Bank Policy Test Cases

(InfoBeyond Technology LLC)

Abstract

This document demonstrates access control test cases using Security Policy Tool, a software tool for Access Control Security Managers, Policy Authors, and other IT Security Professionals specializing in the performance of access control systems. Access control policies are designed to protect the accessibility of online resources in networks, IoTs, healthcare systems, financial service systems, enterprise IT and clouds, military systems, and other online environments. There are several challenges in building robust access control models for these systems including (i) effectively composing secure policies and rules, (ii) testing these policies systematically, (iii) verifying these policies to prevent access control leaks. Security Policy Tool solves these issues by providing powerful access control policy modeling, testing, and verification features that empower organizations to close the door to access control leaks.

Index Terms

Access control, attribute-based access control (abac), role-based access control (rbac), security policy editing, test, verification, deployment, access control leaks, XACML, software tool.

1 Introduction to Test Cases

This document and attached Security Policy Tool – Project Files have been designed to help you gain an understanding of what common access control errors look like, how they are created, and how to resolve them. Organizations who leverage Security Policy Tool's systematic modeling, testing and verification features are empowered to efficiently identify errors and close the door to access control leaks.

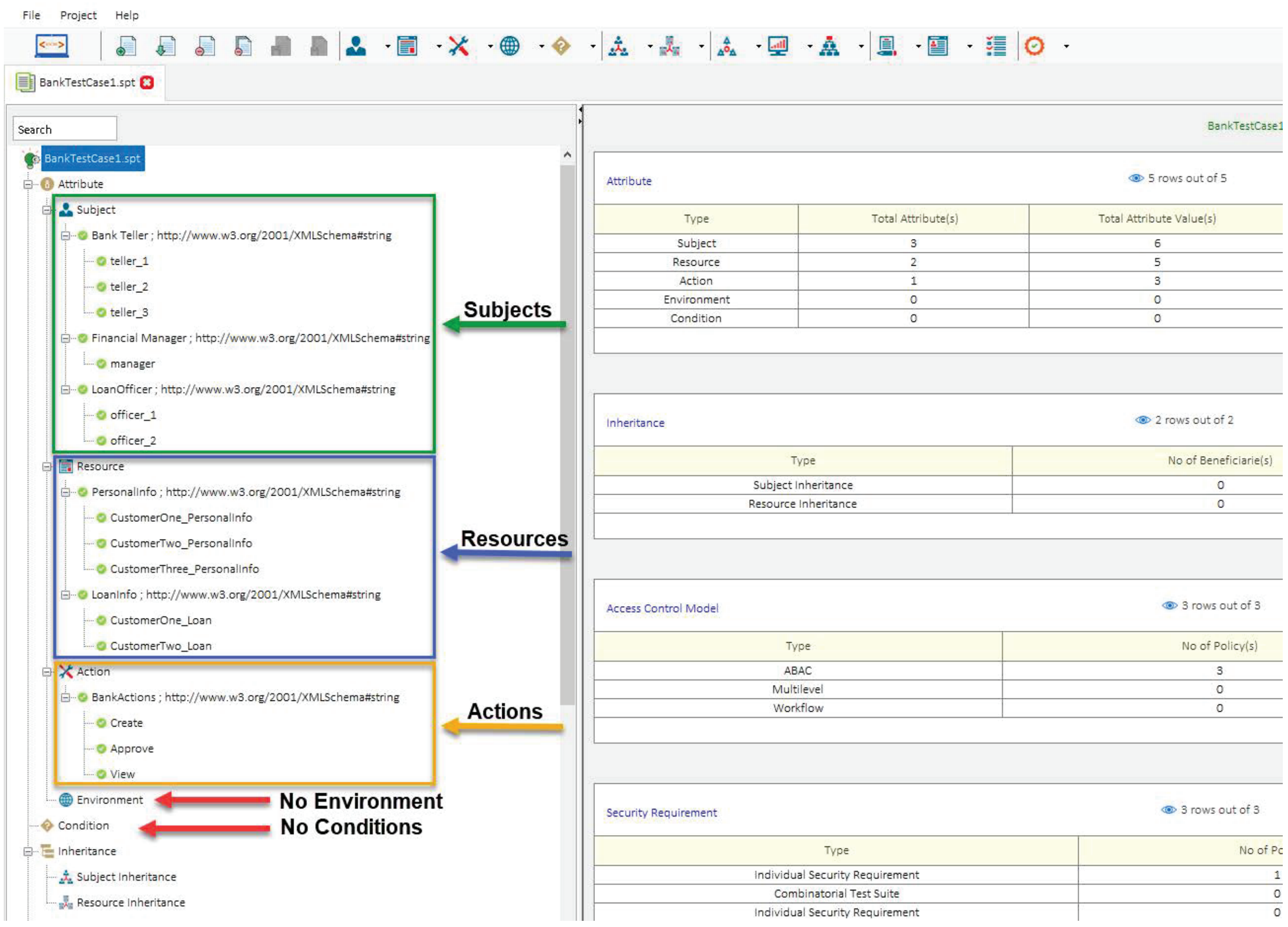
These Bank Policy test cases have been created by InfoBeyond Technology LLC to demonstrate commonly found errors in access control policy design such as Leak Privilege, Block Privilege, Inheritance Loop, Separation of Duty, and Inconsistent Assignment. These test cases consist of policies/rules to better illustrate how Security Policy Tool enhances access control security. The goal of these test cases is to provide a starting point for what to expect as you go on to use Security Policy Tool to analyze your own policy verification results for errors.

2 Setting Up The Policies – Test Case 1 (Leak Privilege)

This bank example contains three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). The Attribute /Attribute Values include in these policies are as shown in Figure 1.

• *Contact us at: E-mail: Info@Securitypolicytool.com*

Security Policy Tool (www.Securitypolicytool.com) is a commercial version of NIST(National Institute of Standards and Technology)'s ACPT (Access Control Policy Tool). With tremendous consultation with NIST experts, Security Policy Tool substantially enhances and expands the NIST's ACPT design with advanced features for achieving high security confidence access control levels such that it can be commercialized. The development of Security Policy Tool is financially sponsored by NIST via a SBIR (Small Business Innovation Research) Phase I and II programs. It specifically improves the NIST's ACPT design to provide a robust, unified, professional, and functionally powerful access control policy tool.



BankTestCase1.spt
- Attribute
  - Subject
    - Bank Teller ; http://www.w3.org/2001/XMLSchema#string
      - teller_1
      - teller_2
      - teller_3
    - Financial Manager ; http://www.w3.org/2001/XMLSchema#string
      - manager
    - LoanOfficer ; http://www.w3.org/2001/XMLSchema#string
      - officer_1
      - officer_2
  - Resource
    - PersonalInfo ; http://www.w3.org/2001/XMLSchema#string
      - CustomerOne_PersonalInfo
      - CustomerTwo_PersonalInfo
      - CustomerThree_PersonalInfo
    - LoanInfo ; http://www.w3.org/2001/XMLSchema#string
      - CustomerOne_Loan
      - CustomerTwo_Loan
  - Action
    - BankActions ; http://www.w3.org/2001/XMLSchema#string
      - Create
      - Approve
      - View
  - Environment
- Condition
- Inheritance
  - Subject Inheritance
  - Resource Inheritance

Attribute — 5 rows out of 5

Type	Total Attribute(s)	Total Attribute Value(s)
Subject	3	6
Resource	2	5
Action	1	3
Environment	0	0
Condition	0	0

Inheritance — 2 rows out of 2

Type	No of Beneficiarie(s)
Subject Inheritance	0
Resource Inheritance	0

Access Control Model — 3 rows out of 3

Type	No of Policy(s)
ABAC	3
Multilevel	0
Workflow	0

Security Requirement — 3 rows out of 3

Type	No of Pc
Individual Security Requirement	1
Combinatorial Test Suite	0
Individual Security Requirement	0

Fig. 1. Test Case 1

3 Modeling Your Policy – Test Case 1 (Leak Privilege)

Now that we have entered our attributes we can model our three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). See the list below of the rules contained in each of these policies. You can open a "New (blank) Project" and build these policies by entering the following rules below:

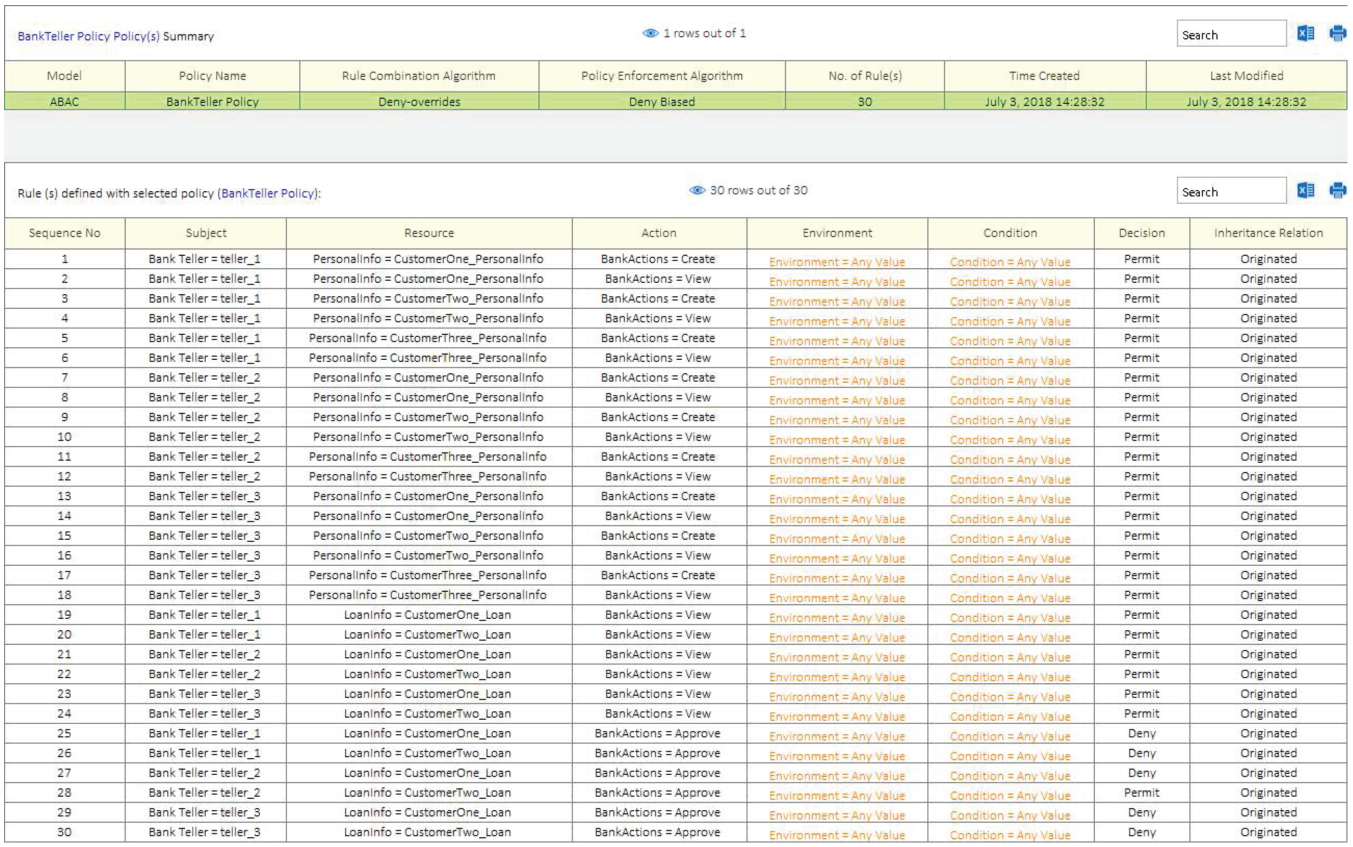
BankTeller Policy:
(Bank Teller = teller_1, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerThree_PersonalInfo, View) →Permit

(Bank Teller = teller_1, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_1, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_2, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_2, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_3, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_3, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_1, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_1, CustomerTwo_Loan, Approve) →Deny
(Bank Teller = teller_2, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_2, CustomerTwo_Loan, Approve) →Permit
(Bank Teller = teller_3, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_3, CustomerTwo_Loan, Approve) →Deny

LoanOfficer Policy:
(Loan Officer = officer_1, CustomerOne_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerOne_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerTwo_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerThree_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerThree_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, View) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_Loan, View) →Permit
(Loan Officer = officer_1, CustomerTwo_Loan, Create) →Permit
(Loan Officer = officer_2, CustomerOne_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerOne_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerTwo_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerTwo_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerThree_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerThree_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerOne_Loan, View) →Permit
(Loan Officer = officer_2, CustomerOne_Loan, Create) →Permit
(Loan Officer = officer_2, CustomerTwo_Loan, View) →Permit
(Loan Officer = officer_2, CustomerTwo_Loan, Create) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, Approve) →Deny
(Loan Officer = officer_1, CustomerTwo_Loan, Approve) →Deny
(Loan Officer = officer_2, CustomerOne_Loan, Approve) →Deny
(Loan Officer = officer_2, CustomerTwo_Loan, Approve) →Deny

FManager Policy:
(Manager = manager, CustomerOne_PersonalInfo, Create) →Permit
(Manager = manager, CustomerOne_PersonalInfo, View) →Permit
(Manager = manager, CustomerTwo_PersonalInfo, Create) →Permit
(Manager = manager, CustomerTwo_PersonalInfo, View) →Permit
(Manager = manager, CustomerThree_PersonalInfo, Create) →Permit
(Manager = manager, CustomerThree_PersonalInfo, View) →Permit
(Manager = manager, CustomerOne_Loan, Create) →Permit
(Manager = manager, CustomerOne_Loan, View) →Permit
(Manager = manager, CustomerTwo_Loan, Create) →Permit
(Manager = manager, CustomerTwo_Loan, View) →Permit
(Manager = manager, CustomerOne_Loan, Approve) →Permit
(Manager = manager, CustomerTwo_Loan, Approve) →Permit

After entering the rules above your modeled policies should look like the screenshots below. If you did not create your own Project File, you can simply open Security Policy Tool – Project File: BankTestCase1 and these policies will have been already created for you.


BankTeller Policy Policy(s) Summary

1 rows out of 1

Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	BankTeller Policy	Deny-overrides	Deny Biased	30	July 3, 2018 14:28:32	July 3, 2018 14:28:32

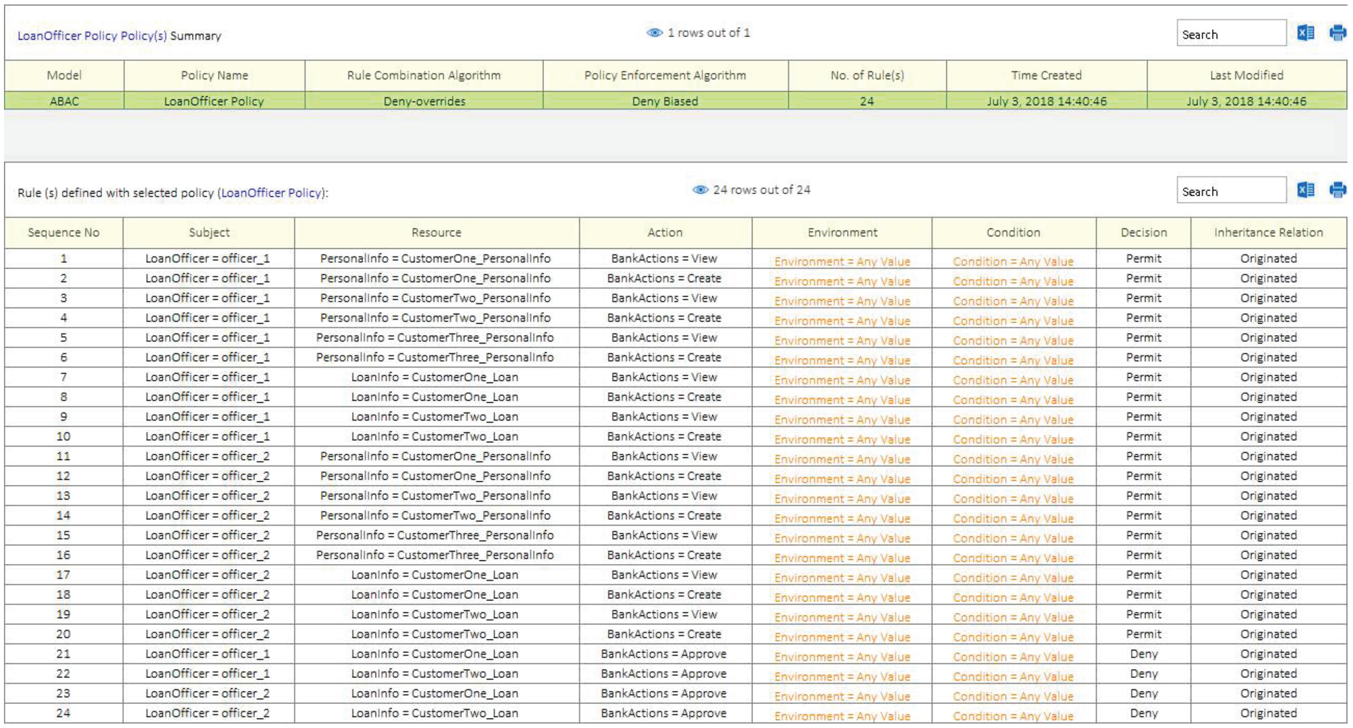
Rule (s) defined with selected policy (BankTeller Policy):

30 rows out of 30

Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
2	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
3	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
4	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
5	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
6	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
7	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
8	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
9	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
10	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
11	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
12	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
13	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
14	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
15	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
16	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
17	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
18	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
19	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
20	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
21	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
22	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
23	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
24	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
25	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
26	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
27	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
28	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated
29	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
30	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 2. BankTeller Policy


LoanOfficer Policy Policy(s) Summary

1 rows out of 1

Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	LoanOfficer Policy	Deny-overrides	Deny Biased	24	July 3, 2018 14:40:46	July 3, 2018 14:40:46

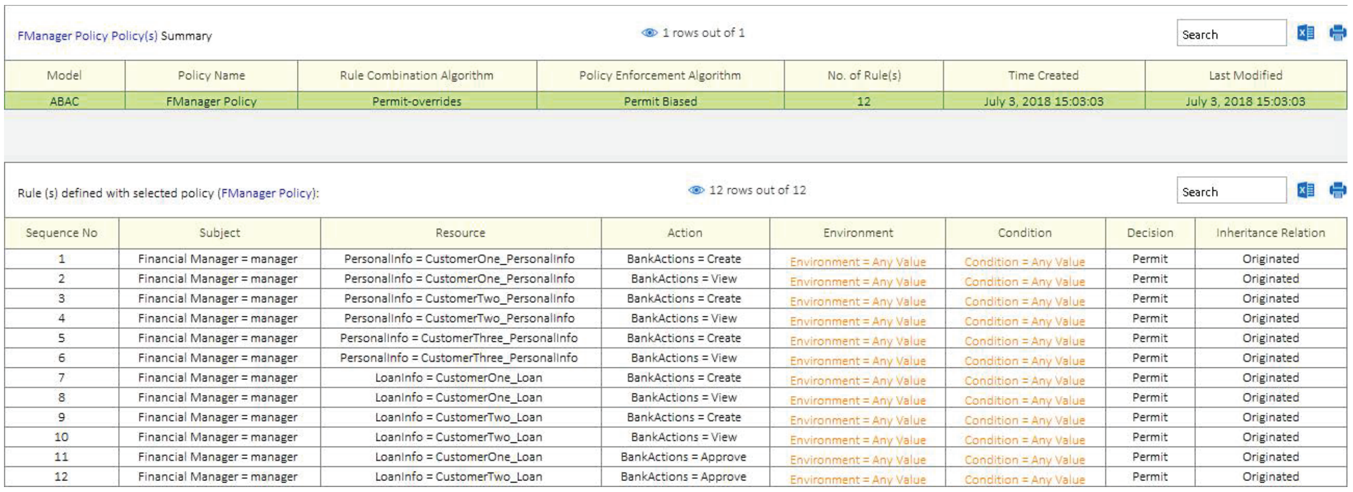
Rule (s) defined with selected policy (LoanOfficer Policy):

24 rows out of 24

Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
2	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
3	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
4	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
5	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
6	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
7	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
8	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
9	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
10	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
11	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
12	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
13	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
14	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
15	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
16	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
17	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
18	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
19	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
20	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
21	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
22	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
23	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
24	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 3. LoanOfficer Policy


FManager Policy Policy(s) Summary — 1 rows out of 1 — Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	FManager Policy	Permit-overrides	Permit Biased	12	July 3, 2018 15:03:03	July 3, 2018 15:03:03

Rule (s) defined with selected policy (FManager Policy): — 12 rows out of 12 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	Financial Manager = manager	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
2	Financial Manager = manager	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
3	Financial Manager = manager	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
4	Financial Manager = manager	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
5	Financial Manager = manager	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
6	Financial Manager = manager	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
7	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
8	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
9	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
10	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
11	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated
12	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated

Fig. 4. FManager Policy

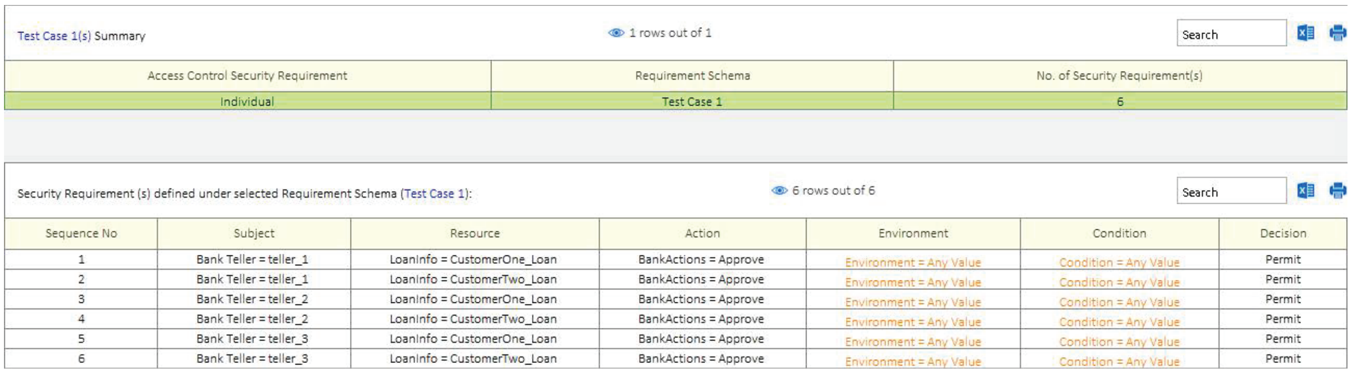
4 Individual Security Requirements - Test Case 1 (Leak Privilege)

The final step before analyzing these policies for errors is to create individual security requirements to use for testing. If you are building a "New (blank) Project" on your own you will enter the security requirements as follows.

Individual Security Requirements:

(BankTeller = teller_1) & (Action = Approve) & (LoanInfo = CustomerOne_Loan) → decision = Permit
(BankTeller = teller_1) & (Action = Approve) & (LoanInfo = CustomerTwo_Loan) → decision = Permit
(BankTeller = teller_2) & (Action = Approve) & (LoanInfo = CustomerOne_Loan) → decision = Permit
(BankTeller = teller_2) & (Action = Approve) & (LoanInfo = CustomerTwo_Loan) → decision = Permit
(BankTeller = teller_3) & (Action = Approve) & (LoanInfo = CustomerOne_Loan) → decision = Permit
(BankTeller = teller_3) & (Action = Approve) & (LoanInfo = CustomerTwo_Loan) → decision = Permit

After entering the rules above your individual security requirements should look like the screenshot below. If you did not create your own Project File you can simply open Security Policy Tool – Project File: BankTestCase1 and these requirements will have been already created for you.


Test Case 1(s) Summary — 1 rows out of 1 — Search

Access Control Security Requirement	Requirement Schema	No. of Security Requirement(s)
Individual	Test Case 1	6

Security Requirement (s) defined under selected Requirement Schema (Test Case 1): — 6 rows out of 6 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision
1	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit
2	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit
3	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit
4	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit
5	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit
6	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit

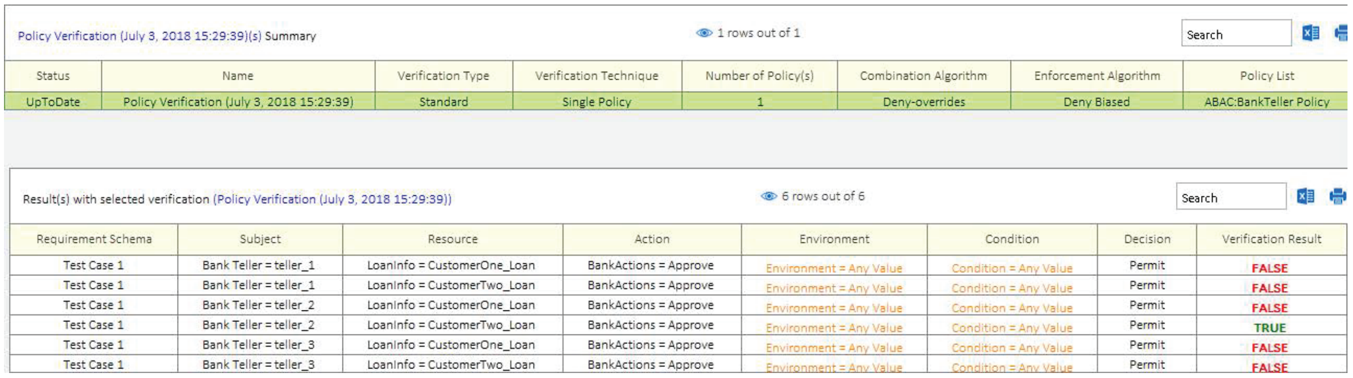
Fig. 5. Individual Security Requirements

5 Policy Verification/Analyzing Results - Test Case 1 (Leak Privilege)

Now that we are ready to test our policies let's discuss the error we will be looking at in this first example. When policies are designed, there is potential for a "Leak Privilege" being created. A Leak Privilege occurs when a flaw in your policy logic is authorizing a subject to take an action you did not intend for them to take. This error can occur due to a mistaken privilege assignment directly or careless privilege inheritance indirectly as well.

In our example, an individual at this bank has been assigned a role of Bank Teller = teller_2 at the facility. We have designed several rules in the BankTeller Policy, a few of which specifying Bank Teller subjects are unable to Approve any LoanInfo resources. In this test case, we are going to verify that this is true.

We will run one "Single Policy" Verification to reveal if their is a Leak Privilege present in our policies. To do this, we will right-click Model Verification and select New Policy Verification. Then we will choose Test Case 1 (security requirement), BankTeller Policy, Single Verification and select run. Again, this will have already been done for you if you open Project File: BankTestCase1.


Policy Verification (July 3, 2018 15:29:39)(s) Summary

1 rows out of 1

Status	Name	Verification Type	Verification Technique	Number of Policy(s)	Combination Algorithm	Enforcement Algorithm	Policy List
UpToDate	Policy Verification (July 3, 2018 15:29:39)	Standard	Single Policy	1	Deny-overrides	Deny Biased	ABAC:BankTeller Policy

Result(s) with selected verification (Policy Verification (July 3, 2018 15:29:39))

6 rows out of 6

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
Test Case 1	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 1	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE

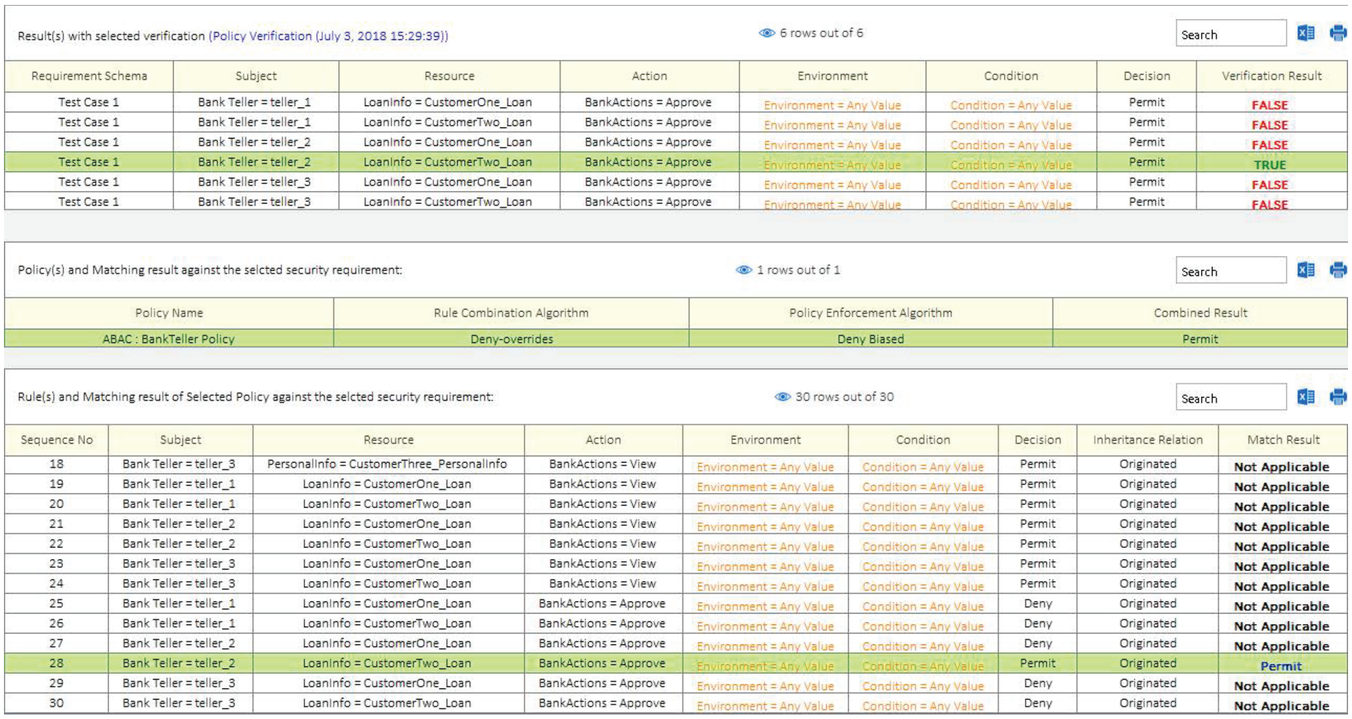
Fig. 6. BankTeller Policy x Test Case 1

As you can see from our verification results our BankTeller Policy is generating a True result for Security Requirement (4) thus Permitting (Bank Teller = teller_2) to approve CustomerTwo_LoanInfo which is known as a Leak Privilege error.

6 Resolving This Error - Test Case 1 (Leak Privilege)

To solve a Leak Privilege the policy author would need to go back and either update or delete the related rule that is creating this error. To view which specific Rules are resulting in these Verification Results we can click on all (6) of our specific Results (False; False; False; True; False; False) and see which Rules have "Match Results". Since we are already aware of which Result is incorrect let's take a look at it.

See the screenshots below of our Policy Match Results to discover which specific rule is related to our unintended Verification Result (e.g., Permit = True).


Result(s) with selected verification (Policy Verification (July 3, 2018 15:29:39)) 6 rows out of 6 Search

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
Test Case 1	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 1	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE

Policy(s) and Matching result against the selcted security requirement: 1 rows out of 1 Search

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
ABAC : BankTeller Policy	Deny-overrides	Deny Biased	Permit

Rule(s) and Matching result of Selected Policy against the selcted security requirement: 30 rows out of 30 Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation	Match Result
18	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
19	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
20	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
21	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
22	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
23	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
24	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
25	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated	Not Applicable
26	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated	Not Applicable
27	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated	Not Applicable
28	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated	Permit
29	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated	Not Applicable
30	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated	Not Applicable

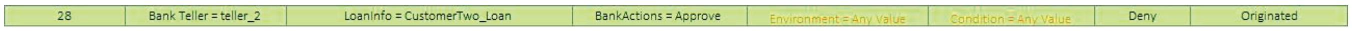
Fig. 7. BankTeller Policy: Match Results

Now that we have pinpointed our rule related to our Leak Privilege Error we can go back and make changes or possibly remove this rule. Depending on your organizational structure the policy author or access control administrator would need to decide what is the most appropriate action to take to resolve the error. There is no “right” or “wrong” solution for this, you would need to determine what is most appropriate based on your organizational needs.

For our example, we are going to modify Rule 28 in the BankTeller Policy to Deny “BankTeller = teller_2” to Approve CustomerTwo_Loan which will in turn resolve the Leak Privilege. For this example, we are fixing our mistake in unintentionally selecting Permit for this individual (teller_2) to be able to Approve the resource.

BankTeller Policy: Modify (28) Rule:

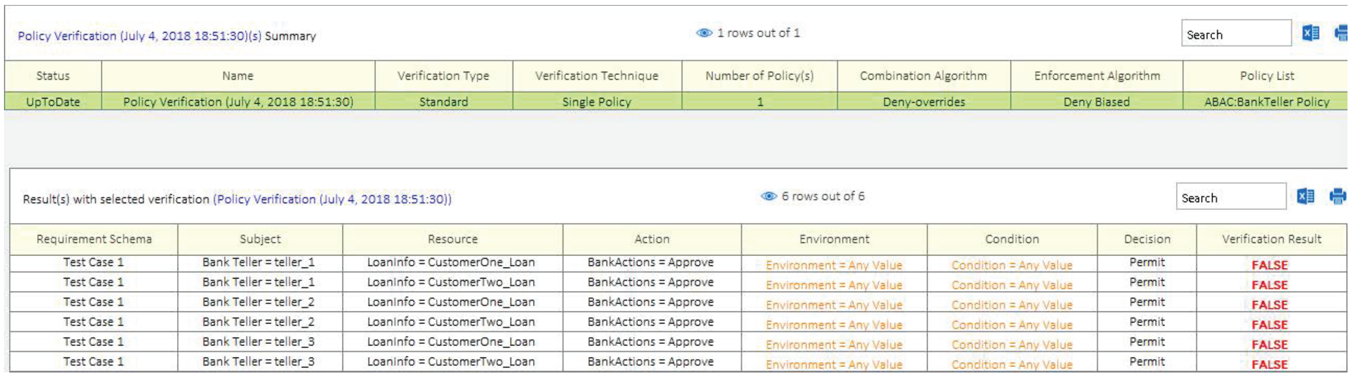
(Rule No. = 28) $\rightarrow$ (Bank Teller = teller_2) $\rightarrow$ (Action = Approve) $\rightarrow$(Resource = CustomerTwo_Loan) $\rightarrow$ decision = Deny



28	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 8. BankTeller Policy: Modified Rule (28)

Which then when we “Refresh” our previous Verification Results we no longer have a Leak Privilege occurring:


Policy Verification (July 4, 2018 18:51:30)(s) Summary — 1 rows out of 1 — Search

Status	Name	Verification Type	Verification Technique	Number of Policy(s)	Combination Algorithm	Enforcement Algorithm	Policy List
UpToDate	Policy Verification (July 4, 2018 18:51:30)	Standard	Single Policy	1	Deny-overrides	Deny Biased	ABAC:BankTeller Policy

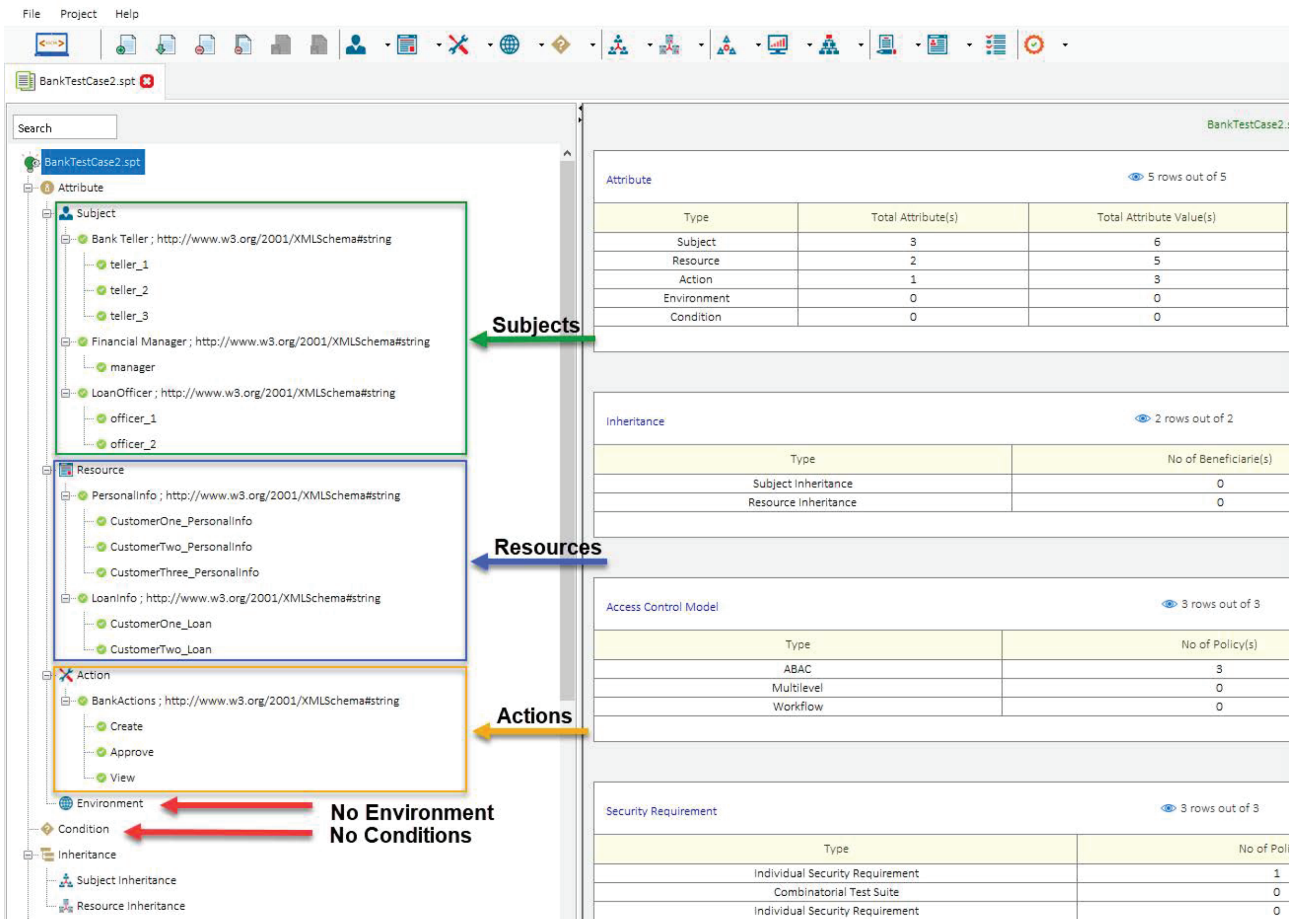
Result(s) with selected verification (Policy Verification (July 4, 2018 18:51:30)) — 6 rows out of 6 — Search

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
Test Case 1	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 1	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE

Fig. 9. Updated Results: BankTeller Policy (No Leak Privilege)

7 Setting Up The Policies – Test Case 2 (Block Privilege)

This bank example contains three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). The attributes in this example have not been changed from previous Test Case 1. The Attribute/Attribute Values included in these policies are as shown in Figure 10.



- BankTestCase2.spt
  - Attribute
    - Subject
      - Bank Teller ; http://www.w3.org/2001/XMLSchema#string
        - teller_1
        - teller_2
        - teller_3
      - Financial Manager ; http://www.w3.org/2001/XMLSchema#string
        - manager
      - LoanOfficer ; http://www.w3.org/2001/XMLSchema#string
        - officer_1
        - officer_2
    - Resource
      - PersonalInfo ; http://www.w3.org/2001/XMLSchema#string
        - CustomerOne_PersonalInfo
        - CustomerTwo_PersonalInfo
        - CustomerThree_PersonalInfo
      - LoanInfo ; http://www.w3.org/2001/XMLSchema#string
        - CustomerOne_Loan
        - CustomerTwo_Loan
    - Action
      - BankActions ; http://www.w3.org/2001/XMLSchema#string
        - Create
        - Approve
        - View
    - Environment
  - Condition
  - Inheritance
    - Subject Inheritance
    - Resource Inheritance

BankTestCase2.s

Attribute — 5 rows out of 5

Type	Total Attribute(s)	Total Attribute Value(s)
Subject	3	6
Resource	2	5
Action	1	3
Environment	0	0
Condition	0	0

Inheritance — 2 rows out of 2

Type	No of Beneficiarie(s)
Subject Inheritance	0
Resource Inheritance	0

Access Control Model — 3 rows out of 3

Type	No of Policy(s)
ABAC	3
Multilevel	0
Workflow	0

Security Requirement — 3 rows out of 3

Type	No of Poli
Individual Security Requirement	1
Combinatorial Test Suite	0
Individual Security Requirement	0

Fig. 10. Test Case 2

8 Modeling Your Policy – Test Case 2 (Block Privilege)

Now that we have entered our attributes we can model our three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). See the list below of the rules contained in each of these policies. You can open a "New (blank) Project" and build these policies by entering the following rules below:

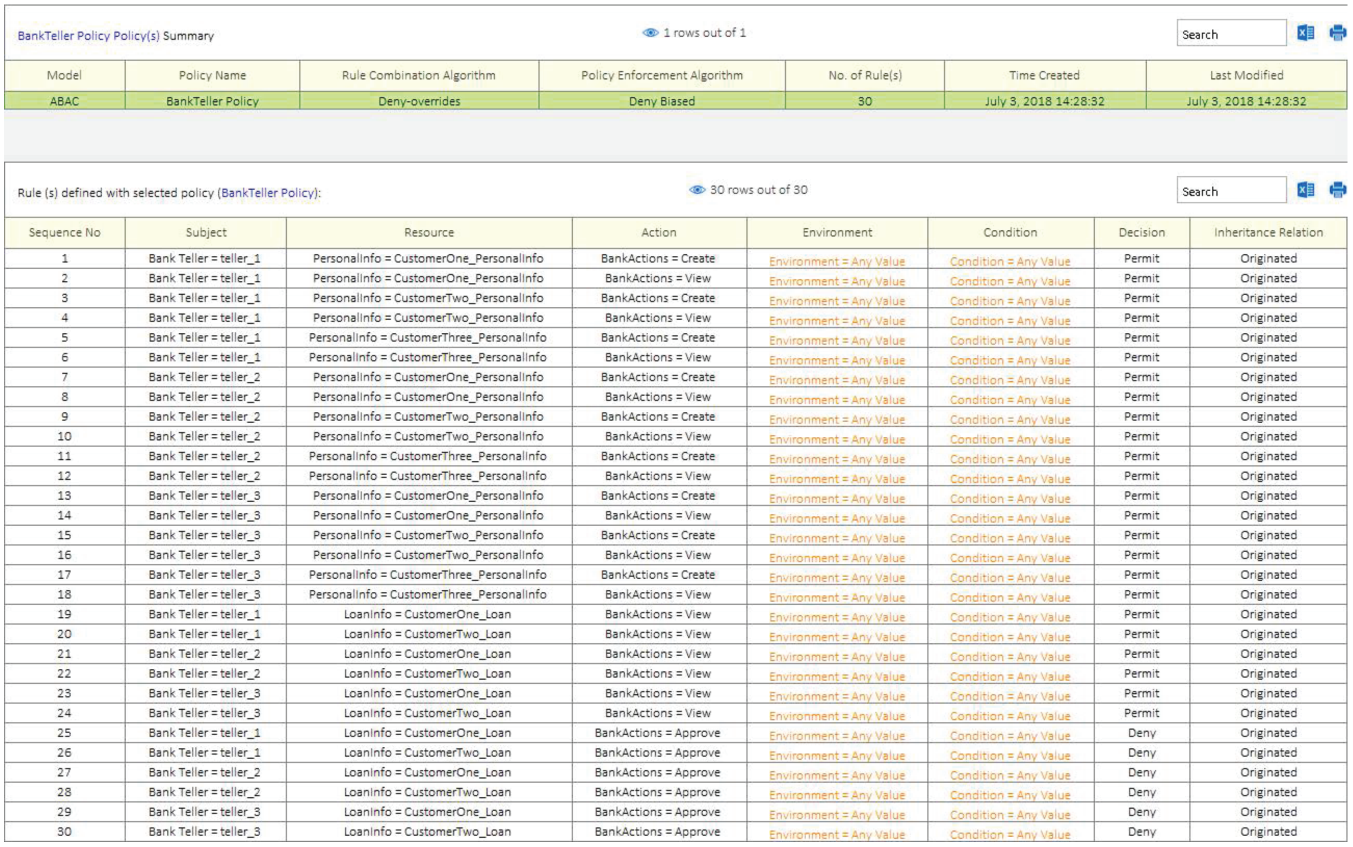
BankTeller Policy:
(Bank Teller = teller_1, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_1, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_2, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_2, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_3, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_3, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_1, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_1, CustomerTwo_Loan, Approve) →Deny
(Bank Teller = teller_2, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_2, CustomerTwo_Loan, Approve) →Deny
(Bank Teller = teller_3, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_3, CustomerTwo_Loan, Approve) →Deny

LoanOfficer Policy:
(Loan Officer = officer_1, CustomerOne_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerOne_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerTwo_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerThree_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerThree_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, View) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_Loan, View) →Deny
(Loan Officer = officer_1, CustomerTwo_Loan, Create) →Permit
(Loan Officer = officer_2, CustomerOne_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerOne_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerTwo_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerTwo_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerThree_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerThree_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerOne_Loan, View) →Permit
(Loan Officer = officer_2, CustomerOne_Loan, Create) →Permit
(Loan Officer = officer_2, CustomerTwo_Loan, View) →Permit
(Loan Officer = officer_2, CustomerTwo_Loan, Create) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, Approve) →Deny
(Loan Officer = officer_1, CustomerTwo_Loan, Approve) →Deny
(Loan Officer = officer_2, CustomerOne_Loan, Approve) →Deny
(Loan Officer = officer_2, CustomerTwo_Loan, Approve) →Deny

FManager Policy:
(Manager = manager, CustomerOne_PersonalInfo, Create) →Permit
(Manager = manager, CustomerOne_PersonalInfo, View) →Permit
(Manager = manager, CustomerTwo_PersonalInfo, Create) →Permit
(Manager = manager, CustomerTwo_PersonalInfo, View) →Permit

(Manager = manager, CustomerThree_PersonalInfo, Create) →Permit
(Manager = manager, CustomerThree_PersonalInfo, View) →Permit
(Manager = manager, CustomerOne_Loan, Create) →Permit
(Manager = manager, CustomerOne_Loan, View) →Permit
(Manager = manager, CustomerTwo_Loan, Create) →Permit
(Manager = manager, CustomerTwo_Loan, View) →Permit
(Manager = manager, CustomerOne_Loan, Approve) →Permit
(Manager = manager, CustomerTwo_Loan, Approve) →Permit

After entering the rules above your modeled policies should look like the screenshots below. If you did not create your own Project File, you can simply open Security Policy Tool – Project File: BankTestCase2 and these policies will have been already created for you.



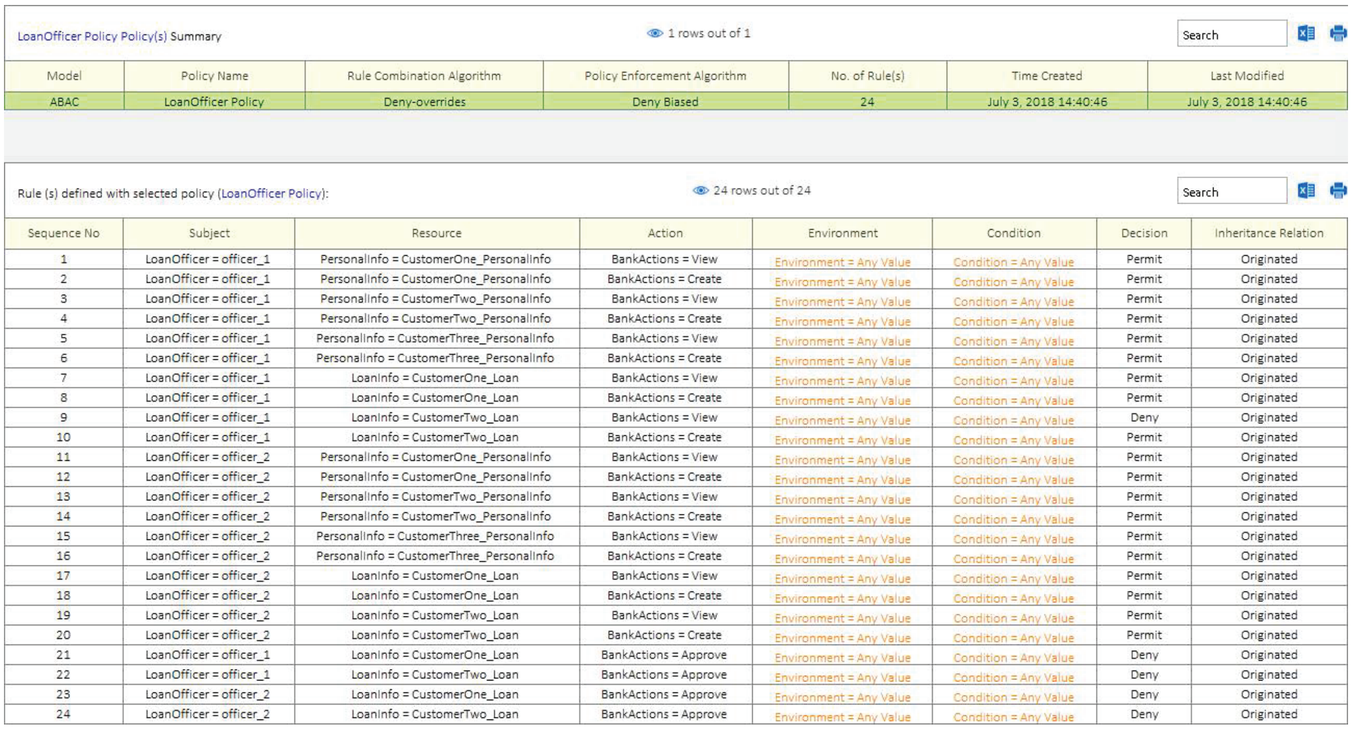
BankTeller Policy Policy(s) Summary 1 rows out of 1 Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	BankTeller Policy	Deny-overrides	Deny Biased	30	July 3, 2018 14:28:32	July 3, 2018 14:28:32

Rule (s) defined with selected policy (BankTeller Policy): 30 rows out of 30 Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
2	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
3	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
4	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
5	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
6	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
7	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
8	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
9	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
10	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
11	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
12	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
13	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
14	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
15	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
16	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
17	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
18	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
19	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
20	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
21	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
22	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
23	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
24	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
25	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
26	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
27	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
28	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
29	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
30	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 11. BankTeller Policy


LoanOfficer Policy Policy(s) Summary

1 rows out of 1

Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	LoanOfficer Policy	Deny-overrides	Deny Biased	24	July 3, 2018 14:40:46	July 3, 2018 14:40:46

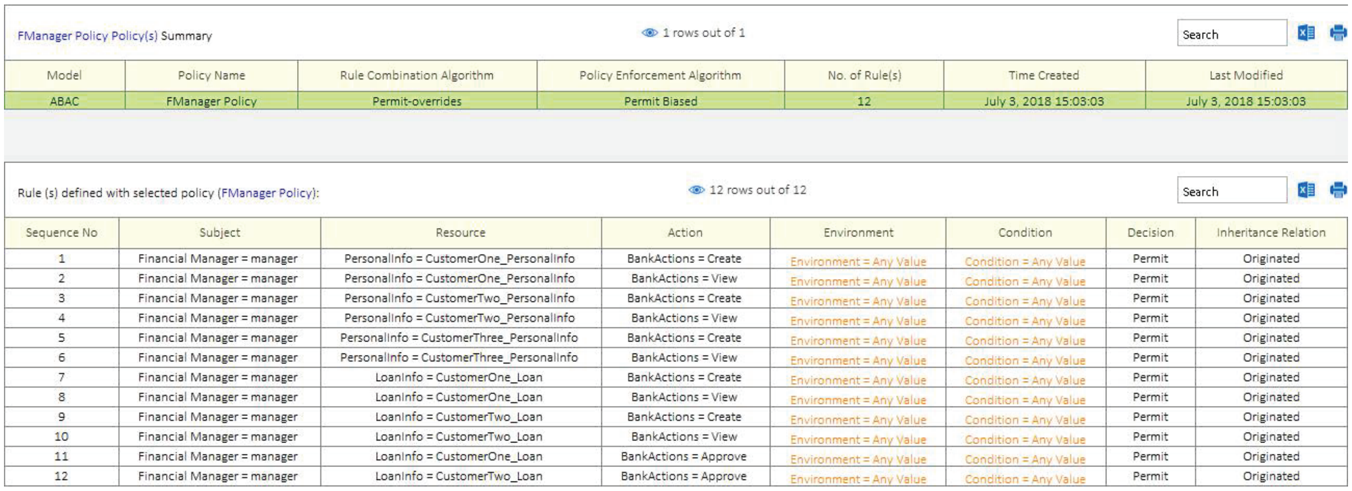
Rule (s) defined with selected policy (LoanOfficer Policy):

24 rows out of 24

Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
2	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
3	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
4	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
5	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
6	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
7	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
8	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
9	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Deny	Originated
10	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
11	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
12	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
13	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
14	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
15	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
16	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
17	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
18	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
19	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
20	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
21	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
22	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
23	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
24	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 12. LoanOfficer Policy


FManager Policy Policy(s) Summary

1 rows out of 1

Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	FManager Policy	Permit-overrides	Permit Biased	12	July 3, 2018 15:03:03	July 3, 2018 15:03:03

Rule (s) defined with selected policy (FManager Policy):

12 rows out of 12

Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	Financial Manager = manager	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
2	Financial Manager = manager	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
3	Financial Manager = manager	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
4	Financial Manager = manager	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
5	Financial Manager = manager	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
6	Financial Manager = manager	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
7	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
8	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
9	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
10	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
11	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated
12	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated

Fig. 13. FManager Policy

9 Individual Security Requirement - Test Case 2 (Block Privilege)

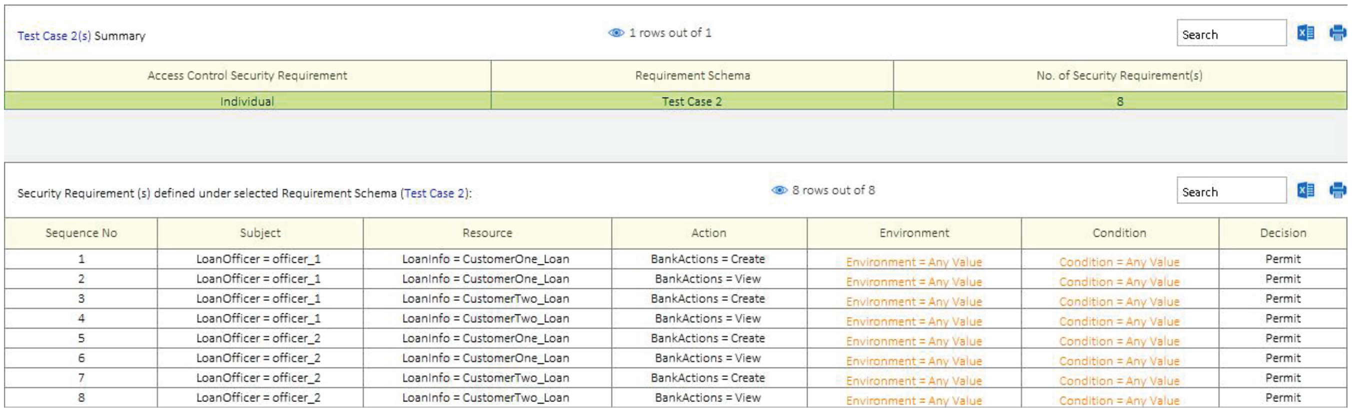
The final step before analyzing these policies for errors is to create individual security requirements to use for testing. If you are building a "New (blank) Project" on your own you will enter the following security requirements below:

Individual Security Requirements:

(Loan Officer = officer_1) & (Action = Create) & (LoanInfo = CustomerOne_Loan) $\rightarrow$ decision = Permit
(Loan Officer = officer_1) & (Action = View) & (LoanInfo = CustomerOne_Loan) $\rightarrow$ decision = Permit
(Loan Officer = officer_1) & (Action = Create) & (LoanInfo = CustomerTwo_Loan) $\rightarrow$ decision = Permit
(Loan Officer = officer_1) & (Action = View) & (LoanInfo = CustomerTwo_Loan) $\rightarrow$ decision = Permit
(Loan Officer = officer_2) & (Action = Create) & (LoanInfo = CustomerOne_Loan) $\rightarrow$ decision = Permit

(Loan Officer = officer_2) & (Action = View) & (LoanInfo = CustomerOne_Loan) → decision = Permit
(Loan Officer = officer_2) & (Action = Create) & (LoanInfo = CustomerTwo_Loan) → decision = Permit
(Loan Officer = officer_2) & (Action = View) & (LoanInfo = CustomerTwo_Loan) → decision = Permit

After entering the rules above your individual security requirements should look like the screenshot below. If you did not create your own Project File you can simply open Security Policy Tool – Project File: BankTestCase2 and these requirements will have been already created for you.



Test Case 2(s) Summary

1 rows out of 1

Search

Access Control Security Requirement	Requirement Schema	No. of Security Requirement(s)
Individual	Test Case 2	8

Security Requirement (s) defined under selected Requirement Schema (Test Case 2):

8 rows out of 8

Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision
1	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit
2	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit
3	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit
4	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit
5	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit
6	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit
7	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit
8	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit

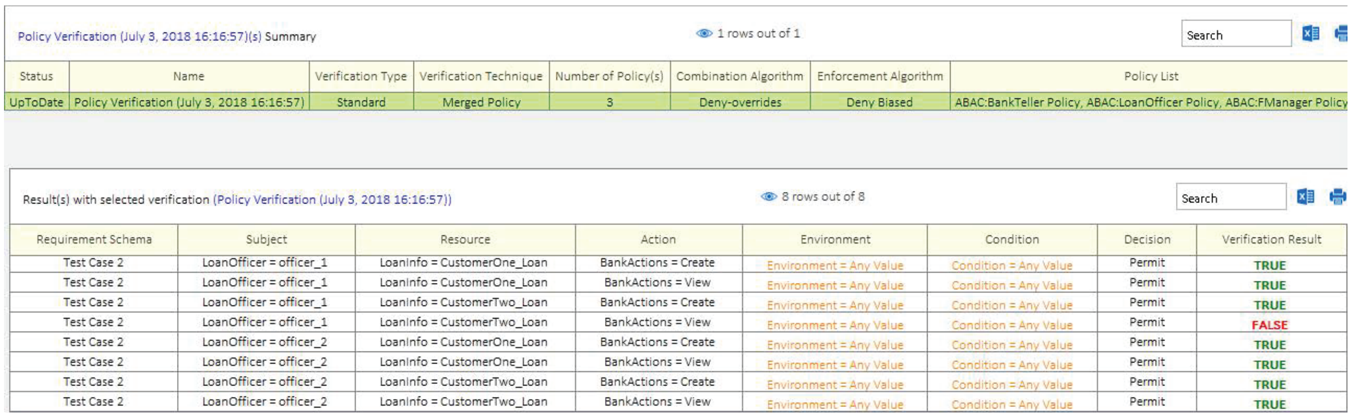
Fig. 14. Individual Security Requirement

10 Policy Verification/Analyzing Results - Test Case 2 (BLock Privilege)

Now that we are ready to test our policies let's discuss the error we will be looking at in this second example. When policies are designed there is potential for a "Block Privilege" error being created. A Block Privilege error occurs when policy rules are blocking a subject's legitimate access to rightful resources. A Block Privilege is created when a policy cannot render a grant or deny decision, no available logic in the AC policy algorithm for evaluating the access request, or by mistaken privilege assignment directly.

For example, when the policy author was designing the logic for these bank policies; the author intended all LoanOfficers to be able to Create all LoanInfo resources. However, to be able to Create all LoanInfo resources, LoanOfficers also need to be able to View all LoanInfo resources. To ensure the policy has been designed correctly let's verify that this is true (e.g., LoanOfficer → View or Create → LoanInfo Resources).

We will run one "Merged Policy" Verification with all three of our policies to reveal the Block Privilege error that is present in our policies. To do this, we will select Test Case 2 (security requirement) and BankTeller Policy & LoanOfficer Policy & FManager Policy as a Merged Policy Verification and analyze our verification result. Again, this will have already been done for you if you open Project File: BankTestCase2.



Policy Verification (July 3, 2018 16:16:57)(s) Summary — 1 rows out of 1 — Search

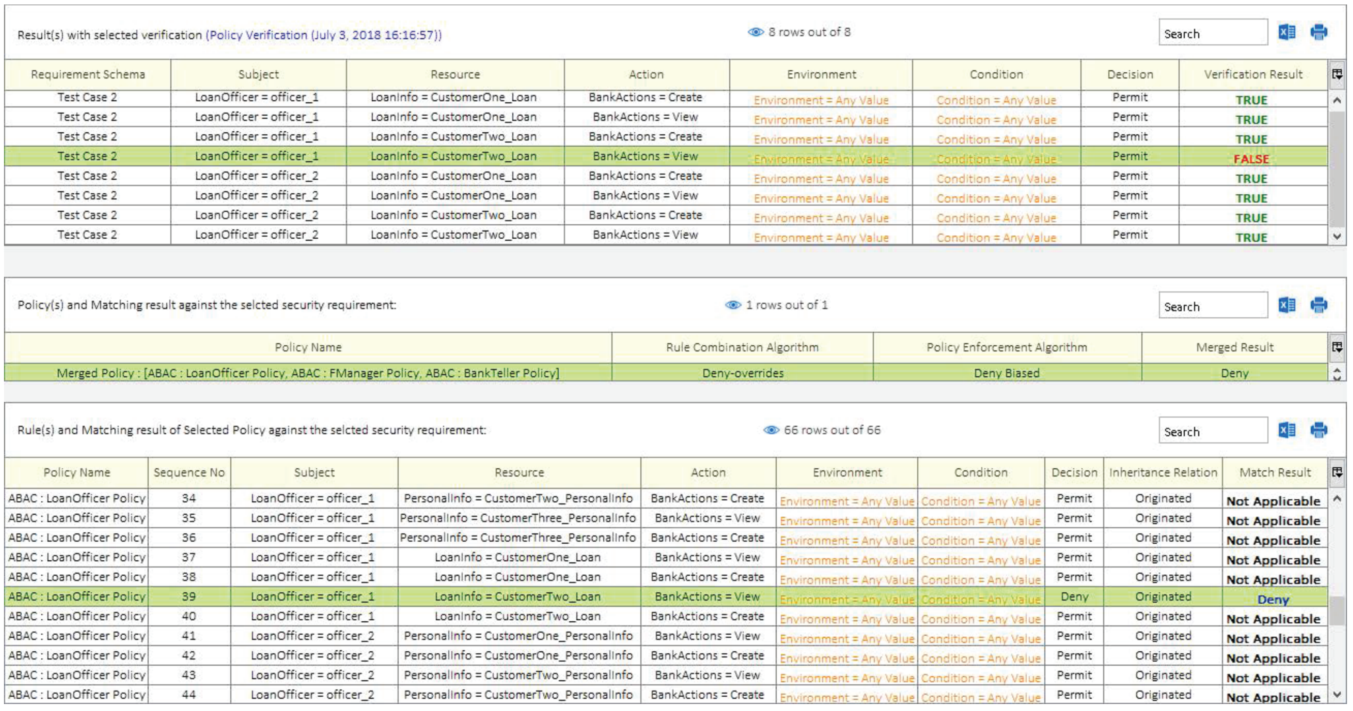
Status	Name	Verification Type	Verification Technique	Number of Policy(s)	Combination Algorithm	Enforcement Algorithm	Policy List
UpToDate	Policy Verification (July 3, 2018 16:16:57)	Standard	Merged Policy	3	Deny-overrides	Deny Biased	ABAC:BankTeller Policy, ABAC:LoanOfficer Policy, ABAC:FManager Policy

Result(s) with selected verification (Policy Verification (July 3, 2018 16:16:57)) — 8 rows out of 8 — Search

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE

Fig. 15. Merged Policy x Test Case 2

Right away we will notice our Verification Results contains a False which is known as Block Privilege. A rule or rules in our policy are incorrectly telling the system to not allow officer_1 to View CustomerTwo_Loan which is required for officer_1 to be able to Create LoanInfo resources. If we click on our Security Requirement (4), we can analyze deeper the reasoning for this result we have received. Here is where we will notice we have mistakenly selected the wrong privilege assignment for this subject's access on this resource.



Result(s) with selected verification (Policy Verification (July 3, 2018 16:16:57)) — 8 rows out of 8 — Search

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	FALSE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE

Policy(s) and Matching result against the selcted security requirement: — 1 rows out of 1 — Search

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Merged Result
Merged Policy : [ABAC : LoanOfficer Policy, ABAC : FManager Policy, ABAC : BankTeller Policy]	Deny-overrides	Deny Biased	Deny

Rule(s) and Matching result of Selected Policy against the selcted security requirement: — 66 rows out of 66 — Search

Policy Name	Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation	Match Result
ABAC : LoanOfficer Policy	34	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	35	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	36	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	37	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	38	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	39	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Deny	Originated	Deny
ABAC : LoanOfficer Policy	40	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	41	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	42	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	43	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
ABAC : LoanOfficer Policy	44	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable

Fig. 16. Merged Policy: Match Results

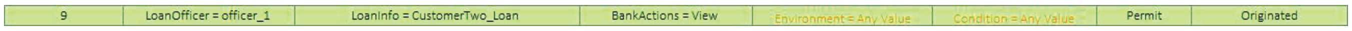
11 Resolving This Error - Test Case 2 (Block Privilege)

To eliminate a Block Privilege error the policy author would need to go back and either update the related rule that is creating this error or delete it and define a Policy Enforcement Algorithm as Permit Biased because there will be no specific rule for this scenario. However, this could potentially create errors for other rules in our policy so we will modify the exact rule in this situation.

For example, if we were to modify this rule below in the LoanOfficer Policy to Permit…

LoanOfficer Policy: Modify Rule (9):

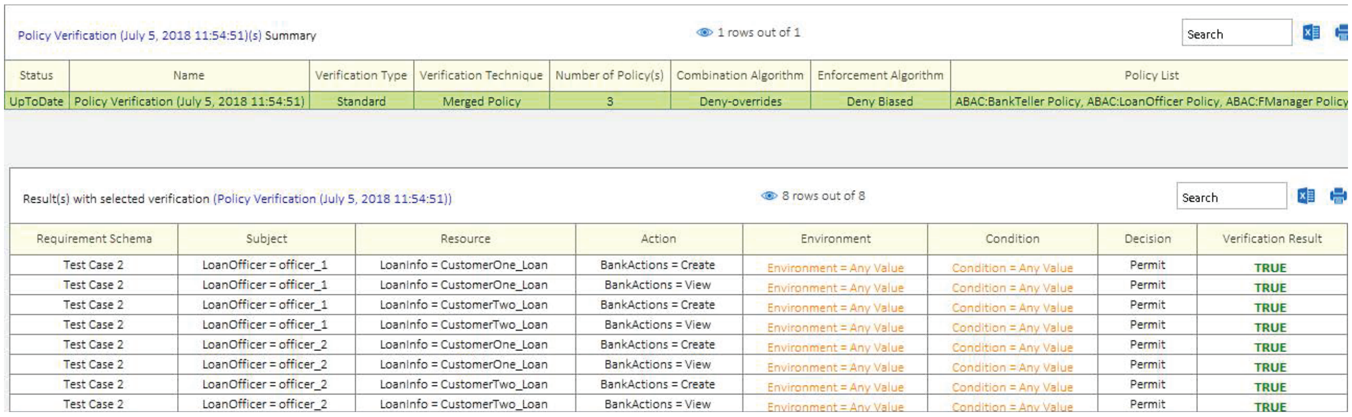
(Rule No. = 9) → (LoanOfficer = officer_1) → (Action = View) → (Resource = CustomerTwo_Loan) → decision = Permit



9	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated

Fig. 17. LoanOfficer Policy: Modified Rule (9)

Then retest using the same Policy Verification selections as last time we will now get a True Verification result showing that we no longer have a Block Privilege error occuring.



Policy Verification (July 5, 2018 11:54:51)(s) Summary

1 rows out of 1

Status	Name	Verification Type	Verification Technique	Number of Policy(s)	Combination Algorithm	Enforcement Algorithm	Policy List
UpToDate	Policy Verification (July 5, 2018 11:54:51)	Standard	Merged Policy	3	Deny-overrides	Deny Biased	ABAC:BankTeller Policy, ABAC:LoanOfficer Policy, ABAC:FManager Policy

Result(s) with selected verification (Policy Verification (July 5, 2018 11:54:51))

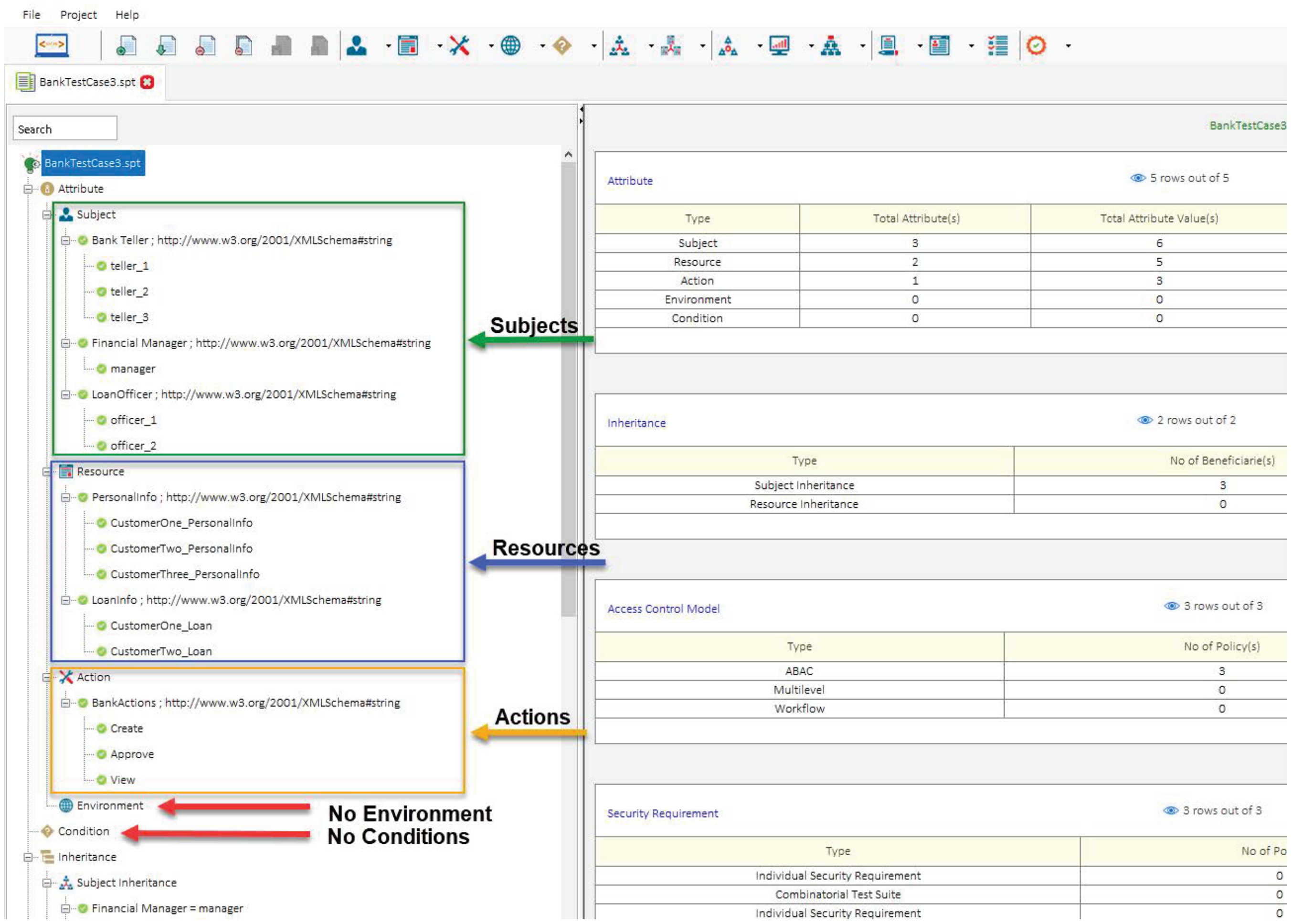
8 rows out of 8

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE
Test Case 2	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	TRUE

Fig. 18. Updated Policy: No Longer Blocking

12 SETTING UP THE POLICIES – TEST CASE 3 (INHERITANCE LOOP)

This bank example contains three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). The attributes in this example have not been changed from previous Test Case 1 and Test Case 2. The Attribute/Attribute Values included in these policies are as shown in Figure 19.



BankTestCase3.spt
- Attribute
  - Subject
    - Bank Teller ; http://www.w3.org/2001/XMLSchema#string
      - teller_1
      - teller_2
      - teller_3
    - Financial Manager ; http://www.w3.org/2001/XMLSchema#string
      - manager
    - LoanOfficer ; http://www.w3.org/2001/XMLSchema#string
      - officer_1
      - officer_2
  - Resource
    - PersonalInfo ; http://www.w3.org/2001/XMLSchema#string
      - CustomerOne_PersonalInfo
      - CustomerTwo_PersonalInfo
      - CustomerThree_PersonalInfo
    - LoanInfo ; http://www.w3.org/2001/XMLSchema#string
      - CustomerOne_Loan
      - CustomerTwo_Loan
  - Action
    - BankActions ; http://www.w3.org/2001/XMLSchema#string
      - Create
      - Approve
      - View
  - Environment
- Condition
- Inheritance
  - Subject Inheritance
    - Financial Manager = manager

BankTestCase3

Attribute — 5 rows out of 5

Type	Total Attribute(s)	Total Attribute Value(s)
Subject	3	6
Resource	2	5
Action	1	3
Environment	0	0
Condition	0	0

Inheritance — 2 rows out of 2

Type	No of Beneficiarie(s)
Subject Inheritance	3
Resource Inheritance	0

Access Control Model — 3 rows out of 3

Type	No of Policy(s)
ABAC	3
Multilevel	0
Workflow	0

Security Requirement — 3 rows out of 3

Type	No of Po
Individual Security Requirement	0
Combinatorial Test Suite	0
Individual Security Requirement	0

Fig. 19. Test Case 3

13 Subject Inheritance – Test Case 3 (Inheritance Loop)

Depending on your security needs or organizational structure you may decide to define Inheritance relationships to help you generate policy Rules more quickly. For this bank policy example we will define (3) Subject Inheritance Relationships. If you are creating your own Blank Project enter the relationship rules below. If you did not create your own Project File, you can simply open Security Policy Tool – Project File: BankTestCase3 and these relationships will have been already created for you.

Subject Inheritance:

Beneficiary Values → Financial Manager = manager
Inherited Values → Loan Officer = officer_1 & officer_2

Beneficiary Values → Loan Officer = officer_1
Inherited Values → Bank Teller = teller_1 & teller_2 & teller_3

Beneficiary Values → Loan Officer = officer_2
Inherited Values → Bank Teller = teller_1 & teller_2 & teller_3

If we have created these Relationships correctly based on the above direction it will look like this in Security Policy Tool:

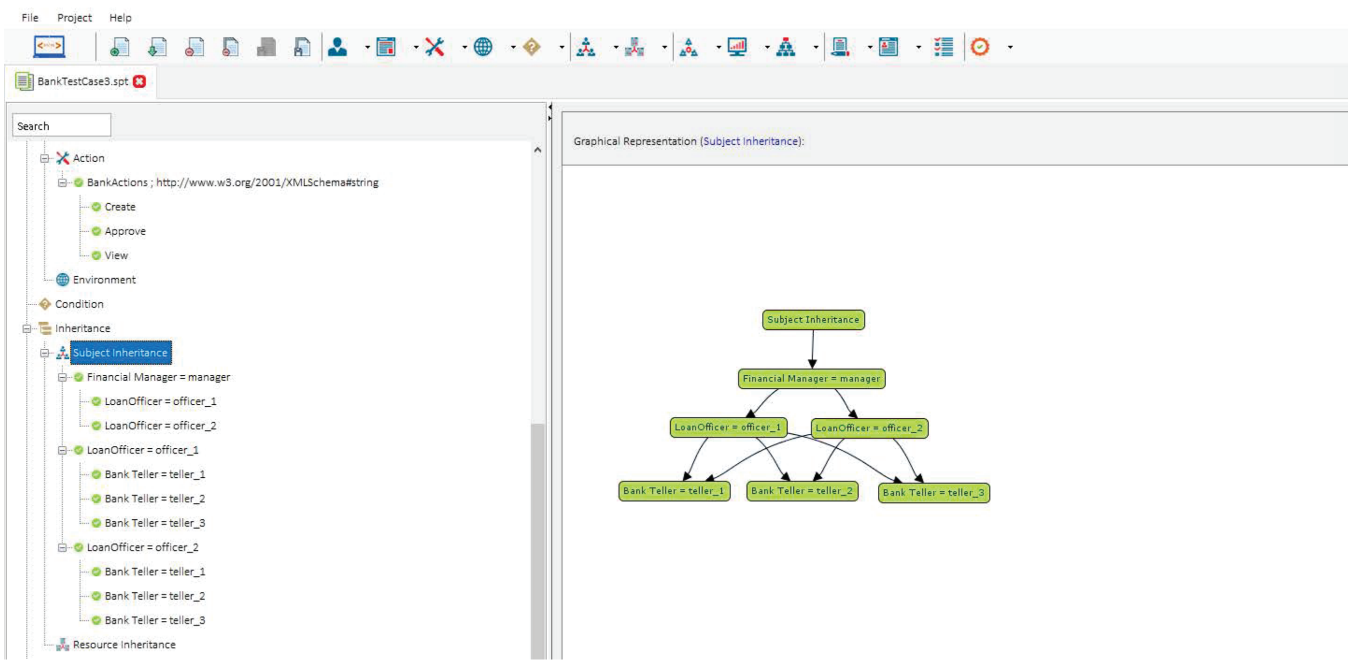


Fig. 20. Subject Inheritance Relationships

By defining these relationships, any (Originated) Rules with Decision = Permit given to Bank Teller subject attributes will now also be given to officer_1 & officer_2 as (Inherited) Rules in our policies. Additionally any Rules with Decision = Permit given to Loan Officers subject attributes will now also be given to the manager subject attribute. Originated Rules with Decision = Deny are never Inherited. This is because typically Beneficiaries in these relationships are higher ranking/senior roles that by nature will have less restrictions (e.g., denying access) than roles that are providing the Inheritance Value (e.g., bank teller/loan officer in our example).

Hence, it will authorize the Beneficiaries to obtain all privileges of Inherited Values (e.g., generally lower-level roles) while not obtaining their typically tighter restrictions. If you would like Beneficiaries to be Denied access similar to their Inherited Value you can still do so by manually creating individual rules when you begin modeling.

14 Understanding This Error - Test Case 3 (Inheritance Loop)

Unlike all other Test Case examples, to demonstrate this error we do not need to run any policy tests. Thus we do not have sections for modeling our policies, creating security requirements, and running verification tests for this example. An Inheritance Loop is an error that occurs when the policy author defines inheritance relations that gives a subject both recursive and subsequent inheritance.

For example, an inheritance loop or sometimes called ”‘cyclical inheritance”’ could look like this..

Person A can inherit $\rightarrow$ Person B Privileges
Person B can inherit $\rightarrow$ Person C Privileges

Person C can inherit $\rightarrow$ Person A Privileges

In our example an Inheritance Loop would look like the screenshot below...

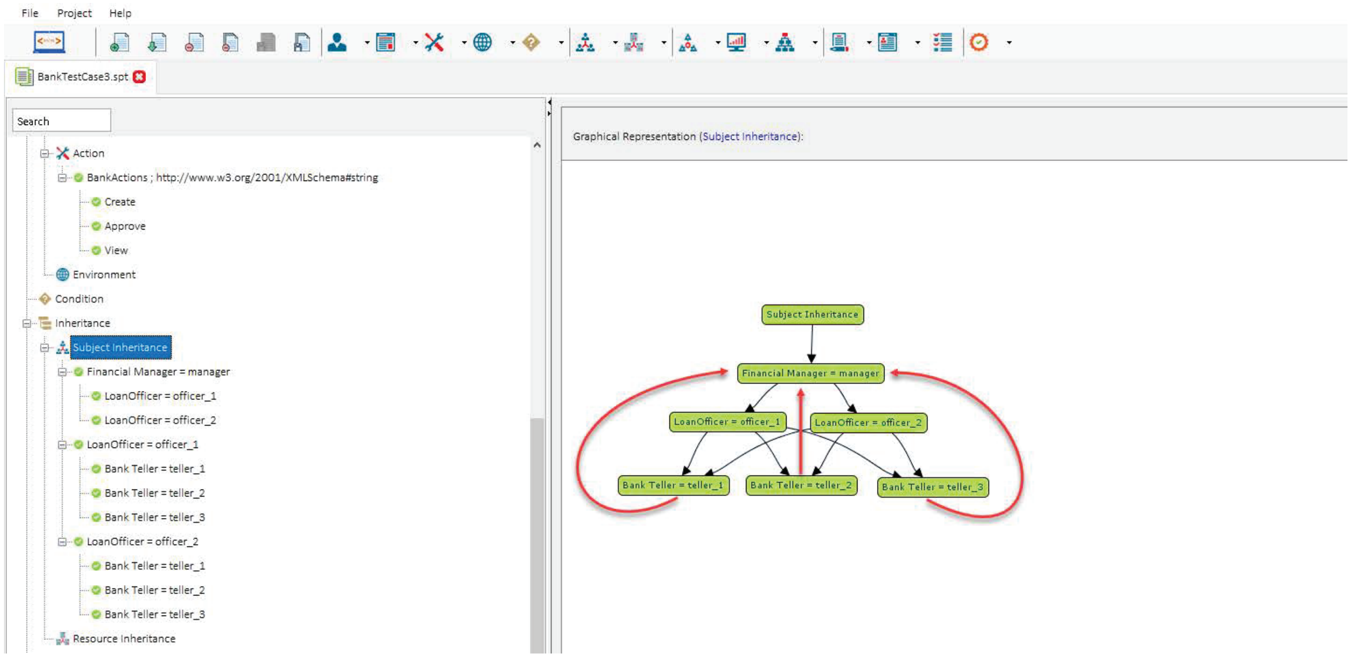


Fig. 21. Inheritance Loop

However, in Security Policy Tool it is not possible to create this type of error. Security Policy Tool automatically detects which attribute values are being selected as Beneficiaries/Inherited Values to prevent Inheritance Loops. It will not allow attributes that are already allocated as Inherited Values to a Beneficiary Value to then also be allocated as a Beneficiary Value to the Beneficiary Value they are already giving their (inheritance) rules to.

See the screenshot below, values that would create an Inheritance Loop are unavailable to be selected...

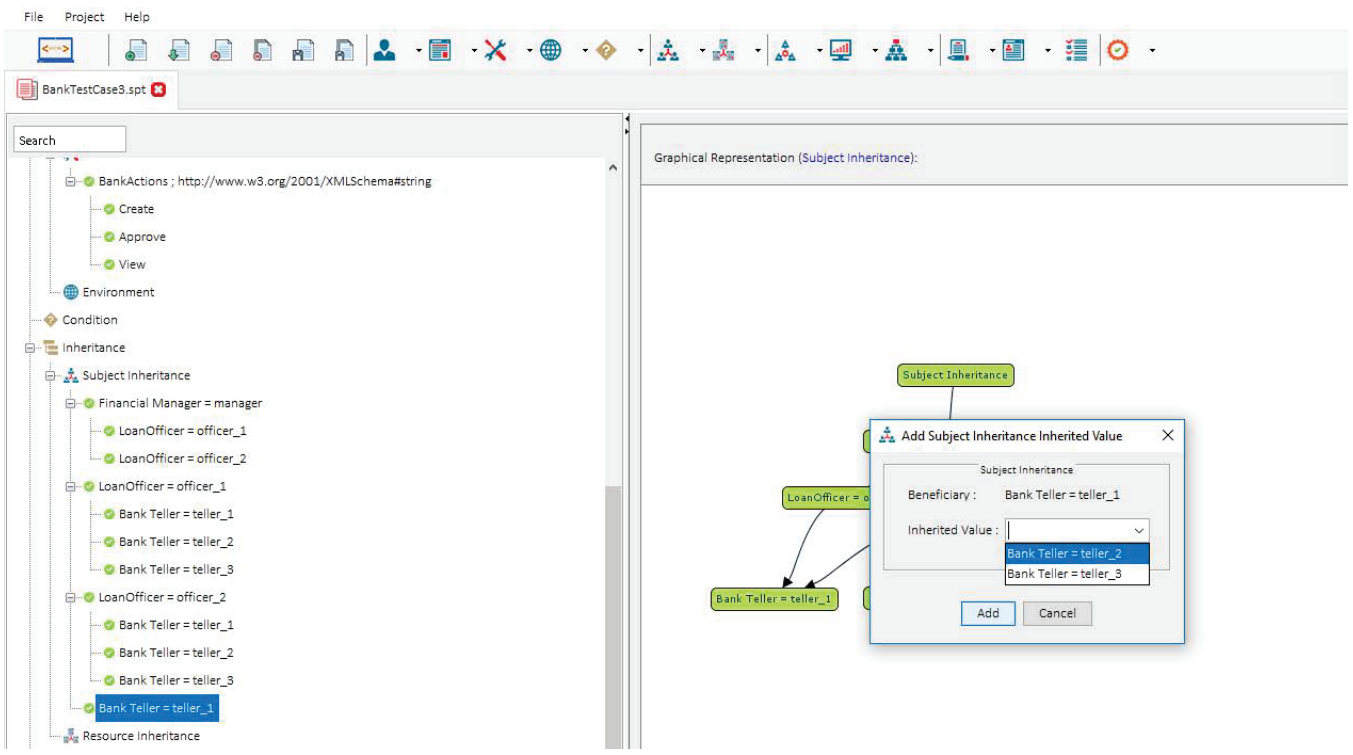
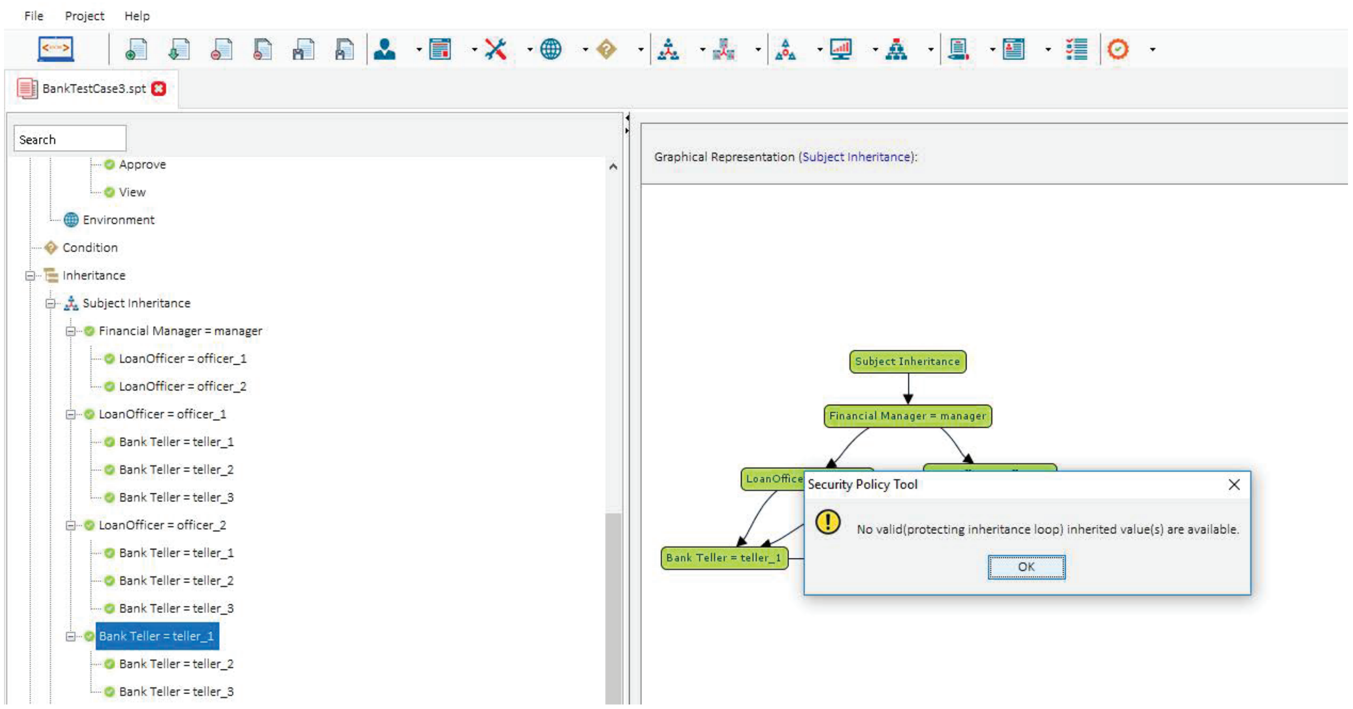


Fig. 22. Subject Inheritance: Cannot Add Any Value

In our example, allowing Bank Teller = teller_1, teller_2, or teller_3 to be able to be a Beneficiary for LoanOfficer = officer_1 or officer_2 or Manager = manager would create an Inheritance Loop. Security Policy Tool detects this issue and only allows Bank Tellers to be able to be defined as a Beneficiary to other Bank Tellers since they are not Inheriting Values from each other at this moment.

If we continue with our example and make teller_1 the beneficiary to teller_2 and teller_3 and attempt to define another Inherited Value for teller_1 we will get this message below and avoid creating an Inheritance Loop. See screenshot...

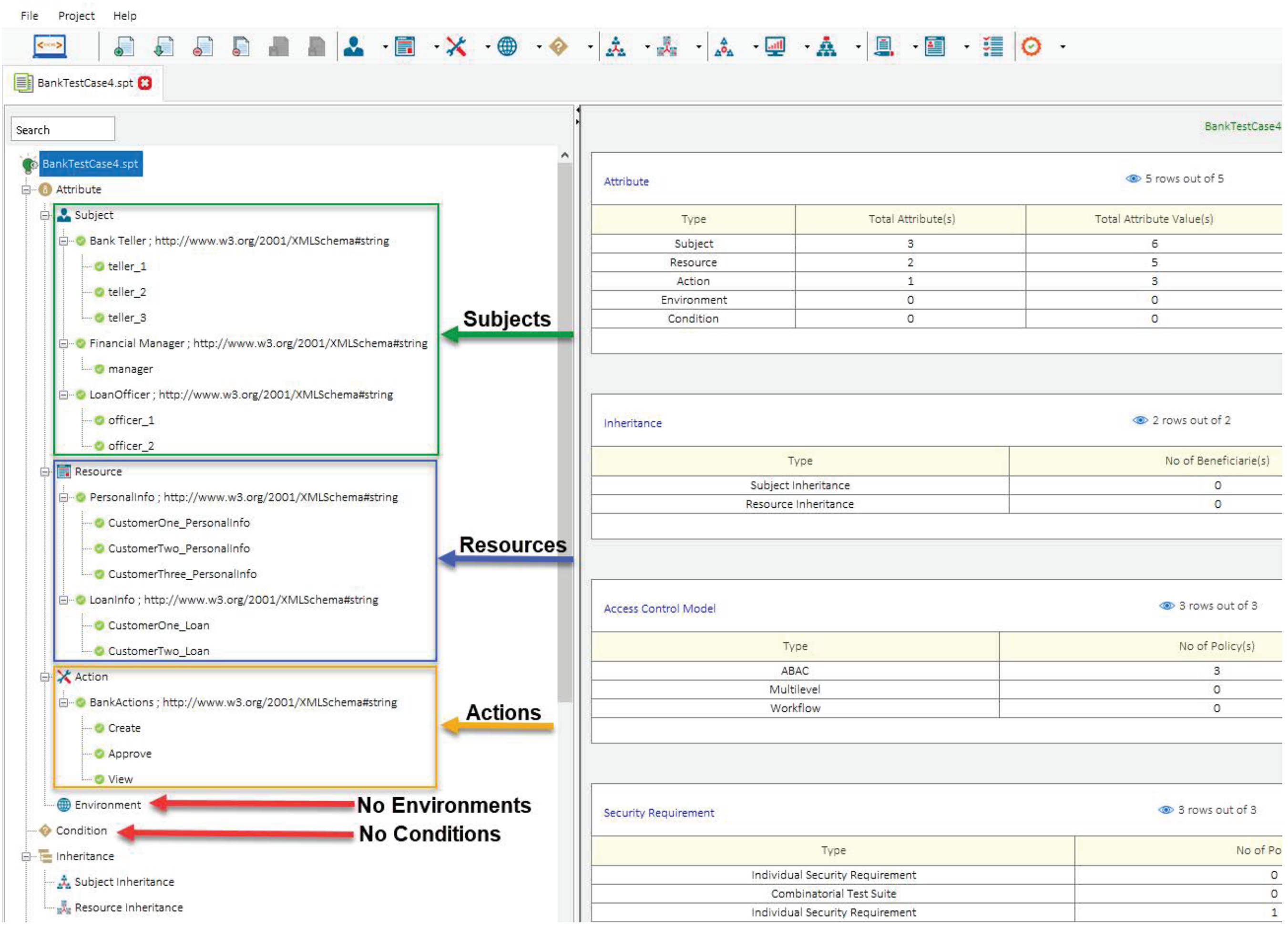


No valid(protecting inheritance loop) inherited value(s) are available.

Fig. 23. Subject Inheritance: Loop Error Prevented

15 Setting Up The Policies – Test Case 4 (Separation of Duty)

This bank example contains three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). The attributes in this example have not been changed from previous Bank Test Cases. The Attribute/Attribute Values included in these policies are as shown in Figure 24.



```
BankTestCase4.spt
  Attribute
    Subject
      Bank Teller ; http://www.w3.org/2001/XMLSchema#string
        teller_1
        teller_2
        teller_3
      Financial Manager ; http://www.w3.org/2001/XMLSchema#string
        manager
      LoanOfficer ; http://www.w3.org/2001/XMLSchema#string
        officer_1
        officer_2
    Resource
      PersonalInfo ; http://www.w3.org/2001/XMLSchema#string
        CustomerOne_PersonalInfo
        CustomerTwo_PersonalInfo
        CustomerThree_PersonalInfo
      LoanInfo ; http://www.w3.org/2001/XMLSchema#string
        CustomerOne_Loan
        CustomerTwo_Loan
    Action
      BankActions ; http://www.w3.org/2001/XMLSchema#string
        Create
        Approve
        View
    Environment
  Condition
  Inheritance
    Subject Inheritance
    Resource Inheritance
```

Attribute — 5 rows out of 5

Type	Total Attribute(s)	Total Attribute Value(s)
Subject	3	6
Resource	2	5
Action	1	3
Environment	0	0
Condition	0	0

Inheritance — 2 rows out of 2

Type	No of Beneficiarie(s)
Subject Inheritance	0
Resource Inheritance	0

Access Control Model — 3 rows out of 3

Type	No of Policy(s)
ABAC	3
Multilevel	0
Workflow	0

Security Requirement — 3 rows out of 3

Type	No of Po
Individual Security Requirement	0
Combinatorial Test Suite	0
Individual Security Requirement	1

Fig. 24. Test Case 4

16 Modeling Your Policy – Test Case 4 (Seperation of Duty)

Now that we have entered our attributes we can model our three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). See the list below of the rules contained in each of these policies. You can open a "New (blank) Project" and build these policies by entering the following rules below:

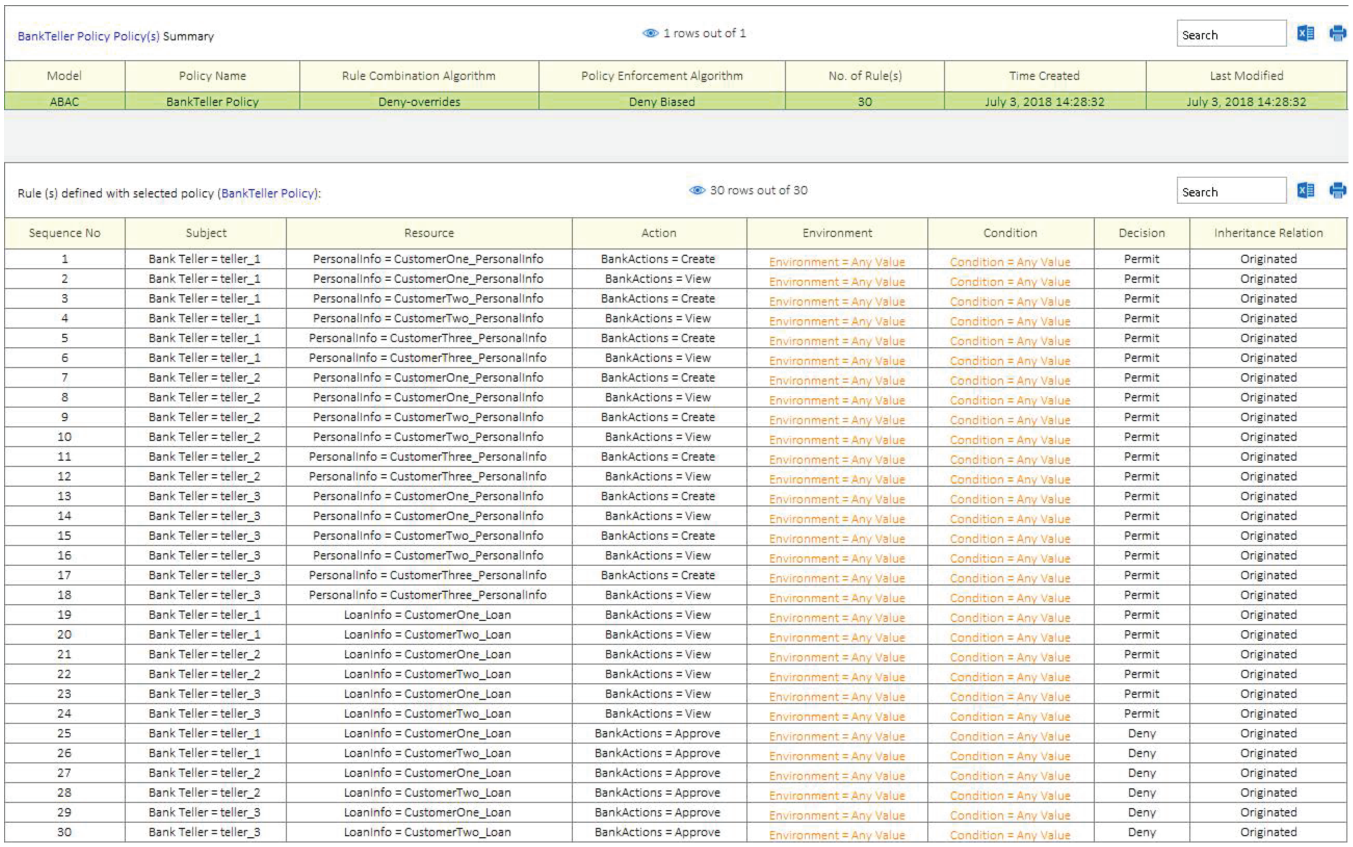
BankTeller Policy:
(Bank Teller = teller_1, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerThree_PersonalInfo, View) →Permit

(Bank Teller = teller_1, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_1, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_2, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_2, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_3, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_3, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_1, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_1, CustomerTwo_Loan, Approve) →Deny
(Bank Teller = teller_2, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_2, CustomerTwo_Loan, Approve) →Deny
(Bank Teller = teller_3, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_3, CustomerTwo_Loan, Approve) →Deny

LoanOfficer Policy:
(Loan Officer = officer_1, CustomerOne_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerOne_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerTwo_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerThree_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerThree_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, View) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_Loan, View) →Permit
(Loan Officer = officer_1, CustomerTwo_Loan, Create) →Permit
(Loan Officer = officer_2, CustomerOne_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerOne_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerTwo_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerTwo_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerThree_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerThree_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerOne_Loan, View) →Permit
(Loan Officer = officer_2, CustomerOne_Loan, Create) →Permit
(Loan Officer = officer_2, CustomerTwo_Loan, View) →Permit
(Loan Officer = officer_2, CustomerTwo_Loan, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_Loan, Approve) →Deny
(Loan Officer = officer_2, CustomerOne_Loan, Approve) →Deny
(Loan Officer = officer_2, CustomerTwo_Loan, Approve) →Deny

FManager Policy:
(Manager = manager, CustomerOne_PersonalInfo, Create) →Permit
(Manager = manager, CustomerOne_PersonalInfo, View) →Permit
(Manager = manager, CustomerTwo_PersonalInfo, Create) →Permit
(Manager = manager, CustomerTwo_PersonalInfo, View) →Permit
(Manager = manager, CustomerThree_PersonalInfo, Create) →Permit
(Manager = manager, CustomerThree_PersonalInfo, View) →Permit
(Manager = manager, CustomerOne_Loan, Create) →Permit
(Manager = manager, CustomerOne_Loan, View) →Permit
(Manager = manager, CustomerTwo_Loan, Create) →Permit
(Manager = manager, CustomerTwo_Loan, View) →Permit
(Manager = manager, CustomerOne_Loan, Approve) →Permit
(Manager = manager, CustomerTwo_Loan, Approve) →Permit

After entering the rules above your modeled policies should look like the screenshots below. If you did not create your own Project File, you can simply open Security Policy Tool – Project File: BankTestCase4 and these policies will have been already created for you.



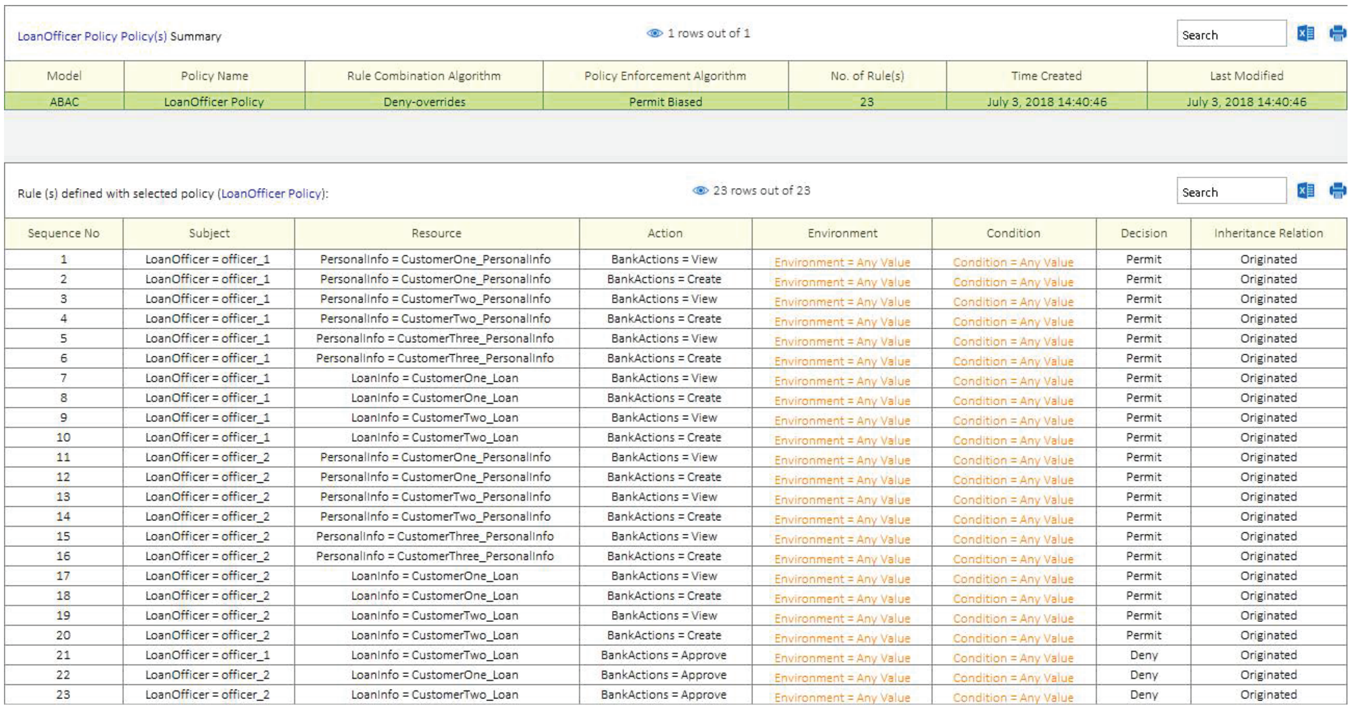
BankTeller Policy Policy(s) Summary — 1 rows out of 1 — Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	BankTeller Policy	Deny-overrides	Deny Biased	30	July 3, 2018 14:28:32	July 3, 2018 14:28:32

Rule (s) defined with selected policy (BankTeller Policy): — 30 rows out of 30 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
2	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
3	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
4	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
5	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
6	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
7	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
8	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
9	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
10	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
11	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
12	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
13	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
14	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
15	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
16	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
17	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
18	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
19	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
20	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
21	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
22	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
23	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
24	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
25	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
26	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
27	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
28	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
29	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
30	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 25. BankTeller Policy



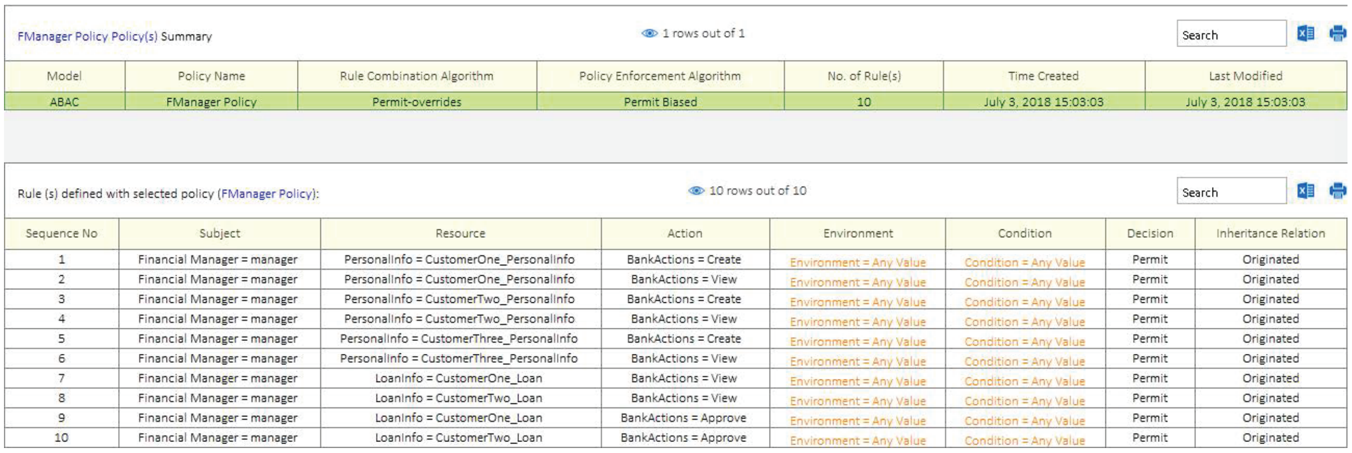
LoanOfficer Policy Policy(s) Summary — 1 rows out of 1 — Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	LoanOfficer Policy	Deny-overrides	Permit Biased	23	July 3, 2018 14:40:46	July 3, 2018 14:40:46

Rule (s) defined with selected policy (LoanOfficer Policy): — 23 rows out of 23 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
2	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
3	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
4	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
5	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
6	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
7	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
8	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
9	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
10	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
11	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
12	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
13	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
14	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
15	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
16	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
17	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
18	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
19	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
20	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
21	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
22	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
23	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 26. LoanOfficer Policy



FManager Policy Policy(s) Summary — 1 rows out of 1 — Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	FManager Policy	Permit-overrides	Permit Biased	10	July 3, 2018 15:03:03	July 3, 2018 15:03:03

Rule (s) defined with selected policy (FManager Policy): — 10 rows out of 10 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	Financial Manager = manager	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
2	Financial Manager = manager	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
3	Financial Manager = manager	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
4	Financial Manager = manager	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
5	Financial Manager = manager	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
6	Financial Manager = manager	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
7	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
8	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
9	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated
10	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated

Fig. 27. FManager Policy

17 SOD Security Requirements - Test Case 4 (Separation of Duty)

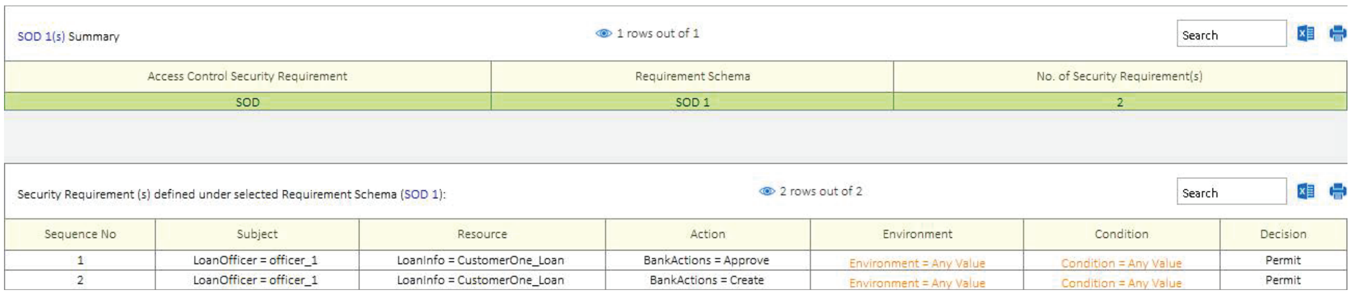
The final step before analyzing these policies for errors is to create security requirements to use for testing. If you are building a "New (blank) Project" on your own you will enter the following security requirements below:

SOD Security Requirements:

(Loan Officer = officer_1) & (Action = Approve) & (LoanInfo = CustomerOne_Loan) → decision = Permit

(Loan Officer = officer_1) & (Action = Create) & (LoanInfo = CustomerOne_Loan) → decision = Permit

After entering the rules above your SOD security requirements should look like the screenshot below. If you did not create your own Project File you can simply open Security Policy Tool – Project File: BankTestCase4 and these requirements will have been already created for you.



SOD 1(s) Summary — 1 rows out of 1 — Search

Access Control Security Requirement	Requirement Schema	No. of Security Requirement(s)
SOD	SOD 1	2

Security Requirement (s) defined under selected Requirement Schema (SOD 1): — 2 rows out of 2 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision
1	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit
2	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit

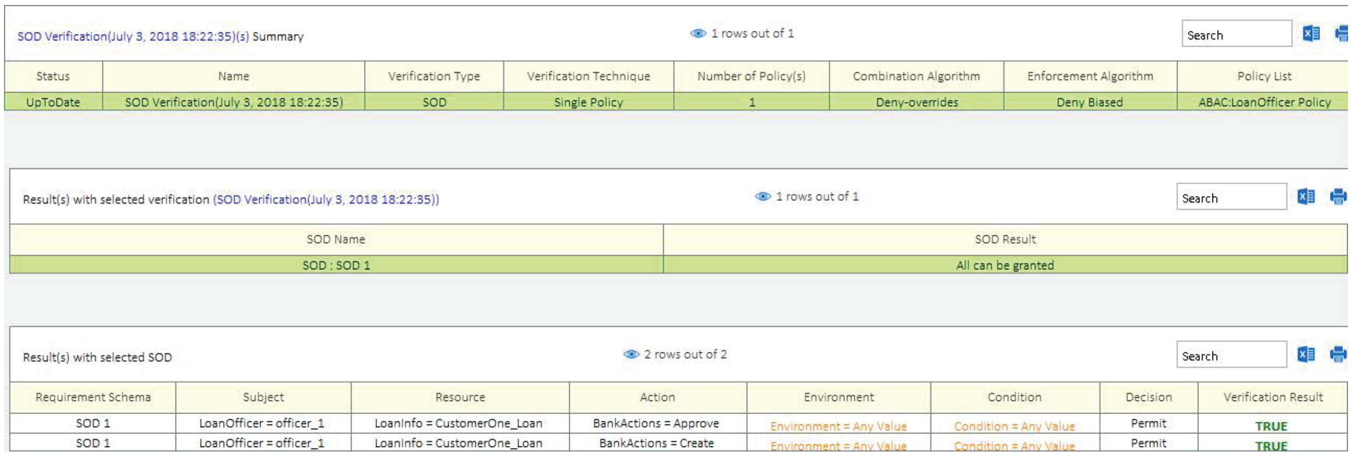
Fig. 28. SOD Security Requirements

18 Policy Verification/Analyzing Results - Test Case 4 (Separation of Duty)

Now that we are ready to test our policies let's discuss the error we will be looking for in this fourth example. When policies are designed there is potential for a "Separation of Duty" error being created. A Separation of Duty error occurs when two or more rules cause competing interests among subjects, resources, or actions. For example, giving one subject too much privileges to the point that they could misuse the system.

For example, when the policy author was designing the logic for these bank policies; the author intended all LoanOfficers to be able to Create all LoanInfo resources. However, they should not be able to Approve loans as way to ensure Loan Officers do not misuse their privileges to create improper loans. To ensure the policy has been designed correctly let's verify that this is false (e.g., LoanOfficer → Approve → LoanInfo Resources).

We will run one "Single Policy" Separation of Duty Verification with our LoanOfficer Policy to reveal the Separation of Duty error that is present in our policies. To do this, we will select SOD 1 (security requirement) and LoanOfficer Policy and analyze our verification results. Again, this will have already been done for you if you open Project File: BankTestCase4.



SOD Verification(July 3, 2018 18:22:35)(s) Summary — 1 rows out of 1

Status	Name	Verification Type	Verification Technique	Number of Policy(s)	Combination Algorithm	Enforcement Algorithm	Policy List
UpToDate	SOD Verification(July 3, 2018 18:22:35)	SOD	Single Policy	1	Deny-overrides	Deny Biased	ABAC:LoanOfficer Policy

Result(s) with selected verification (SOD Verification(July 3, 2018 18:22:35)) — 1 rows out of 1

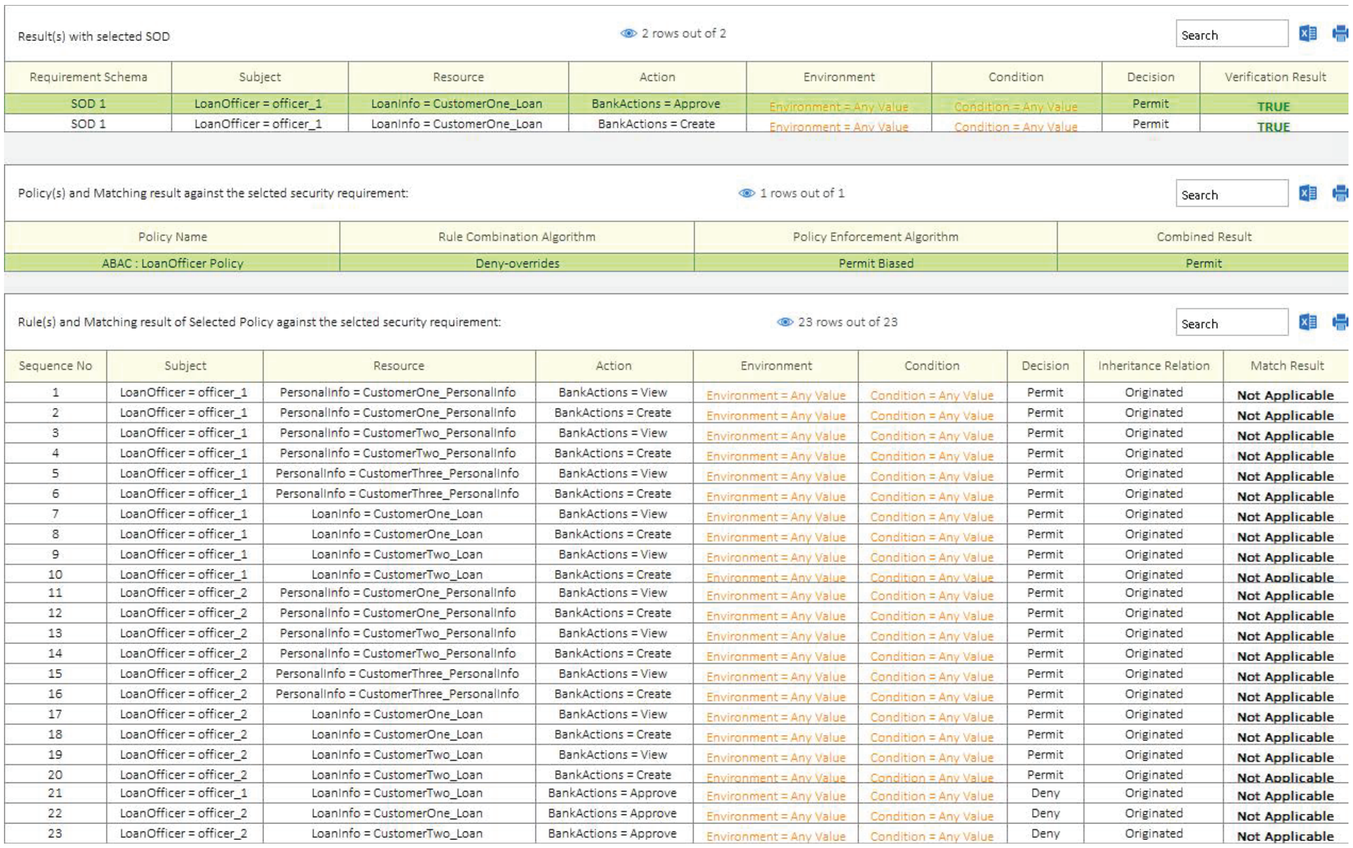
SOD Name	SOD Result
SOD : SOD 1	All can be granted

Result(s) with selected SOD — 2 rows out of 2

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
SOD 1	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	TRUE
SOD 1	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE

Fig. 29. LoanOfficer Policy x SOD 1

Right away we will notice our test came back with two True Results which is not what was intended. LoanOfficers should be able to Create loans but should not be able to Approve loans. A rule or rules in our policy are incorrectly telling the system to allow officer_1 to Approve CustomerTwo_Loan. If we click on our Security Requirement related to Approving the resource, we can analyze deeper the reasoning for this result we have received.



Result(s) with selected SOD — 2 rows out of 2 — Search

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
SOD 1	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	TRUE
SOD 1	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE

Policy(s) and Matching result against the selcted security requirement: — 1 rows out of 1 — Search

Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	Combined Result
ABAC : LoanOfficer Policy	Deny-overrides	Permit Biased	Permit

Rule(s) and Matching result of Selected Policy against the selcted security requirement: — 23 rows out of 23 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation	Match Result
1	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
2	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
3	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
4	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
5	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
6	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
7	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
8	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
9	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
10	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
11	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
12	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
13	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
14	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
15	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
16	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
17	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
18	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
19	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
20	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated	Not Applicable
21	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated	Not Applicable
22	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated	Not Applicable
23	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated	Not Applicable

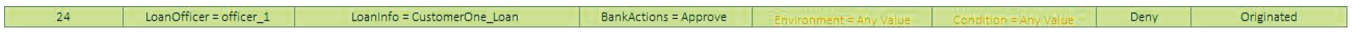
Fig. 30. LoanOfficer Policy: Match Results

19 Resolving This Error - Test Case 4 (Separation of Duty)

Looking at the Match Results we will see all rules in the LoanOfficer have come back as "'Not Applicable"' which can only mean that we are missing a rule to cover this particular access request. Upon further inspection we will see that in the LoanOfficer Policy all Loan Officers are Denied to Approve LoanInfo resources except for the individual (officer_one) on this specific resource (CustomerOne_Loan). In addition to missing this rule the Policy Enforcement Algorithm has been selected as Permit-Biased thus why the result is Permit with no rules defined related to the request.
To eliminate this error the policy author would need to go back and either update the Policy Enforcement Algorithm to Deny Biased or go back to the LoanOfficer Policy and create a rule for this scenario.

For this example, let's create a new rule in the LoanOfficer Policy...

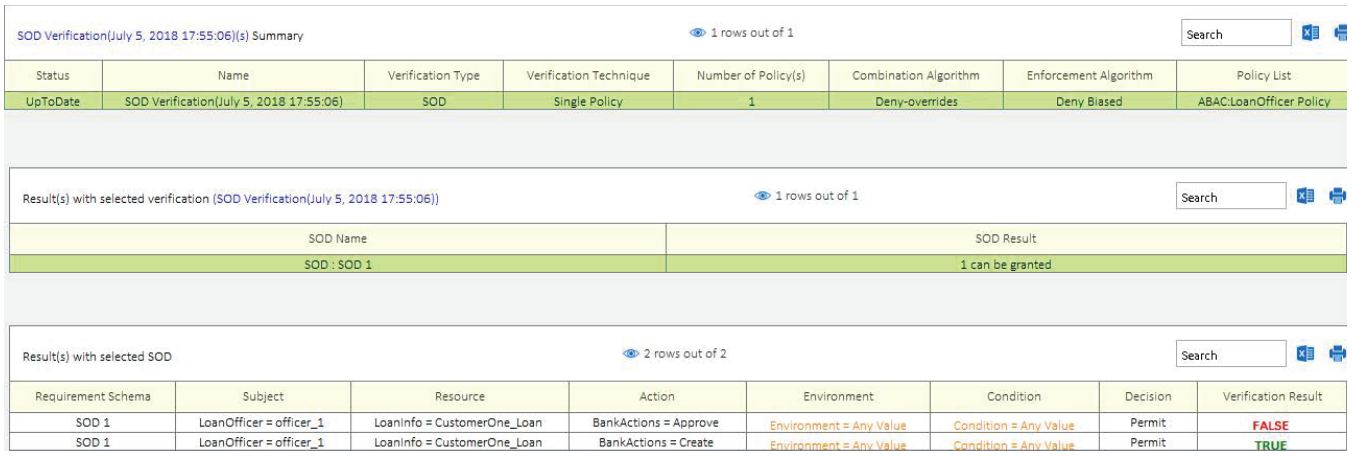
LoanOfficer Policy: New Rule (24):
(Rule No. = 24) $\rightarrow$ (LoanOfficer = officer_1) $\rightarrow$ (Action = Approve) $\rightarrow$ (Resource = CustomerOne_Loan) $\rightarrow$ decision = Deny



24	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 31. LoanOfficer Policy: New Rule (24)

Then retest using the same SOD Verification selections as last time we will now get a False Verification result showing that we no longer have a Seperation of Duty error occurring.



SOD Verification(July 5, 2018 17:55:06)(s) Summary — 1 rows out of 1

Status	Name	Verification Type	Verification Technique	Number of Policy(s)	Combination Algorithm	Enforcement Algorithm	Policy List
UpToDate	SOD Verification(July 5, 2018 17:55:06)	SOD	Single Policy	1	Deny-overrides	Deny Biased	ABAC:LoanOfficer Policy

Result(s) with selected verification (SOD Verification(July 5, 2018 17:55:06)) — 1 rows out of 1

SOD Name	SOD Result
SOD : SOD 1	1 can be granted

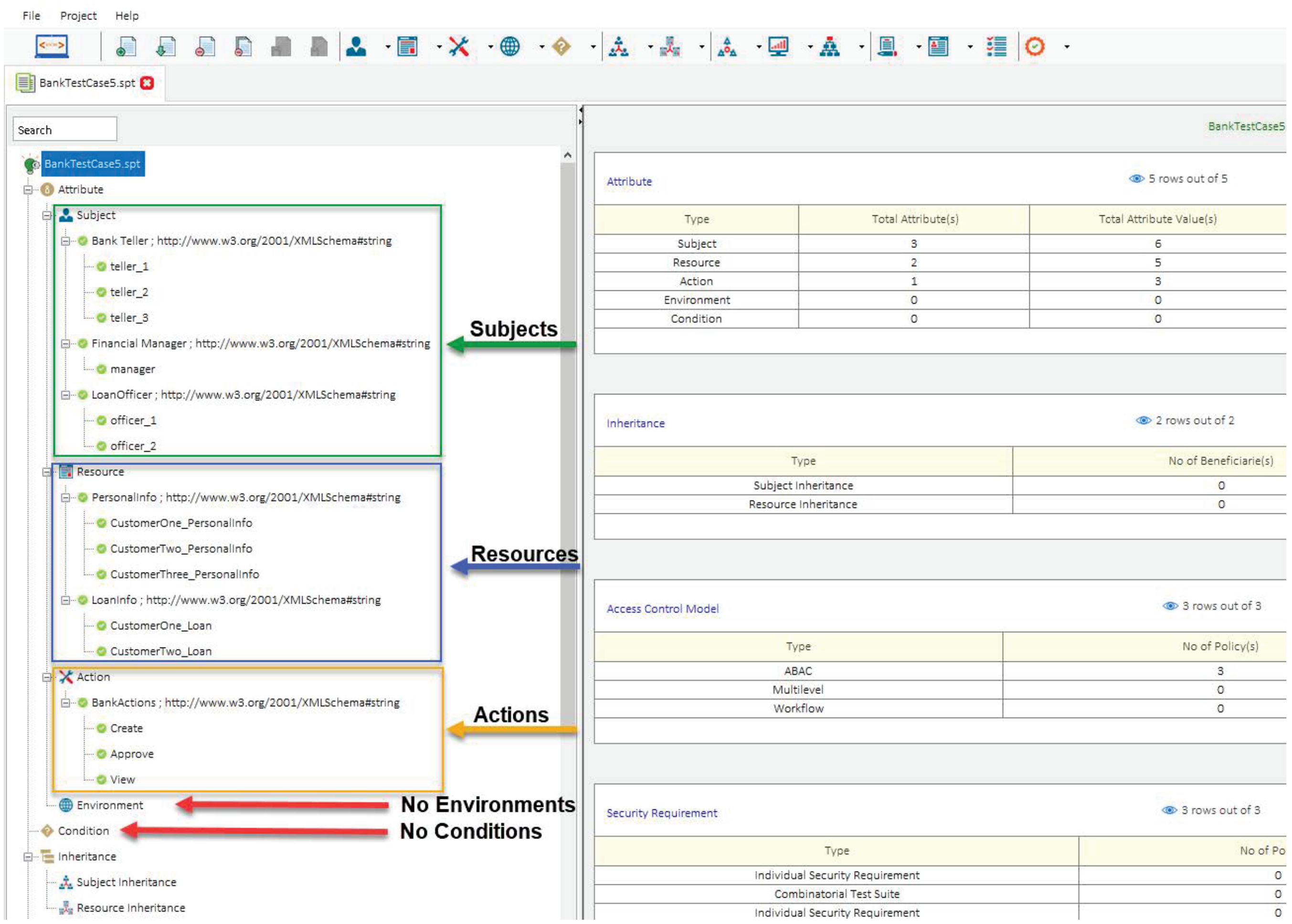
Result(s) with selected SOD — 2 rows out of 2

Requirement Schema	Subject	Resource	Action	Environment	Condition	Decision	Verification Result
SOD 1	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	FALSE
SOD 1	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	TRUE

Fig. 32. Updated Policy: No Seperation of Duty

20 SETTING UP THE POLICIES – TEST CASE 5 (INCONSISTENT ASSIGNMENT)

This bank example contains three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). The attributes in this example have not been changed from previous Test Case examples. The Attribute/Attribute Values included in these policies are as shown in Figure 33.



BankTestCase5.spt
- Attribute
  - Subject
    - Bank Teller ; http://www.w3.org/2001/XMLSchema#string
      - teller_1
      - teller_2
      - teller_3
    - Financial Manager ; http://www.w3.org/2001/XMLSchema#string
      - manager
    - LoanOfficer ; http://www.w3.org/2001/XMLSchema#string
      - officer_1
      - officer_2
  - Resource
    - PersonalInfo ; http://www.w3.org/2001/XMLSchema#string
      - CustomerOne_PersonalInfo
      - CustomerTwo_PersonalInfo
      - CustomerThree_PersonalInfo
    - LoanInfo ; http://www.w3.org/2001/XMLSchema#string
      - CustomerOne_Loan
      - CustomerTwo_Loan
  - Action
    - BankActions ; http://www.w3.org/2001/XMLSchema#string
      - Create
      - Approve
      - View
  - Environment
- Condition
- Inheritance
  - Subject Inheritance
  - Resource Inheritance

BankTestCase5

Attribute — 5 rows out of 5

Type	Total Attribute(s)	Total Attribute Value(s)
Subject	3	6
Resource	2	5
Action	1	3
Environment	0	0
Condition	0	0

Inheritance — 2 rows out of 2

Type	No of Beneficiarie(s)
Subject Inheritance	0
Resource Inheritance	0

Access Control Model — 3 rows out of 3

Type	No of Policy(s)
ABAC	3
Multilevel	0
Workflow	0

Security Requirement — 3 rows out of 3

Type	No of Po
Individual Security Requirement	0
Combinatorial Test Suite	0
Individual Security Requirement	0

Fig. 33. Test Case 5

21 Modeling Your Policy – Test Case 5 (Inconsistent Assignment)

Now that we have entered our attributes we can model our three policies (BankTeller Policy & LoanOfficer Policy & FManager Policy). See the list below of the rules contained in each of these policies. You can open a “New (blank) Project” and build these policies by entering the following rules below:

BankTeller Policy:
(Bank Teller = teller_1, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_1, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_1, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_2, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_2, CustomerThree_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerOne_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerOne_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerTwo_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerTwo_PersonalInfo, View) →Permit
(Bank Teller = teller_3, CustomerThree_PersonalInfo, Create) →Permit
(Bank Teller = teller_3, CustomerThree_PersonalInfo, View) →Permit

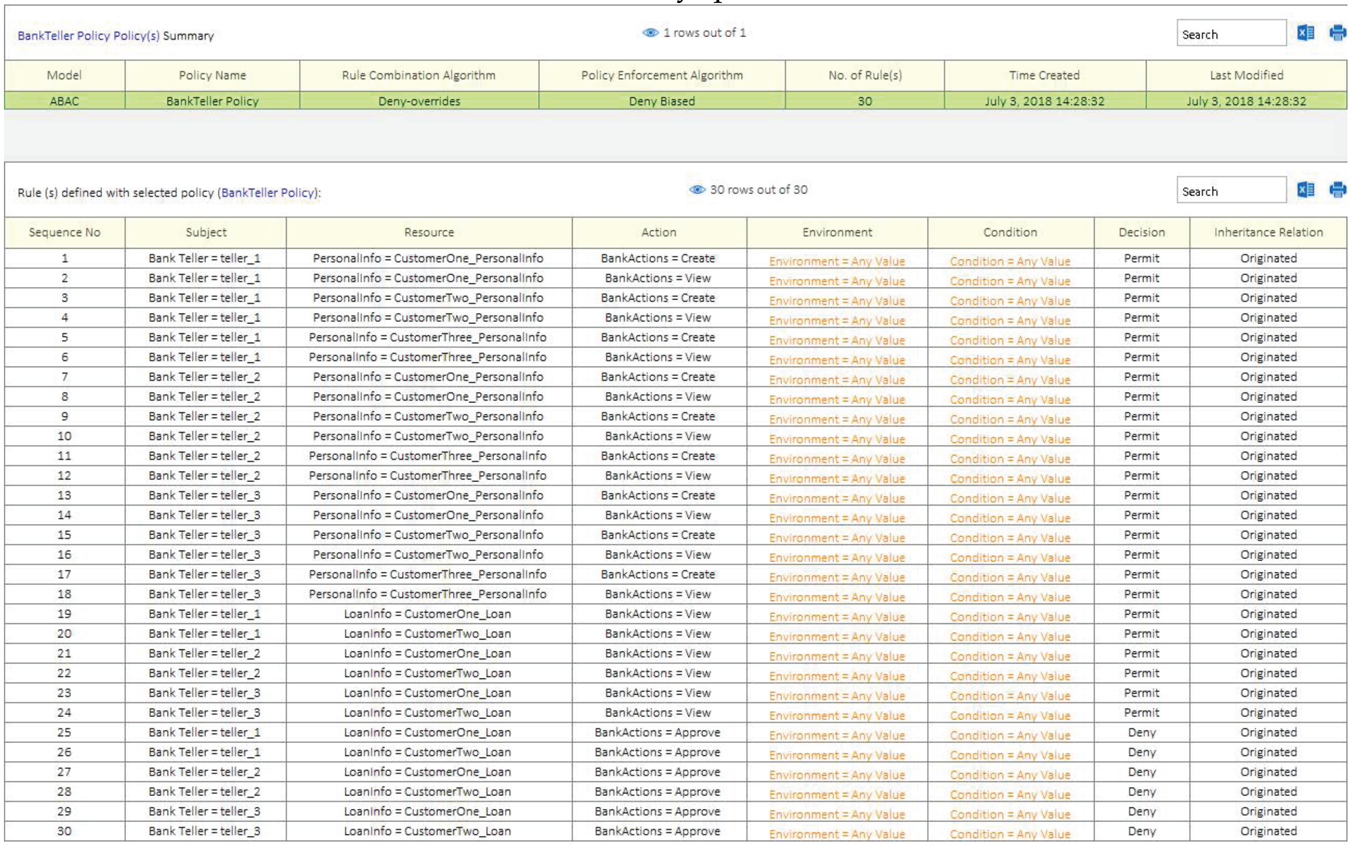
(Bank Teller = teller_1, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_1, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_2, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_2, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_3, CustomerOne_Loan, View) →Permit
(Bank Teller = teller_3, CustomerTwo_Loan, View) →Permit
(Bank Teller = teller_1, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_1, CustomerTwo_Loan, Approve) →Deny
(Bank Teller = teller_2, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_2, CustomerTwo_Loan, Approve) →Deny
(Bank Teller = teller_3, CustomerOne_Loan, Approve) →Deny
(Bank Teller = teller_3, CustomerTwo_Loan, Approve) →Deny

LoanOfficer Policy:
(Loan Officer = officer_1, CustomerOne_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerOne_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerTwo_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerThree_PersonalInfo, View) →Permit
(Loan Officer = officer_1, CustomerThree_PersonalInfo, Create) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, View) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, Create) →Permit
(Loan Officer = officer_1, CustomerTwo_Loan, View) →Permit
(Loan Officer = officer_1, CustomerTwo_Loan, Create) →Permit
(Loan Officer = officer_2, CustomerOne_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerOne_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerTwo_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerTwo_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerThree_PersonalInfo, View) →Permit
(Loan Officer = officer_2, CustomerThree_PersonalInfo, Create) →Permit
(Loan Officer = officer_2, CustomerOne_Loan, View) →Permit
(Loan Officer = officer_2, CustomerOne_Loan, Create) →Permit
(Loan Officer = officer_2, CustomerTwo_Loan, View) →Permit
(Loan Officer = officer_2, CustomerTwo_Loan, Create) →Permit
(Loan Officer = officer_1, CustomerOne_Loan, Approve) →Deny
(Loan Officer = officer_1, CustomerTwo_Loan, Approve) →Deny
(Loan Officer = officer_2, CustomerOne_Loan, Approve) →Deny
(Loan Officer = officer_2, CustomerTwo_Loan, Approve) →Deny

FManager Policy:
(Manager = manager, CustomerOne_PersonalInfo, Create) →Permit
(Manager = manager, CustomerOne_PersonalInfo, View) →Permit
(Manager = manager, CustomerTwo_PersonalInfo, Create) →Permit
(Manager = manager, CustomerTwo_PersonalInfo, View) →Permit
(Manager = manager, CustomerThree_PersonalInfo, Create) →Permit
(Manager = manager, CustomerThree_PersonalInfo, View) →Permit
(Manager = manager, CustomerOne_Loan, Create) →Permit
(Manager = manager, CustomerOne_Loan, View) →Permit
(Manager = manager, CustomerTwo_Loan, Create) →Permit
(Manager = manager, CustomerTwo_Loan, View) →Permit
(Manager = manager, CustomerOne_Loan, Approve) →Permit
(Manager = manager, CustomerTwo_Loan, Approve) →Permit

After entering the rules above your modeled policies should look like the screenshots below. If you did not create your own Project File, you can simply open Security Policy Tool – Project File: BankTestCase5 and these policies will have been already created for you.

Policy.eps



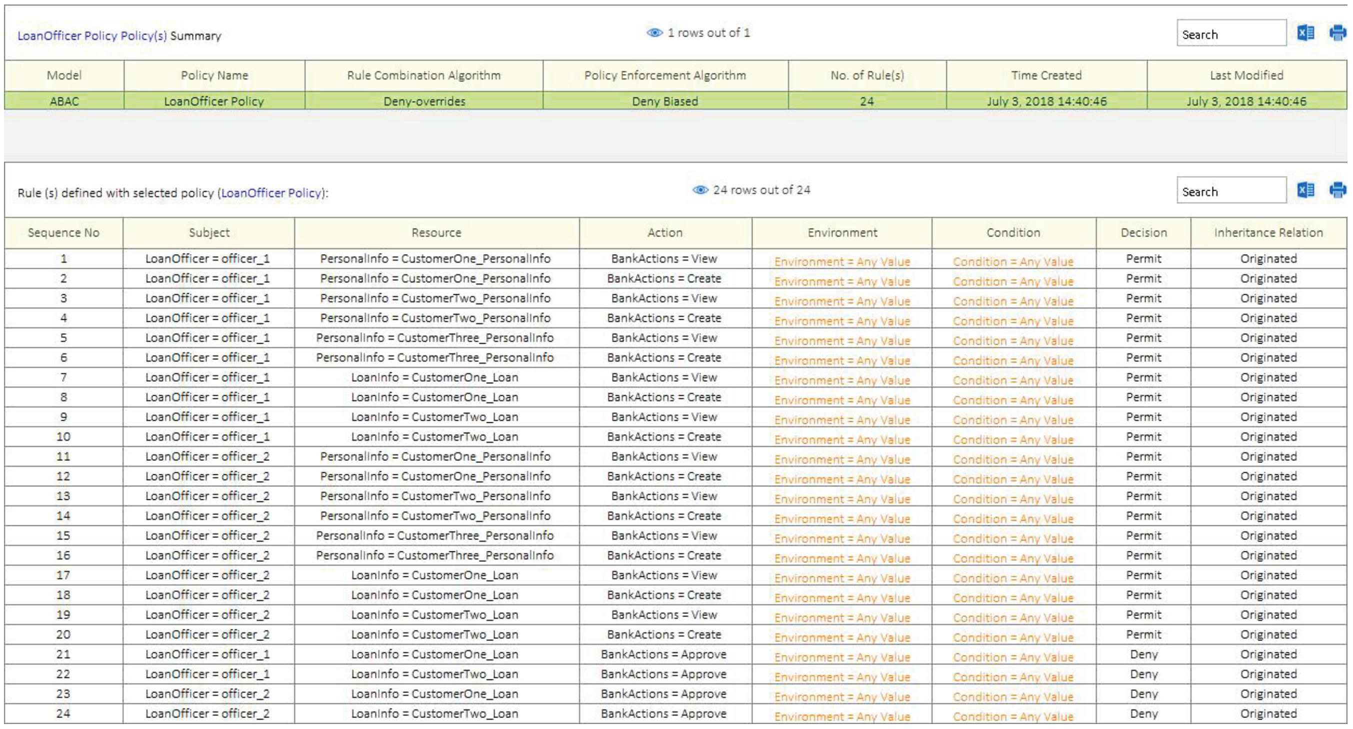
BankTeller Policy Policy(s) Summary — 1 rows out of 1 — Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	BankTeller Policy	Deny-overrides	Deny Biased	30	July 3, 2018 14:28:32	July 3, 2018 14:28:32

Rule (s) defined with selected policy (BankTeller Policy): — 30 rows out of 30 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
2	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
3	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
4	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
5	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
6	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
7	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
8	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
9	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
10	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
11	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
12	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
13	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
14	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
15	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
16	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
17	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
18	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
19	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
20	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
21	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
22	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
23	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
24	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
25	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
26	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
27	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
28	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
29	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
30	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 34. BankTeller Policy



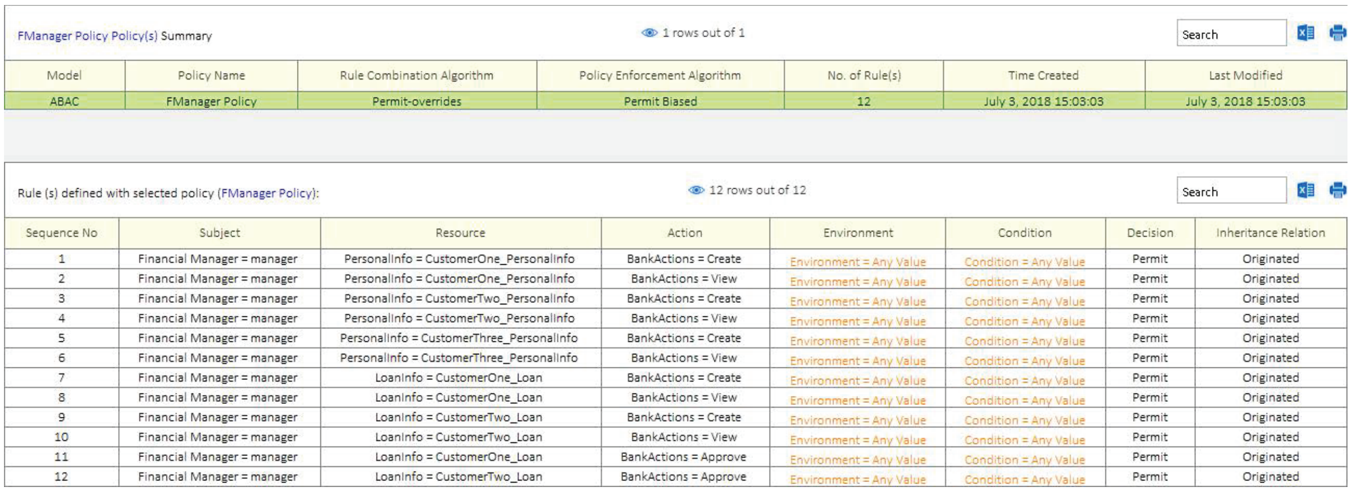
LoanOfficer Policy Policy(s) Summary — 1 rows out of 1 — Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	LoanOfficer Policy	Deny-overrides	Deny Biased	24	July 3, 2018 14:40:46	July 3, 2018 14:40:46

Rule (s) defined with selected policy (LoanOfficer Policy): — 24 rows out of 24 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
2	LoanOfficer = officer_1	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
3	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
4	LoanOfficer = officer_1	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
5	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
6	LoanOfficer = officer_1	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
7	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
8	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
9	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
10	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
11	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
12	LoanOfficer = officer_2	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
13	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
14	LoanOfficer = officer_2	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
15	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
16	LoanOfficer = officer_2	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
17	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
18	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
19	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
20	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
21	LoanOfficer = officer_1	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
22	LoanOfficer = officer_1	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
23	LoanOfficer = officer_2	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated
24	LoanOfficer = officer_2	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Deny	Originated

Fig. 35. LoanOfficer Policy



FManager Policy Policy(s) Summary — 1 rows out of 1 — Search

Model	Policy Name	Rule Combination Algorithm	Policy Enforcement Algorithm	No. of Rule(s)	Time Created	Last Modified
ABAC	FManager Policy	Permit-overrides	Permit Biased	12	July 3, 2018 15:03:03	July 3, 2018 15:03:03

Rule (s) defined with selected policy (FManager Policy): — 12 rows out of 12 — Search

Sequence No	Subject	Resource	Action	Environment	Condition	Decision	Inheritance Relation
1	Financial Manager = manager	PersonalInfo = CustomerOne_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
2	Financial Manager = manager	PersonalInfo = CustomerOne_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
3	Financial Manager = manager	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
4	Financial Manager = manager	PersonalInfo = CustomerTwo_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
5	Financial Manager = manager	PersonalInfo = CustomerThree_PersonalInfo	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
6	Financial Manager = manager	PersonalInfo = CustomerThree_PersonalInfo	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
7	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
8	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
9	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = Create	Environment = Any Value	Condition = Any Value	Permit	Originated
10	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = View	Environment = Any Value	Condition = Any Value	Permit	Originated
11	Financial Manager = manager	LoanInfo = CustomerOne_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated
12	Financial Manager = manager	LoanInfo = CustomerTwo_Loan	BankActions = Approve	Environment = Any Value	Condition = Any Value	Permit	Originated

Fig. 36. FManager Policy

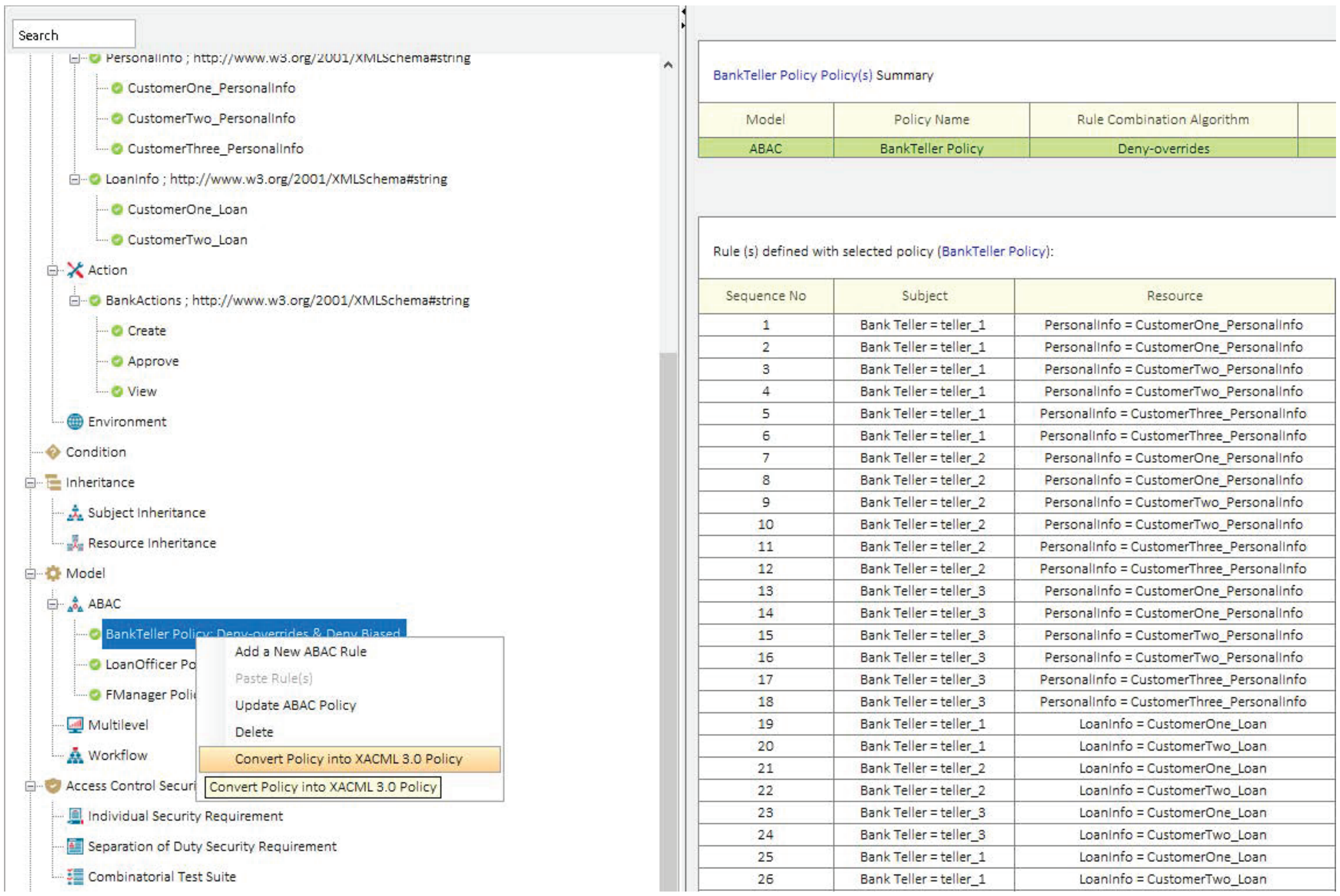
22 Understanding This Error - Test Case 5 (Inconsistent Assignment)

Now that we have our policies set up let's discuss the error we will be looking at in this fifth example. When XACML policies are being constructed there is potential for "Inconsistent Assignment" errors being created. This error occurs when a policy author has unintentionally assigned attributes, conditions, rule or other policy variables/values incorrectly in different policies. For example, Attribute LoanOfficer could be incorrectly termed as LoanOfficr or LoanOficer in different policy documents.

Doing so, could result in a significant security vulnerability due to the system providing unintended access decisions as consequence of the incorrectly defined policy value(s). Organizations with very large and complex policies are especially at risk for this type of error as small inconsistencies could very easily go unnoticed if they do not have a rigorous method for modeling and testing their policies.

23 Resolving This Error - Test Case 5 (Inconsistent Assignment)

Security Policy Tool by default prevents this error from occurring. After modeling your policies you can automatically convert your policies into XACML 3.0. The XACML Editor included with Security Policy Tool contains intuitive features to help you create secure and accurate XACML documents. Several robust features that prevent inconsistent assignment errors include Integrity Verification, Syntax Error Detection, Assistive XACML Code Completion, and others.


Search
PersonalInfo ; http://www.w3.org/2001/XMLSchema#string
CustomerOne_PersonalInfo
CustomerTwo_PersonalInfo
CustomerThree_PersonalInfo
LoanInfo ; http://www.w3.org/2001/XMLSchema#string
CustomerOne_Loan
CustomerTwo_Loan
Action
BankActions ; http://www.w3.org/2001/XMLSchema#string
Create
Approve
View
Environment
Condition
Inheritance
Subject Inheritance
Resource Inheritance
Model
ABAC
BankTeller Policy: Deny-overrides & Deny Biased
LoanOfficer Po
FManager Poli
Multilevel
Workflow
Access Control Securi
Individual Security Requirement
Separation of Duty Security Requirement
Combinatorial Test Suite
Add a New ABAC Rule
Paste Rule(s)
Update ABAC Policy
Delete
Convert Policy into XACML 3.0 Policy
Convert Policy into XACML 3.0 Policy

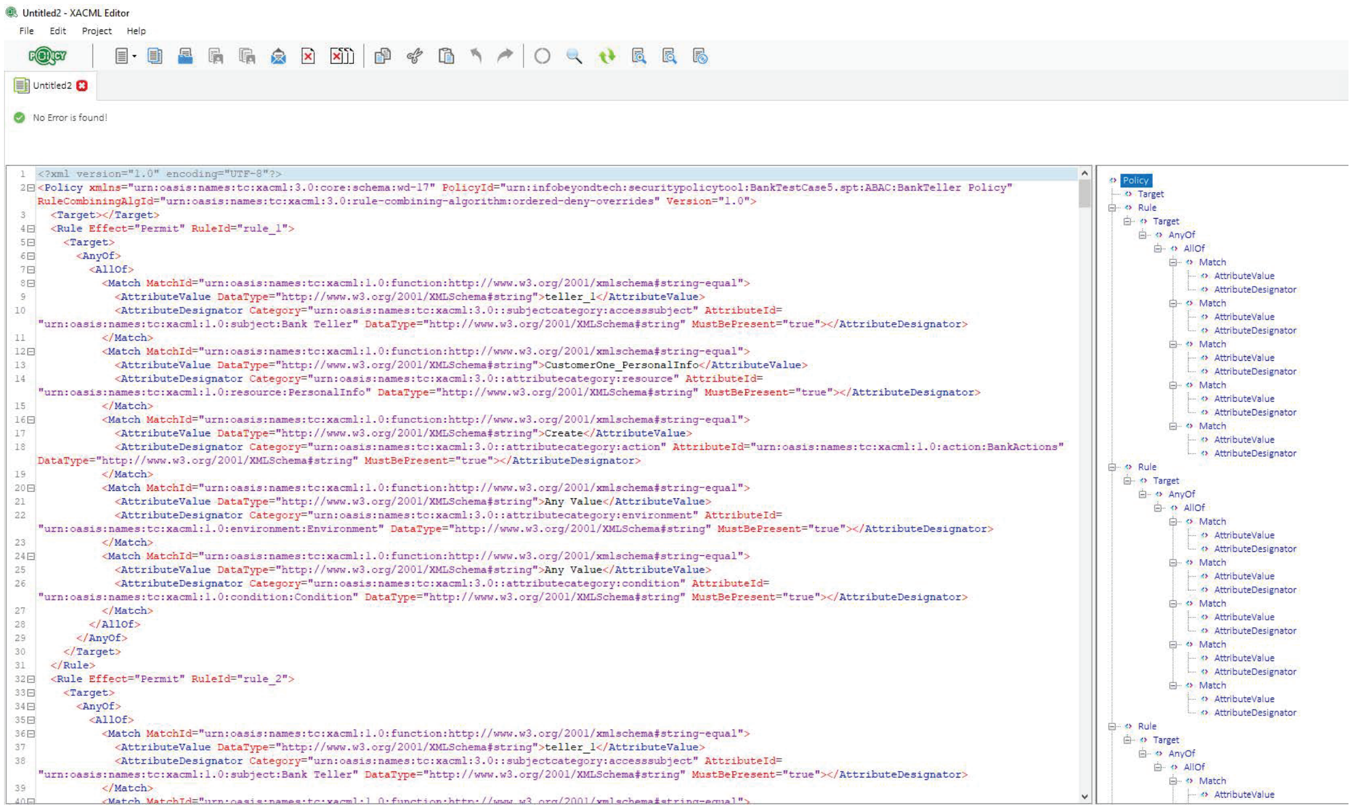
BankTeller Policy Policy(s) Summary

Model	Policy Name	Rule Combination Algorithm
ABAC	BankTeller Policy	Deny-overrides

Rule (s) defined with selected policy (BankTeller Policy):

Sequence No	Subject	Resource
1	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo
2	Bank Teller = teller_1	PersonalInfo = CustomerOne_PersonalInfo
3	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo
4	Bank Teller = teller_1	PersonalInfo = CustomerTwo_PersonalInfo
5	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo
6	Bank Teller = teller_1	PersonalInfo = CustomerThree_PersonalInfo
7	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo
8	Bank Teller = teller_2	PersonalInfo = CustomerOne_PersonalInfo
9	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo
10	Bank Teller = teller_2	PersonalInfo = CustomerTwo_PersonalInfo
11	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo
12	Bank Teller = teller_2	PersonalInfo = CustomerThree_PersonalInfo
13	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo
14	Bank Teller = teller_3	PersonalInfo = CustomerOne_PersonalInfo
15	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo
16	Bank Teller = teller_3	PersonalInfo = CustomerTwo_PersonalInfo
17	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo
18	Bank Teller = teller_3	PersonalInfo = CustomerThree_PersonalInfo
19	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan
20	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan
21	Bank Teller = teller_2	LoanInfo = CustomerOne_Loan
22	Bank Teller = teller_2	LoanInfo = CustomerTwo_Loan
23	Bank Teller = teller_3	LoanInfo = CustomerOne_Loan
24	Bank Teller = teller_3	LoanInfo = CustomerTwo_Loan
25	Bank Teller = teller_1	LoanInfo = CustomerOne_Loan
26	Bank Teller = teller_1	LoanInfo = CustomerTwo_Loan

Fig. 37. Convert Modeled Policies into XACML



No Error is found!

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="urn:infobeyondtech:securitypolicytool:BankTestCase5.spt:ABAC:BankTeller Policy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides" Version="1.0">
  <Target></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">teller_1</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::subjectcategory:accesssubject" AttributeId=
"urn:oasis:names:tc:xacml:1.0:subject:Bank Teller" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"></AttributeDesignator>
          </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">CustomerOne_PersonalInfo</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::attributecategory:resource" AttributeId=
"urn:oasis:names:tc:xacml:1.0:resource:PersonalInfo" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"></AttributeDesignator>
          </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="urn:oasis:names:tc:xacml:1.0:action:BankActions"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"></AttributeDesignator>
          </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">Any Value</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::attributecategory:environment" AttributeId=
"urn:oasis:names:tc:xacml:1.0:environment:Environment" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"></AttributeDesignator>
          </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">Any Value</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::attributecategory:condition" AttributeId=
"urn:oasis:names:tc:xacml:1.0:condition:Condition" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"></AttributeDesignator>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  </Rule>
  <Rule Effect="Permit" 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">teller_1</AttributeValue>
            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0::subjectcategory:accesssubject" AttributeId=
"urn:oasis:names:tc:xacml:1.0:subject:Bank Teller" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"></AttributeDesignator>
          </Match>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:http://www.w3.org/2001/xmlschema#string-equal">
```

Fig. 38. Converted BankTeller Policy: No Errors

24 Conclusion

Now you should have a better understanding of what to look for as you go onto verify your access control policies with Security Policy Tool. In addition to this document there are other resources located in the Learning Center in your My account page that will help you start leveraging Security Policy Tool to prevent access control leaks, today!

If you have not yet, download Security Policy Tool – Lite Version for FREE now! Close the door the Access Control Leaks and save time and cost creating, modeling, testing, and verifying your access control policies, today.

Click here to begin securing your policies now → Lite Version.

InfoBeyond Technology, LLC is an innovative company specializing in Network, Machine Learning and Data Security within the Information Technology industry. The mission of InfoBeyond is to research, develop, and deliver viable software products for network communication and security. Some of our research is sponsored by Department of Defense, Department of Energy, Missile Defense Agency, Department of Transportation, NIST (National Institute of Standards and Technology), etc. Security Policy Tool is Awarded the 2017 Innovative Security Solution Award at the 2017 Big Data and SDN/NFV Summit. NXdrive is a fragment-based cybersecurity storage system and more information can be found at www.NXdrive.com. The company is featured as one of 50 fast growth IT small businesses in 2017 by The Silicon Review.