A formal Equivalence Classes based Method for Security Policy Conformance Checking

Eckehard Hermann, Udo Litschauer and Jürgen Fuß
eckehard.hermann@fh-hagenberg.at
udo.litschauer@fh-hagenberg.at
juergen.fuss@fh-hagenberg.at

Department of Secure Information Systems
University of Applied Sciences Upper Austria, Austria

Abstract. Different security policy models have been developed and published in the past. Proven security policy models, if correctly implemented, guarantee the protection of data objects from unauthorized access or usage or prevent an illegal information flow. To verify that a security policy model has been correctly implemented, it is important to define and execute an exhaustive list of test cases, which verify that the formal security policy neither has been over-constrained nor under-constrained. In this paper we present a method for defining an exhaustive list of test cases, based on formally described equivalence classes that are derived from the formal security policy description.

Keywords: security models, test generation, access control, conformance testing

1 Introduction

Different security policy models have been developed and published in the past. In 1989 Brewer and Nash presented their Chinese Wall Security Policy model that is based on conflict of interest classes [2]. Other security policy models are a formalization of a military security model like the one from Bell and LaPadula [4], they address the integrity of data objects in commercial transactions, as stated by Clark and Wilson [5], control the information flow like the Limes Security Model from Hermann [6] or they are a model of access control like the access matrix defined by Lampson [1]. Each of these models defines the security requirements that have to be correctly implemented by a system for achieving a given security objective. If the security model has not been correctly implemented, the resulting system will be over-constrained or under-constrained. After the correctness of the formal specification of the security model has been verified, the system implementation has to be validated against the security model. As discussed by Hu and Ahn in [7] a system is under-constrained if, based on the security model, undesired system states are granted and over-constrained if desired system states are denied, which probably causes availability problems. Murnane and Reed argument in [8]

testing software after it is completed remains an important aspect of software quality assurance despite the recent emphasis on the use of formal methods and 'defect-free' software development processes.

Our approach, presented in this paper, is a method for defining an exhaustive list of test cases based on formally described equivalence classes that are derived from the formal security policy description. The paper is organized as follows. Section 2 introduces some background on security models and testing, in particular equivalence class testing and section 3 gives an overview of related work. Our approach is presented in section 4. We start with a formal definition of equivalence classes. For explaining the application of our approch, we define the equivalence classes of the Bell and LaPadula model. Section 5 outlines opportunities for future work and draws conclusions.

2 State of the Art

2.1 Security Models

A security model defines rules and demonstrates, that if security requirements are correctly and completely educed from the rules and these requirements are correctly and completely implemented by a system, the system achieves a given security objective. Different security policy models like the one from Bell and LaPadula, are a formalization of a military security model in [4] or they address the integrity of data objects in commercial transactions, as stated by Clark and Wilson [5]. Grimm defined in [9] that all security models contain five elements of description:

1. the definition of a superior security objective (informal),
2. the specification of secure system states (formal),
3. rules for allowed state transitions (formal),
4. a security theorem that proves that an allowed state transition will transfer a secure state always into a secure state (formal),
5. a trust model that describes requirements for the system implementation and for the application environment in order to enable secure system states to achieve the superior security objective (semi-formal or informal).

The Bell and LaPadula Model In 1973 the first complete formal defined security model has been described by David Elliott Bell and Leonard J. LaPadula in [4]. Bell and LaPadula define a partial ordered set of security levels. The security levels are assigned to subjects - the active entities - and to objects - the passive and data containing entities - in the model. The security level of a subject is called clearance level and the security level of an object is called classification. The superior security objective of the model is the prevention of a vertical information flow from top to bottom according to the partially ordered clearance and classification levels. The model defines two properties.

The ss-property defines that in order for a subject to read an object's data, the subject's clearance level must dominate the object's classification. The *-property authorizes a subject, only if the objects classification is more recent than the most sensitive object that it is currently allowed to read. Additionally, Bell and LaPadula define the usage of a Lampson-matrix. In preparation of later usage Table 1 introduces the elements of the Bell and LaPadula model.

Set	**Element**	**Semantic**
$\mathcal{S}$	$\{S_1, S_2, \ldots, S_n\}$	*subjects*; processes, programs in execution, ...
$\mathcal{O}$	$\{O_1, O_2, \ldots, O_m\}$	*objects*; data, files, programs, subjects, ...
$\mathcal{C}$	$\{C_1, C_2, \ldots, C_q\}$, where $C_1 > C_2 > \cdots > C_q$	*classifications*; clearance level of a subject, classification of an object
K	$\{K_1, K_2, \ldots, K_r\}$	*needs-to-know categories*; project numbers, access privileges, ...
A	$\{A_1, A_2, \ldots, A_p\}$	*access attributes*; read, write, copy, append, owner, control, ...
R	$\{R_1, R_2, \ldots, R_u\}$	*requests*; inputs, commands, requests for access to objects by subjects, ...
D	$\{D_1, D_2, \ldots, D_v\}$	*decisions*; outputs, answers, "yes", "no", "error"
T	$\{1, 2, \ldots, t, \ldots\}$	*indices*; elements of the time set; identification of discrete moments; an element t is an index to request and decision sequences
$\mathcal{P}(\alpha)$	all subsets of α	power set of α
α^β	all functions from the set β to the set α	–
$\alpha \times \beta$	$\{(a, b) : a \in \alpha, b \in \beta\}$	cartesian product of the sets α and β
F	$C^{\mathcal{S}} \times C^{\mathcal{O}} \times K^{\mathcal{S}} \times K^{\mathcal{O}}$ an arbitrary element of F is written $f = (f_1, f_2, f_3, f_4)$	*classification/need-to-know vectors*; f_1: subject *classification* fct. f_2: object *classification* fct. f_3: subject *need-to-know* fct. f_4: object *need-to-know* fct.
X	R^T; an arbitrary element of X is written x	*request sequences*
Y	D^T; an arbitrary element of Y is written y	*decision sequences*

$\mathcal{M}$	$\{M_1, M_2, \ldots, M_c\}$, $c = nm2^p$; an element M_K of $\mathcal{M}$ is an $n \times m$-Lampson-matrix with entries from $\mathcal{P}(A)$; the (i,j)-entry of M_k shows S_i access attributes relative to O_j	*access matrices*
V	$P(S \times O) \times M \times F$	*states*
Z	V^T; an arbitrary element of Z is written z; $z_t = z(t)$ is the t-th state in the state sequence z	*state sequences*

Table 1: Elements of the Bell-LaPadula-Model [4]

Additionally the following definitions of system states and state-transition relations are done in [4]: A state $v \in V$ is a triple (b, M, f) where

$b \in \mathcal{P}(\mathcal{S} \times \mathcal{O})$,	indicating which subjects have access to which objects in the state v,
$M \in \mathcal{M}$,	indicating the entries of the Lampson-matrix in the state v and
$f \in F$,	indicating the clearance level of all subjects, the classification level of all objects, and the needs-to-know associated with all subjects and objects in the state v.[4]

Definition 1 (Access-Matrix-Function).
Let $S_i \in \mathcal{S}$, $O_j \in \mathcal{O}$ and $M \in \mathcal{M}$. The function

$$m : \mathcal{S} \times \mathcal{O} \to \mathcal{P}(A)$$

is called Access-Matrix-Function in relation to M if $m(S_i, O_j)$ returns the (i,j)-entry of M.

Let $W \subseteq R \times D \times V \times V$. The system $\Sigma(R, D, W, z_0) \subseteq X \times Y \times Z$ is defined by $(x, y, z) \in \Sigma(R, D, W, z_0)$, if and only if $(x_t, y_t, z_t, z_{t-1}) \in W$ for each t $\in$ T where z_0 is a specified initial state usually of the form $(\emptyset, M, f)$ where $\emptyset$ denotes the empty set [4].

The state $v = (b, M, f) \in V$ is a compromise state (compromise) if there is an ordered pair $(S, O) \in b$, such that

1. $f_1(S) < f_2(O)$ or
2. $f_3(S) \not\supseteq f_4(O)$.

The pair$(S, O) \in \mathcal{S} \times \mathcal{O}$ satisfies the security condition relative to *f (SC rel f)* if

3. $f_1(S) \geq f_2(O)$ and
4. $f_3(S) \supseteq f_4(O)$.

A state $v = (b, M, f) \in V$ is a secure state if each $(S, O) \in b$ satisfies SC rel f.[4].

2.2 Testing

Testing a system is an important aspect of quality assurance. But it does not prove the absence of errors; it can only prove the presence of features. Testing can be divided into functional and non-functional testing. By performing functional tests it is verified whether a system fulfils its functional requirements or not. When non-functional testing aspects are tested, they may not be related to a specific functional requirement, such as performance testing. In the past different test techniques have been developed, which can be split into black box and white box techniques. Black box techniques are testing techniques, where the test cases are primarily derived from the system specifications and without knowledge of the inspected system implementations. These kind of testing techniques are testing the systems input and output behavior. Different types of black box testing techniques are equivalence class testing, boundary testing or fuzz testing [10]. White box techniques are testing techniques, which use knowledge about the internal composition of a system for the test case definition.[8]

Equivalence Class Testing The equivalence class testing technique implies, that the input domain of a system is partitioned into a finte number of sets, called equivalence classes, such that the systems behavior to a test of a representative value, called test case, of one equivalence class is equal to the systems behavior to a test of any other value of the same equivalence class. If one test case of an equivalence class detects an error, any other test case of the same equivalence class will be expected to detect the same error. If one test case of an equivalence class does not detect an error, it is expected that not any of the test case of the same equivalence class will detect an error.[11]

3 Related Work

In [12] Hu et al. propose an approach for conducting conformance checking of access control policies, specified in XACML and they also propose an implementation of conformance checking based on previous XACML policy verification and testing tools. The work is based on a fault model [13], a structural coverage measurement tool for defining policy coverage metrics [15] and a test generator [14], developed by two of the authors in their former work. In [16] De Angelis et al. discuss access policy testing as a vital function of the trust network, in which users and service providers interact. User-centric security management is enabled by using automated compliance testing using an audit bus, sharing audit trails and governance information, to monitor service state and provide users with privacy protection in networks of services and a conceptual framework supporting on-line testing of deployed systems. The authors explain that for each service under test the component continuously verifies that it does not violate any of the declared access control policies running for a set of test cases. Hu and Ahn developed in [7] a methodological attempt to verify formal specifications of a role-based access control model and corresponding policies. They

also derived test cases from formal specifications and validate conformance to the system design and implementation using those test cases. In [17] Mouelhi et al. propose a test selection criteria to produce tests from a security policy. They propose several test criteria to select test cases from a security policy specified in OrBaC (a specification language to define the access control rules). Criterion 1 is satisfied if and only if a test case is generated for each primary access control rule of the security model and criterion 2 is satisfied if and only if a test case is generated for each primary and secondary rule of the security model, except the generic rules. Security test cases obtained with one of the two criteria should test aspects, which are not the explicit objective of functional tests. Mutation analysis, which is used by Mouelhi et al. during security test case generation, is a technique for evaluating the quality of test cases at which a mutant is a copy of the original policy that contains one simple flaw.

4 An equivalence classes based approach for security test case definition

To verify if a security policy model has been correctly implemented, we propose the definition of an equivalence class for valid as well as for invalid system input values for each rule of a security policy. We distinguish between two principal categories of equivalence class: equivalence class for valid and for invalid system input values. We present our approach by defining the equivalence classes of the Bell and LaPadula model. In relation to the Bell and LaPadula model an equivalence class is defined as:

Definition 2 (equivalence class). *Let $a \in A$ and $b \in \mathcal{P}(\mathcal{S} \times \mathcal{O})$. The equivalence class of (a, b) is the set $\mathrm{ec}_{ab} \subseteq A \times \mathcal{P}(\mathcal{S} \times \mathcal{O})$ with the property:*

$$\mathrm{ec}_{ab} := \{(x, y) \mid \ (x, y) \text{ is equivalent to } (a, b)\}$$

Let $EC := \{\mathrm{ec}_1, \mathrm{ec}_2, \ldots, \mathrm{ec}_n\}$ be the set of all equivalence classes of a system.

In principal we distinguish between equivalence classes that satisfy the above defined security conditions and equivalence classes that violate these security conditions.

A proof concerning completeness is trivial and can easy be done by performing a logical AND-conjunction of the satisfying equivalence class, that has to be equal to the policy definition and a logical OR-conjunction of the violating equivalence classes, that has to be equal to the negation of the security policy definition.

In a first preparative step the following security conditions SC are defined: The elements of an equivalence class containing $Subject \times Object$ tuples where the Subject is allowed to read access the Object, have to satisfy the f_{read} security condition.

Definition 3 (security condition f_{read}).
$(S, O) \in \mathcal{S} \times \mathcal{O}$ *satisfies the security condition relative* f_{read}*, if*

1. $f_1(S) \geq f_2(O)$ *and*
2. $f_4(O) \subseteq f_3(S)$.

The elements of an equivalence class containing $Subject \times Object$ tuples where the Subject is allowed to write access the Object, have to satisfy the f_{write} security condition.

Definition 4 (security condition f_{write}).
$(S, O) \in \mathcal{S} \times \mathcal{O}$ *satisfies the security condition relative* f_{write}*, if*

1. $f_1(S) \leq f_2(O)$ *and*
2. $f_4(O) \subseteq f_3(S)$.

The elements of an equivalence class containing $Subject \times Object$ tuples, where the Subject is allowed to read-write access the Object have to satisfy the $f_{\text{read-write}}$ security condition.

Definition 5 (security condition $f_{\text{read-write}}$).
$(S, O) \in \mathcal{S} \times \mathcal{O}$ *satisfies the security condition relative* $f_{\text{read-write}}$*, if*

1. $f_1(S) = f_2(O)$ *and*
2. $f_4(O) \subseteq f_3(S)$.

The elements of an equivalence class containing $Subject \times Object$ tuples, where the Subject is allowed to append access the Object have to satisfy the f_{append} security condition.

Definition 6 (security condition f_{append}).
$(S, O) \in \mathcal{S} \times \mathcal{O}$ *satisfies the security condition relative* f_{append}*, if*

1. $f_1(S) \leq f_2(O)$ *and*
2. $f_4(O) \subseteq f_3(S)$.

The elements of an equivalence class containing $Subject \times Object$ tuples, where the Subject is allowed to execute the Object, have to satisfy the f_{execute} security condition.

Definition 7 (security condition f_{execute}).
$(S, O) \in \mathcal{S} \times \mathcal{O}$ *satisfies the security condition relative* f_{execute}*, if*

1. $f_1(S) \geq f_2(O)$ *and*
2. $f_4(O) \subseteq f_3(S)$.

4.1 Equivalence classes definition

Based on the security conditions defined above, equivalence classes of the Bell and LaPadula model are defined now. As discussed above we distinguish between equivalence classes that satisfy the defined security conditions and equivalence classes that violate these security conditions.

Satisfying equivalence classes The test cases contained by the satisfying equivalence classes can be used to verify if the system is over-constrained.

Definition 8 (equivalence classes $\mathrm{ec}_1, \ldots, \mathrm{ec}_5$).
Let $b \in \mathcal{P}(\mathcal{S} \times \mathcal{O})$, be $a \in A$ an access attribute, $M \in \mathcal{M}$ the current Lampson-matrix and m the Access-Matrix-Function in relation to M. Then

$$\begin{aligned}
\mathrm{ec}_1 := \{(b,a) \mid\ & a = \text{read} \wedge a \in m(b) \wedge \\
& b \textit{ satisfies the security condition relative } f_{\text{read}}\} \\
\mathrm{ec}_2 := \{(b,a) \mid\ & a = \text{write} \wedge a \in m(b) \wedge \\
& b \textit{ satisfies the security condition relative } f_{\text{write}}\} \\
\mathrm{ec}_3 := \{(b,a) \mid\ & a = \text{read-write} \wedge a \in m(b) \wedge \\
& b \textit{ satisfies the security condition relative } f_{\text{read}-\text{write}}\} \\
\mathrm{ec}_4 := \{(b,a) \mid\ & a = \text{execute} \wedge a \in m(b) \wedge \\
& b \textit{ satisfies the security condition relative } f_{\text{execute}}\} \\
\mathrm{ec}_5 := \{(b,a) \mid\ & a = \text{append} \wedge a \in m(b) \wedge \\
& b \textit{ satisfies the security condition relative } f_{\text{append}}\}
\end{aligned}$$

Violating equivalence classes The test cases contained by the violating equivalence classes show if the system is under-constrained.

Definition 9 (equivalence classes $\mathrm{ec}_6, \ldots, \mathrm{ec}_{10}$).
Let $b \in \mathcal{P}(\mathcal{S} \times \mathcal{O})$, be $a \in A$ an access attribute, $M \in \mathcal{M}$ the current Lampson-matrix and m the Access-Matrix-Function in relation to M. Then

$$\begin{aligned}
\mathrm{ec}_6 := \{(b,a) \mid\ & a = \text{read} \wedge (a \notin m(b) \vee \\
& b \textit{ violates the security condition relative } f_{read})\} \\
\mathrm{ec}_7 := \{(b,a) \mid\ & a = \text{write} \wedge (a \notin m(b) \vee \\
& b \textit{ violates the security condition relative } f_{\text{write}})\} \\
\mathrm{ec}_8 := \{(b,a) \mid\ & a = \text{read-write} \wedge (a \notin m(b) \vee \\
& b \textit{ violates the security condition relative } f_{\text{read}-\text{write}})\} \\
\mathrm{ec}_9 := \{(b,a) \mid\ & a = \text{execute} \wedge (a \notin m(b) \vee \\
& b \textit{ violates the security condition relative } f_{\text{execute}})\} \\
\mathrm{ec}_{10} := \{(b,a) \mid\ & a = \text{append} \wedge (a \notin m(b) \vee \\
& b \textit{ violates the security condition relative } f_{\text{append}})\}
\end{aligned}$$

4.2 Test cases definition

After the equivalence classes have been defined, a functional test can be performed. In the following section we define a sample system. We start by defining the subjects, objects, classification, access attributes and Need-to-Know-categories as well as the Lampson-matrix of the sample system.

Definition of the sample system Let $\mathcal{S}$ be the set of all subject in the sample system

$$\mathcal{S} = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8\},$$

Let $\mathcal{O}$ be the set of all objects in the sample system

$$\mathcal{O} = \{O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8\},$$

Let $\mathcal{C}$ be a partial ordered set of classification

$$\mathcal{C} = \{\text{top secret}, \text{secret}, \text{classified}, \text{unclassified}\}, \text{ where}$$
$$\text{top secret} > \text{secret} > \text{classified} > \text{unclassified}.$$

Let A be the set of allowed access attributes in the sample system

$$A = \{\text{read}, \text{write}, \text{read-write}, \text{execute}, \text{append}\}$$

Let K be the *Need-to-Know*-categories in the sample system

$$K = \{\text{Cat}_1, \text{Cat}_2, \text{Cat}_3, \text{Cat}_4, \text{Cat}_5, \text{Cat}_6, \text{Cat}_7, \text{Cat}_8\}$$

$$\begin{array}{ll}
C_{S_1} = \text{top secret} & C_{O_1} = \text{top secret} \\
C_{S_2} = \text{secret} & C_{O_2} = \text{secret} \\
C_{S_3} = \text{classified} & C_{O_3} = \text{classified} \\
C_{S_4} = \text{unclassified} & C_{O_4} = \text{unclassified} \\
C_{S_5} = \text{top secret} & C_{O_5} = \text{top secret} \\
C_{S_6} = \text{secret} & C_{O_6} = \text{secret} \\
C_{S_7} = \text{classified} & C_{O_7} = \text{classified} \\
C_{S_8} = \text{unclassified} & C_{O_8} = \text{unclassified}
\end{array}$$

Figure 1 shows all the subjects and objects and their classifiction level

The *Need-to-Know*-categories are as follows:

$$\begin{array}{l}
K_{S_1} = \{\text{Cat}_1, \text{Cat}_4, \text{Cat}_5, \text{Cat}_7, \text{Cat}_8\} \\
K_{S_2} = \{\text{Cat}_2, \text{Cat}_4, \text{Cat}_5, \text{Cat}_6, \text{Cat}_7, \text{Cat}_8\} \\
K_{S_3} = \{\text{Cat}_3, \text{Cat}_4, \text{Cat}_5, \text{Cat}_7, \text{Cat}_8\} \\
K_{S_4} = \{\text{Cat}_4, \text{Cat}_5, \text{Cat}_8\} \\
K_{S_5} = \{\text{Cat}_1, \text{Cat}_2, \text{Cat}_4, \text{Cat}_5, \text{Cat}_7, \text{Cat}_8\} \\
K_{S_6} = \{\text{Cat}_2, \text{Cat}_4, \text{Cat}_5, \text{Cat}_6, \text{Cat}_7, \text{Cat}_8\} \\
K_{S_7} = \{\text{Cat}_3, \text{Cat}_4, \text{Cat}_5, \text{Cat}_7, \text{Cat}_8\} \\
K_{S_8} = \{\text{Cat}_4, \text{Cat}_5, \text{Cat}_8\}
\end{array}$$

$$\begin{array}{l}
K_{O_1} = \{\text{Cat}_1\} \\
K_{O_2} = \{\text{Cat}_2\} \\
K_{O_3} = \{\text{Cat}_3\} \\
K_{O_4} = \{\text{Cat}_4\} \\
K_{O_5} = \{\text{Cat}_5\}
\end{array}$$

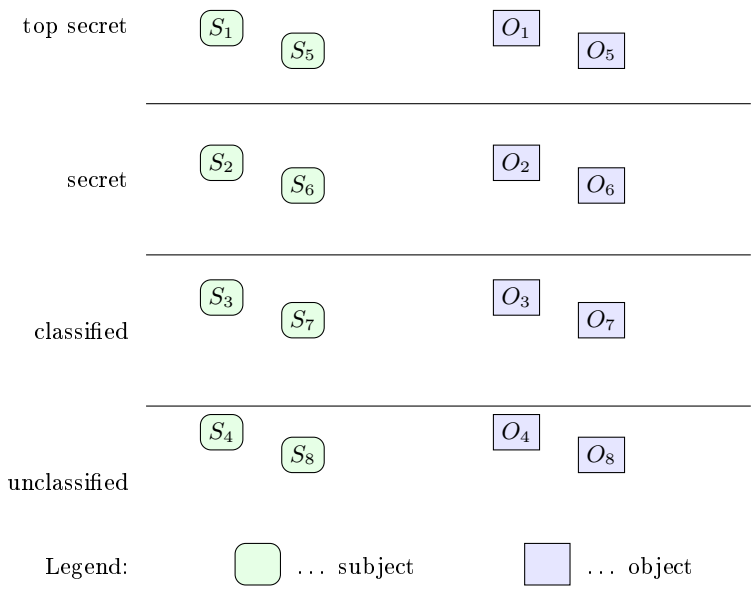

Fig. 1. Subjects, objects and classification levels of the sample system

$$K_{O_6} = \{\mathrm{Cat}_6\}$$
$$K_{O_7} = \{\mathrm{Cat}_7\}$$
$$K_{O_8} = \{\mathrm{Cat}_8\}$$

For testing of the equivalence classes, the following Lampson-matrix is defined.

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
S_1	{r}	{}	{}	{r}	{r}	{}	{r}	{r,w}
S_2	{}	{r,e}	{}	{r}	{a}	{r,w,e}	{r}	{r,w}
S_3	{}	{}	{rw,e}	{r}	{a}	{}	{r,w}	{r,w}
S_4	{}	{}	{}	{r,w,a}	{a}	{}	{}	{r,w}
S_5	{rw,a}	{r}	{}	{r}	{r}	{}	{r}	{r,w}
S_6	{}	{r,w}	{}	{r}	{a}	{r,w,e}	{r}	{r,w}
S_7	{}	{}	{rw,e}	{r}	{a}	{}	{r,w}	{r,w}
S_8	{}	{}	{}	{r,w,a}	{a}	{}	{}	{r,w}

r... read, w... write, rw... read-write, e... execute, a... append

Table 2. Sample Access Matrix

After defining the entities and Lampson-matrix, all entities are assigned to the equivalence classes:

$$
\begin{aligned}
\mathrm{ec}_1 = \{&((S_1, O_1), \text{read}), ((S_1, O_4), \text{read}), ((S_1, O_5), \text{read}), ((S_1, O_7), \text{read}),\\
&((S_1, O_8), \text{read}), ((S_2, O_2), \text{read}), ((S_2, O_4), \text{read}), ((S_2, O_6), \text{read}),\\
&((S_2, O_7), \text{read}), ((S_2, O_8), \text{read}), ((S_3, O_4), \text{read}), ((S_3, O_7), \text{read}),\\
&((S_3, O_8), \text{read}), ((S_4, O_4), \text{read}), ((S_4, O_8), \text{read}), ((S_5, O_2), \text{read}),\\
&((S_5, O_4), \text{read}), ((S_5, O_5), \text{read}), ((S_5, O_7), \text{read}), ((S_5, O_8), \text{read}),\\
&((S_6, O_2), \text{read}), ((S_6, O_4), \text{read}), ((S_6, O_6), \text{read}), ((S_6, O_7), \text{read}),\\
&((S_6, O_8), \text{read}), ((S_7, O_4), \text{read}), ((S_7, O_7), \text{read}), ((S_7, O_8), \text{read}),\\
&((S_8, O_4), \text{read}), ((S_8, O_8), \text{read})\}\\
\mathrm{ec}_2 = \{&((S_2, O_6), \text{write}), ((S_3, O_7), \text{write}), ((S_4, O_4), \text{write}),\\
&((S_4, O_8), \text{write}), ((S_6, O_2), \text{write}), ((S_6, O_6), \text{write}),\\
&((S_7, O_7), \text{write}), ((S_8, O_4), \text{write}), ((S_8, O_8), \text{write})\}\\
\mathrm{ec}_3 = \{&((S_3, O_3), \text{read-write}), ((S_5, O_1), \text{read-write}), ((S_7, O_3), \text{read-write})\}\\
\mathrm{ec}_4 = \{&((S_2, O_2), \text{execute}), ((S_2, O_7), \text{execute}), ((S_3, O_3), \text{execute}),\\
&((S_6, O_6), \text{execute}), ((S_7, O_3), \text{execute})\}\\
\mathrm{ec}_5 = \{&((S_2, O_5), \text{append}), ((S_3, O_5), \text{append}), ((S_4, O_4), \text{append}),\\
&((S_4, O_5), \text{append}), ((S_5, O_1), \text{append}), ((S_6, O_5), \text{append}),\\
&((S_7, O_5), \text{append}), ((S_8, O_4), \text{append}), ((S_8, O_5), \text{append})\}\\
\mathrm{ec}_6 = \{&((S_1, O_2), \text{read}), ((S_1, O_3), \text{read}), ((S_1, O_6), \text{read}), ((S_2, O_1), \text{read}),\\
&((S_2, O_3), \text{read}), ((S_2, O_5), \text{read}), ((S_3, O_1), \text{read}), ((S_3, O_2), \text{read}),\\
&((S_3, O_3), \text{read}), ((S_3, O_5), \text{read}), ((S_3, O_6), \text{read}), ((S_4, O_1), \text{read}),\\
&((S_4, O_2), \text{read}), ((S_4, O_3), \text{read}), ((S_4, O_5), \text{read}), ((S_4, O_6), \text{read}),\\
&((S_4, O_7), \text{read}), ((S_5, O_1), \text{read}), ((S_5, O_3), \text{read}), ((S_5, O_6), \text{read}),\\
&((S_6, O_1), \text{read}), ((S_6, O_3), \text{read}), ((S_6, O_5), \text{read}), ((S_7, O_1), \text{read}),\\
&((S_7, O_2), \text{read}), ((S_7, O_3), \text{read}), ((S_7, O_5), \text{read}), ((S_7, O_6), \text{read}),\\
&((S_8, O_1), \text{read}), ((S_8, O_2), \text{read}), ((S_8, O_3), \text{read}), ((S_8, O_5), \text{read}),\\
&((S_8, O_6), \text{read}), ((S_8, O_7), \text{read})\\
\mathrm{ec}_7 = \{&((S_1, O_1), \text{write}), ((S_1, O_2), \text{write}), ((S_1, O_3), \text{write}),\\
&((S_1, O_4), \text{write}), ((S_1, O_5), \text{write}), ((S_1, O_6), \text{write}), ((S_1, O_7), \text{write}),\\
&((S_1, O_8), \text{write}), ((S_2, O_1), \text{write}), ((S_2, O_2), \text{write}), ((S_2, O_3), \text{write}),\\
&((S_2, O_4), \text{write}), ((S_2, O_5), \text{write}), ((S_2, O_7), \text{write}), ((S_2, O_8), \text{write}),\\
&((S_3, O_1), \text{write}), ((S_3, O_2), \text{write}), ((S_3, O_3), \text{write}), ((S_3, O_4), \text{write}),\\
&((S_3, O_5), \text{write}), ((S_3, O_6), \text{write}), ((S_3, O_8), \text{write}), ((S_4, O_1), \text{write}),\\
&((S_4, O_2), \text{write}), ((S_4, O_3), \text{write}), ((S_4, O_5), \text{write}), ((S_4, O_6), \text{write}),\\
&((S_4, O_7), \text{write}), ((S_5, O_1), \text{write}), ((S_5, O_2), \text{write}), ((S_5, O_3), \text{write}),\\
&((S_5, O_4), \text{write}), ((S_5, O_5), \text{write}), ((S_5, O_6), \text{write}), ((S_5, O_7), \text{write}),
\end{aligned}
$$

$$
\begin{aligned}
&((S_5, O_8), \text{write}), ((S_6, O_1), \text{write}), ((S_6, O_3), \text{write}), ((S_6, O_4), \text{write}),\\
&((S_6, O_5), \text{write}), ((S_6, O_7), \text{write}), ((S_6, O_8), \text{write}), ((S_7, O_1), \text{write}),\\
&((S_7, O_2), \text{write}), ((S_7, O_3), \text{write}), ((S_7, O_4), \text{write}), ((S_7, O_5), \text{write}),\\
&((S_7, O_6), \text{write}), ((S_7, O_8), \text{write}), ((S_8, O_1), \text{write}), ((S_8, O_2), \text{write}),\\
&((S_8, O_3), \text{write}), ((S_8, O_5), \text{write}), ((S_8, O_6), \text{write}), ((S_8, O_7), \text{write})\}\\
\text{ec}_8 = \{&((S_1, O_1), \text{read-write}), ((S_1, O_2), \text{read-write}), ((S_1, O_3), \text{read-write}),\\
&((S_1, O_4), \text{read-write}), ((S_1, O_5), \text{read-write}), ((S_1, O_6), \text{read-write}),\\
&((S_1, O_7), \text{read-write}), ((S_1, O_8), \text{read-write}), ((S_2, O_1), \text{read-write}),\\
&((S_2, O_2), \text{read-write}), ((S_2, O_3), \text{read-write}), ((S_2, O_4), \text{read-write}),\\
&((S_2, O_5), \text{read-write}), ((S_2, O_6), \text{read-write}), ((S_2, O_7), \text{read-write}),\\
&((S_2, O_8), \text{read-write}), ((S_3, O_1), \text{read-write}), ((S_3, O_2), \text{read-write}),\\
&((S_3, O_4), \text{read-write}), ((S_3, O_5), \text{read-write}), ((S_3, O_6), \text{read-write}),\\
&((S_3, O_7), \text{read-write}), ((S_3, O_8), \text{read-write}), ((S_4, O_1), \text{read-write}),\\
&((S_4, O_2), \text{read-write}), ((S_4, O_3), \text{read-write}), ((S_4, O_4), \text{read-write}),\\
&((S_4, O_5), \text{read-write}), ((S_4, O_6), \text{read-write}), ((S_4, O_7), \text{read-write}),\\
&((S_4, O_8), \text{read-write}), ((S_5, O_2), \text{read-write}), ((S_5, O_3), \text{read-write}),\\
&((S_5, O_4), \text{read-write}), ((S_5, O_5), \text{read-write}), ((S_5, O_6), \text{read-write}),\\
&((S_5, O_7), \text{read-write}), ((S_5, O_8), \text{read-write}), ((S_6, O_1), \text{read-write}),\\
&((S_6, O_2), \text{read-write}), ((S_6, O_3), \text{read-write}), ((S_6, O_4), \text{read-write}),\\
&((S_6, O_5), \text{read-write}), ((S_6, O_6), \text{read-write}), ((S_6, O_7), \text{read-write}),\\
&((S_6, O_8), \text{read-write}), ((S_7, O_1), \text{read-write}), ((S_7, O_2), \text{read-write}),\\
&((S_7, O_4), \text{read-write}), ((S_7, O_5), \text{read-write}), ((S_7, O_6), \text{read-write}),\\
&((S_7, O_7), \text{read-write}), ((S_7, O_8), \text{read-write}), ((S_8, O_1), \text{read-write}),\\
&((S_8, O_2), \text{read-write}), ((S_8, O_3), \text{read-write}), ((S_8, O_4), \text{read-write}),\\
&((S_8, O_5), \text{read-write}), ((S_8, O_6), \text{read-write}), ((S_8, O_7), \text{read-write}),\\
&((S_8, O_8), \text{read-write})\}\\
\text{ec}_9 = \{&((S_1, O_1), \text{execute}), ((S_1, O_2), \text{execute}), ((S_1, O_3), \text{execute}),\\
&((S_1, O_4), \text{execute}), ((S_1, O_5), \text{execute}), ((S_1, O_6), \text{execute}),\\
&((S_1, O_7), \text{execute}), ((S_1, O_8), \text{execute}), ((S_2, O_1), \text{execute}),\\
&((S_2, O_3), \text{execute}), ((S_2, O_4), \text{execute}), ((S_2, O_5), \text{execute}),\\
&((S_2, O_6), \text{execute}), ((S_2, O_8), \text{execute}), ((S_3, O_1), \text{execute}),\\
&((S_3, O_2), \text{execute}), ((S_3, O_4), \text{execute}), ((S_3, O_5), \text{execute}),\\
&((S_3, O_6), \text{execute}), ((S_3, O_7), \text{execute}), ((S_3, O_8), \text{execute}),\\
&((S_4, O_1), \text{execute}), ((S_4, O_2), \text{execute}), ((S_4, O_3), \text{execute}),\\
&((S_4, O_4), \text{execute}), ((S_4, O_5), \text{execute}), ((S_4, O_6), \text{execute}),\\
&((S_4, O_7), \text{execute}), ((S_4, O_8), \text{execute}), ((S_5, O_1), \text{execute}),
\end{aligned}
$$

$$
\begin{aligned}
&((S_5, O_2), \text{execute}), ((S_5, O_3), \text{execute}), ((S_5, O_4), \text{execute}),\\
&((S_5, O_5), \text{execute}), ((S_5, O_6), \text{execute}), ((S_5, O_7), \text{execute}),\\
&((S_5, O_8), \text{execute}), ((S_6, O_1), \text{execute}), ((S_6, O_2), \text{execute}),\\
&((S_6, O_3), \text{execute}), ((S_6, O_4), \text{execute}), ((S_6, O_5), \text{execute}),\\
&((S_6, O_7), \text{execute}), ((S_6, O_8), \text{execute}), ((S_7, O_1), \text{execute}),\\
&((S_7, O_2), \text{execute}), ((S_7, O_4), \text{execute}), ((S_7, O_5), \text{execute}),\\
&((S_7, O_6), \text{execute}), ((S_7, O_7), \text{execute}), ((S_7, O_8), \text{execute}),\\
&((S_8, O_1), \text{execute}), ((S_8, O_2), \text{execute}), ((S_8, O_3), \text{execute}),\\
&((S_8, O_4), \text{execute}), ((S_8, O_5), \text{execute}), ((S_8, O_6), \text{execute}),\\
&((S_8, O_7), \text{execute}), ((S_8, O_8), \text{execute})\}\\
\text{ec}_{10} = \{&((S_1, O_1), \text{append}), ((S_1, O_2), \text{append}), ((S_1, O_3), \text{append}),\\
&((S_1, O_4), \text{append}), ((S_1, O_5), \text{append}), ((S_1, O_6), \text{append}),\\
&((S_1, O_7), \text{append}), ((S_1, O_8), \text{append}), ((S_2, O_1), \text{append}),\\
&((S_2, O_2), \text{append}), ((S_2, O_3), \text{append}), ((S_2, O_4), \text{append}),\\
&((S_2, O_6), \text{append}), ((S_2, O_7), \text{append}), ((S_2, O_8), \text{append}),\\
&((S_3, O_1), \text{append}), ((S_3, O_2), \text{append}), ((S_3, O_3), \text{append}),\\
&((S_3, O_4), \text{append}), ((S_3, O_6), \text{append}), ((S_3, O_7), \text{append}),\\
&((S_3, O_8), \text{append}), ((S_4, O_1), \text{append}), ((S_4, O_2), \text{append}),\\
&((S_4, O_3), \text{append}), ((S_4, O_6), \text{append}), ((S_4, O_7), \text{append}),\\
&((S_4, O_8), \text{append}), ((S_5, O_2), \text{append}), ((S_5, O_3), \text{append}),\\
&((S_5, O_4), \text{append}), ((S_5, O_6), \text{append}), ((S_5, O_7), \text{append}),\\
&((S_5, O_8), \text{append}), ((S_6, O_1), \text{append}), ((S_6, O_2), \text{append}),\\
&((S_6, O_3), \text{append}), ((S_6, O_4), \text{append}), ((S_6, O_6), \text{append}),\\
&((S_6, O_7), \text{append}), ((S_6, O_8), \text{append}), ((S_7, O_1), \text{append}),\\
&((S_7, O_2), \text{append}), ((S_7, O_3), \text{append}), ((S_7, O_4), \text{append}),\\
&((S_7, O_6), \text{append}), ((S_7, O_7), \text{append}), ((S_7, O_8), \text{append}),\\
&((S_8, O_1), \text{append}), ((S_8, O_2), \text{append}), ((S_8, O_3), \text{append}),\\
&((S_8, O_6), \text{append}), ((S_8, O_7), \text{append}), ((S_8, O_8), \text{append})\}
\end{aligned}
$$

Because all subjects, objects and access attributes are assigned to equivalence classes, only one subject, object and access attribute combination of each equivalence class has to be tested.

Test	**subject**	**object**	**attribute**	**tested ec**	**result**
1	S_1	O_5	read	ec_1	valid
2	S_7	O_7	write	ec_2	valid
3	S_5	O_1	read-write	ec_3	valid
4	S_6	O_6	execute	ec_4	valid
5	S_8	O_5	append	ec_5	valid

6	S_2	O_3	read	ec_6	invalid
7	S_5	O_4	write	ec_7	invalid
8	S_3	O_8	read-write	ec_8	invalid
9	S_4	O_7	execute	ec_9	invalid
10	S_8	O_2	append	ec_{10}	invalid

Table 3: Result of the equivalence classes test

5 Conclusion

In our paper we presented a method for defining an exhaustive list of test cases based on formally described equivalence classes that are derived from the formal security policy description. We distinguished between satisfying equivalence classes that are used to verify if the system is over-constrained and violating equivalence classes, showing if the system is under-constrained. Additionally we defined the equivalence classes for the Bell and LaPadula model and defined test cases, based on the Bell and LaPadula equivalence classes for a sample system. Our current and further investigations will be in the field of testing formally history based security models like the Limes Security model [6] and the Chinese Wall Security Policy model [2].

6 Acknowledgement

The authors thank Dipl-Inf. Dieter Kessler for his support in the preparation of this paper.

References

1. Lampson, Butler W., *Protection*, Proceedings of the 5th Princeton Conference on Information Sciences and Systems. pp. 437., Princeton,1971
2. Brewer, D. F. C. and Nash, M. J., *The Chinese Wall Security Policy*, in IEEE Symposium on Security and Privacy, Oakland, pp. 206–214, 1989.
3. Lin, T. Y. *Chinese Wall Security Policy-An Aggressive Model*, Proceedings of the Fifth Aerospace Computer Security Application Conference, December 4–8, pp. 286–293, 1989.
4. Bell, D. and LaPadula, L., *Secure Computer Systems: Mathematical Foundations*, MITRE Corporation, Bedford, MA, Technical Report MTR-2547, Vol. I., 1973
5. Clark, D. and Wilson, D., *A Comparison of Commercial and Military Security Policies*, in IEEE Symposium on Security and Privacy, pp. 184–194, 1987.
6. Hermann, Eckehard, *The Limes Security Model for Information Flow Control*, FARES Workshop of the Sixth International Conference on Availability, Reliability and Security (ARES 2011), Vienna, Austria, Aug 22–26, 2011.
7. Hu, Hongxin and Ahn, Gail-Joon, *Enabling Verification and Conformance Testing for Access Control Model*, SACMAT 08, Estes Park, Colorado, USA, June 11–13, 2008.

8. Murnane, Tafline and Reed, Karl, *On the Effectiveness of Mutation Analysis as a Black Box Testing Technique*, 13th Australian Software Engineering Conference (ASWEC'01), Canberra, Australia, August 27–28, 2001.
9. Grimm, Ruediger, *A Formal IT-Security Model for a Weak Fair-Exchange Cooperation with Non-Repudiation Proofs*, International Conference on Emerging Security Information, Systems and Technologies, Athens, June 18–23, 2009.
10. Godefroid, Patrice; Levin, Michael Y.; Molnar, David, *Automated Whitebox Fuzz Testing*, Network and IT Security Conference, San Diego, CA, February 8–11, 2008
11. Myers, Glenford, *The Art of Software Testing*, Wiley-Interscience Publication, 1979
12. Hu, Vincent C.; Martin, Evan; Hwang, JeeHyun; Xie, Tao, *Conformance Checking of Access Control Policies Specified in XACML*, 31st Annual International Computer Software and Applications Conference, Beijing, 2007
13. Martin, Evan and Xie, Tao, *A fault model and mutation testing of access control policies*, 16th International Conference on World Wide Web, May 2007.
14. Martin, Evan and Xie, Tao, *Automated test generation for access control policies*, 17th IEEE International Conference on Software Reliability Engineering, November 2006.
15. Martin, Evan and Xie, Tao, *Defining and measuring policy coverage in testing access control policies*, International Conference on Information and Communications Security, pages 139–158, December 2006.
16. De Angelis, Guglielmo; Kirkham, Tom; Winfield, Sandra, *Access Policy Compliance Testing in a User Centric Trust Service Infrastructure*, QASBA '11, Lugano, Switzerland, September 14, 2011
17. Traon,Yves Le; Mouelhi, Tejeddine; Baudry, Benoit, *Testing security policies: going beyond functional testing*, 18th IEEE International Symposium on Software Reliability (ISSRE '07), Novembre 5–9, Sweden, 2007.