the following subsections.

7.12.1 Integrity and Consistence Check

During the AC model composition, $\mathcal{SPT}$ provides automatic integrity and consistency check internally to exclude Error Type 5 (i.e., Inconsistent Assignment). It also performs the detection of and Error Type 6 (i.e., AC Inheritance Loop) to prevent adding loop relations into the hierarchical inheritance. As a result, Error Types 5 and 6 can be automatically excluded and it is worry-free for a policy author by using $\mathcal{SPT}$. Meanwhile, $\mathcal{SPT}$ prevents duplicate rule composition in a policy while $\mathcal{SPT}$ will discard a duplicate rule when it is composed. In addition, $\mathcal{SPT}$ checks the Error Type 4 (i.e., Rule Conflict). Considering the following example for Nursing Service, Rule Conflict occurs:

$Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicare = Dressing\ Care;$
$Action = Delete; Decision : Permit$
$Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicare = Dressing\ Care;$
$Action = Delete; Decision : Deny$

where these two rules are conflicts on the Decision (i.e., One for Permit and one for Deny).


Security Policy Tool

Conflict of rule is found. Do you want to replace following existing rule :

Subject : Role = Nursing Attendant; Resource : Patient Medicare = Dressing care; Action : Actions = Delete; Environment : Any; Condition : Any; Decision : Deny

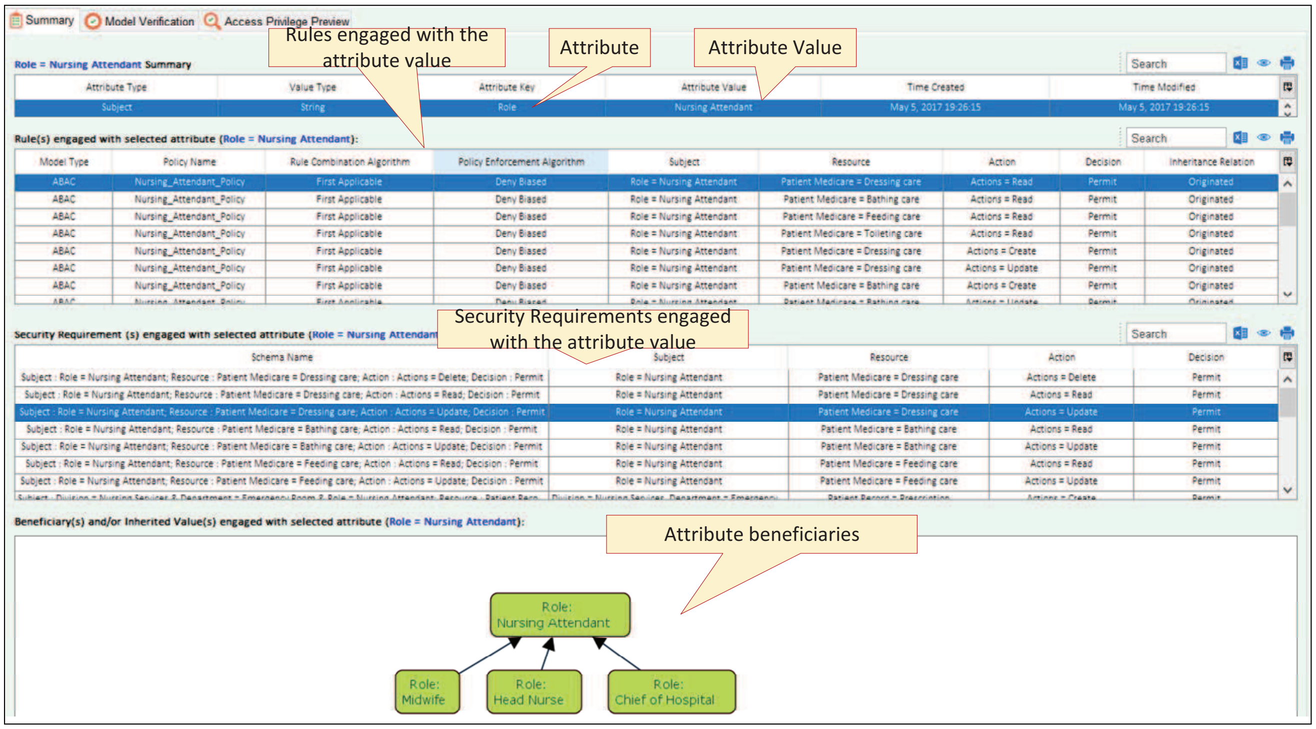
with new rule :

Subject : Role = Nursing Attendant; Resource : Patient Medicare = Dressing care; Action : Actions = Delete; Environment : Any; Condition : Any; Decision : Permit ?

☐ Apply for all conflicted rule

Figure 57: Rule Conflict Detection

During the composition of the policy rule, $\mathcal{SPT}$ will prompt the policy author to resolve the conflict by choosing the desirable one. Figure 57 shows the example when a conflicting rule is composed and $\mathcal{SPT}$ allows policy author to choose the correct one. Due to this, it is worry- free for Rule Conflict error as $\mathcal{SPT}$ automatically performs the detection.



Role = Nursing Attendant Summary

Attribute Type	Value Type	Attribute Key	Attribute Value	Time Created	Time Modified
Subject	String	Role	Nursing Attendant	May 5, 2017 19:26:15	May 5, 2017 19:26:15

Rule(s) engaged with selected attribute (Role = Nursing Attendant):

Model Type	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Subject	Resource	Action	Decision	Inheritance Relation
ABAC	Nursing_Attendant_Policy	First Applicable	Deny Biased	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Read	Permit	Originated
ABAC	Nursing_Attendant_Policy	First Applicable	Deny Biased	Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Read	Permit	Originated
ABAC	Nursing_Attendant_Policy	First Applicable	Deny Biased	Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Read	Permit	Originated
ABAC	Nursing_Attendant_Policy	First Applicable	Deny Biased	Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Read	Permit	Originated
ABAC	Nursing_Attendant_Policy	First Applicable	Deny Biased	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Create	Permit	Originated
ABAC	Nursing_Attendant_Policy	First Applicable	Deny Biased	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Update	Permit	Originated
ABAC	Nursing_Attendant_Policy	First Applicable	Deny Biased	Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Create	Permit	Originated
ABAC	Nursing_Attendant_Policy	First Applicable	Deny Biased	Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Update	Permit	Originated

Security Requirement (s) engaged with selected attribute (Role = Nursing Attendant

Schema Name	Subject	Resource	Action	Decision
Subject : Role = Nursing Attendant; Resource : Patient Medicare = Dressing care; Action : Actions = Delete; Decision : Permit	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Permit
Subject : Role = Nursing Attendant; Resource : Patient Medicare = Dressing care; Action : Actions = Read; Decision : Permit	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Read	Permit
Subject : Role = Nursing Attendant; Resource : Patient Medicare = Dressing care; Action : Actions = Update; Decision : Permit	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Update	Permit
Subject : Role = Nursing Attendant; Resource : Patient Medicare = Bathing care; Action : Actions = Read; Decision : Permit	Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Read	Permit
Subject : Role = Nursing Attendant; Resource : Patient Medicare = Bathing care; Action : Actions = Update; Decision : Permit	Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Update	Permit
Subject : Role = Nursing Attendant; Resource : Patient Medicare = Feeding care; Action : Actions = Read; Decision : Permit	Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Read	Permit
Subject : Role = Nursing Attendant; Resource : Patient Medicare = Feeding care; Action : Actions = Update; Decision : Permit	Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Update	Permit
Subject : Division = Nursing Services & Department = Emergency Room & Role = Nursing Attendant; Resource : Patient Reco	Division = Nursing Services, Department = Emergency	Patient Record = Prescription	Actions = Create	Permit

Beneficiary(s) and/or Inherited Value(s) engaged with selected attribute (Role = Nursing Attendant):

Figure 58: Summary of an Attribute Value

In addition to these automatic detections, $\mathcal{SPT}$ enables policy author to preview and check the AC model from different ways. Figure 58 shows the information about attribute value of $Nursing\ Attendent$ that allows a policy author to check if there are unintended definitions of any attributes or not. For example, from the table, a policy author can check if the rules are composed as the AC intention. If there are any errors, the rules can be updated by right click on the rule in the project tree. As shown in Figure 58, the summary shows up the inheritance beneficiaries of the attribute value and the policy author can check if there are any issues. A similar summary is provided for the composed policies as well as rules such that the policy author can check the rule combining algorithm, policy enforcement algorithm, and the rule decisions. Policy and rule updating function is provided for policy author to revise a policy and its rules.

In order to find other types of AC flaws, $\mathcal{SPT}$ has ”Model Verification” and ”Access Privilege Preview” functions for various policy testing methods from which analysis can be conducted for AC flaw detections as illustrated in the next Subsections.

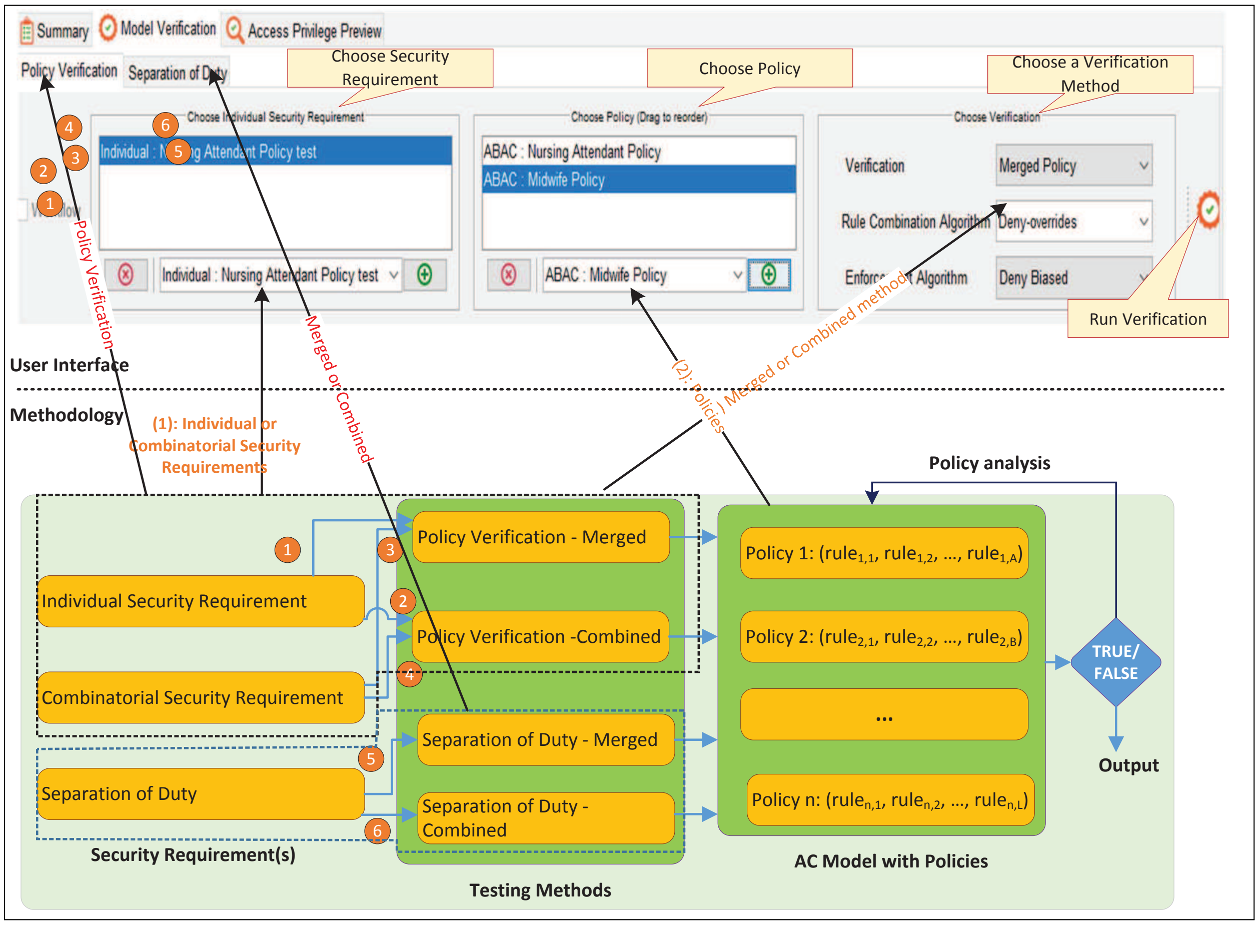

Summary
Model Verification
Access Privilege Preview
Policy Verification
Separation of Duty
Choose Security Requirement
Choose Policy
Choose a Verification Method
Choose Individual Security Requirement
Individual : Nursing Attendant Policy test
Choose Policy (Drag to reorder)
ABAC : Nursing Attendant Policy
ABAC : Midwife Policy
Choose Verification
Verification
Merged Policy
Rule Combination Algorithm
Deny-overrides
Enforcement Algorithm
Deny Biased
Run Verification
Policy Verification
Merged or Combined
(2): policies
(3) Merged or Combined method
User Interface
Methodology
(1): Individual or Combinatorial Security Requirements
Policy analysis
Individual Security Requirement
Combinatorial Security Requirement
Separation of Duty
Security Requirement(s)
Policy Verification - Merged
Policy Verification -Combined
Separation of Duty - Merged
Separation of Duty - Combined
Testing Methods
Policy 1: (rule1,1, rule1,2, ..., rule1,A)
Policy 2: (rule2,1, rule2,2, ..., rule2,B)
...
Policy n: (rulen,1, rulen,2, ..., rulen,L)
AC Model with Policies
TRUE/ FALSE
Output

Figure 59: Policy Verification

7.12.2 Testing Policy and Method Configuration

As illustrated in Figure 8 in Section 6.3, the policy verification is to test the AC effectiveness (i.e., Permit or Deny) of the selected policies of an AC model in response to one or more Security Requirements via a testing method, where

- Security Requirements are access requests,
- Policies are a set of access control rules with policy algorithms, and
- A testing method could be (i) Single Policy, (ii) Merged Policy, (iii) Combined Policy, which state how the policies/rules are operated to achieve a AC decision. Merged Policy and Combined Policy tests are used for the test of multiple policies.

Figure 59 shows the Policy Verification interface and how it is functionally mapped to Policy Test Methods in Figure 8. By clicking on "Model Verification", it shows two functional tabs:

- **"Policy Verification"**: This provides the functions to conduct ① – ④ testing types, as discussed in Section 6.3.
- **"Separation of Duty"**: This provides the functions to conduct ⑤ – ⑥ testing types, as discussed in Section 6.3.

For the Policy Verification, Figure 59 shows the Policy Verification interface which allows the testing configuration of: (1): Choosing Security Requirement (Individual or Combinatorial), (2): Choosing an AC Model with Policies, (3) Choosing Testing Methods (Merged, or Combined). As shown in Figure 59, a "Run Verification" button will be activated if a testing request is ready by choosing the Security Requirement, Policy, and a testing method.

Order the Policies: The order of the policies may play a role on the policy-combining algorithm. For such a consideration, $\mathcal{SPT}$ allows the selected policies to be reordered by a function called $Drag\ to\ reorder$. This function allows you to choose a policy in the "Choose Policy (Drag to Reorder)" window and drag it to a new sequenced place in the window.

In the following subsection, we explain how to conduct policy analysis with various testing methods.

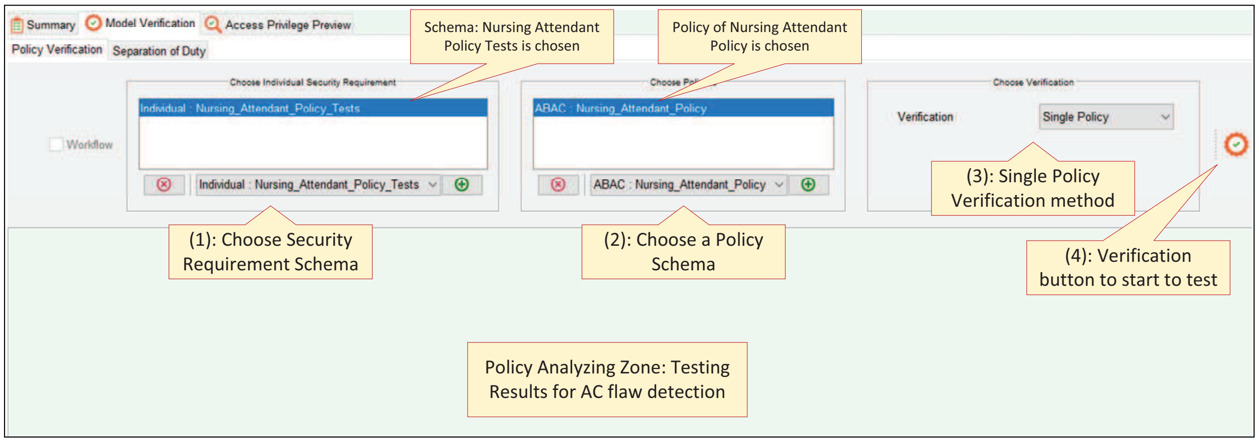

Summary
Model Verification
Access Privilege Preview
Policy Verification
Separation of Duty
Schema: Nursing Attendant Policy Tests is chosen
Policy of Nursing Attendant Policy is chosen
Choose Individual Security Requirement
Individual : Nursing_Attendant_Policy_Tests
Workflow
Individual : Nursing_Attendant_Policy_Tests
ABAC : Nursing_Attendant_Policy
ABAC : Nursing_Attendant_Policy
Choose Verification
Verification
Single Policy
(3): Single Policy Verification method
(1): Choose Security Requirement Schema
(2): Choose a Policy Schema
(4): Verification button to start to test
Policy Analyzing Zone: Testing Results for AC flaw detection

Figure 60: Single Policy Verification

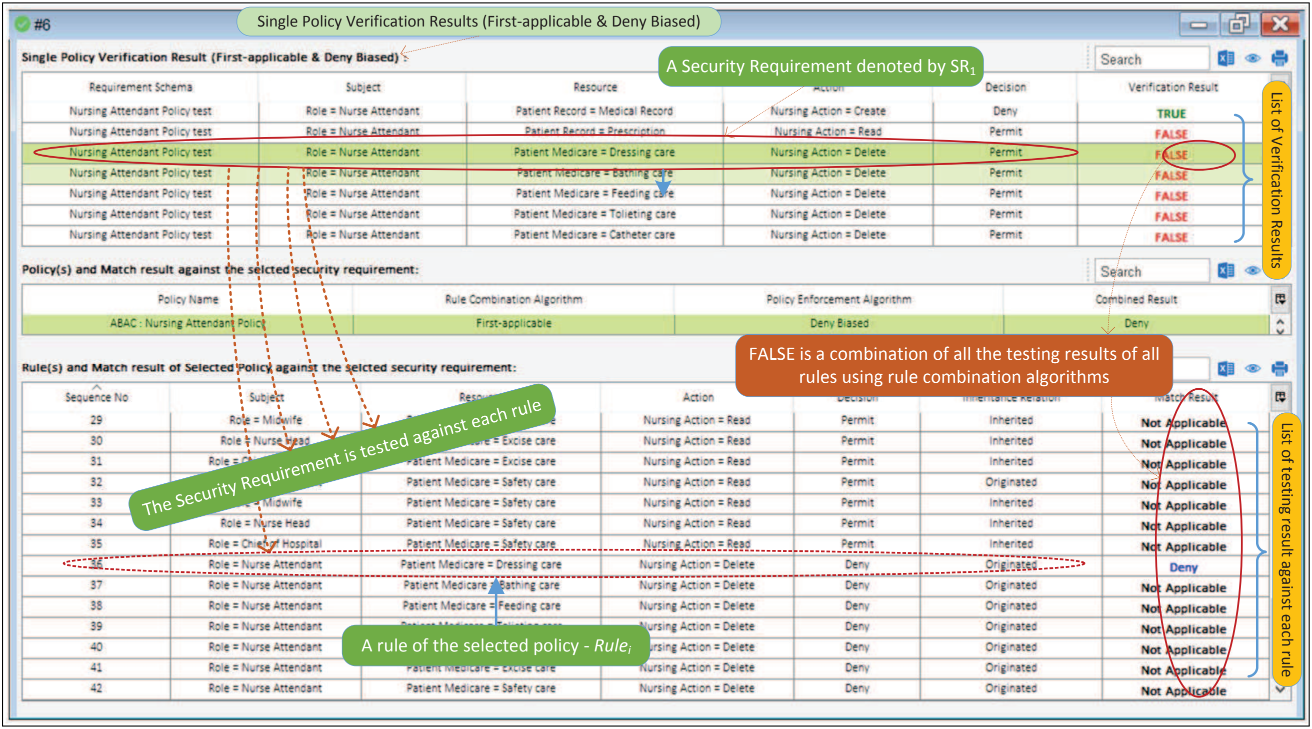
7.12.3 Single Policy Verification

If only one policy is chosen for verification, the merged policy verification and combine policy verification are reduced to Single Policy Verification. Figure 60 demonstrates a Single Policy Verification example using the Nursing Service and shows three steps:

- **Choose Security Requirement Schema(s)**: Navigate to Policy Verification by clicking "Model Verification" and "Policy Verification" subsequently. Under the "Choose Security Requirement", a schema can be selected from the dropdown menu which lists all the composed schemas, such as $Nursing\ Attendant\ Policy\ Tests$, as shown in Figure 60. The selected schema is added into the "Choose Security Requirement" list upon the click of $\oplus$. Repeating this operation, multiple schemas can be selected. Meanwhile, $\otimes$ can be used to remove a selected schema from the "Choose Security Requirement" list.

- **Choose a Policy**: This step selects the policy for testing. Under the "Choose Policy (Drag to Reorder)", a policy in an AC Model can be selected from the dropdown menu which lists all the composed policies, such as $ABAC : Nursing\ Attendant\ Policy\ Tests$, as shown in Figure 60. The selected policy is added into the "Choose Policy (Drag to Reorder)" list upon the click of $\oplus$. $\otimes$ allows to remove a selected policy from the "Choose Policy (Drag to Reorder)" list. Single Policy Verification only chooses one policy.

- **Start to the Verification**: As only one policy is chosen, Single Policy Verification is the available option in the "Choose Verification" method. Please click the "Run Verification" button to start the test. Upon the execution of the test, the verification results will be presented in the Policy Analyzing Zone and then policy analysis can be conducted as illustrated below.


#6
Single Policy Verification Results (First-applicable & Deny Biased)
A Security Requirement denoted by SR_1
List of Verification Results
Search
FALSE is a combination of all the testing results of all rules using rule combination algorithms
The Security Requirement is tested against each rule
A rule of the selected policy - $Rule_i$
List of testing result against each rule

Single Policy Verification Result (First-applicable & Deny Biased):

Requirement Schema	Subject	Resource	Action	Decision	Verification Result
Nursing Attendant Policy test	Role = Nurse Attendant	Patient Record = Medical Record	Nursing Action = Create	Deny	TRUE
Nursing Attendant Policy test	Role = Nurse Attendant	Patient Record = Prescription	Nursing Action = Read	Permit	FALSE
Nursing Attendant Policy test	Role = Nurse Attendant	Patient Medicare = Dressing care	Nursing Action = Delete	Permit	FALSE
Nursing Attendant Policy test	Role = Nurse Attendant	Patient Medicare = Bathing care	Nursing Action = Delete	Permit	FALSE
Nursing Attendant Policy test	Role = Nurse Attendant	Patient Medicare = Feeding care	Nursing Action = Delete	Permit	FALSE
Nursing Attendant Policy test	Role = Nurse Attendant	Patient Medicare = Tolieting care	Nursing Action = Delete	Permit	FALSE
Nursing Attendant Policy test	Role = Nurse Attendant	Patient Medicare = Catheter care	Nursing Action = Delete	Permit	FALSE

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
ABAC : Nursing Attendant Policy	First-applicable	Deny Biased	Deny

Rule(s) and Match result of Selected Policy against the selcted security requirement:

Sequence No	Subject	Resource	Action	Decision	Inheritance Relation	Match Result
29	Role = Midwife	[illegible]	Nursing Action = Read	Permit	Inherited	Not Applicable
30	Role = Nurse Head	Patient Medicare = Excise care	Nursing Action = Read	Permit	Inherited	Not Applicable
31	Role = Chief of Hospital	Patient Medicare = Excise care	Nursing Action = Read	Permit	Inherited	Not Applicable
32	[illegible]	Patient Medicare = Safety care	Nursing Action = Read	Permit	Originated	Not Applicable
33	Role = Midwife	Patient Medicare = Safety care	Nursing Action = Read	Permit	Inherited	Not Applicable
34	Role = Nurse Head	Patient Medicare = Safety care	Nursing Action = Read	Permit	Inherited	Not Applicable
35	Role = Chief of Hospital	Patient Medicare = Safety care	Nursing Action = Read	Permit	Inherited	Not Applicable
36	Role = Nurse Attendant	Patient Medicare = Dressing care	Nursing Action = Delete	Deny	Originated	Deny
37	Role = Nurse Attendant	Patient Medicare = Bathing care	Nursing Action = Delete	Deny	Originated	Not Applicable
38	Role = Nurse Attendant	Patient Medicare = Feeding care	Nursing Action = Delete	Deny	Originated	Not Applicable
39	Role = Nurse Attendant	Patient Medicare = Tolieting care	Nursing Action = Delete	Deny	Originated	Not Applicable
40	Role = Nurse Attendant	[illegible]	Nursing Action = Delete	Deny	Originated	Not Applicable
41	Role = Nurse Attendant	Patient Medicare = Excise care	Nursing Action = Delete	Deny	Originated	Not Applicable
42	Role = Nurse Attendant	Patient Medicare = Safety care	Nursing Action = Delete	Deny	Originated	Not Applicable

Figure 61: Single Policy Test Result

Policy Analysis: Policy analysis is to verify if each of the testing results of the Security Requirement is matched with the AC expectation or not. The AC expectation is the trust of a Permit or Deny decision that should be intentionally given to a specific Security Requirement without AC errors. If the AC expectation not matched with Decision result, an AC error occurs and the policy needs to be revised to avoid the error and meanwhile generate no new errors. Figure 61 shows an example to illustrate the policy analysis. This example shows the following information:

Security Requirement Schema (e.g., $Nursing\ Attendant\ Policy\ Tests$): The schema is selected in Figure 60. It has 22 Security Requirements that are listed in Table of "Single Policy Verification Result" in Figure 61. It is noted that they are also listed in the schema summary by Clicking $Nursing\ Attendant\ Policy\ Tests$ in the project tree. The Security Requirements are:

$Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicare = Dressing\ Care;$ $Actions = Delete; Decision : Permit$ (Marked in Figure 61; Denote this rule by SR_1 for illustration purpose.)
. . .
where "· · ·" represents other Security Requirements in the table (same in below).

Testing Policy ($ABAC : Nursing\ Attendant\ Policy$): This is the testing policy selected in Figure 60. This policy has a set of rules which are listed in the Table of "Rules and Match result of selected security requirement" in Figure 61. Similarly, these rules are listed in the policy summary by clicking $Nursing\ Attendant\ Policy$ under ABAC in the project tree. These rules are:

...
$Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicare = Dressing\ Care;$
$Actions = Delete; Decision : Deny$ (Marked in Figure 61; Denote this rule by $Rule_i$ for illustration purpose.)
...

Policy Algorithms: The policy algorithms are configured during the composition of the policy as shown in Figure 32. The example has: (i) Rule Combination Algorithm - First Applicable, (ii) Policy Enforcement Algorithm - Deny Biased.

Verification Results: The final verification results are presented by $TRUE/FALSE$. Policy author reviews the verification result by following ACFD theory:

If the verification result of $TRUE$ or $FALSE$ is matched to AC expectation, then we say: $No\ AC\ flow$; Otherwise, $AC\ flow$

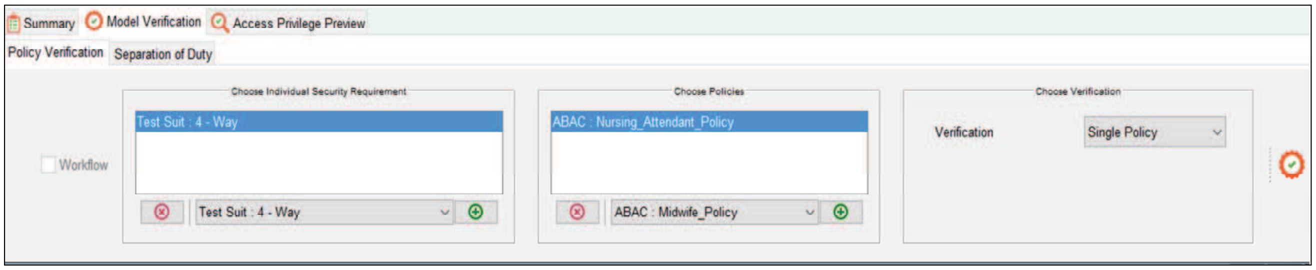
For example, the verification result for SR_1 is read as: It is $FALSE$ to give Permission for $Nursing\ Attendant$ to $Delete$s the patient medicate record. This result is contradict to the SR_1 that states: It is $Permitted$ for $Nursing\ Attendant$ to $Delete$s the patient medicate record. In this example, it has:

$$\begin{cases} AC\ Flaw, & \text{if } SR_1 \text{ is AC Expectation} \\ No\ AC\ Flaw, & \text{Otherwise.} \end{cases} \tag{10}$$

Suppose SR_1 is the AC expectation. Under this assumption, the result is not matched with the AC expectation. According to the AC flaw Error Type definition, this can be identified as Error Type 1 (Block Privilege, see subsection 6.4) as the intended permission of the $Nursing\ Attendant$ is declined by the given policy. By reviewing the rules, it can be seen that $Rule_i$ in the policy declines the request for $Nursing\ Attendant$ to $Delete$ the patient medicate record, and all other rules are "Not Applicable" as shown in Figure 61. Therefore, a modification should be made to $Rule_i$, such as revising the Deny to Permit such that SR_1 will be matched with the rule Decision. The modification can be done by right clicking on the rule in the project tree with selection of "Update ABAC Rule".

The above analyzing procedure repeats for all Security Requirements. As each Security Requirement is independently tested against the policy. The above example has 22 results, i.e., 22 $TRUE/FALSE$s, which are shown in the last column of the Table "Single Policy Verification Results" in Figure 61, and marked as List of Verification Result. Therefore, the policy author needs to review all the 22 results to verify if they are matched to AC expectation. If a flaw is found, the corresponding rule can be identified and modified to fix the flaw. After the revision, the policy should be reverified by repeat policy test in Figure 60 and the above analysis in Figure 61 till all AC expectations are satisfied.

Exhaustive Single Policy Verification: Exhaustive Single Policy Verification is to test a policy with Combinatorial Security Requirements. The above example considers the schema named by $Nursing\ Attendant\ Policy\ Tests$ which is built in the category of the Individual Security Requirement. As an result, the request cases are normally limited due to manual Individual Security Requirement generation, e.g, 22 Security Requests. Differently, Exhaustive Single Policy Verification verifies the AC requests to achieve a certain Degree of Coverage (See Subsection 7.11.3) to offer a higher AC security confidence. It could verify the policy via a large number of Security Requests. For the example in Subsection 7.11.3, $4-way$ Combinatorial Test Suite has generated 2907 Security Requirements. The analysis of the results of all these Security Requirements achieves 27.83% Degree of Coverage. Such a coverage will reaches to a Flaw Detection Probability of $90\%-100\%$ proximately. Therefore, an affordable Exhaustive Single Policy Verification is recommended for policy verification.



Test Suit : 4 - Way

ABAC : Nursing_Attendant_Policy

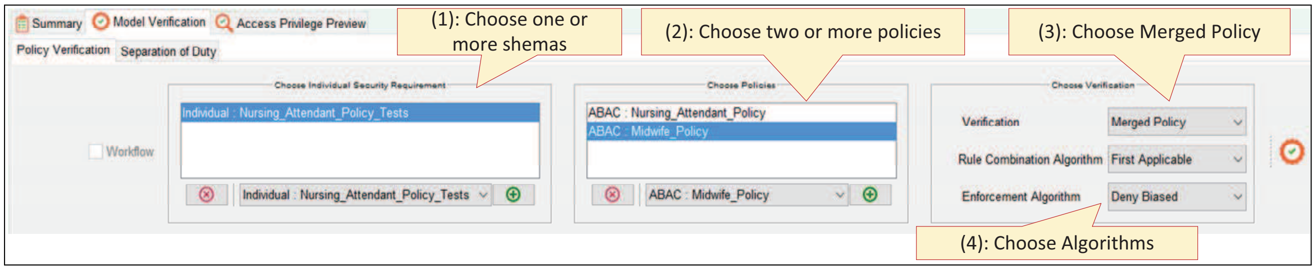
Verification: Single Policy

Test Suit : 4 - Way

ABAC : Midwife_Policy

Figure 62: Policy Verification Steps

Figure 62 shows how to conduct Exhaustive Single Policy Verification. Instead of choosing a schema of Individual Security Requirement, Exhaustive Single Policy Verification choose a Combinatorial Test Suite, such as $4-way$ as shown in Figure 62. The operations are to first navigate Policy Verification by clicking “Model Verification” and “Policy Verification”, choose a Combinatorial Test Suite, e.g., $4-way$, from the dropdown menu of the“Choose Security Requirement”, and then choose the policy as shown Figure 62. It is then to click on the Run Verification icon to perform the test and analyze the verification results, e.g., 2907 results in the example, to fix the policy flaw if it has. $\mathcal{SPT}$ provides search, sort, print, and Excel output functions to facilitate the analyzing process as it could be time-consuming.



Individual : Nursing_Attendant_Policy_Tests

ABAC : Nursing_Attendant_Policy
ABAC : Midwife_Policy

Verification: Merged Policy
Rule Combination Algorithm: First Applicable
Enforcement Algorithm: Deny Biased

Individual : Nursing_Attendant_Policy_Tests

ABAC : Midwife_Policy

Figure 63: Merged Policy Verification

7.12.4 Merged Policy Verification

Instead of a single policy, Merged Policy Verification verifies the AC effectiveness of multiple policies while these policies are merged as one policy. In this way, the decision of an AC request, i.e., a Security Request, is determined by all the rules of multiple policies. Consider an example of two policies while each policy has a set of rules. In this case, a Security Requirement will be tested against each rule in other policies. The testing results of all these rules will be merged by a rule combining algorithm and a policy enforcement algorithm. The merged Decision is compared with the Decision in the Security Requirement to yield a $TRUE$ or $FALSE$. Please refer to Subsection 6.3.2 for the detailed principle.

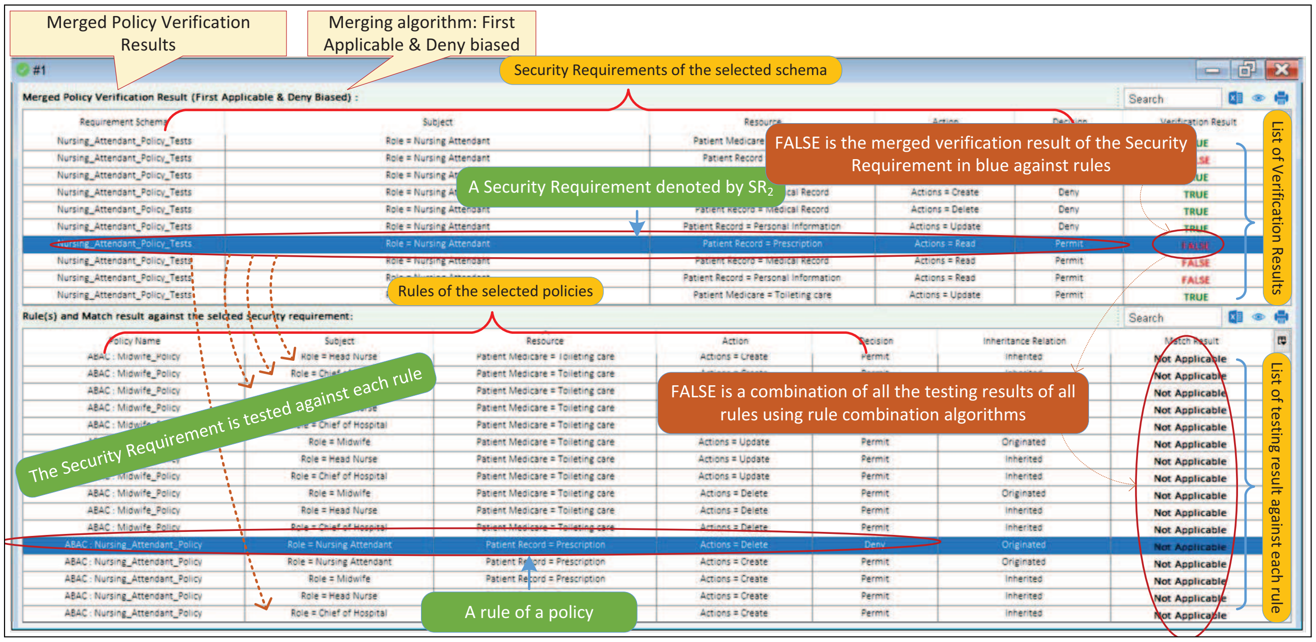
Policy Test: Figure 63 shows how the merged policy verification is conducted.

- **Choose Security Requirement Schema(s)**: Click "Model Verification" and "Policy Verification" subsequently. Choose a schema from the dropdown menu and add it to the "Choose Security Requirement" by clicking "$\oplus$" button, e.g., $Nursing\ Attendant\ Policy\ Tests$, as shown in Figure 63. One or more schemas can be chosen for the test.

- **Choose Two or More Policies**: The purpose of Merged Policy Verification is to test the AC effectiveness of the selected schema against all the rules of two or more policies. Therefore, at least two or more policies (e.g., $Policy\ 1 - n$) are selected for Merged Policy Verification. This is done by repeatedly choosing a policy from the dropdown menu and clicking "$\oplus$" button to add the policy into "Choose Policies", as shown in Figure 63. Figure 63 shows two policies that are $ABAC : Nursing\ Attendant\ Policy$ and $ABAC : Midwife\ Policy$.

- **Choose Merged Policy and Algorithms**: As two policies are chosen, "Choose Verification" will show up three dropdown menus for Verification, Rule Combining Algorithm, and Enforcement Algorithm respectively. "Merged Policy" is chosen for Merged Policy Verification. Meanwhile, a Rule Combining Algorithm and an Enforcement Algorithm should be chosen. During the policy composition, each policy is configured with its own Rule Combining Algorithm and Enforcement Algorithm. These algorithms may be different for policies, e.g., $Policy\ 1$ has First Applicable while $Policy\ 2$ has Deny Override. When Merged Policy Verification is applied, these algorithms will become invalid as these algorithms could be unified into the chosen algorithms. For example, First Applicable and Deny Biased as shown in Figure 63 will be applied to the Merged Policy Verification for $ABAC : Nursing\ Attendant\ Policy$ and $ABAC : Midwife\ Policy$ regardless of the initial algorithm configurations of these two policies.

- **Start to the Verification**: Finally, it is to click the Run Verification icon and the test starts immediately.


Merged Policy Verification Results
Merging algorithm: First Applicable & Deny biased
Security Requirements of the selected schema
#1
A Security Requirement denoted by SR$_2$
FALSE is the merged verification result of the Security Requirement in blue against rules
List of Verification Results
Rules of the selected policies
The Security Requirement is tested against each rule
FALSE is a combination of all the testing results of all rules using rule combination algorithms
List of testing result against each rule
A rule of a policy
Search

Merged Policy Verification Result (First Applicable & Deny Biased) :

Requirement Schema	Subject	Resource	Action	Decision	Verification Result
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Medicare [illegible]	[illegible]	[illegible]	[illegible]
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record [illegible]	[illegible]	[illegible]	[illegible]
Nursing_Attendant_Policy_Tests	Role = Nursing At[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Nursing_Attendant_Policy_Tests	Role = Nursing A[illegible]	[illegible]cal Record	Actions = Create	Deny	TRUE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Medical Record	Actions = Delete	Deny	TRUE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Personal Information	Actions = Update	Deny	TRUE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Prescription	Actions = Read	Permit	FALSE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Medical Record	Actions = Read	Permit	FALSE
Nursing_Attendant_Policy_Tests	[illegible]	Patient Record = Personal Information	Actions = Read	Permit	FALSE
Nursing_Attendant_Policy_Tests	[illegible]	Patient Medicare = Toileting care	Actions = Update	Permit	TRUE

Rule(s) and Match result against the selcted security requirement:

Policy Name	Subject	Resource	Action	Decision	Inheritance Relation	Match Result
ABAC : Midwife_Policy	Role = Head Nurse	Patient Medicare = Toileting care	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Midwife_Policy	Role = Chief[illegible]	Patient Medicare = Toileting care	[illegible]	[illegible]	[illegible]	Not Applicable
ABAC : Midwife_Policy	[illegible]	Patient Medicare = Toileting care	[illegible]	[illegible]	[illegible]	Not Applicable
ABAC : Midwife_Policy	[illegible]	Patient Medicare = Toileting care	[illegible]	[illegible]	[illegible]	Not Applicable
ABAC : Midw[illegible]	[illegible] = Chief of Hospital	Patient Medicare = Toileting care	[illegible]	[illegible]	[illegible]	Not Applicable
[illegible]	Role = Midwife	Patient Medicare = Toileting care	Actions = Update	Permit	Originated	Not Applicable
[illegible]	Role = Head Nurse	Patient Medicare = Toileting care	Actions = Update	Permit	Inherited	Not Applicable
[illegible] Midwife_Policy	Role = Chief of Hospital	Patient Medicare = Toileting care	Actions = Update	Permit	Inherited	Not Applicable
ABAC : Midwife_Policy	Role = Midwife	Patient Medicare = Toileting care	Actions = Delete	Permit	Originated	Not Applicable
ABAC : Midwife_Policy	Role = Head Nurse	Patient Medicare = Toileting care	Actions = Delete	Permit	Inherited	Not Applicable
ABAC : Midwife_Policy	Role = Chief of Hospital	Patient Medicare = Toileting care	Actions = Delete	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Record = Prescription	Actions = Delete	Deny	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Record = Prescription	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	[illegible]	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Chief of Hospital	[illegible]	Actions = Create	Permit	Inherited	Not Applicable

Figure 64: Merged Policy Verification Results

Policy Analysis: The policy analysis of the Merged Policy Verification has a similar procedure of the Single Policy Verification and the difference is that Merged Policy Verification involves two policies. Specifically, after clicking the Run Verification icon, the testing results will show up and Figure 64 is an example that includes the following information:

Security Requirement Schema ($Nursing\ Attendant\ Policy\ Tests$): As configured in Figure 63, $Nursing\ Attendant\ Policy\ Tests$ is the selected schema that has 22 Security Requirements that are listed in Table of "Merged Policy Verification Results" in Figure 64. The following Security Requirement is highlighted in the table by blue and we use its result for AC analysis:

$Subject : Role = Nursing\ Attendent; Resource : Patient\ Record = Prescription;$
$Actions = Read; Decision : Permit$ (Denote this rule by SR_2 for illustration purpose.)

Testing Policies: As shown in Figure 63, the Security Requirements in the selected schema (e.g., $Nursing\ Attendant\ Policy\ Tests$) are tested against all rules of two selected policies: $ABAC : Nursing\ Attendant\ Policy$ and $ABAC : Midwife\ Policy$. The rule that is highlighted in Figure 64 is below:

$Subject : Role = Nursing\ Attendent; Resource : Patient\ Record = Prescription;$
$Actions = Delete; Decision : Deny$ (Denote this rule by $Rule_i$ for illustration purpose.)

Policy Algorithms: As shown in Figure 63, the chosen policy algorithms are (i) Rule Combining Algorithm - First Applicable, (ii) Policy Enforcement Algorithm - Deny Biased.

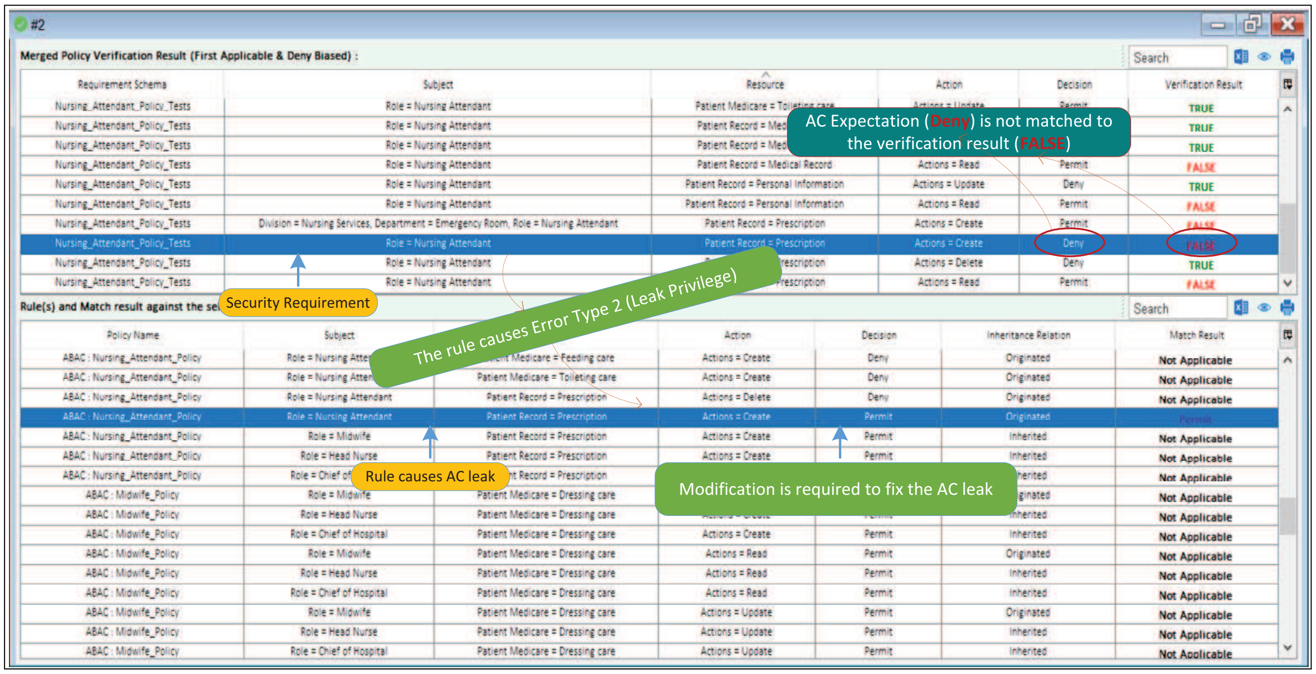
Verification Results: Policy author subsequently reviews the verification result (i.e., $FALSE/TRUE$) of all Security Requirements in the schema. The verification result of SR_2 in the above example is $FALSE$ which is highlighted in Figure 64. As noted in Figure 64, the $FALSE$ is the merged verification result of the Security Requirement SR_2 against all rules, and in other words the $FALSE$

is a combination of all testing results of all rules using rule combination algorithm. According to the verification principle:

$$\begin{cases} AC\ Flaw, & \text{if } SR_2 \text{ is AC Expectation} \\ No\ ACFlaw, & \text{Otherwise.} \end{cases} \tag{11}$$

If the $Nursing\ Attendant$ is $Permit$ted to $Read$ the patient's $Prescription$ as stated in SR_2 is the AC Expectation, there is an AC Flaw since the testing result is $FALSE$. According to the AC flaw Error Type definition, this AC Flaw is Error Type 1 (Block Privilege, see subsection 6.4) as an intended permission of the $Nursing\ Attendant$ is declined by the given policy. As shown in the List of testing result against each rule in Figure 64, the SR_2 testing results against all the rules are $Not\ Applicable$ which means no rule governs the SR_2 security requirement. In order to fix AC flaw, a new rule needs to be added, such as:

$Subject : Role = Nursing\ Attendent; Resource : Patient\ Record = Prescription;$
$Actions = Read; Decision : Deny$
which satisfies the AC expectation.


#2
Search
AC Expectation (Deny) is not matched to the verification result (FALSE)
Security Requirement
The rule causes Error Type 2 (Leak Privilege)
Rule causes AC leak
Modification is required to fix the AC leak

Merged Policy Verification Result (First Applicable & Deny Biased) :

Requirement Schema	Subject	Resource	Action	Decision	Verification Result
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Update	Permit	TRUE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Med	[illegible]	[illegible]	TRUE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Med	[illegible]	[illegible]	TRUE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Medical Record	Actions = Read	Permit	FALSE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Personal Information	Actions = Update	Deny	TRUE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Personal Information	Actions = Read	Permit	FALSE
Nursing_Attendant_Policy_Tests	Division = Nursing Services, Department = Emergency Room, Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	FALSE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Deny	FALSE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	rescription	Actions = Delete	Deny	TRUE
Nursing_Attendant_Policy_Tests	Role = Nursing Attendant	rescription	Actions = Read	Permit	FALSE

Rule(s) and Match result against the se

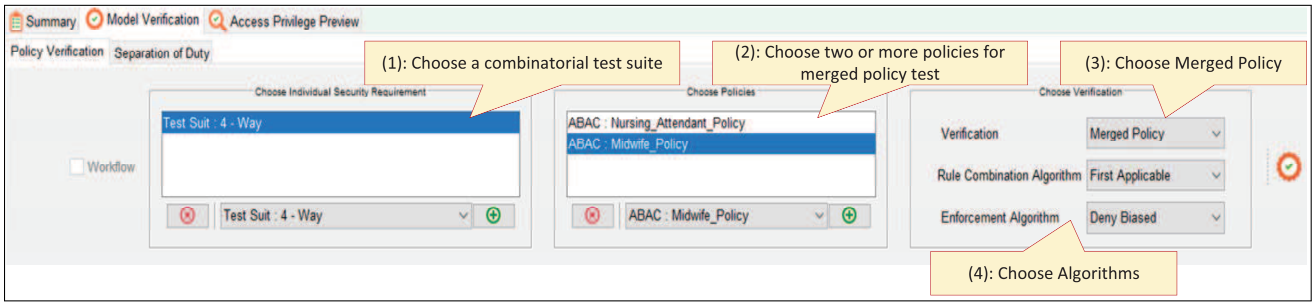
Policy Name	Subject	[illegible]	Action	Decision	Inheritance Relation	Match Result
ABAC : Nursing_Attendant_Policy	Role = Nursing Atte	nt Medicare = Feeding care	Actions = Create	Deny	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Atten	Patient Medicare = Toileting care	Actions = Create	Deny	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Record = Prescription	Actions = Delete	Deny	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	Originated	Permit
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Record = Prescription	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Record = Prescription	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Chief of	t Record = Prescription	[illegible]	[illegible]	herited	Not Applicable
ABAC : Midwife_Policy	Role = Midwife	Patient Medicare = Dressing care	[illegible]	[illegible]	ginated	Not Applicable
ABAC : Midwife_Policy	Role = Head Nurse	Patient Medicare = Dressing care	[illegible]	[illegible]	nherited	Not Applicable
ABAC : Midwife_Policy	Role = Chief of Hospital	Patient Medicare = Dressing care	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Midwife_Policy	Role = Midwife	Patient Medicare = Dressing care	Actions = Read	Permit	Originated	Not Applicable
ABAC : Midwife_Policy	Role = Head Nurse	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited	Not Applicable
ABAC : Midwife_Policy	Role = Chief of Hospital	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited	Not Applicable
ABAC : Midwife_Policy	Role = Midwife	Patient Medicare = Dressing care	Actions = Update	Permit	Originated	Not Applicable
ABAC : Midwife_Policy	Role = Head Nurse	Patient Medicare = Dressing care	Actions = Update	Permit	Inherited	Not Applicable
ABAC : Midwife_Policy	Role = Chief of Hospital	Patient Medicare = Dressing care	Actions = Update	Permit	Inherited	Not Applicable

Figure 65: Example of Error Type 2 (Leak Privilege)

The above policy analysis should be conducted for all Security Requirements. Each security requirement will generate a testing result ($FALSE/True$) and the policy author verify if all the results (i.e., each result in the "List of Verification Result" in Figure 64) are matched with the AC Expectation. If any AC flaw, the policy author can review the testing results of each rule (i.e., each result in the "List of testing result against each rule" in Figure 64) and identify which rules cause the AC flaw and correct them in a way to achieve the expected results. Figure 65 shows an example of Error Type 2 (Leak Privilege) where the AC expectation is:

$Subject : Role = Nursing\ Attendent; Resource : Patient\ Record = Prescription;$
$Actions = Create; Decision : Deny$

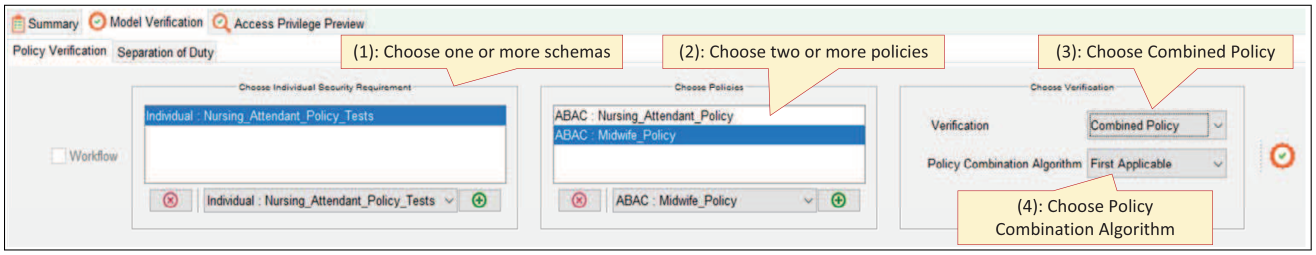
while the testing result is $FALSE$, which causes a consequence that $Nursing\ Attendent$ can $Create$ a $Prescription$, which is supposed the privilege of the Midwife and a $Nursing\ Attendent$ should not have the privilege to create a $Prescription$. In order to fix the AC leak, the policy author clicks the corresponding rule that causes the leakage in the project tree and revises the permission as the AC expectation. After the revision, a new Merged Policy Verification can be conducted to confirmation the correction of the AC leak.


Summary
Model Verification
Access Privilege Preview
Policy Verification
Separation of Duty
Workflow
(1): Choose a combinatorial test suite
(2): Choose two or more policies for merged policy test
(3): Choose Merged Policy
(4): Choose Algorithms
Choose Individual Security Requirement
Choose Policies
Choose Verification

Test Suit : 4 - Way
ABAC : Nursing_Attendant_Policy
ABAC : Midwife_Policy
Verification: Merged Policy
Rule Combination Algorithm: First Applicable
Enforcement Algorithm: Deny Biased

Figure 66: Example of Exhaustive Merged Policy Verification

Exhaustive Merged Policy Verification: Exhaustive Merged Policy Verification is to test two or more policies by merged verification with Combinatorial Security Requirements. Figure 66 shows an example of $4 - way$ Combinatorial Security Requirements. Comparing to Individual Security Requirement, the difference is that the Exhaustive Merged Policy Verification should choose a Combinatorial Test Suite in Step 1 as shown in Figure 66. Other steps are the same as shown in Figure 63. Meanwhile, the policy analysis and leakage correction are the same as the Merged Policy Verification.


Summary
Model Verification
Access Privilege Preview
Policy Verification
Separation of Duty
Workflow
(1): Choose one or more schemas
(2): Choose two or more policies
(3): Choose Combined Policy
(4): Choose Policy Combination Algorithm
Choose Individual Security Requirement
Choose Policies
Choose Verification

Individual : Nursing_Attendant_Policy_Tests
ABAC : Nursing_Attendant_Policy
ABAC : Midwife_Policy
Verification: Combined Policy
Policy Combination Algorithm: First Applicable

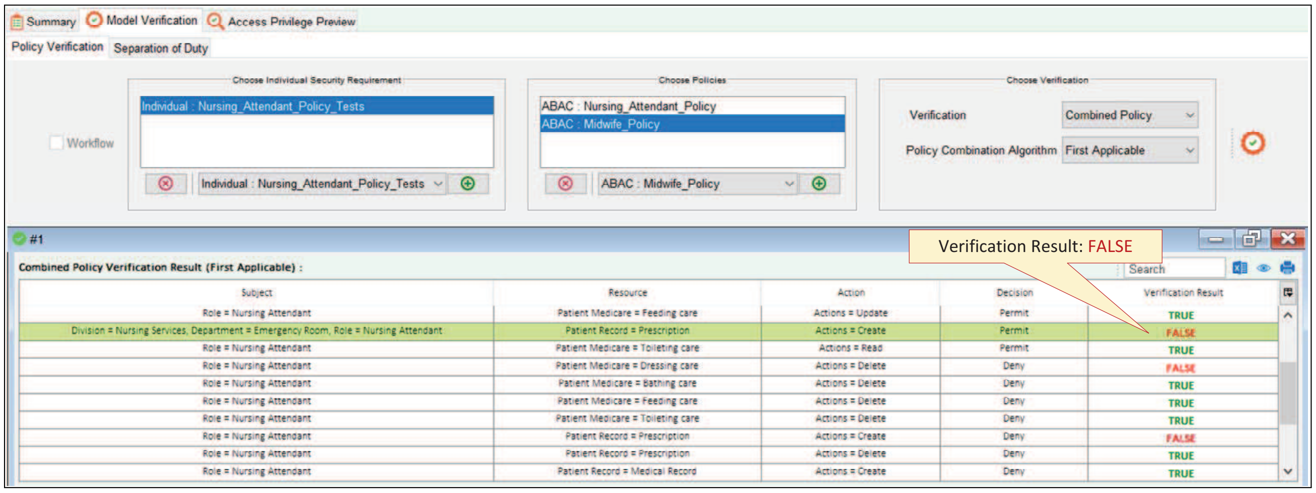
Figure 67: Merged Policy Verification Steps

7.12.5 Combined Policy Verification

Combined Policy Verification verifies the AC effectiveness of Security Requirements against multiple policies and the all the results are combined into a final Decision. Specifically, the final Decision (Permit or Deny) of a Security Requirement is made by two steps: (i) the Security Requirement is first against each rule with each policy and generates a Decision with the use of the policy algorithm, (ii) the Decisions from the individual policies are combined into a Decision through a policy combining algorithm. In the end, the combined Decision is compared with Decision in the Security Requirement for a judge of $TRUE$ or $FALSE$. Please see Figure 10 for the process of the Merged Policy Verification.

Policy Test: Figure 67 shows how the combined policy verification is conducted. The first two steps are the same as the Merged Policy Verification.

- **Choose Security Requirement Schema(s)**: Click "Model Verification" and "Policy Verification" subsequently. Choose a schema from the dropdown menu and add it to the "Choose Security Requirement" by clicking ⊕ button, e.g., *Nursing Attendant Policy Tests*, as shown in Figure 63. One or more schemas can be chosen for the test.
- **Choose Two or More Policies**: Combined Policy Verification should choose with two or more policies (e.g., $Policy\ 1 - n$). This is done by repeatedly choosing a policy from the dropdown menu and clicking ⊕ button to add the policy into "Choose Policies", as shown in Figure 63. Figure 63 shows two policies that are $ABAC : Nursing\ Attendant\ Policy$ and $ABAC : Midwife\ Policy$.
- **Choose Combined Policy and Algorithms**: Upon the selection of two policies, "Combined Policy" can be chosen from "Choose Verification" and it shows up "Policy Combined Algorithm", as shown in Figure 67. From the dropdown menu under "Policy Combined Algorithm", an algorithm, e.g., First Applicable, can be chosen, which will be applied to combine the Decisions from each policy.
- **Start to the Verification**: Finally, it is to click the Run Verification icon and the test starts immediately.



Individual : Nursing_Attendant_Policy_Tests

ABAC : Nursing_Attendant_Policy
ABAC : Midwife_Policy

Verification: Combined Policy
Policy Combination Algorithm: First Applicable

Verification Result: FALSE

Combined Policy Verification Result (First Applicable) :

Subject	Resource	Action	Decision	Verification Result
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Update	Permit	TRUE
Division = Nursing Services, Department = Emergency Room, Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	FALSE
Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Read	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Deny	FALSE
Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Delete	Deny	TRUE
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Delete	Deny	TRUE
Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Delete	Deny	TRUE
Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Deny	FALSE
Role = Nursing Attendant	Patient Record = Prescription	Actions = Delete	Deny	TRUE
Role = Nursing Attendant	Patient Record = Medical Record	Actions = Create	Deny	TRUE

Figure 68: Merged Policy Verification Results

Policy Analysis: Policy analysis of the combined policy verification allows us to: (i) detect the AC flaw, (ii) identify the source of the AC flaw. The sources of a AC flaw could be a rule, the policy, and policy algorithms, or the combination algorithm. Therefore, our policy testing and analysis should have display their correlation clearly. After clicking the Run Verification icon, the testing results will show up. Figure 64 is an example of the following information:

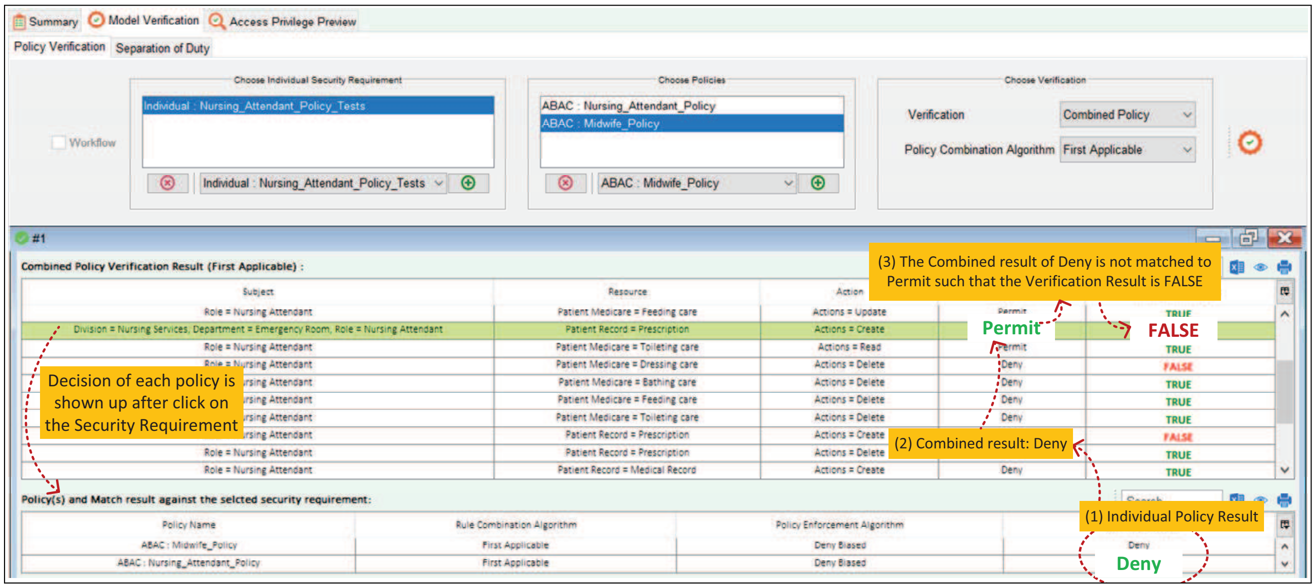
Combined Testing Results: Figure 64 has a table named as "Combined Policy Verification Result" which lists a sequence of Security Requirement and the testing results ($TRUE/FALSE$). As configured in Figure 67, *Nursing Attendant Policy Tests* is the selected schema with a set of Security Requirement. The following Security Requirement is highlighted in the table by yellow and we use its result for AC analysis:

$Subject : Division = Nursing\ Service\ \&Department = Emergency\ Room$
$\&Role = Nursing\ Attendent;$
$Resource : Patient\ Record = Prescription;$
$Actions = Create;$
$Decision : Permit$ (Denote this rule by SR_1 for illustration purpose.)
$Verification\ Result = FALSE$
which says that SR_1 is not Permitted, e.g., it is False for SR_1 to be Permit.

The combined testing results allow policy author to determine if the it is correct or an AC flaw with the following principle:

$$\begin{cases} AC\ Flaw, & \text{if } SR_2 \text{ is AC Expectation \& } Result = TRUE \\ AC\ Flaw, & \text{if } SR_2 \text{ isn't AC Expectation \& } Result = FALSE \\ No\ ACFlaw, & \text{Otherwise.} \end{cases} \tag{12}$$

By following the above principle, the policy author can review all the Security Requirements with identification of the AC flaw. Suppose there is an AC flow. Clicking on the Security Requirement in the table of "Combined Policy Verification Result" will lead to "Policy (s) and Match result against the selected security requirement" table, which is shown in Figure 69.


Summary
Model Verification
Access Privilege Preview
Policy Verification
Separation of Duty
Workflow
#1
(3) The Combined result of Deny is not matched to Permit such that the Verification Result is FALSE
Decision of each policy is shown up after click on the Security Requirement
(2) Combined result: Deny
(1) Individual Policy Result
Permit
FALSE
Deny

Choose Individual Security Requirement: Individual : Nursing_Attendant_Policy_Tests

Choose Policies: ABAC : Nursing_Attendant_Policy; ABAC : Midwife_Policy

Choose Verification: Verification: Combined Policy; Policy Combination Algorithm: First Applicable

Combined Policy Verification Result (First Applicable) :

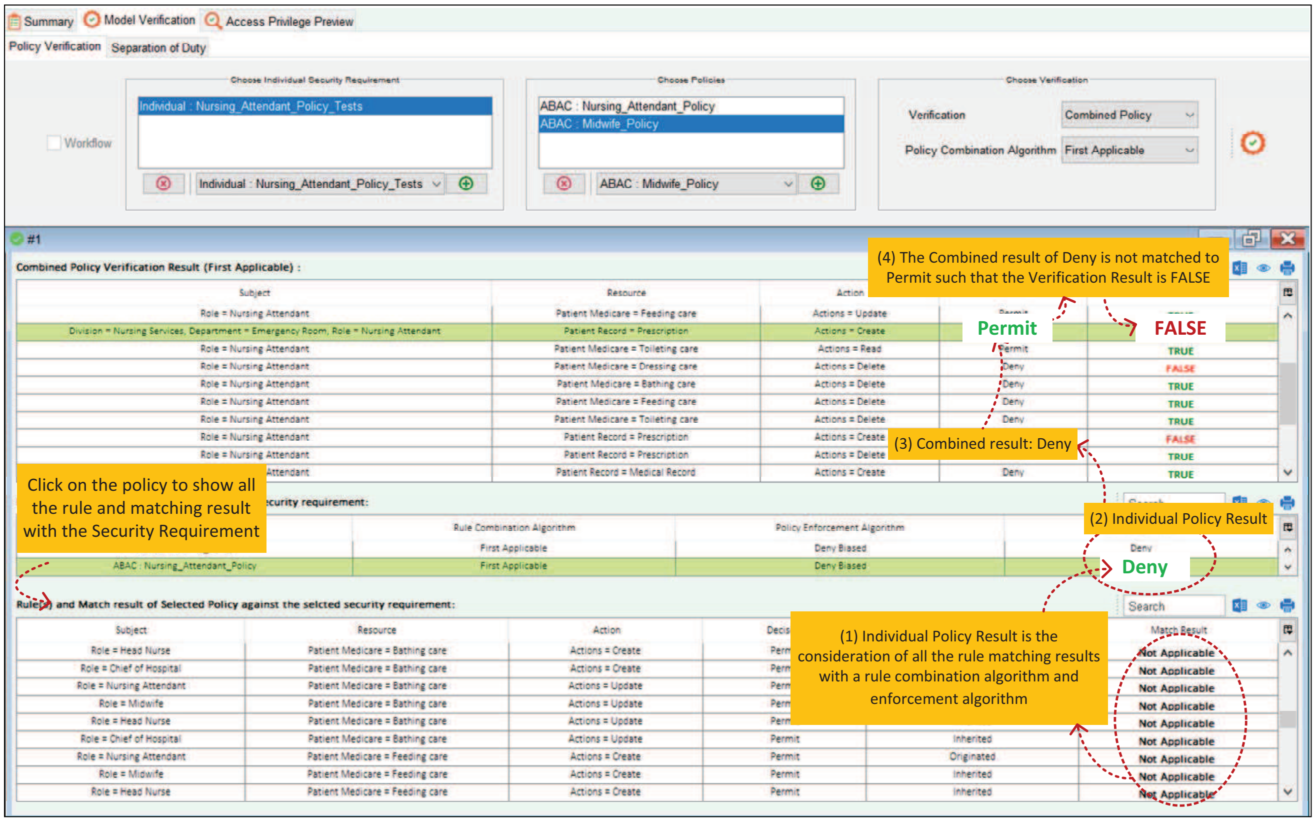
Subject	Resource	Action		
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Update	Permit	TRUE
Division = Nursing Services, Department = Emergency Room, Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	FALSE
Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Read	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Deny	FALSE
rsing Attendant	Patient Medicare = Bathing care	Actions = Delete	Deny	TRUE
rsing Attendant	Patient Medicare = Feeding care	Actions = Delete	Deny	TRUE
rsing Attendant	Patient Medicare = Toileting care	Actions = Delete	Deny	TRUE
rsing Attendant	Patient Record = Prescription	Actions = Create		FALSE
Role = Nursing Attendant	Patient Record = Prescription	Actions = Delete		TRUE
Role = Nursing Attendant	Patient Record = Medical Record	Actions = Create	Deny	TRUE

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm		
ABAC : Midwife_Policy	First Applicable	Deny Biased		Deny
ABAC : Nursing_Attendant_Policy	First Applicable	Deny Biased		Deny

Figure 69: Identify the Policy and the Individual Testing Results

Individual Policy Testing Result: Figure 69 gives a list of the policies and the individual policy testing results. Figure 69 has two policies that are selected for Combined Policy test, i.e., $ABAC$: $Nursing\ Attendant\ Policy$ and $ABAC : Midwife\ Policy$. Both of the policies give an result of Deny. These two individual testing results applies the Policy Combining Algorithm, e.g., First Applicable. It will yield a Deny as the final result, which again is contradict with Permit stated in the SR_2 Decision. The Combined result of Deny is not matched to the Permit such that the Verification Result is FALSE.


Summary
Model Verification
Access Privilege Preview
Policy Verification
Separation of Duty
Workflow
#1
(4) The Combined result of Deny is not matched to Permit such that the Verification Result is FALSE
Permit
FALSE
(3) Combined result: Deny
Click on the policy to show all the rule and matching result with the Security Requirement
(2) Individual Policy Result
Deny
(1) Individual Policy Result is the consideration of all the rule matching results with a rule combination algorithm and enforcement algorithm
Search

Choose Individual Security Requirement: Individual : Nursing_Attendant_Policy_Tests

Choose Policies: ABAC : Nursing_Attendant_Policy; ABAC : Midwife_Policy

Choose Verification — Verification: Combined Policy; Policy Combination Algorithm: First Applicable

Combined Policy Verification Result (First Applicable) :

Subject	Resource	Action		
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Update	Permit	TRUE
Division = Nursing Services, Department = Emergency Room, Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	FALSE
Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Read	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Deny	FALSE
Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Delete	Deny	TRUE
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Delete	Deny	TRUE
Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Delete	Deny	TRUE
Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	[illegible]	FALSE
Role = Nursing Attendant	Patient Record = Prescription	Actions = Delete	[illegible]	TRUE
[illegible] Attendant	Patient Record = Medical Record	Actions = Create	Deny	TRUE

[illegible] ecurity requirement:

	Rule Combination Algorithm	Policy Enforcement Algorithm	
	First Applicable	Deny Biased	Deny
ABAC : Nursing_Attendant_Policy	First Applicable	Deny Biased	Deny

Rule(s) and Match result of Selected Policy against the selcted security requirement:

Subject	Resource	Action	Decis			Match Result
Role = Head Nurse	Patient Medicare = Bathing care	Actions = Create	Perm	[illegible]	[illegible]	Not Applicable
Role = Chief of Hospital	Patient Medicare = Bathing care	Actions = Create	Perm	[illegible]	[illegible]	Not Applicable
Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Update	Perm	[illegible]	[illegible]	Not Applicable
Role = Midwife	Patient Medicare = Bathing care	Actions = Update	Perm	[illegible]	[illegible]	Not Applicable
Role = Head Nurse	Patient Medicare = Bathing care	Actions = Update	Perm	[illegible]	[illegible]	Not Applicable
Role = Chief of Hospital	Patient Medicare = Bathing care	Actions = Update	Permit	Inherited		Not Applicable
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Create	Permit	Originated		Not Applicable
Role = Midwife	Patient Medicare = Feeding care	Actions = Create	Permit	Inherited		Not Applicable
Role = Head Nurse	Patient Medicare = Feeding care	Actions = Create	Permit	Inherited		Not Applicable

Figure 70: Identify the Rules and the Matching Results

Rule against Security Requirement: Clicking on the policy (i.e., $ABAC : Nursing\ Attendant\ Policy$) in the "Policy (s) and Match result against the selected security requirement" table, a new table of "Rule (s) and Match result of the Selected Policy against the selected Security Requirement" is pop out as shown in Figure 70. This table tells how the Individual Policy Result (e.g., Deny) is tested out. The Individual Policy Result is the Security Requirement tests again each policy rule with rule combining algorithm and policy enforcement algorithm.

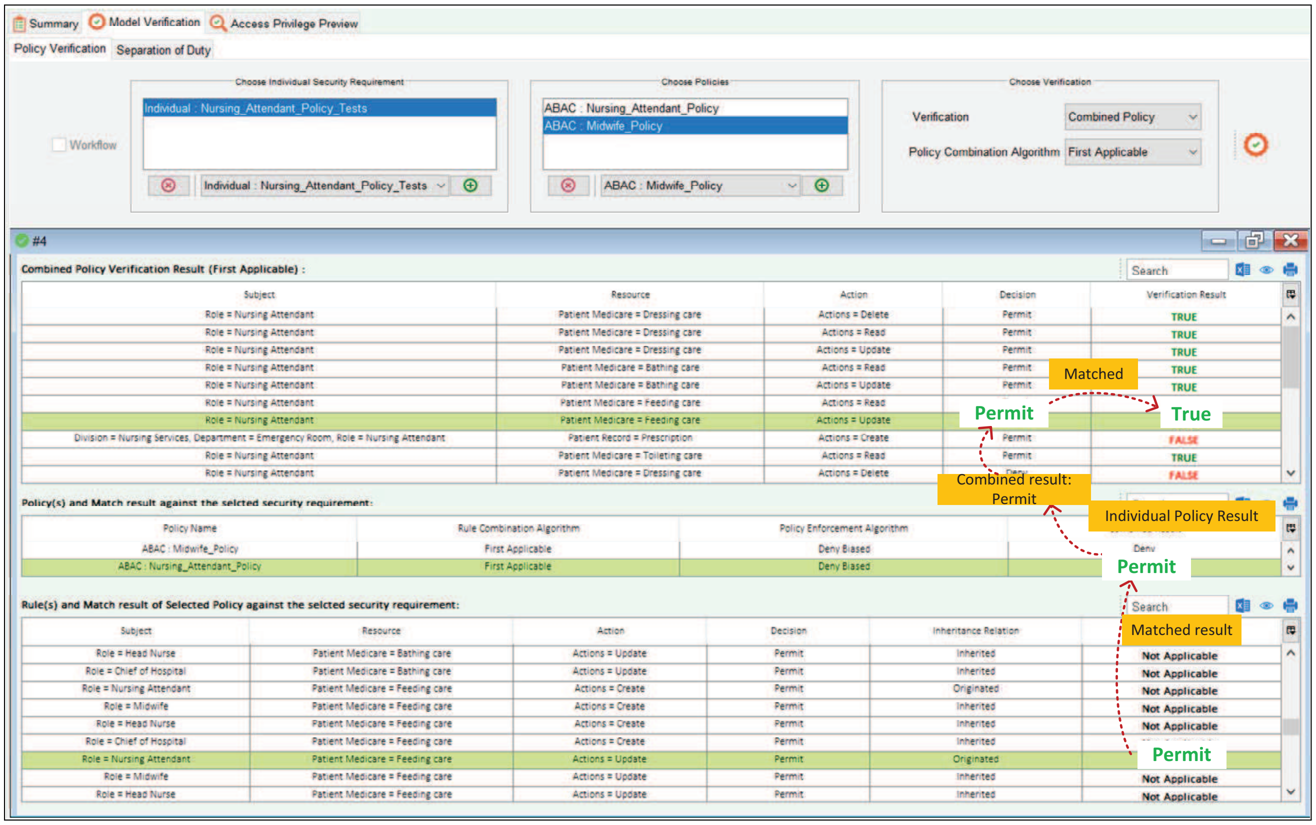
Reversely reviewing the above correlations, we can summarize how the Verification Result is specifically derived:

$$Security\ Requirement \xrightarrow{\text{Test against each rule}} Matched\ Result$$

$$Matched\ Result \xrightarrow{\text{Rule combining algorithm and policy enforcement algorithm}} Individual\ Policy\ Result$$

$$Individual\ Policy\ Result \xrightarrow{\text{Policy Combination Algorithm}} Combined\ Result$$

$$Combined\ Result \xrightarrow{\text{Is the Combined Result matched with the } SR_2 \text{ Decision}} VerificationResult(TRUEorFALSE)$$



Choose Individual Security Requirement: Individual : Nursing_Attendant_Policy_Tests

Choose Policies: ABAC : Nursing_Attendant_Policy; ABAC : Midwife_Policy

Choose Verification — Verification: Combined Policy; Policy Combination Algorithm: First Applicable

Combined Policy Verification Result (First Applicable) :

Subject	Resource	Action	Decision	Verification Result
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Read	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Update	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Read	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Update	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Read	[illegible]	[illegible]
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Update	[illegible]	[illegible]
Division = Nursing Services, Department = Emergency Room, Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	FALSE
Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Read	Permit	TRUE
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Deny	FALSE

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	[illegible]
ABAC : Midwife_Policy	First Applicable	Deny Biased	Deny
ABAC : Nursing_Attendant_Policy	First Applicable	Deny Biased	[illegible]

Rule(s) and Match result of Selected Policy against the selcted security requirement:

Subject	Resource	Action	Decision	Inheritance Relation	[illegible]
Role = Head Nurse	Patient Medicare = Bathing care	Actions = Update	Permit	Inherited	Not Applicable
Role = Chief of Hospital	Patient Medicare = Bathing care	Actions = Update	Permit	Inherited	Not Applicable
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Create	Permit	Originated	Not Applicable
Role = Midwife	Patient Medicare = Feeding care	Actions = Create	Permit	Inherited	Not Applicable
Role = Head Nurse	Patient Medicare = Feeding care	Actions = Create	Permit	Inherited	Not Applicable
Role = Chief of Hospital	Patient Medicare = Feeding care	Actions = Create	Permit	Inherited	[illegible]
Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Update	Permit	Originated	[illegible]
Role = Midwife	Patient Medicare = Feeding care	Actions = Update	Permit	Inherited	Not Applicable
Role = Head Nurse	Patient Medicare = Feeding care	Actions = Update	Permit	Inherited	Not Applicable

Figure 71: Example of Combined Policy Test

The above derivation can be illustrated with the example in Figure 71:

Security Requirement:
$Subject : Role = Nursing\ Attendant; Resource : Patient\ Record = Prescription; Actions = Create; Decision : Permit$ (Denote this rule by SR_3 for illustration purpose.)
Matched Result: Permit (Highlighted in Figure 71) and some results of Not Applicable
Individual Policy Results: Permit for policy $ABAC : Nursing\ Attendant\ Policy$ and Deny for policy $ABAC : Midwife\ Policy$
Combined Result: Permit (First Applicable from the first policy $ABAC : Nursing\ Attendant\ Policy$ yields Permit.
Verification Result = TRUE
Applying the logic in Express 12 says that: If the Permit of the combined result for SR_3 is AC expectation, then it has no flaw according to SR_3 Security Requirement. Otherwise, it is an error and the highlighted rule or the rule or policy algorithms can be modified to fix the error.

The above policy analysis should be conducted for all Security Requirements. As SR_3, each security requirement will generate a testing result ($FALSE/True$) and the above analysis can be employed to check if the verification result is matched with the AC Expectation or not. If any AC flaw, the policy author can review the verification process of the rule and identify which rules or algorithms cause the AC flaw and correct them in a way to achieve the expected results.

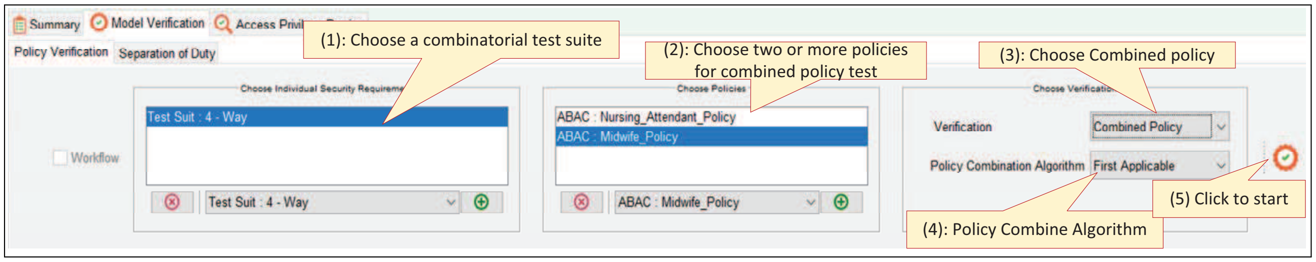

Summary
Model Verification
Access Privi
Policy Verification
Separation of Duty
(1): Choose a combinatorial test suite
(2): Choose two or more policies for combined policy test
(3): Choose Combined policy
Choose Individual Security Requirements
Choose Policies
Choose Verification
Workflow
Test Suit : 4 - Way
ABAC : Nursing_Attendant_Policy
ABAC : Midwife_Policy
Verification
Combined Policy
Policy Combination Algorithm
First Applicable
(5) Click to start
(4): Policy Combine Algorithm

Figure 72: Example of Exhaustive Combined Policy Verification

Exhaustive Combined Policy Verification: Exhaustive Combined Policy Verification is to test two or more policies by combined verification with Combinatorial Security Requirements. Figure 72 shows an example of $4 - way$ Combinatorial Security Requirements. Comparing to Individual Security Requirement, the difference is that the Exhaustive Combined Policy Verification should choose a Combinatorial Test Suite in Step 1 as shown in Figure 72. Step 2 selects the policies, Step 3 chooses the "Combined Policy", and Step 4 is the selection of Policy Combination Algorithm. Finally click the Run Verification icon to start the test. Similar results as shown in Figure 71 will pop up. Meanwhile, the policy analysis and leakage correction are the same as the individual Combined Policy Verification.

7.12.6 Separation of Duty

Separation of Duty test provides a way to avoid Conflicts of Interest associated with conflicting roles. Before conducting a Separation of Duty policy test, Separation of Duty Security Requirements should be generated as illustrated in Subsection 7.11.2. A Separation of Duty schema represents a case of Conflict of Interest, i.e., an AC concern if the Decision of a Security Requirement is conflicted with the Decision of another Security Requirement.

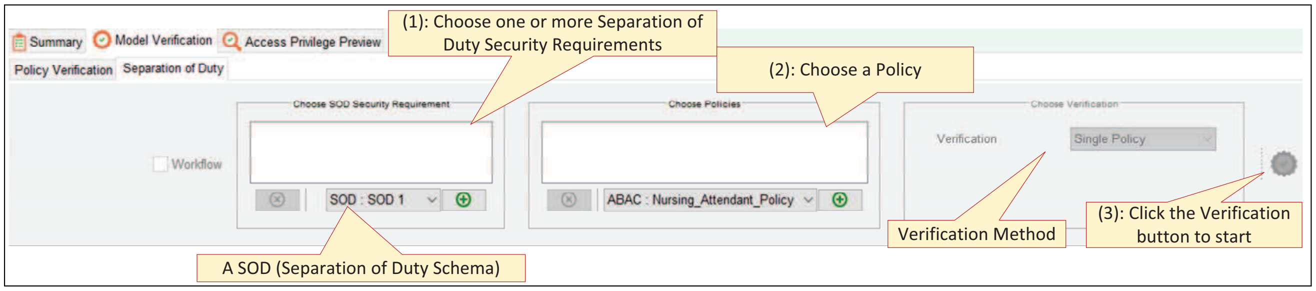


Figure 73: Starting Interface for Separation of Duty Policy Tests

Policy Tests: A Separation of Duty policy test tarts by clicking "Model Verification" and then "Separation of Duty". Figure 73 shows the starting user interface of the Separation of Duty policy test. At first, one or more schemas of SoD (Separation of Duty) Security Requirements should be chosen from the dropdown menu. As such a test is dedicated to Separation of Duty policy tests, all the schemas for Individual Separation Requirements and Combinatorial Security Requirement are not contained in the dropdown menu. After the schema selection of Separation of Duty, one or more policies should be chosen. Separation of Duty policy tests can be conducted by (i) single policy, (ii) merged policy, and (iii) combined policy, depending on the chosen verification method. The composition of these tests are described below. It is noted that the merged policy and combined policy tests are unified to single policy test as the merge and combination of one policy is the same original policy.

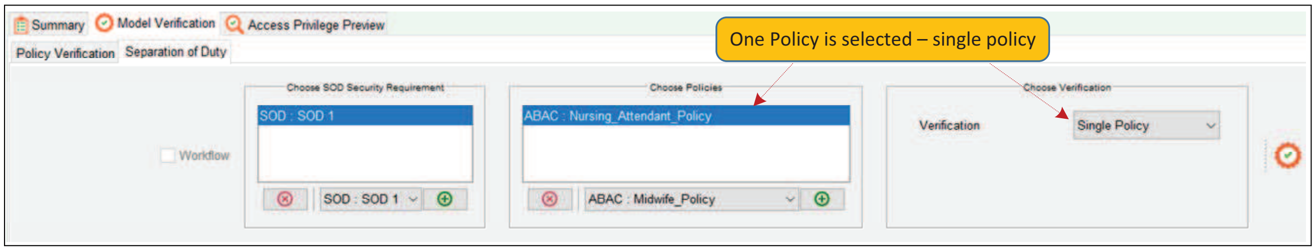


Figure 74: Separation of Duty - Single Policy

Single Policy: This is case that the Security Requirements of a Separation of Duty schema is tested against one chosen policy. It verifies if any Conflict of Interests are caused by the selected policy. Figure 74 shows an example that $SOD\ 1$ is tested against the policy of $Nursing\ Attendant\ Policy$. You click on the Run Verification icon to start the test.

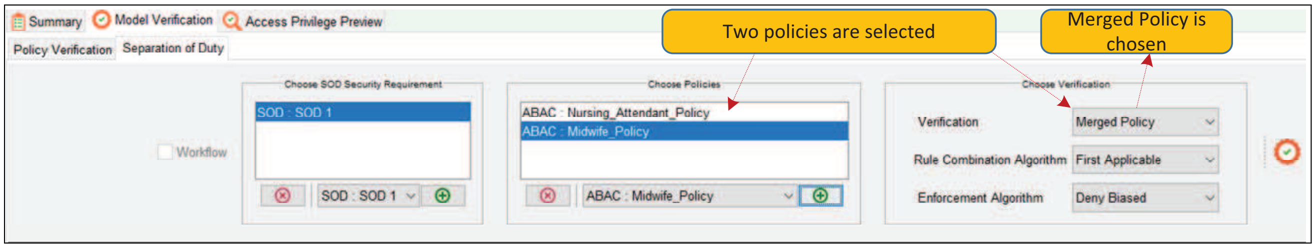


Figure 75: Separation of Duty - Merged Policy

Merged Policy: The Security Requirements of the Separation of Duty can be verified against two or more policies. This involves a Merged Policy or Combined Policy test. Once two or policies are chosen, "Choose Verification" will show up three dropdown menus for Verification, Rule Combining Algorithm, and Enforcement Algorithm respectively. "Merged Policy" is chosen for Merged Policy Verification. In this case, the Separation of Duty is evaluated over the merged policy which verifies Conflict of Interest by merging all policies as one. Figure 75 shows two selected policies (i) $ABAC\ :\ Nursing\ Attendant\ Policy$ and (ii) $ABAC\ :\ Midwife\ Policy$. It considers all the rules of all policies and then applies the chosen rule combining algorithm and enforcement algorithm, e.g., First Applicable and Deny Biased as shown in Figure 75. In the end, clicking on the Run Verification icon starts the test.

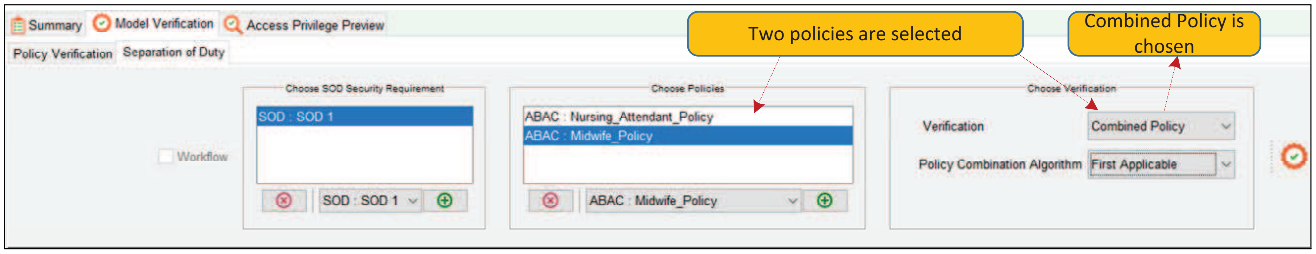
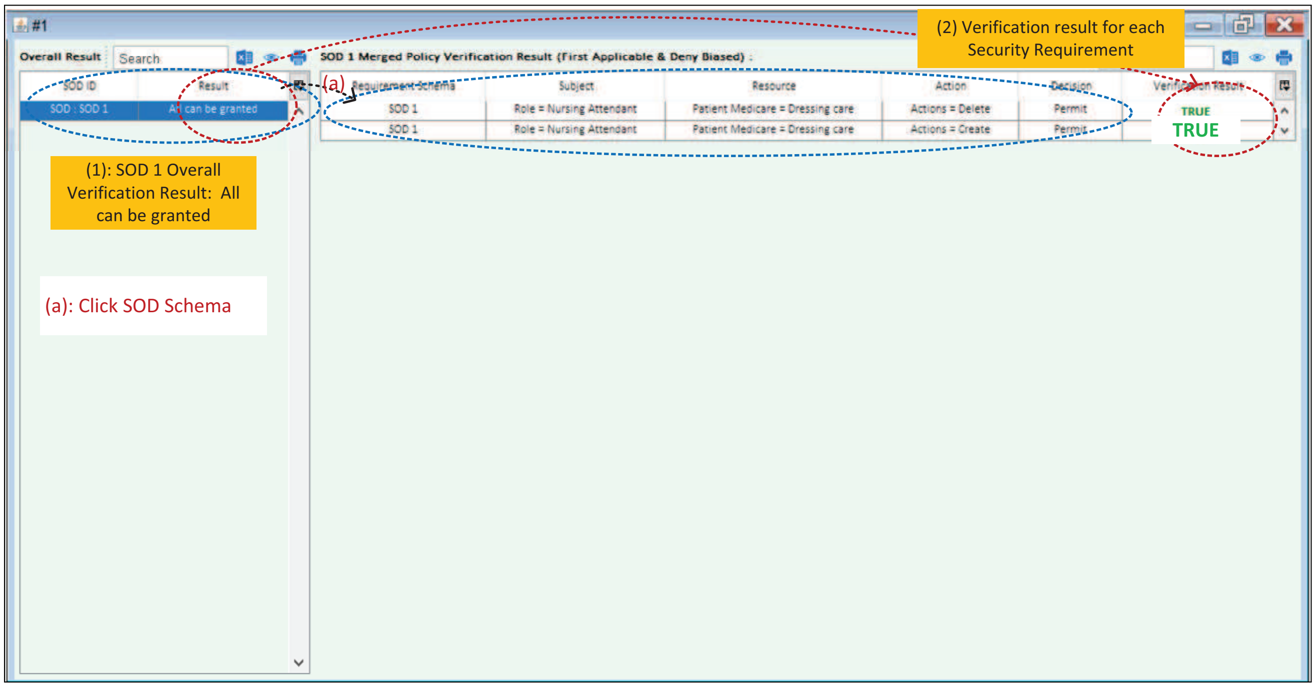


Figure 76: Separation of Duty - Combined Policy

Combined Policy: In addition to Merged Policy, "Combined Policy" can be chosen for Combined Policy Verification, as shown in Figure 75. When Combined Policy Verification is applied, each Security Requirement will be individually verified with each policy with the use of its own Rule Combining Algorithm and Enforcement Algorithm (see Figure 10). The Rule Combining Algorithm and Enforcement Algorithms for two different policies (e.g., $ABAC : Nursing\ Attendant\ Policy$ and $ABAC : Midwife\ Policy$) are separately configured when these policies are composed, and thus they could be different. Then, a Policy Combination Algorithm will be utilized to integrate the verification results of all the policies, e.g., First Applicable as shown in Figure 75. In the end, clicking on the Run Verification icon starts the test.


#1
Overall Result
Search
(1): SOD 1 Overall Verification Result: All can be granted
(a): Click SOD Schema
(a)
(2) Verification result for each Security Requirement
TRUE

SOD ID	Result
SOD : SOD 1	All can be granted

SOD 1 Merged Policy Verification Result (First Applicable & Deny Biased) :

Requirement Schema	Subject	Resource	Action	Decision	Verification Result
SOD 1	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Permit	TRUE
SOD 1	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Create	Permit	

Figure 77: Separation of Duty Verification Result

Policy Analysis: The policy analysis of the Separation of Duty policy test is to identify if there are Conflict of Interests hidden in one ore more policies. The Conflict of Interests, i.e., Error Type 8 (Separation of Duty Error), is indicated by the Separation of Duty overall results:

$$Conflict\ of\ Interest = \begin{cases} Yes & \text{if } SR_i \text{ and } SR_j\text{'s verification results are conflict, } i \neq j \\ No & \text{Otherwise.} \end{cases} \tag{13}$$

where SR_i and SR_j represent two different Security Requirements in a Separation of Duty Schema.

Equation 13 means that an error occurs if the SR_i's verification result is conflicted with that of SR_j. We use an example in Figure 75 to illustrate the above principle. Figure 77 shows an "Overall Result" table that lists the overall verification results of each Separation of Duty, e.g., $SOD\ 1$. As shown in Figure 75, the overall verification result is:

$All\ can\ be\ granted,$

which indicates a Conflict of Interest if only one can be granted due to conflict. $All\ can\ be\ granted$ says that all Security Requirements are $TRUE$. It indicates

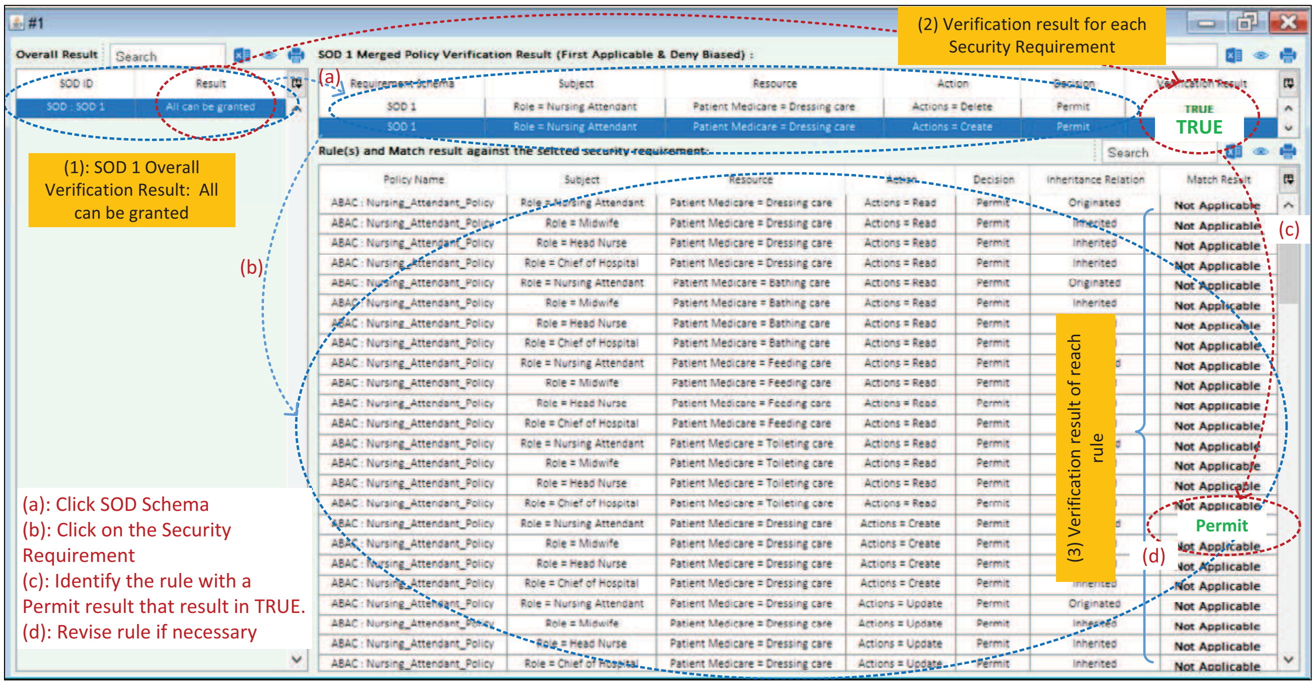
$Conflict\ of\ Interest = Yes,$

if the Conflict of Interest is defined that these two Security Requirements cannot be permitted together, such as two actions (e.g., create and approve an invoice) cannot be taken by the same person. Clicking "SOD:SOD 1" in the "Overall Result" table, it shows a table of "SOD Merged Policy Verification Result (First Applicable & Deny Biased", which lists two Security Requirements that are configured in Figure 75:

$Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicate = Dressing\ care;$
$Actions = Delete; Decision : Permit$ (Denote this rule by SR_1 for illustration purpose.)
$Subject : Role = Nursing\ Attendent; Resource : Patient\ Medicate = Dressing\ care;$
$Actions = Create; Decision : Permit$ (Denote this rule by SR_2 for illustration purpose.) where the verifications of these two Security Requirements are $TRUE$, e.g., permissions are given to two actions together.

More specifically, Conflict of Interest appears if a $Nursing\ Attendent$ cannot take $Create$ and $Delete$ actions in the Nursing Service practice, due to discipline and liability for malpractice, breach of fiduciary duty, or any other reasons. In other words, SR_1 and SR_2 cannot be $TRUE$ together. However, no Conflict of Interest if it is allowable for $Nursing\ Attendent$ to take both $Create$ and $Delete$ actions. Therefore, the determination of Separation of Duty error relies on the actual AC practice to evaluate if two or more verification results are conflict or not. A similar example can be given as below:

$Subject : Loan\ Manager; Resource : Client = Credit;$
$Actions = Update; Decision : Permit$ (Verification results= $TRUE$)
$Subject : Loan\ Manager; Resource : Client = Mortgage;$
$Actions = Approve; Decision : Permit$ (Verification results= $TRUE$.)
where Conflict of Interest is stated by a loan manager cannot change a client's credit score and meanwhile play an impact on the mortgage approvement. This is because increasing the credit may low the mortgage rate and the load manager can abuse it to benefit himself/herself and the client while infringing the bank's benefit.


#1
(2) Verification result for each Security Requirement
(1): SOD 1 Overall Verification Result: All can be granted
(3) Verification result of reach rule
(a)
(b)
(c)
(d)
(a): Click SOD Schema
(b): Click on the Security Requirement
(c): Identify the rule with a Permit result that result in TRUE.
(d): Revise rule if necessary
Search

Overall Result

SOD ID	Result
SOD : SOD 1	All can be granted

SOD 1 Merged Policy Verification Result (First Applicable & Deny Biased) :

Requirement Schema	Subject	Resource	Action	Decision	Verification Result
SOD 1	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Permit	TRUE
SOD 1	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Create	Permit	TRUE

Rule(s) and Match result against the seltced security requirement:

Policy Name	Subject	Resource	Action	Decision	Inheritance Relation	Match Result
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Read	Permit	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Chief of Hospital	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Read	Permit	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Medicare = Bathing care	Actions = Read	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Medicare = Bathing care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Chief of Hospital	Patient Medicare = Bathing care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Medicare = Feeding care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Medicare = Feeding care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Medicare = Feeding care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Chief of Hospital	Patient Medicare = Feeding care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Medicare = Toileting care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Medicare = Toileting care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Chief of Hospital	Patient Medicare = Toileting care	Actions = Read	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Create	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Medicare = Dressing care	Actions = Create	Permit	[illegible]	Permit
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Medicare = Dressing care	Actions = Create	Permit	[illegible]	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Chief of Hospital	Patient Medicare = Dressing care	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Update	Permit	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Medicare = Dressing care	Actions = Update	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Medicare = Dressing care	Actions = Update	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Chief of Hospital	Patient Medicare = Dressing care	Actions = Update	Permit	Inherited	Not Applicable

Figure 78: Separation of Duty Error and Correction

Separation of Duty Policy Correction: When a Separation of Duty error is identified by following the principle of Conflict of Interest, the policy author can further analyze the policy rules and revise them. Figure 78 shows that the verification results are presented in three tables, namely (i) "Overall Result", (ii) "SOD i Merged Policy Verification Result", and (iii) "Result (s) and Matched result against the selected security requirement". The policy analysis and error corrections has the following steps:

- **Separation of Duty Error Detection**: Clicking the Separation of Duty schema, e.g., "SOD: SOD $i, i = 1, \cdots,$", Step (a) as shown in Figure 78, a table titled "SOD i Merged Policy Verification Result" appears. It then applies the principle in Equation 13 with the consideration of the Security Requirements and their verification results ($TRUE/FALSE$) to identify if any Separation of Duty error. If no error, reviewing the next SOD schema till all schemas are reviewed and no errors are found.

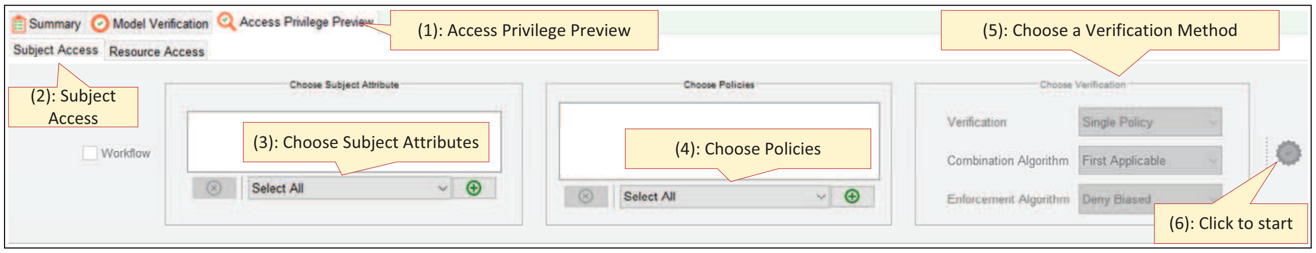
- **Policy Correction**: If an "SOD i is found with Separation of Duty Error, it needs to identify the rule (s) that cause the error. The corresponding operations are Step $b, c, andd$ as shown in Figure 78. Step b clicks on the Security Requirement (e.g., SR_2) that has a conflict result, e.g., SR_2 with $TRUE$ that is highlighted by blue in Figure 78. A table of "Result (s) and Matched result against the selected security requirement" appears and this table presents all the verification results (Permit, Deny, or Not Applicable). Step c reviews the rules and the verification results against the Security Requirement. The rules that give $TRUE$ or $FALSE$ can be identified and their impacts on the results can be checked. As shown in Figure 78, the rule that yields a Permit (highlighted by green) causes a $TRUE$ (Highlight by green) as the SR_2's verification result.

- **Separation of Duty Resolve**: Finally, the rule yielding the Conflict of Interest can be revised to resolve the Separation of Duty error. Similarly, it would be a solution to revise the rule associated with SR_2's result to solve the conflict.

It is noted it has no exhaustive tests for Separation of Duty. This is because Combinatorial Security Requirements are not Separation of Duty schemas. The definition of a Separation of Duty Security Requirement schema is nontrivial. The policy author needs to identify the Separation of Duty cases based on the practice of the access control system, properly define the schemas of multiple Security Requirements, and finally test them.

7.12.7 Subject Access Privilege Preview

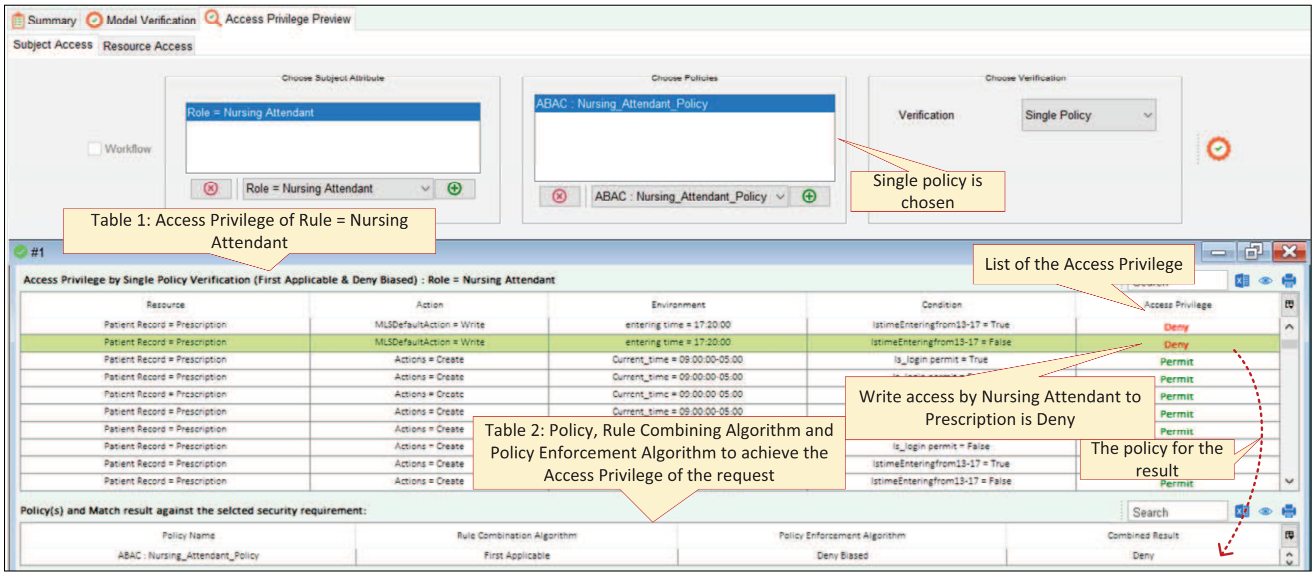
Access privilege preview is to check the privilege of subjects or resources that satisfy certain attribute values. It includes (i) Subject Privilege Preview and (ii) Resource Privilege Preview. Please refer to Subsection 6.3.6 for the principle.



Verification: Single Policy
Combination Algorithm: First Applicable
Enforcement Algorithm: Deny Biased

Figure 79: Subject Access Privilege Preview

Subject Privilege Preview is to preview the resource accessibility of given subject with certain attribute values, e.g., previewing the list of data resources that an employee can access, which provides a way for a policy author to verify if the intended AC security goals are achieved. Figure 79 shows the steps to generate a Subject Access Privilege Preview: (1) Click on "Access Privilege Preview", (2) Click on the "Subject Access", (3) Choose a set of Subject Attributes from the dropdown menu under "Choose Subject Attribute", (4) Choose a set of policies from the dropdown menu under "Choose Policy (Drag to reorder)" box, (5) Choose a verification method, (6) Click the Vellication button to start a query.


Summary
Model Verification
Access Privilege Preview
Subject Access
Resource Access
Choose Subject Attribute
Choose Policies
Choose Verification
Workflow
Single policy is chosen
Table 1: Access Privilege of Rule = Nursing Attendant
#1
List of the Access Privilege
Write access by Nursing Attendant to Prescription is Deny
Table 2: Policy, Rule Combining Algorithm and Policy Enforcement Algorithm to achieve the Access Privilege of the request
The policy for the result
Search

Role = Nursing Attendant
ABAC : Nursing_Attendant_Policy
Verification: Single Policy

Access Privilege by Single Policy Verification (First Applicable & Deny Biased) : Role = Nursing Attendant

Resource	Action	Environment	Condition	Access Privilege
Patient Record = Prescription	MLSDefaultAction = Write	entering time = 17:20:00	IstimeEnteringfrom13-17 = True	Deny
Patient Record = Prescription	MLSDefaultAction = Write	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Deny
Patient Record = Prescription	Actions = Create	Current_time = 09:00:00-05:00	Is_login permit = True	Permit
Patient Record = Prescription	Actions = Create	Current_time = 09:00:00-05:00	[illegible]	Permit
Patient Record = Prescription	Actions = Create	Current_time = 09:00:00-05:00	[illegible]	Permit
Patient Record = Prescription	Actions = Create	Current_time = 09:00:00-05:00	[illegible]	Permit
Patient Record = Prescription	Actions = Create	[illegible]	[illegible]	Permit
Patient Record = Prescription	Actions = Create	[illegible]	Is_login permit = False	[illegible]
Patient Record = Prescription	Actions = Create	[illegible]	IstimeEnteringfrom13-17 = True	[illegible]
Patient Record = Prescription	Actions = Create	[illegible]	IstimeEnteringfrom13-17 = False	Permit

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
ABAC : Nursing_Attendant_Policy	First Applicable	Deny Biased	Deny

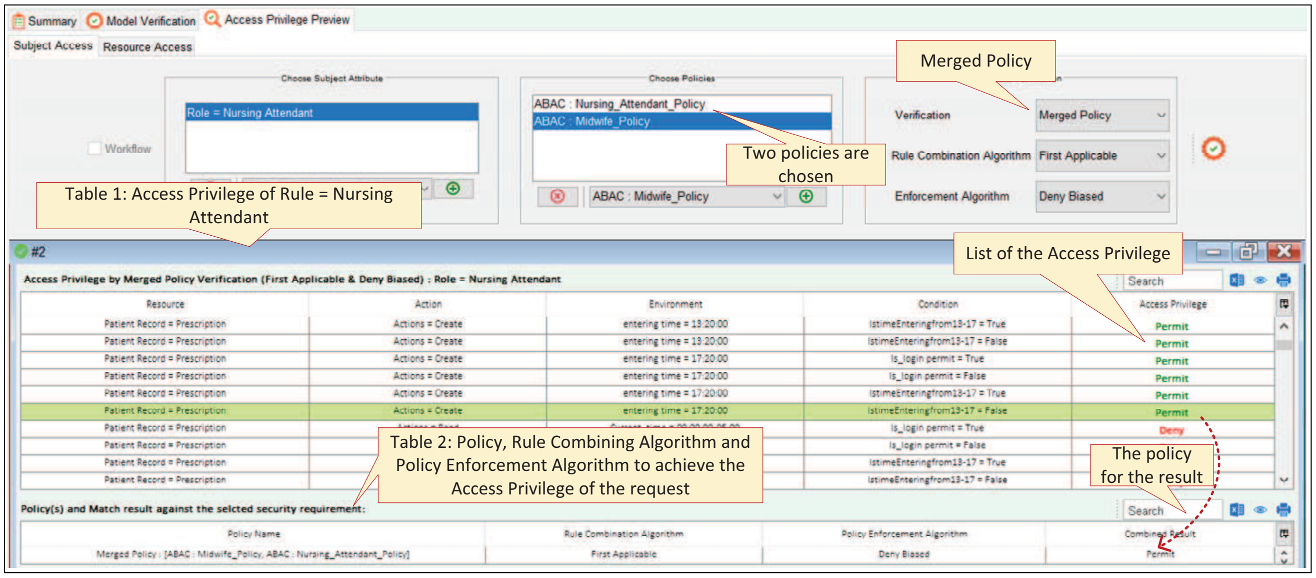
Figure 80: Subject Access Privilege Preview by Single Policy

Single Policy: Figure 80 shows the configuration of Subject Privilege Preview from a single policy. As shown in Figure 80, subject attribute value of $Role = Nursing\ Attendant$ is given. This is query what the Resource that $Role = Nursing\ Attendant$ can access with what actions. As only one policy is chosen for the query, the verification method is "Single Policy", as shown in Figure 80. After clicking on the Run Verification icon, the access privilege results are demonstrated that match the query. Particularly, the access privilege query is an verification process and the results are shown in two tables:

- **Access Privilege by Single Policy Verification**: This table shows all possible accessing requests of the given subject (e.g., $Role = Nursing\ Attendant$) and the results of access privilege (Deny or Permit). The example highlighted in Figure 80 can be read as an $Write-Action$ *access by* $Nursing\ Attendant - Subject$ *to* $Prescription - Resource$ *is* $Deny - Privilige$.

- **Policy (s) and Matching result against the selected security requirement**: This table shows the select policy configuration including the Policy Name, Rule Combining Algorithm, and Policy Enforcement algorithm.

Subject Privilege Preview answers the question the access privilege of a subject, e.g. a role, and from which the policy author can check if any AC flaws. The Subject Privilege Preview results can be exported as Excel table, enlarged view, and printout.


Summary
Model Verification
Access Privilege Preview
Subject Access
Resource Access
Choose Subject Attribute
Workflow
Choose Policies
Merged Policy
Two policies are chosen
Table 1: Access Privilege of Rule = Nursing Attendant
List of the Access Privilege
Table 2: Policy, Rule Combining Algorithm and Policy Enforcement Algorithm to achieve the Access Privilege of the request
The policy for the result
#2
Search

Role = Nursing Attendant

ABAC : Nursing_Attendant_Policy
ABAC : Midwife_Policy

ABAC : Midwife_Policy

Verification: Merged Policy
Rule Combination Algorithm: First Applicable
Enforcement Algorithm: Deny Biased

Access Privilege by Merged Policy Verification (First Applicable & Deny Biased) : Role = Nursing Attendant

Resource	Action	Environment	Condition	Access Privilege
Patient Record = Prescription	Actions = Create	entering time = 13:20:00	IstimeEnteringfrom13-17 = True	Permit
Patient Record = Prescription	Actions = Create	entering time = 13:20:00	IstimeEnteringfrom13-17 = False	Permit
Patient Record = Prescription	Actions = Create	entering time = 17:20:00	Is_login permit = True	Permit
Patient Record = Prescription	Actions = Create	entering time = 17:20:00	Is_login permit = False	Permit
Patient Record = Prescription	Actions = Create	entering time = 17:20:00	IstimeEnteringfrom13-17 = True	Permit
Patient Record = Prescription	Actions = Create	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Permit
Patient Record = Prescription	Actions = Read	[illegible]	Is_login permit = True	Deny
Patient Record = Prescription	[illegible]	[illegible]	Is_login permit = False	[illegible]
Patient Record = Prescription	[illegible]	[illegible]	IstimeEnteringfrom13-17 = True	[illegible]
Patient Record = Prescription	[illegible]	[illegible]	IstimeEnteringfrom13-17 = False	[illegible]

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
Merged Policy : [ABAC : Midwife_Policy, ABAC : Nursing_Attendant_Policy]	First Applicable	Deny Biased	Permit

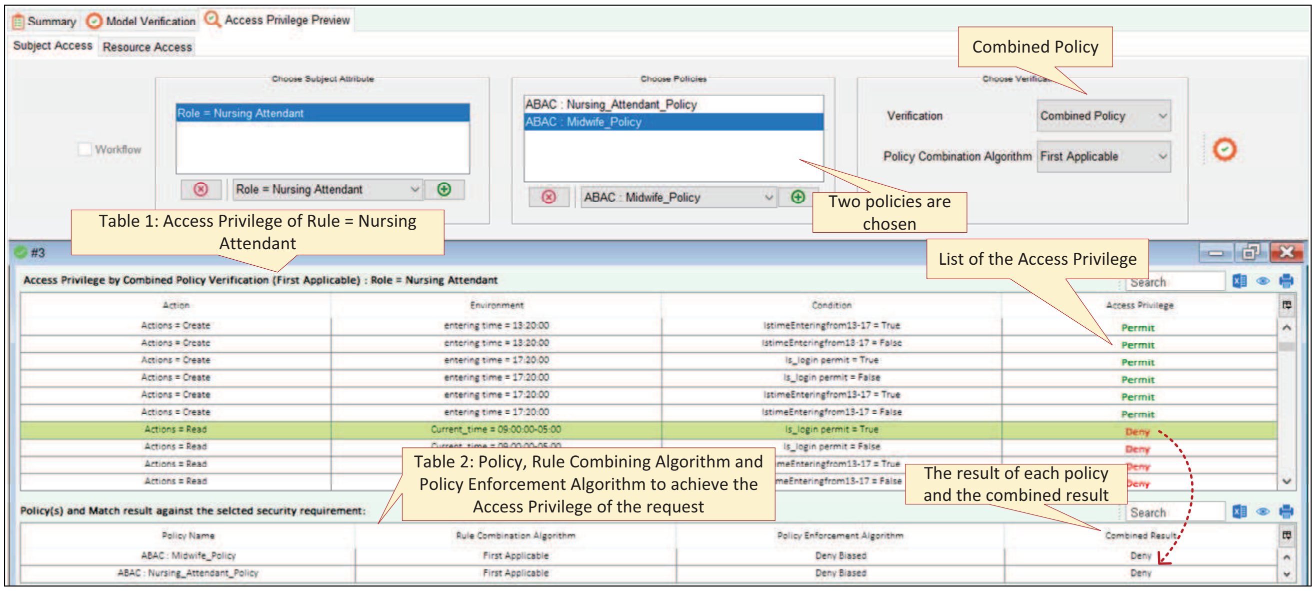
Figure 81: Subject Access Privilege Preview by Merged Policy

Merged Policy: Figure 81 shows the configuration and results of Subject Privilege Preview with merged policy. Similar to Figure 80, it considers one subject attribute value, i.e., $Role = Nursing\ Attendant$, and query what the Resource that $Role = Nursing\ Attendant$) can access with what actions. Differently, two policies are chosen and these two policies are merged as one by "Merged Policy", as shown in Figure 81. The merged policy is configured with rule combination algorithm and policy enforcement algorithm, i.e., First Applicable and Deny Biased respectively. After clicking on the Run Verification icon, the access privilege results are demonstrated that match the query. Particularly, the results are shown in two tables:

- **Access Privilege by Merged Policy Verification**: This table shows all possible accessing requests of the given subject (e.g., $Role = Nursing\ Attendant$) and the results of access privilege (Deny or Permit) against all rules of two policies. The highlighted line in Figure 81 can be read as an request of $Nursing\ Attendant - Subject$ for $Create - Action$ *to* $Prescription - Resource$ *is* $Permit - AccessPrivilige$. If such a Decision is not the intended one, the policy author can revise the rule in order to achieve an expected result, e.g., Deny. The policy author can review all the results to check if they satisfy the AC security requirement.

- **Policy (s) and Matching result against the selected security requirement**: This table shows the select policy configuration including the Policy Names, Rule Combining Algorithm, and Policy Enforcement algorithm.

The policy author can review all the results to check if they satisfy all the AC security requirements.


Summary
Model Verification
Access Privilege Preview
Subject Access
Resource Access
Choose Subject Attribute
Workflow
Choose Policies
Choose Verification
Combined Policy
Two policies are chosen
Table 1: Access Privilege of Rule = Nursing Attendant
#3
List of the Access Privilege
Search
Table 2: Policy, Rule Combining Algorithm and Policy Enforcement Algorithm to achieve the Access Privilege of the request
The result of each policy and the combined result
Search

Role = Nursing Attendant

ABAC : Nursing_Attendant_Policy
ABAC : Midwife_Policy

Verification: Combined Policy
Policy Combination Algorithm: First Applicable

Access Privilege by Combined Policy Verification (First Applicable) : Role = Nursing Attendant

Action	Environment	Condition	Access Privilege
Actions = Create	entering time = 13:20:00	IstimeEnteringfrom13-17 = True	Permit
Actions = Create	entering time = 13:20:00	IstimeEnteringfrom13-17 = False	Permit
Actions = Create	entering time = 17:20:00	Is_login permit = True	Permit
Actions = Create	entering time = 17:20:00	Is_login permit = False	Permit
Actions = Create	entering time = 17:20:00	IstimeEnteringfrom13-17 = True	Permit
Actions = Create	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Permit
Actions = Read	Current_time = 09:00:00-05:00	Is_login permit = True	Deny
Actions = Read	Current_time = 09:00:00-05:00	Is_login permit = False	Deny
Actions = Read	[illegible]	meEnteringfrom13-17 = True	Deny
Actions = Read	[illegible]	meEnteringfrom13-17 = False	Deny

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
ABAC : Midwife_Policy	First Applicable	Deny Biased	Deny
ABAC : Nursing_Attendant_Policy	First Applicable	Deny Biased	Deny

Figure 82: Subject Access Privilege Preview by Combined Policy

Combined Policy: Figure 82 shows the configuration of Subject Privilege Preview by combined policy. Different than merged policy, combined policy tests the query against each policy and the results are combined by policy combining algorithm. Figure 80 shows the same query as in Figure 81 with a subject attribute values of $Role = Nursing\ Attendant$: what the Resource that $Role = Nursing\ Attendant$) can access with what actions.The verification method is "Combined Policy" as shown in Figure 80. After clicking on the Run Verification icon, the access privilege results are shown in two tables:

- **Access Privilege by Single Policy Verification**: It shows all possible accessing requests for $Role = Nursing\ Attendant$ with access privilege (Deny or Permit). The highlighted line in Figure 81 can be read as an request of $Nursing\ Attendant - Subject$ for $Create - Action$ *to* $Prescription - Resource$ *is* $Permit - AccessPrivilige$.

- **Policy (s) and Matching result against the selected security requirement**: This table shows two policies, including the Policy Name, Rule Combining Algorithm, and the Policy

Enforcement algorithm. Meanwhile, it shows the verification results of these two policies, and both results are *Deny*. The *Deny* result in the table of “Access Privilege by Single Policy Verification” is the combined results of *Deny* and *Deny* in the table of “Policy (s) and Matching result against the selected security requirement” via the Policy Combination Algorithm. In other words, the privilege result is the combined results of all policy through the policy combination algorithm.

Subject Privilege Preview answers the question the access privilege of a subject, e.g. a role, and from which the policy author can check if any AC flaws. The Subject Privilege Preview results can be exported as Excel table, enlarged view, and printout.

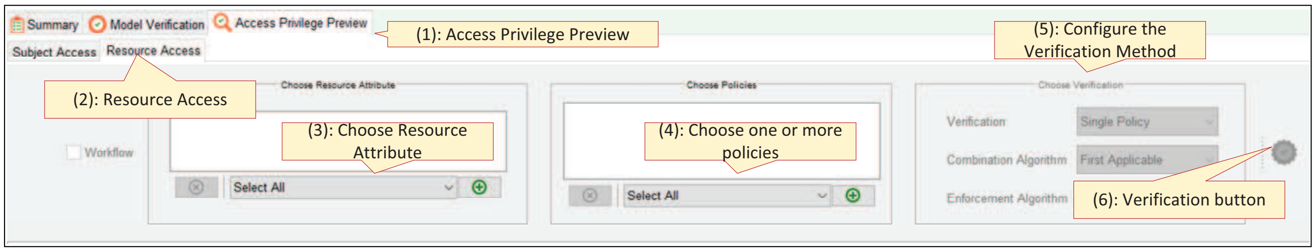
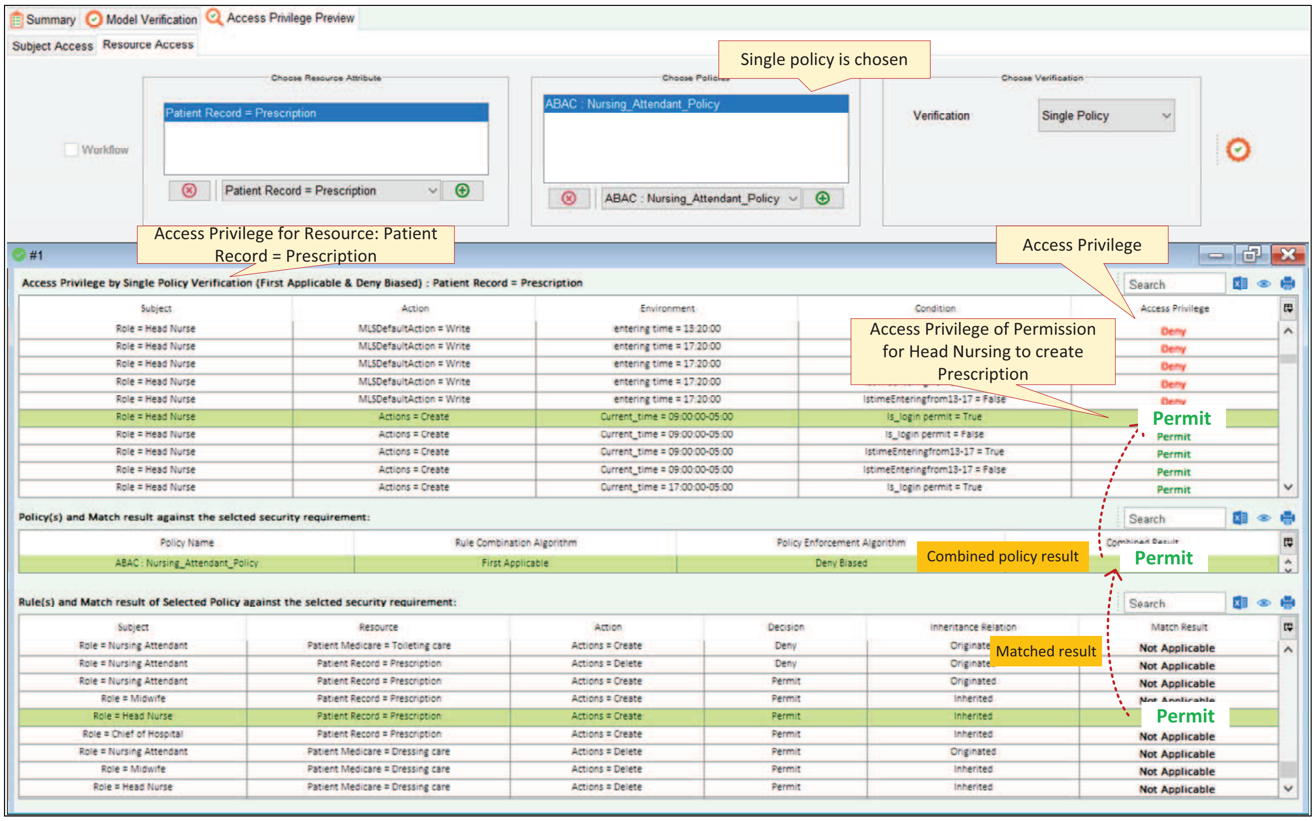


Figure 83: Resource Access Privilege Preview

7.12.8 Resource Access Privilege Preview

Resource Privilege Preview is to preview who (e.g., the subject attribute) can access a given resource, e.g., who can access a resource, and what action can be taken for the specific resource, which provides a way for a policy author to verify if the resource is really protected as intention. Figure 83 shows the steps to generate a Resource Access Privilege Preview: (1) Click “Access Privilege Preview”, (2) Click the “Resource Access”, (3) Choose a set of Resource Attributes from the dropdown menu under “Choose Resource Attribute”, (4) Choose a set of policies from the dropdown menu under “Choose Policy (Drag to reorder)” box, (5) Configure a verification method, (6) Click the Vellication button to start a query.



Choose Resource Attribute: Patient Record = Prescription

Choose Policies: ABAC : Nursing_Attendant_Policy

Verification: Single Policy

Access Privilege by Single Policy Verification (First Applicable & Deny Biased) : Patient Record = Prescription

Subject	Action	Environment	Condition	Access Privilege
Role = Head Nurse	MLSDefaultAction = Write	entering time = 13:20:00		Deny
Role = Head Nurse	MLSDefaultAction = Write	entering time = 17:20:00		Deny
Role = Head Nurse	MLSDefaultAction = Write	entering time = 17:20:00		Deny
Role = Head Nurse	MLSDefaultAction = Write	entering time = 17:20:00		Deny
Role = Head Nurse	MLSDefaultAction = Write	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Deny
Role = Head Nurse	Actions = Create	Current_time = 09:00:00-05:00	Is_login permit = True	Permit
Role = Head Nurse	Actions = Create	Current_time = 09:00:00-05:00	Is_login permit = False	Permit
Role = Head Nurse	Actions = Create	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = True	Permit
Role = Head Nurse	Actions = Create	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = False	Permit
Role = Head Nurse	Actions = Create	Current_time = 17:00:00-05:00	Is_login permit = True	Permit

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
ABAC : Nursing_Attendant_Policy	First Applicable	Deny Biased	Permit

Rule(s) and Match result of Selected Policy against the selcted security requirement:

Subject	Resource	Action	Decision	Inheritance Relation	Match Result
Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Create	Deny	Originate	Not Applicable
Role = Nursing Attendant	Patient Record = Prescription	Actions = Delete	Deny	Originate	Not Applicable
Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	Originated	Not Applicable
Role = Midwife	Patient Record = Prescription	Actions = Create	Permit	Inherited	Not Applicable
Role = Head Nurse	Patient Record = Prescription	Actions = Create	Permit	Inherited	Permit
Role = Chief of Hospital	Patient Record = Prescription	Actions = Create	Permit	Inherited	Not Applicable
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Permit	Originated	Not Applicable
Role = Midwife	Patient Medicare = Dressing care	Actions = Delete	Permit	Inherited	Not Applicable
Role = Head Nurse	Patient Medicare = Dressing care	Actions = Delete	Permit	Inherited	Not Applicable

Figure 84: Resource Access Privilege Preview by Single Policy

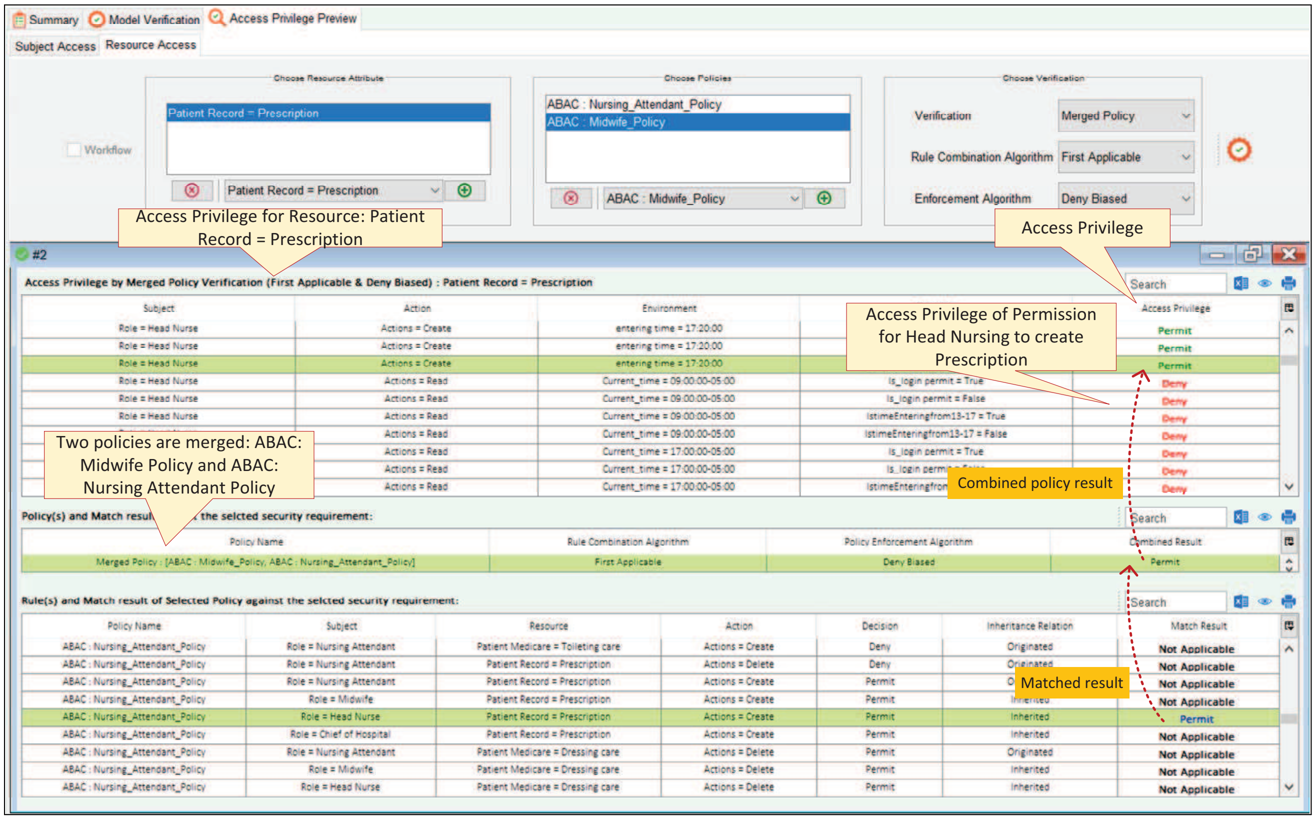
Single Policy: Figure 84 shows the configuration of Resource Privilege Preview from a single policy. As shown in Figure 84, a resource attribute value of $Patient\ Record = Prescription$ is given. This is to query who can access to the Resource of $Patient\ Record = Prescription$ with what actions. As only one policy is chosen for the query, the verification method is "Single Policy", as shown in Figure 84. After clicking on the Run Verification icon, the access privilege results have presented that match with the query. Particularly, the access privilege query is a verification process and the results are shown in three tables:

- **Access Privilege by Single Policy Verification**: This table shows all possible subjects against the resource (e.g., $Patient\ Record = Prescription$) with the privilege (Deny or Permit). The example highlighted in Figure 84 can be read as $Rule = Head\ Nurse - Subject\ Create - Action\ Prescription - Resource$ is $Permit - Privilige$.

- **Policy (s) and Match result against the selected security requirement**: This table shows the privilege (i.e., Permit) of the select policy configuration including the Policy Name, Rule Combining Algorithm, and Policy Enforcement algorithm.

- **Rule (s) and Match result of selected security requirement**: This table shows the rules and the testing results against the query.

Resource Privilege Preview answers the question the resource access privilege, e.g., who can access the resource of concern, and from which the policy author can check if any AC flaws. The Resource Privilege Preview results can be exported as Excel table, enlarged view, and printout.



Summary
Model Verification
Access Privilege Preview
Subject Access
Resource Access
Workflow
Access Privilege for Resource: Patient Record = Prescription
Access Privilege
Access Privilege of Permission for Head Nursing to create Prescription
Two policies are merged: ABAC: Midwife Policy and ABAC: Nursing Attendant Policy
Combined policy result
Matched result
#2
Search

Choose Resource Attribute: Patient Record = Prescription

Choose Policies: ABAC : Nursing_Attendant_Policy; ABAC : Midwife_Policy

Choose Verification:
Verification: Merged Policy
Rule Combination Algorithm: First Applicable
Enforcement Algorithm: Deny Biased

Access Privilege by Merged Policy Verification (First Applicable & Deny Biased) : Patient Record = Prescription

Subject	Action	Environment	[illegible]	Access Privilege
Role = Head Nurse	Actions = Create	entering time = 17:20:00	[illegible]	Permit
Role = Head Nurse	Actions = Create	entering time = 17:20:00	[illegible]	Permit
Role = Head Nurse	Actions = Create	entering time = 17:20:00	[illegible]	Permit
Role = Head Nurse	Actions = Read	Current_time = 09:00:00-05:00	Is_login permit = True	Deny
Role = Head Nurse	Actions = Read	Current_time = 09:00:00-05:00	Is_login permit = False	Deny
Role = Head Nurse	Actions = Read	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = True	Deny
[illegible]	Actions = Read	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = False	Deny
[illegible]	Actions = Read	Current_time = 17:00:00-05:00	Is_login permit = True	Deny
[illegible]	Actions = Read	Current_time = 17:00:00-05:00	Is_login permit = False	Deny
[illegible]	Actions = Read	Current_time = 17:00:00-05:00	IstimeEnteringfrom[illegible]	Deny

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
Merged Policy : [ABAC : Midwife_Policy, ABAC : Nursing_Attendant_Policy]	First Applicable	Deny Biased	Permit

Rule(s) and Match result of Selected Policy against the selcted security requirement:

Policy Name	Subject	Resource	Action	Decision	Inheritance Relation	Match Result
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Medicare = Toileting care	Actions = Create	Deny	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Record = Prescription	Actions = Delete	Deny	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Record = Prescription	Actions = Create	Permit	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Record = Prescription	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Record = Prescription	Actions = Create	Permit	Inherited	Permit
ABAC : Nursing_Attendant_Policy	Role = Chief of Hospital	Patient Record = Prescription	Actions = Create	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Delete	Permit	Originated	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Midwife	Patient Medicare = Dressing care	Actions = Delete	Permit	Inherited	Not Applicable
ABAC : Nursing_Attendant_Policy	Role = Head Nurse	Patient Medicare = Dressing care	Actions = Delete	Permit	Inherited	Not Applicable

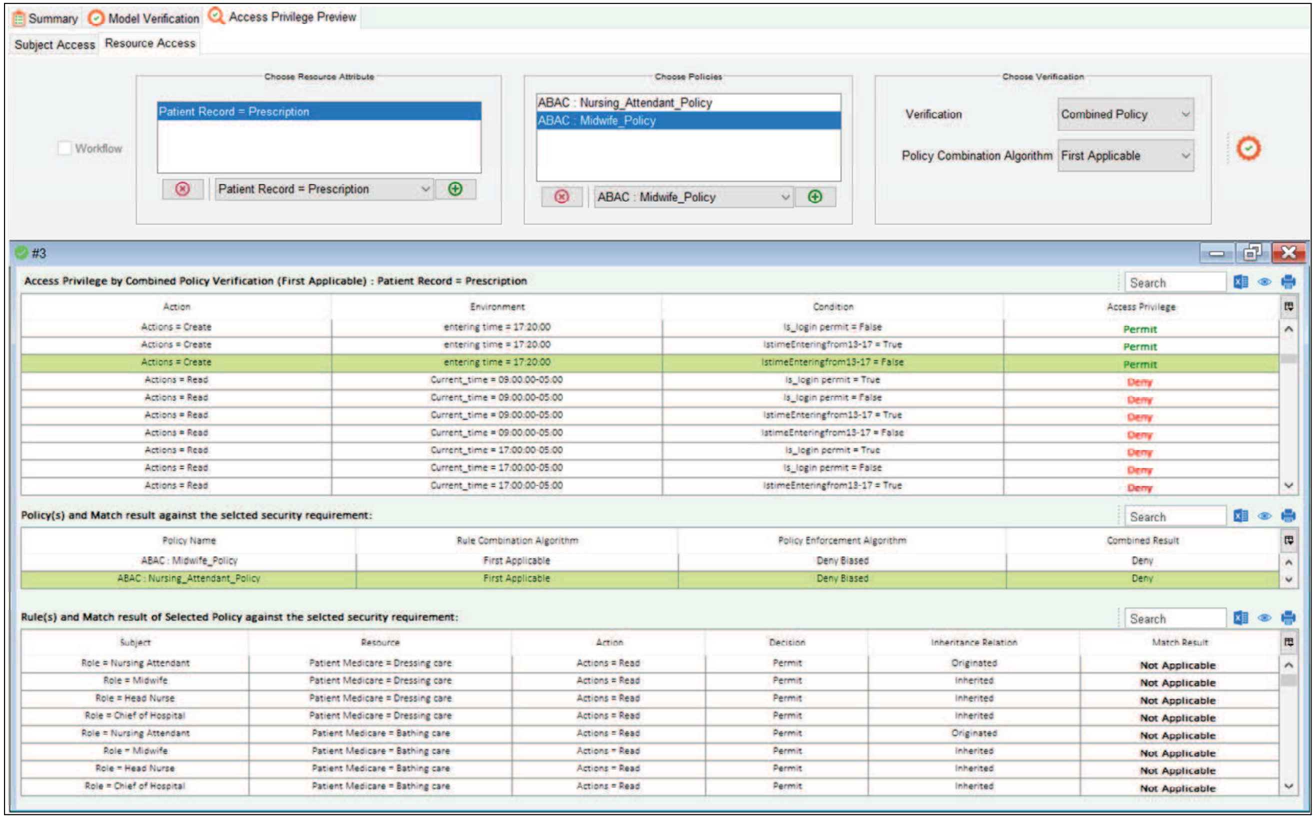
Figure 85: Resource Access Privilege Preview by Merged Policy

Merged Policy: Figure 81 shows the configuration and results of Resource Privilege Preview with merged policy. Similar to Figure 84, this example queries who can access the resource attribute value, i.e., $Patient\ Record = Prescription$, with what actions. Differently, two policies are chosen and these two policies are individually tested against the query. It chooses a "Combined Policy" for policy combination, as shown in Figure 81. The combined policy is configured with rule combination algorithm and policy enforcement algorithm, i.e., First Applicable and Deny Biased respectively. After clicking on the Run Verification icon, the access privilege results are demonstrated that match the query. Particularly, the results are shown in two tables:

- **Access Privilege by Merged Policy Verification**: This table shows all possible accessing requests of the given subject (e.g., $Role = Nursing\ Attendant$) and the results of access privilege (Deny or Permit) against all rules of the policies. The highlighted line in Figure 81 can be read as an request of $Nursing\ Attendant - Subject$ for $Create - Action\ to\ Prescription - Resource\ is\ Permit - AccessPrivilege$. If such a Decision is not the intended one, the policy author can revise the rule in order to achieve an expected result, e.g., Deny. The policy author can review all the results to check if they satisfy the AC security requirement.

- **Policy (s) and Matching result against the selected security requirement**: This table shows the select policy configuration including the Policy Names, Rule Combining Algorithm, and Policy Enforcement algorithm.

The policy author can review all the results to check if they satisfy all the AC security requirements.



Choose Resource Attribute: Patient Record = Prescription

Choose Policies: ABAC : Nursing_Attendant_Policy; ABAC : Midwife_Policy

Verification: Combined Policy

Policy Combination Algorithm: First Applicable

Access Privilege by Combined Policy Verification (First Applicable) : Patient Record = Prescription

Action	Environment	Condition	Access Privilege
Actions = Create	entering time = 17:20:00	Is_login permit = False	Permit
Actions = Create	entering time = 17:20:00	IstimeEnteringfrom13-17 = True	Permit
Actions = Create	entering time = 17:20:00	IstimeEnteringfrom13-17 = False	Permit
Actions = Read	Current_time = 09:00:00-05:00	Is_login permit = True	Deny
Actions = Read	Current_time = 09:00:00-05:00	Is_login permit = False	Deny
Actions = Read	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = True	Deny
Actions = Read	Current_time = 09:00:00-05:00	IstimeEnteringfrom13-17 = False	Deny
Actions = Read	Current_time = 17:00:00-05:00	Is_login permit = True	Deny
Actions = Read	Current_time = 17:00:00-05:00	Is_login permit = False	Deny
Actions = Read	Current_time = 17:00:00-05:00	IstimeEnteringfrom13-17 = True	Deny

Policy(s) and Match result against the selcted security requirement:

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
ABAC : Midwife_Policy	First Applicable	Deny Biased	Deny
ABAC : Nursing_Attendant_Policy	First Applicable	Deny Biased	Deny

Rule(s) and Match result of Selected Policy against the selcted security requirement:

Subject	Resource	Action	Decision	Inheritance Relation	Match Result
Role = Nursing Attendant	Patient Medicare = Dressing care	Actions = Read	Permit	Originated	Not Applicable
Role = Midwife	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited	Not Applicable
Role = Head Nurse	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited	Not Applicable
Role = Chief of Hospital	Patient Medicare = Dressing care	Actions = Read	Permit	Inherited	Not Applicable
Role = Nursing Attendant	Patient Medicare = Bathing care	Actions = Read	Permit	Originated	Not Applicable
Role = Midwife	Patient Medicare = Bathing care	Actions = Read	Permit	Inherited	Not Applicable
Role = Head Nurse	Patient Medicare = Bathing care	Actions = Read	Permit	Inherited	Not Applicable
Role = Chief of Hospital	Patient Medicare = Bathing care	Actions = Read	Permit	Inherited	Not Applicable

Figure 86: Resource Access Privilege Preview by Combined Policy

Combined Policy: Figure 82 shows the configuration of Subject Privilege Preview by combined policy. Different than the merged policy, the combined policy tests the query against each policy and the results are combined by the policy combining algorithm. Figure 80 shows the same query as in Figure 81 with a subject attribute values of $Role = Nursing\ Attendant$: what the Resource that $Role = Nursing\ Attendant$) can access with what actions. The verification method is "Combined Policy" as shown in Figure 80. After clicking on the Run Verification icon, the access privilege results are shown in two tables:

- **Access Privilege by Single Policy Verification**: It shows all possible accessing requests for $Role = Nursing\ Attendant$ with access privilege (Deny or Permit). The highlighted line in Figure 81 can be read as an request of $Nursing\ Attendant - Subject$ for $Create - Action\ to\ Prescription - Resource\ is\ Permit - AccessPrivilige$.

- **Policy (s) and Matching result against the selected security requirement**: This table shows two policies, including the Policy Name, Rule Combining Algorithm, and the Policy Enforcement algorithm. Meanwhile, it shows the verification results of these two policies, and both results are $Deny$ in the example. The $Deny$ result in the table of "Access Privilege by Single Policy Verification" is the combined results of $Deny$ and $Deny$ in the table of "Policy (s) and Match result against the selected security requirement" via the Policy Combination Algorithm. In other words, the privilege result is the combined results of all policy through the policy combination algorithm.

Subject Privilege Preview answers the question the access privilege of a subject, e.g. a role, and from which the policy author can check if any AC flaws. The Subject Privilege Preview results can be exported as Excel table, enlarged view, and printout.

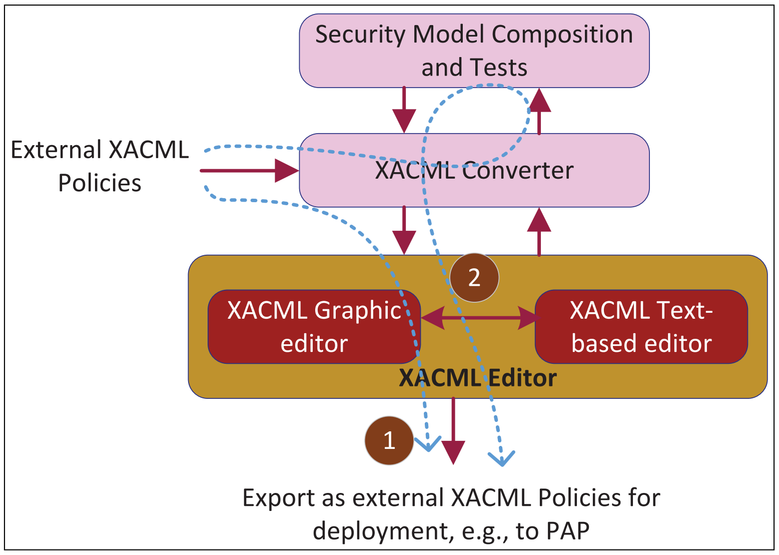

Security Model Composition and Tests
External XACML Policies
XACML Converter
2
XACML Graphic editor
XACML Text-based editor
XACML Editor
1
Export as external XACML Policies for deployment, e.g., to PAP

Figure 87: XACML Functional Structure

7.13 XACML Functions

XACML is a verbose language at the cost of complexity for policy editing and verification. It lacks a user-friendly representation of the policy as the number of policy elements increases. Writing and correcting a policy in a text-based way is difficult for a policy author as typing errors often occur. Graphic XACML editors allow simplified policy editing. However, the current graphic XACML editors have several limitations for professional policy authors:

- Unable to user-friendly view and track XACML elements in text due to graphic interface,
- Cumbersome to get an overview of all XACML elements in a policy.
- Doesn't fully conform all XACML 2.0 and 3.0 specifications (e.g., data types, functions, and algorithms).

Different than the existing graphic XACML editors, $\mathcal{SPT}$ provides XACML policy import, convertor, editing, verification, and export. Figure 87 shows the functional structure which provides more powerful and user-friendly XACML functions, compared to current XACML policy editors. $\mathcal{SPT}$ has the unique functions:

- **Security Model Converter**: The XACML policies can be imported into $\mathcal{SPT}$ for editing and testing, e.g., functions in path ① in Figure 87. Meanwhile, a security model (e.g., ABAC) composed by $\mathcal{SPT}$ can be automatically converted into XACML documents, e.g., functions in path ②. In this way, all the ABAC policies that are graphically composed and tested can be automatically translated into portable XACML policies.

- **Graphic/Text Policy Editor**: $\mathcal{SPT}$ enables graphic policy editing as well as text-based policy editing in an integrated way. This is different than the current graphic policy editors. The current graphic policy editors are all limited in its flexibility and usability as they could not allow policy editing and modification of an XACML element by text editing. A policy author can edit a policy in both graphic or text ways, giving a clear view of the XACML elements. XACML text editing is especially suitable for advanced users. An XACML policy can be edited from empty or from an existing policy imported from external. Furthermore, $\mathcal{SPT}$ can automatically synchronize with each other between the graphic and text-based editing.

- **XACML 2.0 and 3.0 Compatibility**: $\mathcal{SPT}$ policy editor is XACML 2.0 and 3.0 compatible in supporting all their functions.

The other benefit is that $\mathcal{SPT}$ XACML editor is a user-friendly tool to concurrently edit and manage a number of policies.

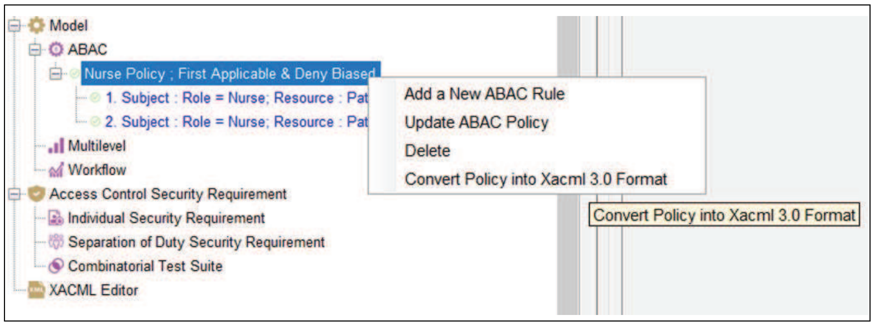
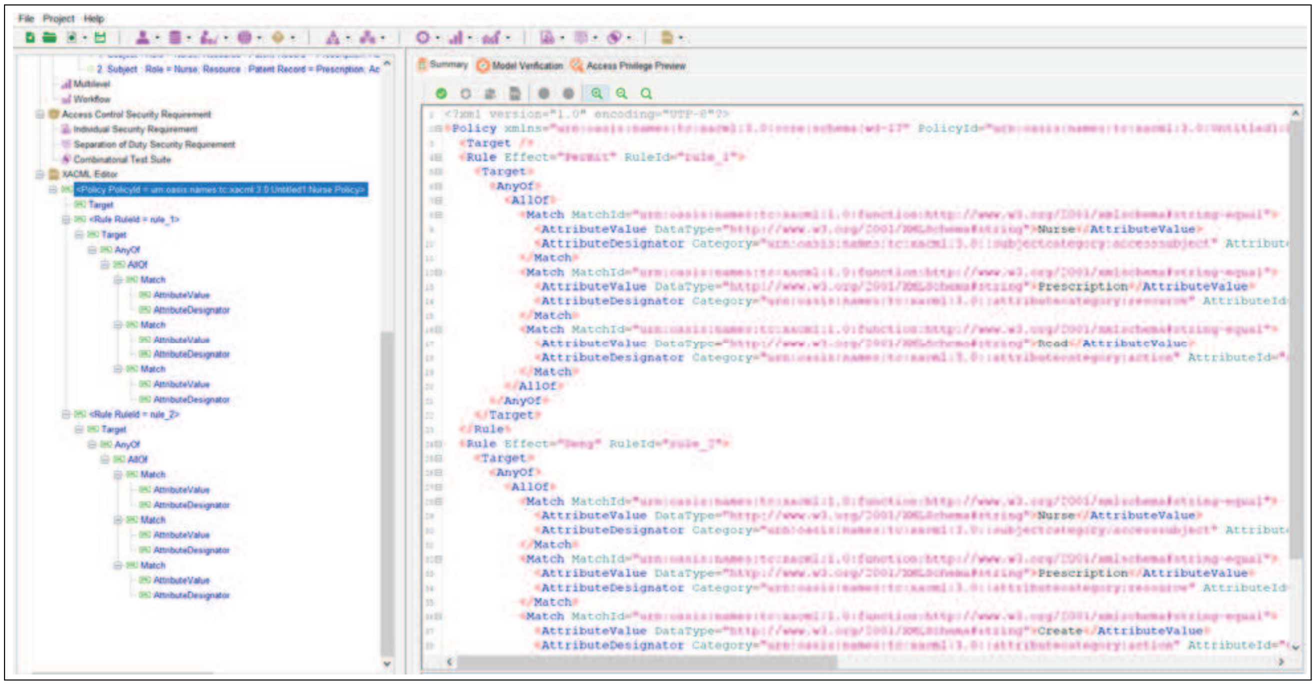

Model
ABAC
Nurse Policy ; First Applicable & Deny Biased
1. Subject : Role = Nurse; Resource : Pat
2. Subject : Role = Nurse; Resource : Pat
Multilevel
Workflow
Access Control Security Requirement
Individual Security Requirement
Separation of Duty Security Requirement
Combinatorial Test Suite
XACML Editor
Add a New ABAC Rule
Update ABAC Policy
Delete
Convert Policy into Xacml 3.0 Format
Convert Policy into Xacml 3.0 Format

Figure 88: Converting an ABAC Policy into XACML Format

7.13.1 XACML Policy Converter

XACML policy converter is a function to convert the ABAC, Multilevel, or Workflow policies composed in the security model into standard XACML policy format. It has the operational steps:

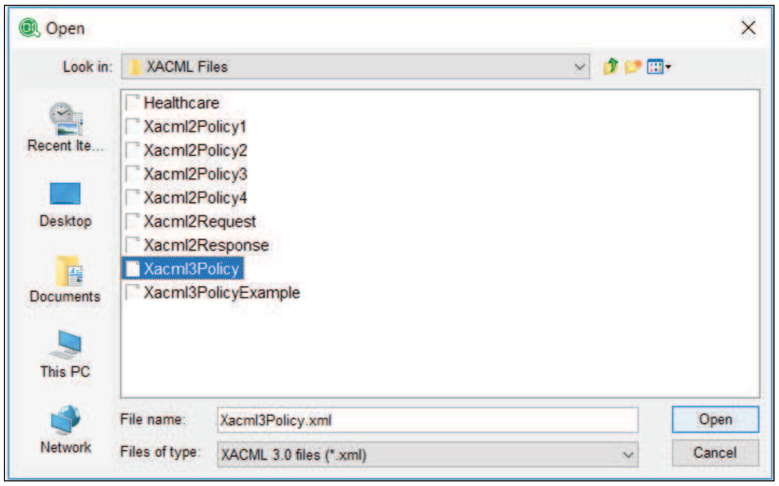

File Project Help
2 Subject Role = Nurse, Resource Patient Record = Prescription, Ac
Multilevel
Workflow
Access Control Security Requirement
Individual Security Requirement
Separation of Duty Security Requirement
Combinatorial Test Suite
XACML Editor
<Policy PolicyId = urn:oasis:names:tc:xacml:3.0:Untitled1:Nurse Policy>
Target
<Rule RuleId = rule_1>
<Rule RuleId = rule_2>
Target
AnyOf
AllOf
Match
AttributeValue
AttributeDesignator
Summary
Model Verification
Access Privilege Preview

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="urn:oasis:names:tc:xacml:3.0:Untitled1:
  <Target />
  <Rule Effect="Permit" RuleId="rule_1">
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:http://www.w3.org/2001/XMLSchema#string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Nurse</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::subjectcategory:accesssubject" Attribut
          </Match>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:http://www.w3.org/2001/XMLSchema#string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Prescription</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::attributecategory:resource" AttributeId
          </Match>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:http://www.w3.org/2001/XMLSchema#string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Read</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::attributecategory:action" AttributeId="
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  </Rule>
  <Rule Effect="Deny" RuleId="rule_2">
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:http://www.w3.org/2001/XMLSchema#string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Nurse</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::subjectcategory:accesssubject" Attribut
          </Match>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:http://www.w3.org/2001/XMLSchema#string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Prescription</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::attributecategory:resource" AttributeId
          </Match>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:http://www.w3.org/2001/XMLSchema#string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Create</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::attributecategory:action" AttributeId="
```

Figure 89: Converted an ABAC Policy in XACML Format

- Right Click on a specific policy in the security model and choose "Convert Policy into XACML 3.0 Format". Figure 88 shows the Nurse Policy and it has an option of "Convert Policy into XACML 3.0 Format".

- Click on "Convert Policy into XACML 3.0 Format" and the policy will be converted into XACML format. Figure 89 demonstrates the results for the example Nurse Policy. The results include the XACML policy tree and XACML text. Further graphic and text-based editing can be conducted as illustrated in the following subsections.



Look in: XACML Files

Healthcare
Xacml2Policy1
Xacml2Policy2
Xacml2Policy3
Xacml2Policy4
Xacml2Request
Xacml2Response
Xacml3Policy
Xacml3PolicyExample

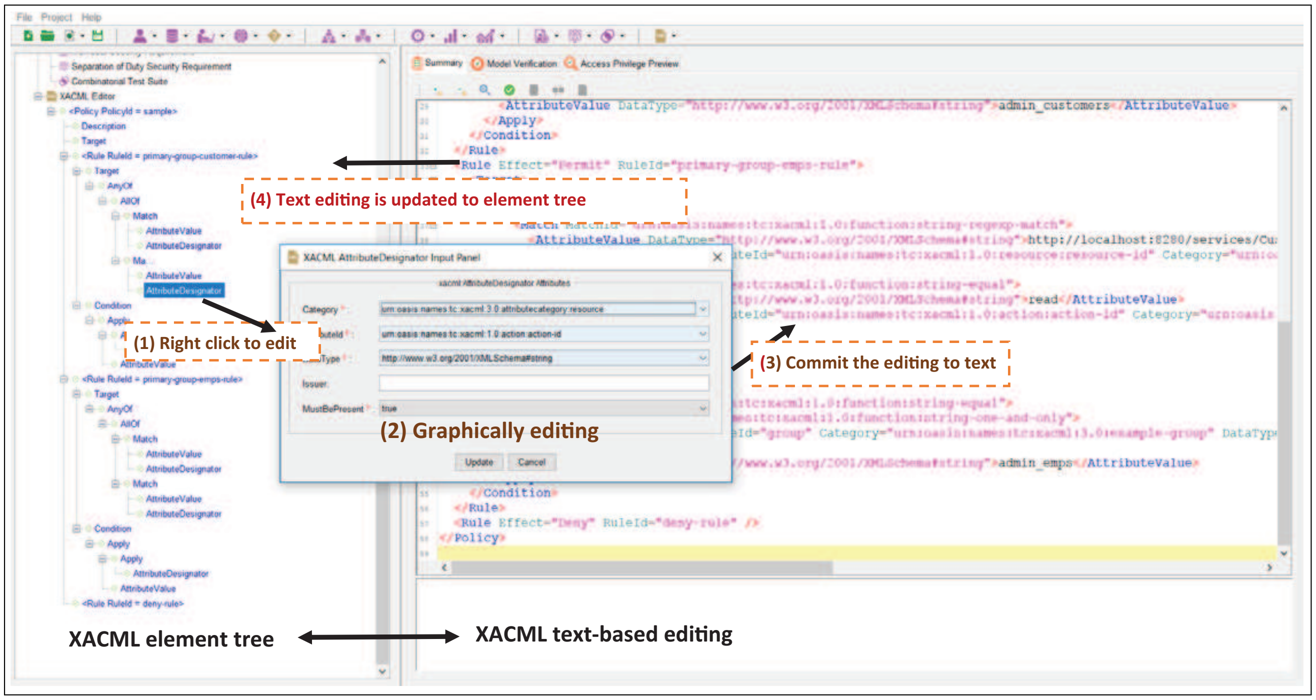
File name: Xacml3Policy.xml

Files of type: XACML 3.0 files (*.xml)

Figure 90: Import/Open an XACML File

7.13.2 XACML Policy Import

SPT allows you to import one or more external XACML 2.0 or 3.0 policy/request documents and edit them concurrently. The operation is that first navigate you to "File", select "Import", and then choose the XACML documents. An interface as shown in Figure 90 appears for you to select the XACML documents to open. Multiple XACML policies can be opened at one time from the interface and each policy is identified by a PloicyId.


File Project Help
Summary
Model Verification
Access Privilege Preview
Separation of Duty Security Requirement
Combinatorial Test Suite
XACML Editor
<Policy PolicyId = sample>
Description
Target
<Rule RuleId = primary-group-customer-rule>
Target
AnyOf
AllOf
Match
AttributeValue
AttributeDesignator
Condition
Apply
<Rule RuleId = primary-group-emps-rule>
<Rule RuleId = deny-rule>
XACML AttributeDesignator Input Panel
xacml AttributeDesignator Attributes
Category
Issuer
MustBePresent
Update
Cancel
(1) Right click to edit
(2) Graphically editing
(3) Commit the editing to text
(4) Text editing is updated to element tree
XACML element tree
XACML text-based editing

```
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">admin_customers</AttributeValue>
          </Apply>
        </Condition>
      </Rule>
      <Rule Effect="Permit" RuleId="primary-group-emps-rule">
              <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
                <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">http://localhost:8280/services/Cu
uteId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" Category="urn:o
s:tc:xacml:1.0:function:string-equal">
tp://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
uteId="urn:oasis:names:tc:xacml:1.0:action:action-id" Category="urn:oasis
:tc:xacml:1.0:function:string-equal">
es:tc:xacml:1.0:function:string-one-and-only">
eId="group" Category="urn:oasis:names:tc:xacml:3.0:example-group" DataTyp
/www.w3.org/2001/XMLSchema#string">admin_emps</AttributeValue>
        </Condition>
      </Rule>
      <Rule Effect="Deny" RuleId="deny-rule" />
    </Policy>
```

Category: urn:oasis:names:tc:xacml:3.0:attributecategory:resource
urn:oasis:names:tc:xacml:1.0:action:action-id
http://www.w3.org/2001/XMLSchema#string
MustBePresent: true

Figure 91: An XACML Security Policy Example

Figure 91 shows an XACML example loaded in the $\mathcal{SPT}$ XACML editor. It includes (i) XACML element tree which allows graphic policy editing, and (ii) XACML text-based editing zone. The policy editing functions are illustrated in the following subsection.

7.13.3 XACML Policy Editing

Figure 91 shows the integrated graphic and text-based policy editing:

- **Graphic Editing**: XACML element tree gives a tree-like structure of the policy, as shown in Figure 91. All the policy XACML elements can be added, edited, modified, and removed one by one through a graphical user interface, by right clicking on a selected element. It is a process of:
  - *Right Click to Edit*: Right click on a policy element, a set of operational options will appear, e.g., Step 1 in Figure 91.
  - *Graphic Editing*: The user interface simplifies the process to edit or correct any policy element, e.g., Step 2 in Figure 91 has the checklist to choose the functions, algorithms, and other schema element parameters.
  - *Commit the Editing to Text*: The graphically edited element is automatically committed to XACML verbose text and the addition or revision will be shown in the XACML text.

- **Text-based Editing**: $\mathcal{SPT}$ enables graphic policy editing as well as text-based policy editing in an integrated way. This is different to current graphic policy editor. The current graphic policy editors are all limited in its flexibility and usability as they could not allow policy editing and modifying an XACML element by text editing. A policy author can edit a policy in both graphic or text ways, giving a clear view of the XACML elements. XACML text editing is suitable for advanced users. Furthermore, $\mathcal{SPT}$ can automatically synchronize the graphic and text-based editing.
- **Synchronization**: $\mathcal{SPT}$ policy editor is XACML 2.0 and 3.0 Compatible.

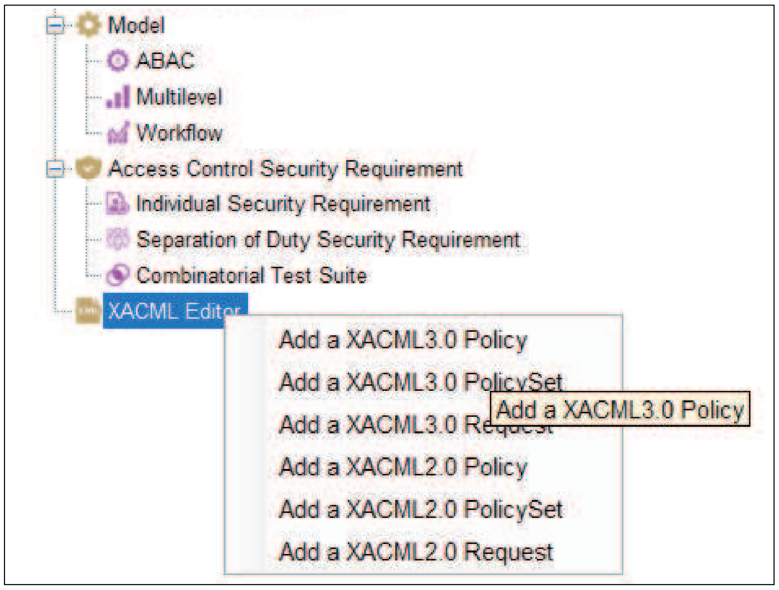


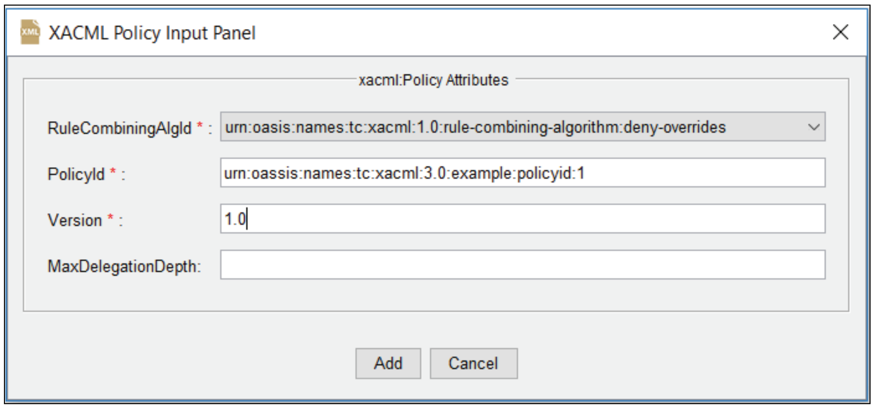
Figure 92: Create a new XACML Policy

7.13.3.1 Graphic-based XACML Editing

Basically, graphic-based XACML editing is performed by right clicking on the policy element to edit and add additional elements in the XACML element tree. We show this with the following steps to graphically create and edit a new policy.

Create a New XACML Policy: Navigating to “XACML Editor” in the project tree and right clicking “XACML Editor”, you will see the policy editing options, as shown in Figure 92. $\mathcal{SPT}$ enables you to create a policyset, policy, or policy request in the XACML 2.0 or 3.0 format:

- **Policyset**: A policyset represents a policy container that can hold a set of Policies, other PolicySets, a policy-combining algorithm, and (optionally) a set of obligations and advice.
- **Policy**: A policy is a single access control policy that consists of a set of Rules, an identifier for the rule-combining algorithm, and (optionally) a set of obligations or advice.
- **Policy Request**: It represents a request in XACML format to access control system.


XACML Policy Input Panel

xacml:Policy Attributes

RuleCombiningAlgId * : urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

PolicyId * : urn:oassis:names:tc:xacml:3.0:example:policyid:1

Version * : 1.0

MaxDelegationDepth:

Figure 93: Create a new XACML Policy

Figure 93 shows the graphic interface to generate a policy where RuleCombiningAlgId, PolicyId, and Version are required parameters to create a policy. The mandatory parameters are marked with a red star (*). MaxDelegationDepth is an optional parameter. After filling these parameters, "Add" is clicked to start the editing of the policy.

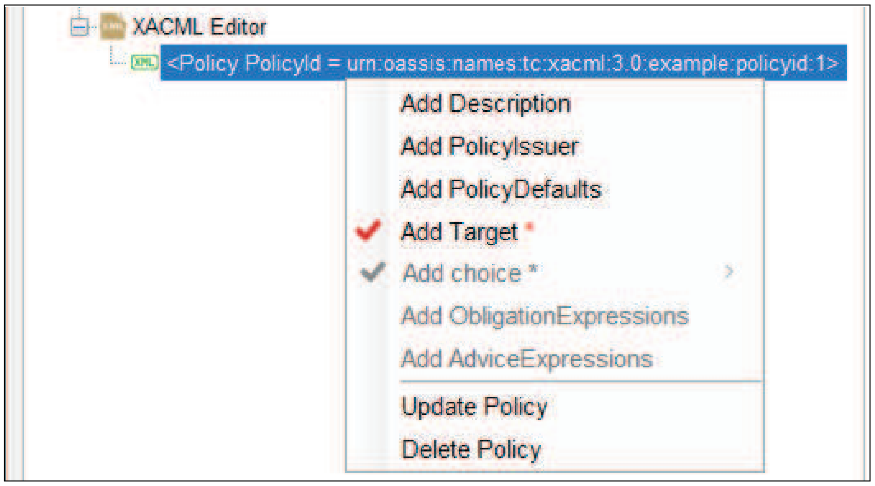


Figure 94: Add Policy Elements

Add a Target: By right clicking on the policy, policy XACML elements can be added as shown in Figure 94, and the elements could be Description, Policy Issuer, PolicyDefaults, and Target. Meanwhile, Figure 94 indicates that the policy can be updated or deleted. Among the elements, a target is mandatory. Basically, a Target is a set of simplified conditions for the Subject, Resource, and Action that must be met for a PolicySet, Policy or Rule to apply to a given request. Right clicking on the "Target", you can add "Anyof", "Allof", "Match", "AttributeValue", "AttributeDesignator", and other elements subsequently.

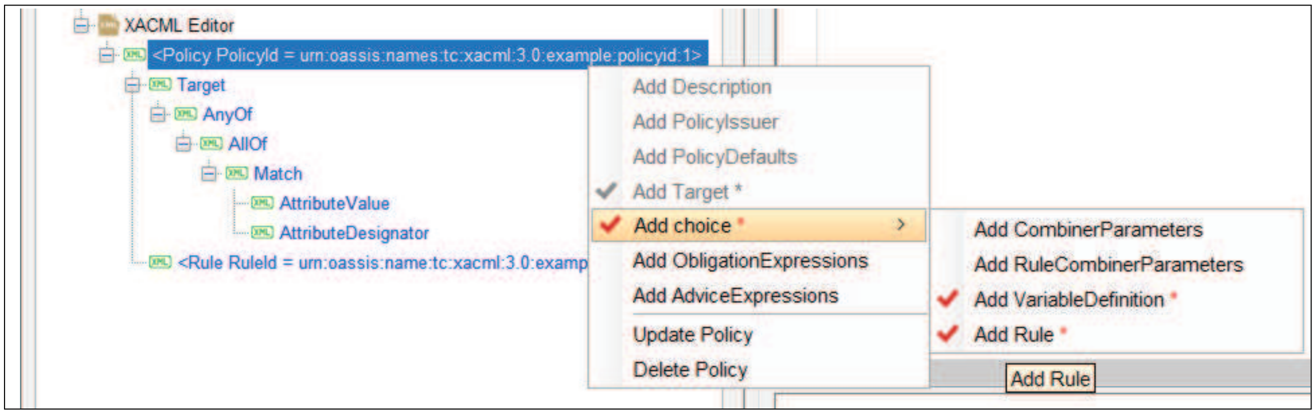
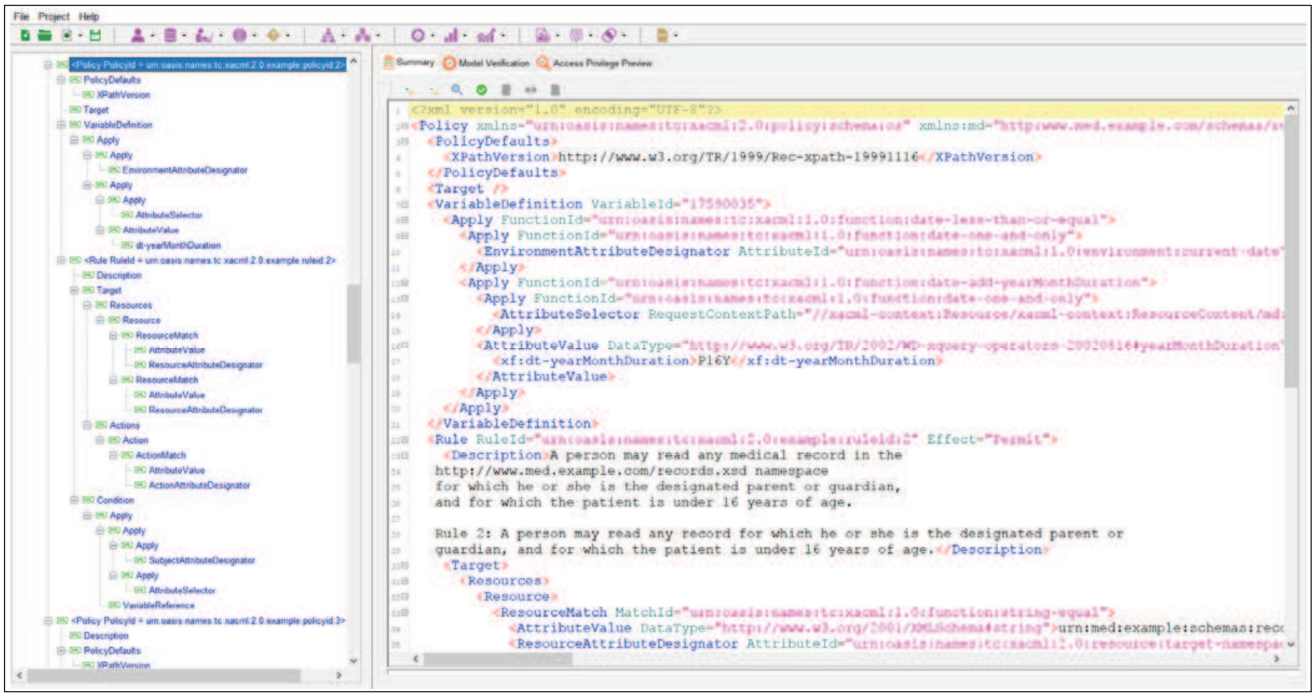


Figure 95: Add a Policy Rule

Add a Rule: By clicking on the policy, a rule can be added and Figure 95 shows the graphic interface to add a rule. By choosing "Add Rule" under "Add Choice", a graphic interface allows the definition of "RuleId" and "Effect". Then, right-clicking on the rule, "Description", "Target", "Condition","ObligationExpressions",and "AdviceExpression" can be added to the rule. By adding target, "Anyof", "Allof", "Match", "AttributeValue", and "AttributeDesignator" can be graphically added for the rule. Multiple rules can be added to the policy. Moreover, "Condition" can be added to the rule.

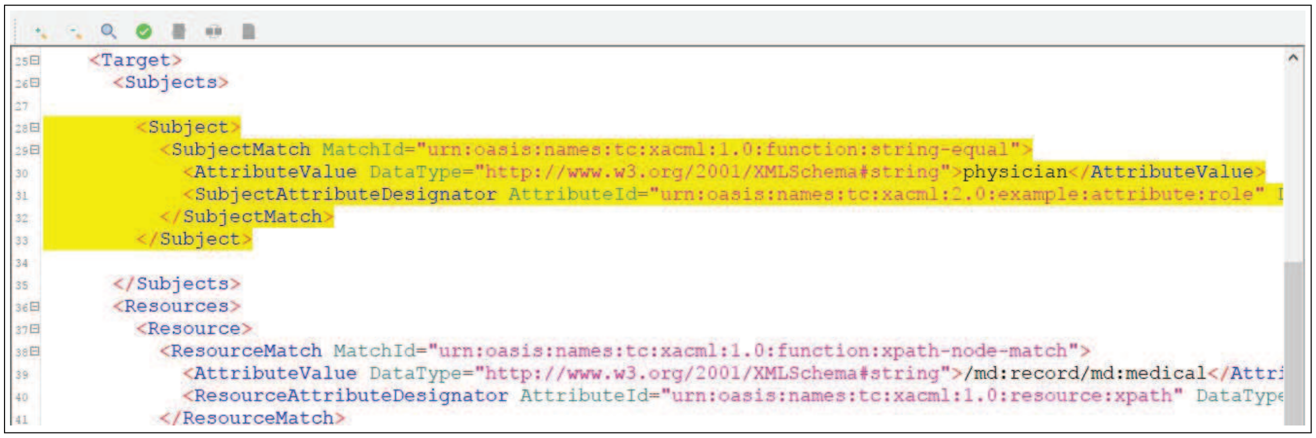


```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:md="http:www.med.example.com/schemas/r
  <PolicyDefaults>
    <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</XPathVersion>
  </PolicyDefaults>
  <Target />
  <VariableDefinition VariableId="17590035">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
        <EnvironmentAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
          <AttributeSelector RequestContextPath="//xacml-context:Resource/xacml-context:ResourceContent/md:
        </Apply>
        <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration"
          <xf:dt-yearMonthDuration>P16Y</xf:dt-yearMonthDuration>
        </AttributeValue>
      </Apply>
    </Apply>
  </VariableDefinition>
  <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:2" Effect="Permit">
    <Description>A person may read any medical record in the
  http://www.med.example.com/records.xsd namespace
  for which he or she is the designated parent or guardian,
  and for which the patient is under 16 years of age.

  Rule 2: A person may read any record for which he or she is the designated parent or
  guardian, and for which the patient is under 16 years of age.</Description>
    <Target>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">urn:med:example:schemas:rec
            <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespa
```

Figure 96: Policy Examples

A policy could have a set of rules that are defined in a similar way. Furthermore, $\mathcal{SPT}$ allows the editing of multiple policies. The policy XACML element tree can be folded/unfolded at each branch. Figure 96 shows multiple policies and each policy have multiple rules.



```
    <Target>
      <Subjects>

        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeValue>
            <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:role" I
          </SubjectMatch>
        </Subject>

      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/md:record/md:medical</Attri
            <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath" DataType
          </ResourceMatch>
```

Figure 97: Highlight of the Subject Element with Text-editing

7.13.3.2 Text-based XACML Editing

For some policy authors, they would be like to use the text-based editing function to flexibly edit and modify a policy or the policy elements. In addition, a policy author would like to have a clear global view of all policy elements by text. Figure 97 shows the highlight of a Subject element and from which the policy author can edit SubjectMatch, Attribute Value, and SubjectAttributeDesignator. By reviewing the XACML code, the text-based XACML editing is very flexible and easy to use for advanced XACML users who well understand XACML.

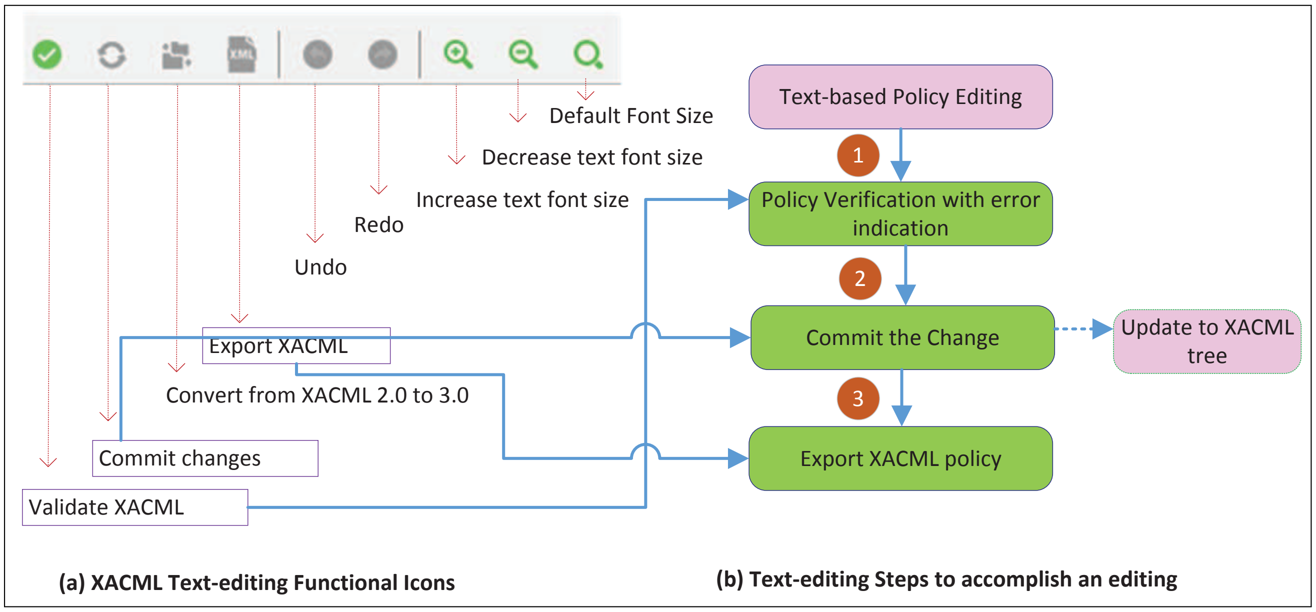

Default Font Size
Decrease text font size
Increase text font size
Redo
Undo
Export XACML
Convert from XACML 2.0 to 3.0
Commit changes
Validate XACML
Text-based Policy Editing
1
Policy Verification with error indication
2
Commit the Change
Update to XACML tree
3
Export XACML policy
(a) XACML Text-editing Functional Icons
(b) Text-editing Steps to accomplish an editing

Figure 98: XACML Panel Functions

Figure 98 shows the principle of text-based XACML editing. Figure 98 (a) are the functions under the "Summary" tab. The basic idea is to (i) edit the policy from the policy editing zone, (ii) verify the editing content, (iii) commit the modification to $\mathcal{SPT}$. Figure 98 (b) shows the editing process:

- **Policy Editing**: A policy author can edit the policy from the policy editing zone. He/she can visually see the editing of a policy.

- **Policy Verification**: A policy author can click on the circled green mark in Figure 98 (a) to verify if the modified policy has any syntax error. $\mathcal{SPT}$ verifies if there are grammar errors. If errors, $\mathcal{SPT}$ will report them with identify the line with error occurs.
- **Commit the Change**: A policy author then click on the commitment button (e.g., Commit changes as shown in Figure 98 (a)) to write the modified policy into the $\mathcal{SPT}$ engine, which meanwhile synchronizes the policy at the XACML element tree.

In the end, the policy author can export/save the policy into an XACML format file via the Export XACML button as shown in Figure 98 (a).

8 Access Control System Implementation

The XACML policies generated from $\mathcal{SPT}$ could be incorporated into the XACML framework for policy enforcement. The XACML framework [3] includes PEP (Policy Enforcement Point), PDP (Policy Decision Point), PIP (Policy Information Point), PAP (Policy Administration Point), and PRP (Policy Retrieval Point). In order to maintain the consistency between the tests and the actual implementation, the XACML policies generated from $\mathcal{SPT}$ should be configured in the PAP and the policy/rule combining algorithms should be exactly configured as that in the $\mathcal{SPT}$ tests.

9 References

1. Security Policy Tool, "Access Control Flaws: What Are They and How Can They Be Avoided?", Available at: www.Securitypolicytool.com, 2017.

2. Vincent C. Hu, Rick Kuhn, Dylan Yaga, "NIST SP 800-192: Verification and Test Methods for Access Control Policies/Models", Computer Security Division, National Institute of Standards and Technology, 2017, Alliable at: https://beta.csrc.nist.gov/News/2017/NIST-Release-SP-800-192.

3. Vincent C. Hu, and Karen Scarfone "NIST IR 800-7874: Guidelines for Access Control System Evaluation Metrics", Computer Security Division, National Institute of Standards and Technology, 2012, Alliable at: http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7874.pdf.

4. Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone "NIST SP 800-162: Guide to Attribute Based Access Control (ABAC) Definition and Considerationss", Computer Security Division, National Institute of Standards and Technology, 2014, Alliable at: http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-162.pdf.

5. V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn, "Assessment of Access Control Systems," NIST IR 7316, Computer Security Division, National Institute of Standards and Technology, 2006.

6. OASIS, "eXtensible Access Control Markup Language(XACML)," http://www.oasis-open.org.

7. NIST, "Computer Security Resource Center, Access Control Policy Tool (ACPT)," Available https://www.nist.gov/programs-projects/access-control-policy-tool-acpt.

8. K. Jayaraman, V. Ganesh, M. Tripunitara, M. C. Rinard, and S. J. Chapin, "ARBAC Policy for a Large Multi-National Bank," IEEE International Conference on Software Testing, Verification, and Validation Workshops 2014.

9. RFC 4949: Internet Security Glossary, Version 2, Network Working Group, 2007.

10. Federal Financial Institutions Examination Council: Authentication in an Internet Banking Environment.

11. MIL-STD-188.