How To Interpret Counterexamples

In [1], we defined three different properties for access control policies namely policy-completeness, policy-consistency, and obligation-safety.

1. Interpreting counterexamples violating the Policy-Completeness property

The Policy-Completeness property is defined as follows:

Policy-Completeness: *A set of policies is complete if it covers all the access requests. More formally, for every Request action, the policy set will inevitably provide a Response action.*

If there is a request that cannot be covered by the policies, it would be considered a violation of the Policy-Completeness property. As a result, the mCRL2 IDE generates a counterexample, as shown in Figure 1. The counterexample violates the Policy-Completeness property by demonstrating the existence of a ***Request*** action for a set of attributes (in this case, subjectid=Doctor, resourceid=MedicalRecords, and actionid=Read), for which there exists no corresponding ***Response*** action. In other words, Figure 1 shows that the examined policies do not cover a scenario where a *Doctor* wants to *Read MedicalRecords*.

To address this issue, the policy authors can define a new rule or modify an existing one to cover such requests. It is worth noting that, according to our approach, every ***Request*** action may contain up to four sets of attributes: Subject Attributes, Object Attributes, Action Attributes, and Environment Attributes. Each attribute is represented as attribute(name, value), and a ***Request*** action is represented as follows:

Request({Subject Attribute 1, …, Subject Attribute N}, {Object Attribute 1, …, Object Attribute N}, {Action Attribute 1, …, Action Attribute N}, {Environment Attribute 1, …, Environment Attribute N}) or simply Request({Attributes}).

Or simply:

Request({*Attributes*})

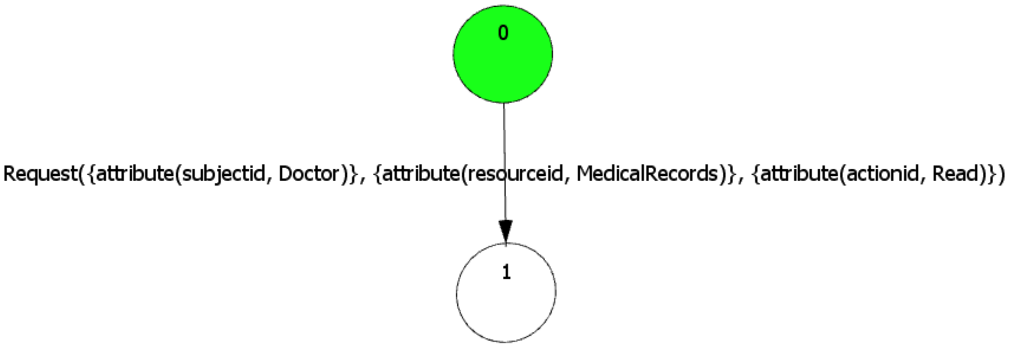
0
Request({attribute(subjectid, Doctor)}, {attribute(resourceid, MedicalRecords)}, {attribute(actionid, Read)})
1

Figure 1: A counterexample violating the Policy-Completeness property.

2. Interpreting counterexamples violating the Policy-Consistency property

The Policy-Consistency property is defined as follows:

Policy-Consistency: *A set of policies is conflict-free if there is no inconsistency between policies*.

This property requires that executing a Request action (with specific attributes) cannot result in both a Deny and a Permit decision.

Hence, if there are two different decisions for a single request, it would be deemed as a violation of the Policy-Consistency property. As a consequence, the mCRL2 IDE generates a counterexample, which is shown in Figure 2.

Based on our approach, every ***Response*** action carries a **Decision**, which can either be *Permit* or *Deny*, in addition to the sets of attributes for subject, object, action, and environment attributes. A ***Response*** action is represented as follows:

Response({*Subject Attribute 1, …, Subject Attribute N*}, {*Object Attribute 1, …, Object Attribute N*}, {*Action Attribute 1, …, Action Attribute N*}, {*Environment Attribute 1, …, Environment Attribute N*}, **Decision**)

Or simply:

Response({*Attributes*}, **Decision**)

Figure 2 indicates that there are two different ***Response*** actions after the same ***Request*** action. In other words, there is an inconsistency among examined policies since two different rules with different rule effects (decisions) cover the same request where *CareGiverA* wants to *Read HealthData* (*subjectid* is *CareGiverA*, *resourceid* is *HealthData*, and *actionid* is *Read*).

The request is:

Request({*attribute*(subjectid, CareGiverA)}, {attribute(resourceid, HealthData)},
{attribute(actionid, Read)})

And responses are:

Response({attribute(subjectid, CareGiverA)}, {attribute(resourceid, HealthData)},
{attribute(actionid, Read)}, **Permit**)

Response({attribute(subjectid, CareGiverA)}, {attribute(resourceid, HealthData)},
{attribute(actionid, Read)}, **Deny**)

The policy authors can remove the inconsistency by modifying the existing rules that cover such a request.

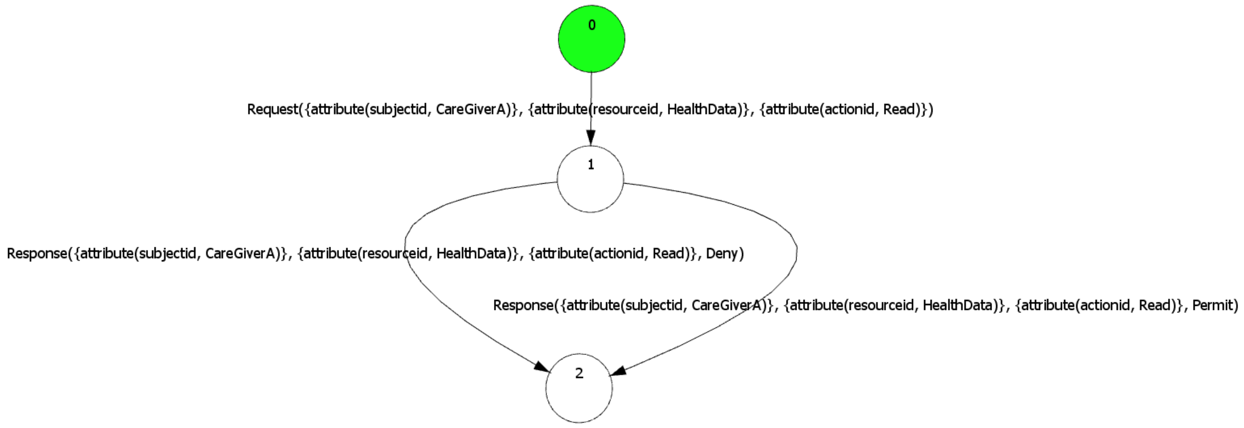
0
Request({attribute(subjectid, CareGiverA)}, {attribute(resourceid, HealthData)}, {attribute(actionid, Read)})
1
Response({attribute(subjectid, CareGiverA)}, {attribute(resourceid, HealthData)}, {attribute(actionid, Read)}, Deny)
Response({attribute(subjectid, CareGiverA)}, {attribute(resourceid, HealthData)}, {attribute(actionid, Read)}, Permit)
2

Figure 2: A counterexample violating the Policy-Consistency property.

3. Interpreting counterexamples violating the Obligation-Safety property

The Obligation-Safety property is defined as follows:

Obligation-Safety: *A concrete Request either will always yield an Obligation, or it will never yield an Obligation*.

In other words, for every Request action, if there is an Obligation action after the Request action, then there should not exist a Response action without a preceding Obligation action for the same request. If this condition is not met, then the mCRL2 IDE generates a counterexample.

For instance, Figure 3 illustrates a counterexample violating the Obligation-Safety property for a request where a *Doctor* wants to *Read HealthData*. As shown in Figure 3, there exists an ***Obligation*** action after the ***Request*** action. However, there also exists a ***Response*** action that is not preceded by an ***Obligation*** action. In other words, a *Doctor* would be able to *Read HealthData* without the enforcement of the obligation.

Based on our approach, every ***Obligation*** action carries an **Obligation ID** in addition to the sets of attributes for subject, object, action, and environment attributes. An ***Obligation*** action is represented as follows:

Obligation({*Subject Attribute 1, …, Subject Attribute N*}, {*Object Attribute 1, …, Object Attribute N*}, {*Action Attribute 1, …, Action Attribute N*}, {*Environment Attribute 1, …, Environment Attribute N*}, **ObligationID**)

Or simply:

Obligation ({*Attributes*}, **ObligationID**)

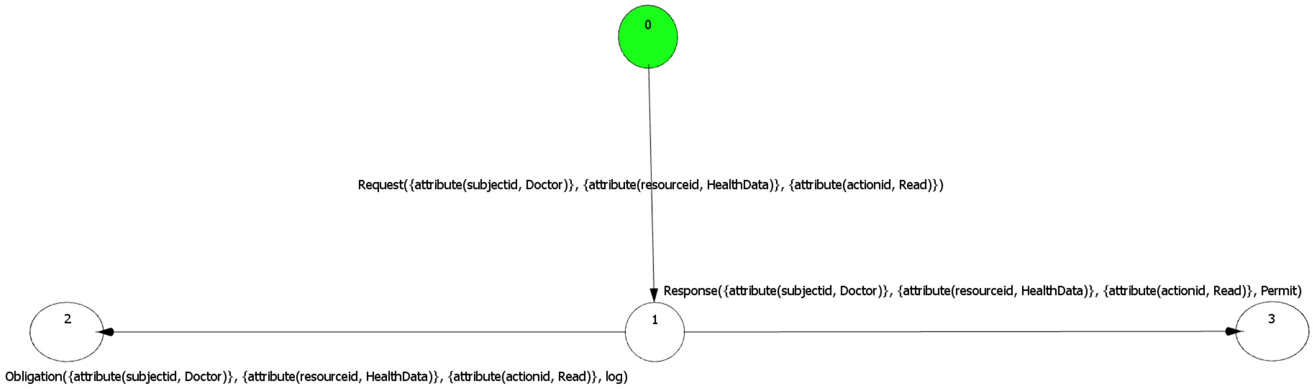
0
Request({attribute(subjectid, Doctor)}, {attribute(resourceid, HealthData)}, {attribute(actionid, Read)})
Response({attribute(subjectid, Doctor)}, {attribute(resourceid, HealthData)}, {attribute(actionid, Read)}, Permit)
2
1
3
Obligation({attribute(subjectid, Doctor)}, {attribute(resourceid, HealthData)}, {attribute(actionid, Read)}, log)

Figure 3: A counterexample violating the Obligation-Safety property.

References:

1. Arshad, H., Horne, R., Johansen, C., Owe, O., Willemse, T.A.C. (2022). Process Algebra Can Save Lives: Static Analysis of XACML Access Control Policies Using mCRL2. In: Mousavi, M.R., Philippou, A. (eds) Formal Techniques for Distributed Objects, Components, and Systems. FORTE 2022. Lecture Notes in Computer Science, vol 13273. Springer, Cham. https://doi.org/10.1007/978-3-031-08679-3_2