Evaluating test characteristics and effectiveness of FSM-based testing methods on RBAC systems

Carlos Diego Nascimento Damasceno
University of São Paulo – USP
São Carlos, SP, Brazil
damascenodiego@usp.br

Paulo Cesar Masiero
University of São Paulo – USP
São Carlos, SP, Brazil
masiero@icmc.usp.br

Adenilso Simao
University of São Paulo – USP
São Carlos, SP, Brazil
adenilso@icmc.usp.br

ABSTRACT

Access control mechanisms demand rigorous software testing approaches, otherwise they can end up with security flaws. Finite state machines (FSM) have been used for testing Role-Based Access Control (RBAC) mechanisms and complete, but significantly large, test suites can be obtained. Experimental studies have shown that recent FSM testing methods can reduce the overall test suite length for random FSMs. However, since the similarity between random FSMs and these specifying RBAC mechanisms is unclear, these outcomes cannot be necessarily generalized to RBAC. In this paper, we compare the characteristics and effectiveness of test suites generated by traditional and recent FSM testing methods for RBAC policies specified as FSM models. The methods W, HSI and SPY were applied on RBAC policies specified as FSMs and the test suites obtained were evaluated considering test characteristics (number of resets, average test case length, and test suite length) and effectiveness on the RBAC fault domain. Our results corroborate some outcomes of previous investigations in which test suites presented different characteristics. On average, the SPY method generated test suites with 32% less resets, average test case length 78% greater than W and HSI, and overall length 46% lower. There were no differences among FSM testing methods for RBAC regarding effectiveness. However, the SPY method significantly reduced the overall test suite length and the number of resets.

CCS Concepts

•Security and privacy → Access control; •Software and its engineering → Formal software verification; Empirical software validation;

Keywords

Finite state machine; Role-Based Access Control (RBAC); Experiments; Conformance Testing

SBES '16, September 19 - 23, 2016, Maringá, Brazil

ISBN 978-1-4503-4201-8/16/09…$15.00

DOI: http://dx.doi.org/10.1145/2973839.2973849

1. INTRODUCTION

Preserving the confidentiality, integrity and availability of personal and critical data has become a mandatory requirement for most industrial-scale information systems. To achieve this goal, access control mechanisms can be used to enforce security policies and data protection [12]. In short, access control ensures that only the intended users can access resources and only the required access to accomplish some task will be given. In this context, the Role-Based Access Control (RBAC) model has been established as one of the most significant access control paradigms [10]. The RBAC model is conceptually simple: In an organization, users receive responsibilities and privileges through roles; Analogously, in RBAC systems, permissions are granted via roles assigned to users. Despite its simplicity, the RBAC model can reduce the complexity of security management routines by grouping privileges in roles [19]. Nevertheless, mistakes can occur during the development of RBAC systems threatening users' privacy, lead to faults, or either security breaches. Therefore, a careful verification, validation, and testing (VV&T) processes must be executed.

Access control testing is the software testing process performed on access control systems. It consists of the evaluation of obtained responses from access control systems given access requests against the expected responses [17]. Formal testing approaches, such as Finite State Machine (FSM) based testing, have been applied on RBAC domain and experimental analysis have revealed that, although the representation of RBAC policies as complete FSMs can be used to generate very effective test suites, they tend to be very costly, or *astronomically large*, as they say [14]. Later, Endo and Simao [8] presented evidences that recent FSM testing methods, such as SPY, can reduce the overall test suite length for random FSMs. Although, the similarity between random FSMs and the ones used in practice, such as these specifying RBAC policies, is unclear. Thus, the outcomes of investigations using random FSM models cannot be generalized to the RBAC domain. In this sense, investigations on RBAC testing using FSM are required. Studies in this domain can improve understanding the potential of FSM based testing methods on RBAC and support VV&T decisions.

This paper presents an experimental investigation of the characteristics and the effectiveness of test suites generated by traditional and recent FSM testing methods on the RBAC domain. Two traditional methods, W and HSI, and a recent one, SPY, were included. Five RBAC policies were specified as complete finite state models and tested using each

FSM-based testing method. The generated test suites were evaluated based on their characteristics (number of resets, test case length, and test suite length) and effectiveness using the RBAC fault model. The contributions of this paper are twofold: it proposes a framework to compare FSM-based testing methods on RBAC and reanalyzes Endo and Simao [8] and Masood et al. [14] outcomes using three different FSM-based testing methods on RBAC.

The remainder of this paper is organized as follows: In section 2, the definitions of FSM and RBAC testing are presented. Section 3 introduces the experimental framework proposed to this investigation. Section 4 shows the results obtained and section 5 presents the discussions about them. Threats to validity, related works, and conclusions are presented respectively in sections 6, 7, and 8.

2. PRELIMINARIES

A Finite State Machine (FSM) is an hypothetical machine M composed by states and transitions [11], as shown in Figure 1. Formally, an FSM is a tuple $M =< S, s_0, I, O, D, \delta, \lambda >$ where

- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- I is the finite set of input symbols,
- O is the finite set of output symbols,
- $D \subseteq S \times I$ is the specification domain,
- $\delta : D \to S$ is the transition function, and
- $\lambda : D \to O$ is the output function.

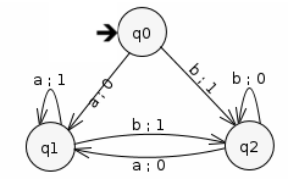


Figure 1: Example of FSM model

An input x is defined for s if $(s, x) \in D$, which means that in state s there is a *defined transition* which consumes input x. In each given moment, an FSM has a single current state $s_i \in S$ which can change to $s_j = \delta(s_i, x)$ by applying an input $x \in I$ defined in s_i, producing the output $y = \lambda(s, x)$. An FSM is *complete* if all inputs are defined for all the states, i.e. $D = S \times I$, otherwise it is called *partial*.

The concatenation of two sequences α and ω is denoted as $\alpha\omega$. A sequence α is prefix of a sequence β, denoted by $\alpha \leqslant \beta$, if $\beta = \alpha\omega$, for some given sequence ω. An empty sequence is denoted by ϵ and a sequence α is a proper prefix of a sequence β, denoted by $\alpha < \beta$, if $\beta = \alpha\omega$ for a given $\omega \neq \epsilon$. The set of prefix sequences of a set T is defined as $pref(T) = \{\alpha \mid \exists\beta \in T \; and \; \alpha < \beta\}$. If $T = pref(T)$, T is *prefix-closed*. Using the prefix definition and the empty sequence, the concept of transition and output functions can be extended to input sequences.

A sequence $\alpha = x_1x_2...x_n \in I^*$ is an input sequence defined for state $s \in S$, if there exist states $s_1, s_2, ..., s_{n+1}$ such that $s = s_1$ and $\delta(s_i, x_i) = s_{i+1}$, for all $1 \leq i \leq n$. The transition and output functions is lifted to input sequences as usual; for the empty sequence ϵ, we have that $\delta(s, \epsilon) = s$ and $\lambda(s, \epsilon) = \epsilon$. For an input sequence αx defined for state s, we have that $\delta(s, \alpha x) = \delta(\delta(s, \alpha), x)$ and $\lambda(s, \alpha x) = \lambda(s, \alpha)\lambda(\delta(s, \alpha), x)$. A sequence $\alpha \in I^*$ is a transfer sequence from s to s_{n+1} and s_{n+1} is said reachable from s if $\delta(s, \alpha) = s_{n+1}$. If every state is reachable from s_0 the FSM is *initially connected* and if all states are reachable from every state the FSM is *strongly connected*.

The symbol $\Omega(s)$ denotes all defined input sequences to the state s and Ω_M abbreviates $\Omega(s_0)$ and depicts all defined input sequences of an FSM M. An FSM M can have a *reset operation*, denoted by r, which takes to s_0 regardless the current state.

A separating sequence for two states s_i and s_j is a sequence γ such that $\gamma \in \Omega(s_i) \cap \Omega(s_j)$ and $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$. In addition, if γ is able to distinguish every pair of states of a machine, it is a distinguishing sequence. Formally, if $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$ is valid for all pairs of state $s_i, s_j \in S$, then γ is a distinguishing sequence. Considering the FSM presented in Figure 1, the sequence a is a separating sequence for states q_0 and q_1 since $\lambda(q_0, a) = 0$ and $\lambda(q_1, a) = 1$.

Two FSM models $M_S =< S, s_0, I, O, D, \delta, \lambda >$ and $M_I =< S', s'_0, I, O', D', \delta', \lambda' >$ are equivalent if for each state of M_S there exists an equivalent state in M_I. Two states are equivalent if $\forall \; \alpha \in I, \; \lambda(s_i, \alpha) = \lambda'(s_j, \alpha)$. An input sequence $\alpha \in \Omega_M$ starting with a reset symbol r is a *test case* of M. Given two test sequences $\alpha, \beta \in T$, if α is a proper prefix of the test case β, the execution of β implies the execution of α, thus α can be removed from T without changing results.

A test suite of M consists of a finite set T of test cases of M, such that there are no $\alpha, \beta \in T$ such that $\alpha < \beta$. The number of symbols of a sequence α is represented by $|\alpha|$ and describes the test case length. Given a test case α, the cost of execution is defined as $|\alpha| + 1$, which stands for the test case length $|\alpha|$ plus one reset operation. The number of test cases of one test suite T also describes the number of resets of T which is depicted as $|T|$.

2.1 Mutation analysis of FSMs

In FSM-based testing, given a specification M, the symbol $\Im(M)$ denotes the set of all deterministic FSMs with the same input alphabet of M for which all sequences in Ω_M are defined. Given $m \geq 1$, then $\Im_m(M)$ denotes all FSMs of $\Im(M)$ with at most m states. Given a specification M with n states, a test suite $T \subseteq \Omega_M$ is m-complete if for each $N \in \Im_m$ distinguishable from M, there exists a test $t \in T$ that distinguish M from N. The set $\Im(M)$ is called *fault domain* for M and can be used to evaluate test effectiveness.

A fault domain can be generated either manually or by automatically generating variants of the FSM, named *mutants*, using *mutation operators* [2]. If the result of running a mutant is different from the original FSM for any test case, the mutant is *killed* and the seeded fault denoted by the mutant is detected. However, some mutants can be syntactically different but functionally equivalent to the original model. These are called *equivalent mutants*.

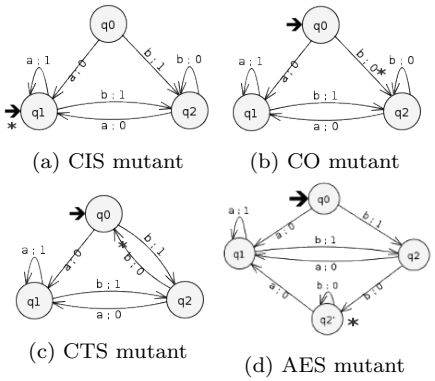
The process of analyzing if mutants are killed and test cases trigger such failures is called *mutation analysis* and is often used on software testing research [13, 9]. The main metric of the mutation analysis is the *mutation score*, which

indicates the effectiveness of a test suite. Given the test suites T, the mutation score (or test effectiveness) is calculated as

$$T_{\text{eff}} = \frac{\#km}{(\#tm - \#em)}$$

, where $\#km$ represents the number of killed mutants, $\#tm$ defines the total number of generated mutants, and $\#em$ defines the number of equivalent mutants. An *m-complete* test suite has *full fault coverage*, and thus, has score 1.0, by definition.

In FSM-based testing, the following mutation operators are often used [6]: *change initial state* (CIS), which changes the s_0 of an FSM to s_k, such that $s_0 \neq s_k$; *change output* (CO), which modifies the output of a transition (s, x), using a different function $\Lambda(s, x)$ instead of $\lambda(s, x)$; *change tail state* (CTS), which modifies the tail (destination) state of a transition (s, x), using a different function $\Delta(s, x)$ instead of $\delta(s, x)$; and *add extra state* (AES), which inserts a new state such that mutant N is equivalent to M. Figures 2a, 2b, 2c, and 2d respectively show examples of mutants generated from the FSM of Figure 1 using CIS, CO, CTS, and AES operators. Changes are marked with an asterisk (*).



(a) CIS mutant (b) CO mutant

(c) CTS mutant (d) AES mutant

Figure 2: Examples of FSM mutants

2.2 FSM-based testing methods

Two basic sequences are used to obtain partial information about the models: state cover (Q set) and transition cover (P set). A set of input sequences Q is a *state cover set* of M, if for each state $s_i \in S$ there exists a sequence $\alpha \in Q$ such that $\delta(s_0, \alpha) = s_i$ and it includes the empty sequence ϵ to reach the initial state. A set of inputs P is named *transition cover set* of M if for each transition $(s, x) \in D$ there exist sequences $\alpha, \alpha x \in P$, such that $\delta(s_0, \alpha) = s$, and it includes the empty sequence ϵ. The P set can be generated from the *testing tree* of an FSM under test [5]. The nodes of the testing tree correspond to the states of the FSM, the tree is rooted at the initial state, and each tree edge correspond to one FSM transition that appear exactly one single time. The state cover and transition cover sets of the FSM presented in Figure 1 are respectively $Q = \{\epsilon, a, b\}$ and $P = \{\epsilon, a, aa, ba, b, ab, bb\}$.

To identify states and transitions of FSM models, traditional methods, such as W [6] and HSI [18], require some pre-defined sets. These sets are the *characterization set* and *separating families*. A characterization set (W set) is a set of defined input sequences containing at least a sequence which distinguishes each pair of states of an FSM. Formally, it means that for all pairs of different states $s_i, s_j \in S$, there exists $\alpha \in W$, such that $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. A separating family is a set of state identifiers H_i for each state $s_i \in S$ which satisfies the condition that for all pairs of different states $s_i, s_j \in S$ there are $\beta \in H_i, \gamma \in H_j$ with a common prefix α such that $\alpha \in \Omega(s_i) \cap \Omega(s_j)$ and $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. The characterization set of the FSM presented in Figure 1 is $W = \{a, b\}$.

Recent methods, such as SPY [21], rely on sufficient conditions to support test generation. However, these conditions are not necessary, i.e. if a test suite does not satisfy them it may still be *m-complete*. The W, HSI and SPY methods are described below.

2.2.1 W method

The W method is the most classical FSM-based test generation method [6]. It uses the transition cover set (P set) to traverse all transitions and then it applies the W set to identify each state reached. The W method can also be extended to detect an estimated number of n states in an implementation by concatenating the traversal set $\bigcup_{i=0}^{m-n}(I^i)$, such that $(m-n)$ depicts the number of extra states, before the W set. The set I^i contains all sequences of length i combining all input symbols of I and the traversal set consists of the union of all sets I^i with sequences of length ranging from 0 to $(m-n)$. Assuming the FSM in Figure 1, no extra states ($m = n$) or proper prefixes, the test suite generated by the W method is $T_W = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$, and the number of resets equals to $|T_W| = 8$.

2.2.2 HSI method

The Harmonized State Identifiers (HSI) method [18] uses state identifiers H_i to distinguish each state $s_i \in S$ of the FSM model. First, the HSI method concatenates the state cover set to the state identifiers set which is, in the worst case, the W set itself. Later, the transition cover set is also concatenated to the state identifiers in order to cover non-traversed transitions. The HSI method can also be used on partial FSM. Assuming the FSM in Figure 1, no extra states or proper prefixes, the test suite generated by the HSI method is equals to $T_{HSI} = \{aaa, aba, abb, baa, bba, bbb\}$, and the number of resets is equals to $|T_{HSI}| = 6$.

2.2.3 SPY method

The SPY method [21] is a recent test generation method for complete and partial FSMs able to generate *m-complete* test suites and reduce test suite length by concatenating sequences *on-the-fly*. First, all sequences of the state cover set are concatenated to state identifiers. Later, differently from the existing methods, the traversal set is distributed over the test set obtained from the concatenation of the Q set with the state identifiers based on sufficient conditions. Thus, test tree branching can be avoided as much as possible and the test suite length and the number of resets can be reduced. Experimental results have indicated that the SPY method can generate test suites on average 40% shorter, and longer test cases compared to traditional methods, such as W and HSI. Moreover, SPY method can achieve higher fault detection effectiveness even if the number of extra states is underestimated [8]. Assuming the FSM in Figure 1, no extra states or proper prefixes, the test suite generated by the SPY method is equals to $T_{SPY} = \{aaaba, abbb, baa, bba\}$, and the number of resets is equals to $|T_{SPY}| = 4$.

2.3 Role Based Access Control

Access Control (AC) is one of the most frequently used approaches to guarantee data confidentiality, integrity and availability in information systems. AC mechanisms are used to implement security policies for mediating the access to resources and granting the use only for authorized personnel based on security models. In this context, the RBAC model is considered one of the most important innovations in security management [1]. *Roles* consist of organizational figures (e.g. functions or jobs) assigned to responsibilities (e.g. *permissions*) which intermediate the assignment and revoking of permissions to *users*. *Role hierarchies* can be defined to specify inheritance of permissions [19].

Masood et al. [14] define an RBAC policy as a 16-tuple $\mathcal{P} = (U, R, Pr, UR, PR, \leq_A, \leq_I, I, S_u, D_u, S_r, D_r, SSoD, DSoD, S_s, D_s)$, where:

- U and R are the finite sets of users and roles;
- Pr is the finite set of permissions;
- $UR \subseteq U \times R$ is the set of user-role assignments;
- $PR \subseteq Pr \times R$ is the set of permission-role assignments;
- $\leq_A \subseteq R \times R$ and $\leq_I \subseteq R \times R$ are the activation and inheritance role hierarchy relationships;
- $I = \{AS, DS, AC, DC, AP, DP\}$ is the set of types of requests: user-role assignment, deassignment, activation and deactivation, and permission-role activation and deactivation, respectively;
- $S_u, D_u : U \rightarrow \mathbb{Z}^+$ are static and dynamic cardinality constraints on users;
- $S_r, D_r : R \rightarrow \mathbb{Z}^+$ are static and dynamic cardinality constraints on roles;
- $SSoD, DSoD \subseteq 2^R$ are the Static and Dynamic Separation of Duty (SoD) sets, respectively;
- $S_s : SSoD \rightarrow \mathbb{Z}^+$ specifies the cardinality of SSoD sets;
- $D_s : DSoD \rightarrow \mathbb{Z}^+$ specifies the cardinality of DSoD sets.

Moreover, two types of operators for RBAC mutation analysis are proposed [14]: Mutation operators and Element modification operators. Given a policy $\mathcal{P}$, the *mutation operators* generate a set of $\mathcal{P}'$ by replacing users, roles and permissions from UR, PR, $\leq_A$, and $\leq_I$, and adding, removing and replacing users from $SSoD$ and $DSoD$ sets. The *element modification operators* generate mutants by incrementing and decrementing cardinality constraints of user (S_u, D_u,), role (S_r, D_r,), SSoD (S_s) and DSoD (D_s). All the categories of RBAC faults have a correspondent type of FSM fault [6] and a single RBAC fault can be exhibited across many different transitions of an FSM [15]. Policy 1 presents an example of RBAC policy with two users (line 1), one role (line 2), and two permissions (line 3). User $u1$ is assigned to role $r1$ (line 4) and has access to permissions $pr1$ and $pr2$ (line 5). Both users only can be assigned to at most one role (line 6). Role $r1$ can be assigned to at most two users (line 7), however it can be activated by only one user per time (line 8).

Policy 1 Example of RBAC policy

```
1: U = {u1, u2}
2: R = {r1}
3: Pr = {pr1,pr2}
4: UR = { (u1,r1) }
5: PR = { (r1,pr1), (r1,pr2) }
6: Su(u1) = Su(u2) = Du(u1) = Du(u1) = 1
7: Sr(r1) = 2
8: Dr(r1) = 1
```

2.3.1 Modeling RBAC policies as FSM

An FSM specifying a policy P, named $FSM(P)$, describes all the access control decisions which an RBAC mechanism should enforce given P. Essentially, an $FSM(P)$ can be described as a tuple $< S_P, s_0, I_P, O, D, \delta_P, \lambda_P >$ such that S_P is the set of reachable states given P, $s_0 \in S$ is the initial state where P currently stands given UR and PR, I_P is the input domain formed by all combinations of I, U and R elements of P, O is the output domain formed by the symbols *granted* and *denied*, $D = S_P \times I_P$ is the specification domain, $\delta : D \rightarrow S_P$ is the state transition function, and $\lambda : D \rightarrow O$ is the output function.

By analyzing the $FSM(P)$ modeling approach, three invariants can be identified: (i) self-loop transitions return *denied* and express that the information necessary for access control decisions does not change; (ii) the diameter of the $FSM(P)$ model is equals to or less than $2 \times |U| \times |R|$, the number of requests necessary to assign and activate all users to every role from the state where nobody is assigned to any role; and (iii) this modelling approach has $3^{|U|\times|R|}$ as upper bound limit of states and the number of reachable states depends on P mutable elements.

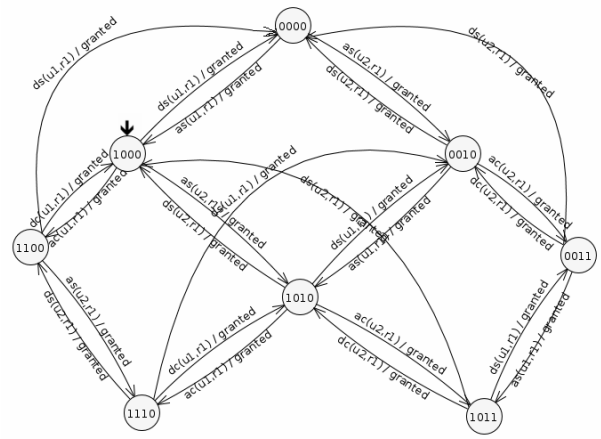


Figure 3: FSM specifying access control decisions

The $FSM(P)$ states are labeled using sequences of pairs of bits, one for each *user-role* and *permission-role* combination. A pair of user-role can become assigned (10), activated (11), or not assigned (00), and a pair of permission-role can become assigned (10), or not assigned (00). The pair 01 is not used since user cannot activate roles they are not assigned. Figure 3 depicts the $FSM(P)$ correspondent to Policy 1. It has eight states due to $Dr(r1) = 1$ which denies transitions to 1111. Self-loop transitions and *permission-role* bits are not shown to keep the figure uncluttered.

2.3.2 FSM-Based Testing of RBAC systems

After modelling an $FSM(P)$, any FSM-based test method can be used for RBAC testing. Figure 4 illustrates a part of a test tree generated from four test cases applied to the $FSM(P)$ in Figure 3 with states and transitions labelled.

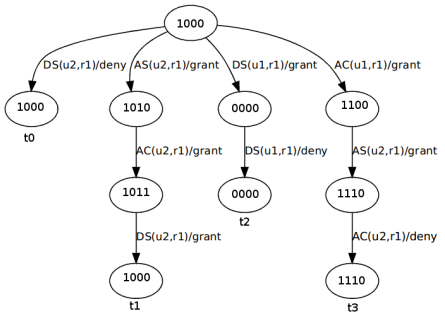


Figure 4: Test tree of an $FSM(P)$ and four test cases

It is important to highlight that the RBAC mutation analysis is performed under the RBAC fault domain instead of mutating the $FSM(P)$ model. For example, an $FSM(P)$ generated from a mutant of the Policy 1 where $Dr(r1) = 1$ is incremented to $Dr(r1) = 2$ has the state 1111 as reachable. Thus, since the test sequence $t3$ presented in Figure 4 covers the transition $(1110, AC(u2, r1))$, it is able to kill a RBAC mutant where $Dr(r1) = 2$. Besides, testing RBAC policies with many users and roles can become very costly due to the upper bound limit, hence, test suites also tend to become large [14].

3. EXPERIMENTAL STUDY

Previous investigation on test suites generated by FSM testing methods on RBAC domain [14] pointed out that, although complete FSMs specifying RBAC policies can generate tests with 100% of effectiveness, they tend to be very large. Later, Endo and Simao [8] investigated different FSM-based testing methods using random FSMs and detected that recent methods can outperform traditional ones by reducing the overall test cost. In this sense, since the similarity between random FSM and these used in practice, such as $FSM(P)$, is unclear [8], the outcomes obtained using random FSM are not necessarily generalizable to RBAC domain. Thus, we designed an experimental framework to compare the characteristics and effectiveness of test suites generated by FSM-based testing methods on RBAC.

Two traditional (W, and HSI) and one recent (SPY) FSM testing methods were applied on the $FSM(P)$ models. As test characteristics, we considered test suite length, number of resets, and average test case length and as effectiveness, we considered the fault detection capability on the RBAC fault model [14]. Thus, instead of mutating FSM models, the RBAC policies were mutated. This experiment was designed to reanalyze Endo and Simao's outcomes [8] on RBAC domain using Masood et al. [14] modelling approach. The seven steps constituting the proposed framework are presented in Figure 5.

In the first (1) step, scientific papers discussing RBAC testing were systematically searched and analyzed to extract policies to be used as system under test (SUT). In the second (2) and third (3) steps, two modules, *rbac2fsm* and *rbacMutation*, were designed and developed to automate the conversion of RBAC policies to FSM models and the RBAC mutation analysis process, respectively. Afterwards, these modules were merged into one software named RBAC-Based Testing (RBAC-BT). RBAC-BT is a java open-source tool developed to support the execution of this experiment. In the fourth (4) and fifth (5) steps, the RBAC-BT tool was used to generate RBAC mutants and FSM models from the policies used as SUT, respectively. In the sixth (6) step, three FSM testing methods (W, HSI and SPY) were executed to generate test suites from each $FSM(P)$. The same set of tools used by Endo and Simao [8] for test generation were adopted. A time limit of 24h for processing each $FSM(P)$ was defined and any processes with duration above this limit would be canceled and the state explosion problem would be assumed. In the last seventh (7) step, the characteristics and effectiveness of the test suites were evaluated using the RBAC-BT tool. In case of state explosion during the fifth step, we decided that constraints would be included in order to reduce the number of reachable states. All the artifacts used in this experiment are available on-line[1].

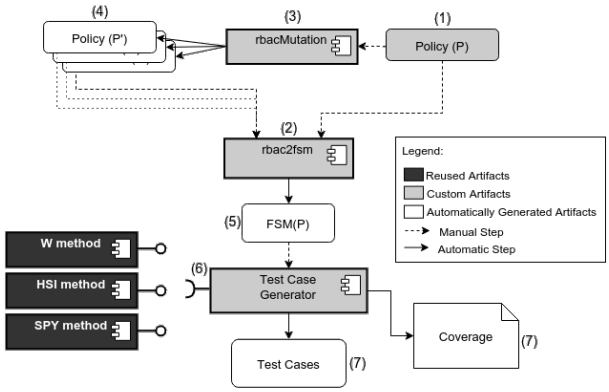


Figure 5: Schematic overview of the experiment

4. ANALYSIS OF RESULTS

In this section we present the results of our experiment. The computational environment used in this experiment was an Intel Core i7-4770 CPU 3.40GHz, 8 Gb RAM, 1Tb of hard disk running Ubuntu 14.04 LTS 64 bits.

Five policies were obtained from studies discussing RBAC testing and used in this experiment. We generated mutants from each policy and attempted to convert them to FSMs using the RBAC-BT tool. However, due to the state explosion problem, we only generated FSMs from *01_Masood2010Example1* and *02_SeniorTraineeDoctor*. Three policies spent more than 24h being converted and, in order to reduce the number of reachable states and make feasible the $FSM(P)$ generation, we redesigned them with constraints. Essentially, these policies were adapted with cardinality constraints, SoD sets and by removing users and roles. We named these policies as *03_ExperiencePointsv2*, *04_users11roles2v2*, and *05_Masood2009P2v2*. A summary of the policies is shown in Table 1. The occurrence of constraints (e.g. Su, Du) is depicted with an • symbol.

[1]https://github.com/damascenodiego/rbac-bt/

Table 1: Summary of the characteristics of the RBAC policies used in this experiment

Policy	|U|	|R|	$\|I_P\|$	$log_{10}(3^{\|U\|\times\|R\|})$	$\leq_A$	$\leq_I$	S_u	D_u	S_r	D_r	$SSoD$	$DSoD$
01_Masood2010Example1	2	1	8	0.9542			•	•	•	•		
02_SeniorTraineeDoctor	2	2	16	1.9084			•	•	•	•	•	
03_ExperiencePoints	3	4	48	3.2375			•					•
03_ExperiencePointsv2	2	4	32	2.7092					•		•	•
04_users11roles2v2	11	2	88	10.4966			•		•		•	
04_users11roles2	11	3	132	15.745								
05_Masood2009P2	2	6	48	5.7254				•		•	•	•
05_Masood2009P2v2	2	5	40	3			•	•		•	•	•

After refinement, we converted the adapted versions of the RBAC policies to FSM models. The characteristics of the FSMs generated from all five policies and the total number of mutants are shown in Table 2.

Table 2: FSMs and mutants generated from policies

Alias	Policy name	States	Transitions	Mutants
P01	01_Masood2010Example1	8	64	9
P02	02_SeniorTraineeDoctor	21	336	17
P03	03_ExperiencePointsv2	203	6496	11
P04	04_users11roles2v2	485	42680	28
P05	05_Masood2009P2v2	857	34280	48

The number of states and transitions of the generated FSM models ranged from 8 to 857 and from 64 to 42680, respectively, and the number of mutants generated ranged from 9 to 48. The number of outputs was omitted since this modeling approach assumes only two outputs (*granted* or *denied*). The upper limit of the number of states is shown in logarithmic scale. Although *P04* presented the highest upper limit, cardinality constraints were defined to all 11 users and 2 roles limiting the number of assignments to one and, consequently, the total number of reachable states.

The methods W, HSI and SPY were applied using the same artifacts of Endo and Simao [8] to generate test suites for each $FSM(P)$ and assuming no extra states. The duration of the test generation ranged from 5 milliseconds (*P01*) to 21 hours (*P05*). The whole test generation process took around 15 hours for all three methods applied on the five policies. Table 3 shows the duration of the test generation process in milliseconds (ms).

Table 3: Test generation - duration

Alias	W (ms)	HSI (ms)	SPY (ms)
P01	137	187	5
P02	306	54	116
P03	260700	193397	207771
P04	37841706	7116335	9306981
P05	18110699	76918751	78933280

After test generation, we used the RBAC-BT tool to measure test suites length, numbers of resets, and average test case lengths. The Pearson Correlation Coefficient (PCC), Spearman Correlation Coefficient (SCC) and Kendall Correlation Coefficient (KCC) were calculated between the aforementioned characteristic and the numbers of states and inputs of the FSMs using the R statistical software [2].

[2]https://www.r-project.org/

4.1 Test Suite Length

Table 4 shows the correlation between the test suite length (TS) and the number of states and inputs. On average, there is a very strong positive correlation between test suite length and both numbers of inputs and states. Although we reduced the number of reachable states using cardinality constraints and SoD sets, such as in *P04*, the test suite length was still influenced by the numbers of states and inputs of $FSM(P)$ which are proportional to the number of users and roles.

Table 4: Correlation between TS, inputs and states

	Correlation Inputs / TS			Correlation States / TS		
Method	PCC	SCC	KCC	PCC	SCC	KCC
W	0.950	1	1	0.717	0.9	0.8
HSI	0.958	1	1	0.682	0.9	0.8
SPY	0.906	1	1	0.823	0.9	0.8

In Table 5, the number of resets for each method and policy are presented in tabular format and, in Figure 6, in graphical format on logarithm scale.

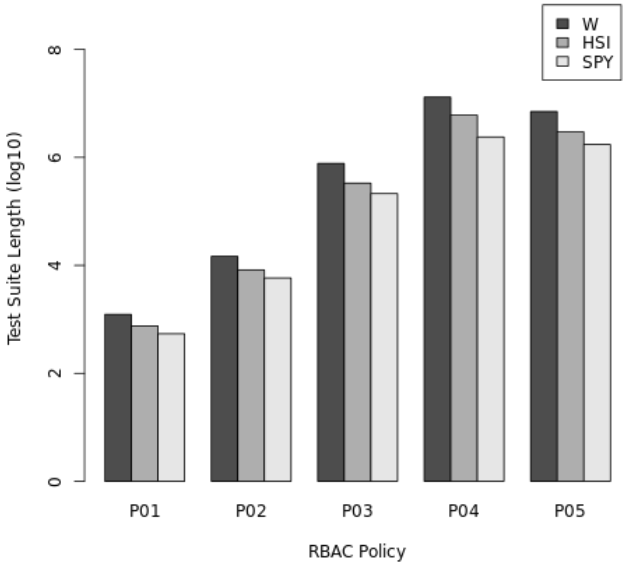


Figure 6: Test Suite Length for each policy

Table 5: Test Suite Length, number of states and inputs

Alias	States	Inputs	W - TS	HSI - TS	SPY - TS
P01	8	8	1240	753	542
P02	21	16	14704	8238	5841
P03	203	32	776074	333550	213799
P04	485	88	13125662	6085633	2392981
P05	857	40	7086325	2970528	1735818

TS: Test Suite Length

On average, test suites generated by the SPY method had 46% of the total length of test suites generated by W and HSI methods. In special, the SPY test suite generated for *P04* was 39% and 18% shorter than the test suites generated by W and HSI, respectively. Thus, SPY generated the shortest test suites in all cases and the following order was observed $SPY_{TS} < HSI_{TS} < W_{TS}$, as presented in [8]. Moreover, a nonlinear behavior was also identified when test suite length was analysed as a function of the number of states, specially in *P04*, reason why the correlation values between test suite length and number of states were lower.

4.2 Number of Resets

Table 6 shows the correlation between the Number of Resets (NR), the number of states, and the number of inputs. On average, there was a very high positive correlation between the number of resets and both numbers of inputs and states. The methods W and HSI presented a stronger correlation between the number of resets and inputs. The nonlinearity detected on *P04* persisted only for W and HSI test methods. There was a very strong correlation between states and resets for the SPY method. Thus, the number of resets of SPY test suites tend to increase as long as the number of states increases. Although, as shown in Table 7 and Figure 7, the number of resets of SPY test suites increased in a lower rate, compared to W and HSI.

Table 6: Correlation between NR, inputs and states

	Correlation Inputs / NR			Correlation States / NR		
Method	PCC	SCC	KCC	PCC	SCC	KCC
W	0.971	1	1	0.595	0.9	0.8
HSI	0.972	1	1	0.580	0.9	0.8
SPY	0.765	0.9	0.8	0.955	1	1

Table 7 and Figure 7 show the number of resets of each test suites generated for the five policies in tabular and graphical format using logarithmic scale. On average, SPY method generated test suites with a number of resets 42% lower than HSI and 22% lower than W. These outcomes corroborate previous results where SPY test suites presented a number of resets approximately 40% lower than HSI method [21]. Since SPY focuses on reducing the branches of the testing trees, a decrement of resets can be expected. Our experiment shows that this behavior still occur on the RBAC domain with greater chances when the number of reachable states increases.

In $P04$, the methods W and HSI generated test suites with the highest number of resets. However, the number of resets of SPY test suites decreased to 13% and 6% compared to these same methods. The SPY method enabled to reduce test tree branching significantly, specially in the *P04* scenario. A reason for that can be the disparity of self-loops in the $FSM(P04)$ of since it has many cardinality constraints

Table 7: Number of Resets, states and inputs

Alias	States	Inputs	W - NR	HSI - NR	SPY - NR
P01	8	8	285	176	93
P02	21	16	2528	1408	751
P03	203	32	119586	51451	24001
P04	485	88	2236388	993492	138766
P05	857	40	835600	353836	159463

NR: Number of Resets

limiting user and role assignments and, consequently, the number of reachable states. These outcomes corroborate previous investigations where SPY generated test suites with lower number of resets when compared to W and HSI. Thus, the order $SPY_{NR} < HSI_{NR} < W_{NR}$ detected by Endo and Simao's [8] is still valid.

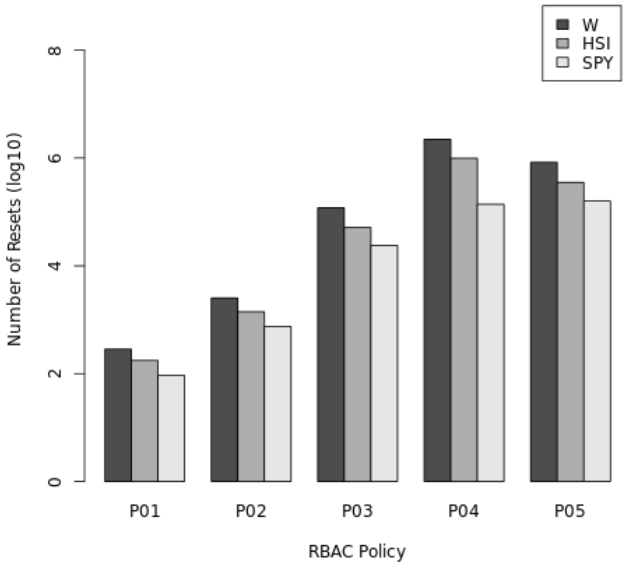


Figure 7: Number of Resets for each policy

4.3 Average Test Case Length

Table 8 shows the correlation between the average test case length (L) and the numbers of states and inputs. There was a moderate to very strong positive correlation between average test case length and the number of states while the correlation between average test case length and number of inputs was weak to very strong. SPY test suites presented a very strong correlation between inputs and average test case length (over 0.99) and a strong correlation to the number of states. The methods W and HSI presented weak to moderate correlation between average test case length and number of inputs due its the low variation of the average test case length.

This behavior can be seen in Table 9 and Figure 8 where average test case length is shown in tabular and graphical format. The average length did not increase significantly for W and HSI along the five test scenarios, while the average length of SPY test cases presented higher variation.

Table 8: Correlation between L and number of inputs and states

	Correlation Inputs / L			Correlation States / L		
Method	PCC	SCC	KCC	PCC	SCC	KCC
W	0.267	0.7	0.6	0.850	0.9	0.8
HSI	0.348	0.7	0.6	0.866	0.9	0.8
SPY	0.994	1	1	0.598	0.9	0.8

These outcomes contradict previous investigations where a negative correlation was found between the average test case length and the number of inputs [8]. Due to the $FSM(P)$ invariant (ii), a positive correlation between the test case length the number of inputs was expected.

Table 9: Average test case length, number of states and inputs

Alias	States	Inputs	W - L	HSI - L	SPY - L
P01	8	8	3.350	3.278	4.827
P02	21	16	4.816	4.850	6.777
P03	203	32	5.489	5.482	7.907
P04	485	88	4.869	5.125	16.24
P05	857	40	7.480	7.395	9.885

L: Average Test Case Length

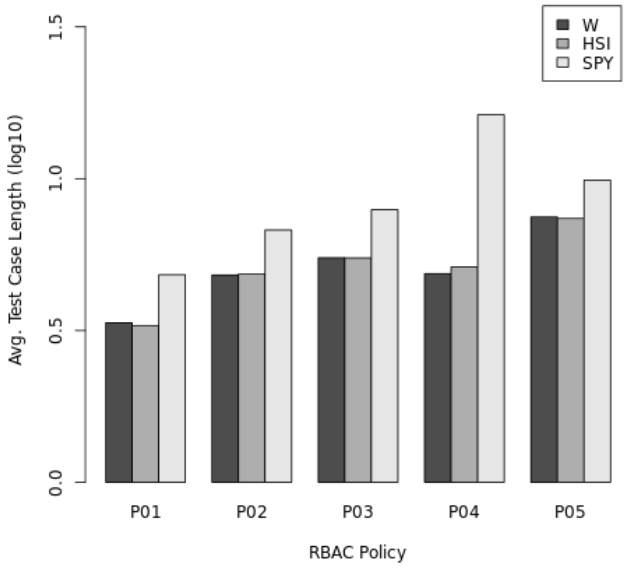


Figure 8: Avg. Test Case Length for each policy

The maximum test case length analysis also did not show high variation. Table 10 shows that the maximum test case length of W and HSI methods were similar. On the other hand, the SPY method presented a maximum test case length 14 times greater than W and HSI, on average. In the *P04* scenario, SPY test cases were 44 times longer than the ones generated by the traditional methods. Moreover, although the SPY method generated longer test cases, the SPY test suites presented an overall length lower than the other two methods. These test characteristics can be useful on VV&T processes and support test planning since the overall test suite length also describes the total cost of a test suite. In this sense, the order $W_L < HSI_L < SPY_L$ detected by Endo and Simao's [8] study also remains valid.

Table 10: Maximum test case length, number of states and inputs

Alias	States	Inputs	W max(L)	HSI max(L)	SPY max(L)
P01	8	8	4	4	14
P02	21	16	8	8	32
P03	203	32	7	7	98
P04	485	88	6	6	261
P05	857	40	12	12	106

max(L): Maximum Test Case Length

4.4 Test Effectiveness

Given each test suite T generated by the W, HSI and SPY methods to the five policies, the test effectiveness was measured using the mutation score on the RBAC fault domain. All the element modification and mutation operators were used in this experiment. However, the operators related to role hierarchies did not generate any mutant since the policies considered did not present this feature. Table 11 shows the number of generated and non-equivalent mutants for each policy.

Table 11: Policies and Mutants

Alias	Mutants	Non-Equivalent
P01	9	7
P02	17	9
P03	11	8
P04	28	26
P05	48	44

The test suite generated by the W, HSI and SPY testing methods were applied on each of the RBAC policies and the number of killed mutants was measured using the RBAC-BT tool. At the end, 100% of effectiveness was obtained in all test scenarios, corroborating Masood et al. [14] results and the assertion that as long as all transitions and states are covered, the fault detection effectiveness for RBAC faults will remain the same. In this sense, any of the testing methods can be used for testing RBAC policies and guarantee complete fault detection on the RBAC fault domain. Although, there is still an overall cost assigned to test execution which, as seen in previous sections, can be reduced by adopting recent testing methods, such as SPY, instead of the traditional ones.

5. DISCUSSION

In this experiment, we have found evidences which corroborated Endo and Simao's results [8], but some divergences were also detected. The SPY testing method enabled significant reduction on the overall cost of the test suites. On average, the SPY method reduced the test suite length to 61% compared to HSI, and 30% compared to W. The number of

test cases (NR) generated by the method SPY also represented 42% and 22% of the total number of test cases generated by W and HSI, respectively. The length of SPY test cases also increased 77% compared to traditional methods. The maximum test case length of the traditional methods did not vary along the test scenarios but, on the other hand, SPY test cases presented a maximum length 14% greater. In one case (*P04*), the maximum length of the SPY test cases increased 43%.

Only positive correlations were detected between test characteristics and the numbers of states and inputs, contradicting Endo and Simao's results [8] which showed that FSMs with more inputs tend to generate test cases with lower average length. Since $FSM(P)$ models have both numbers of states and inputs directly proportional to $|U| \times |R|$, test dimension will always tend to increase as long as the total number of *user-role* combinations increase, even if policies with many constraints are taken as SUT. The effectiveness of the test suites also corroborated Masood et al. [14] that found 100% of effectiveness and no different effectiveness was detected for any testing method. As conclusion, we have found that the SPY method can still generate 100% effective and shorter test suites, composed by longer and less test cases on the RBAC domain. Thus, SPY can significantly improve access control testing processes by reducing test costs and maintaining the same effectiveness.

6. THREATS TO VALIDITY

In this section, the observed threats to validity and limitations related to the process used in this research are presented.

External Validity: It concerns with the generalization of the outcomes to other scenarios. The representativeness of the policies may be an issue; nevertheless, the selection and modification of the policies was performed taking into consideration the inclusion all kinds of RBAC elements, as shown in Table 1. The RBAC elements related with hierarchical relationships were the only characteristics we did not consider in this study as no policy with this sort of constraint was found. Test characteristics may change on different policies, specially policies with greater number of users and roles. Policies with high number of *user-role* combinations can be still used at the cost of state explosion, but we believe they are more suitable for non-functional testing (e.g. scalability or performance), which was not the focus of this investigation.

Conclusion Validity: This category of threats to validity relates with the ability to draw correct conclusions about the relation between the treatment (e.g. test generation, policies under test) and the outcomes (e.g. test effectiveness and test characteristics). Test length and number of resets may increase on real world policies where more users, roles, and permissions are included. Although, the effectiveness may not change as long as transition and state coverage is guaranteed [14]. The whole test analysis was automatically performed using bash scripts and the RBAC-BT tool in order to mitigate mistakes during test execution, test coverage and characteristics analysis.

Internal Validity: Threats to internal validity are related with influences that can affect independent variables with respect to causality. The RBAC-BT tool was designed using the ANSI RBAC standard [3], conceptual models and a taxonomy proposed to this domain [4]. The *rbac2fsm* module was also verified and validated by using toy examples and tested considering early known properties such as *every FSM transition with denied transitions must be a self-loop*. The source code is also available to further assessments.

Construct Validity: Construct validity concerns with generalizing outcomes to the concept or theory behind the experiment. This experiment measured the test effectiveness using RBAC mutation analysis. Mutation analysis is a common assessment approach on software testing investigations [13], thus it can be used to emulate functional faults which may emerge during the design and development of RBAC mechanisms.

7. RELATED WORK

Previous experimental investigations applying FSM-based testing methods on RBAC have shown that complete FSMs specifying RBAC policies can be used to generate complete test suites [14]. However, the test suites generated from these FSMs tend to be very large. Alternatively, a set of test heuristics for reducing FSM models and a Constrained Random Test Selection (CRTS) approach were proposed in order to reduce the size of test suites. Despite degradating the test suite effectiveness when applied individually, the combination of the test suites generated by each proposed technique can still provide complete fault detection on RBAC fault domain. Due to the nature of the RBAC fault model, which tend to exhibit a single RBAC fault across many different transitions of an FSM, a probabilistic formal model was proposed for evaluating the fault coverage of heuristics and random approaches, such as CRTS [15]. Similar approaches were also proposed for testing a variation of the RBAC model which supports temporal constraints, the Temporal RBAC [16].

At the same time, there is a recurrent interest in comparing previously proposed FSM testing methods with new techniques. The paper introducing the SPY method compared SPY test suites to the HSI methods using randomly designed minimal FSMs with different configurations [21]. In Simao et al. [20], structural coverage criteria, such as state and transition coverage, were used to empirically compare random test generation methods in FSM testing domain. Dorofeeva et al. [7] compared the methods W, Wp, UIO, UIOv, DS, HSI, and H measuring their complexity, applicability, completeness, fault detection capability, test length and derivation time and, besides using random FSMs, it also included two specifications of realistic protocols as SUT. In Endo and Simao [8] the number of resets, the overall length of test suites and the average length of test cases generated by the W, HSI, H, SPY, and P methods were compared using 5200 random FSM models of different configurations. Despite the wide range of methods and the variety of FSMs, the presented studies essentially used random FSM models as SUT and, since the resemblance between random and real world models is not clear, their conclusions cannot be extended to specific domains, such as RBAC.

8. CONCLUSION AND FUTURE WORK

In this paper, we presented an experimental framework to investigate FSM-based test generation methods on RBAC. Two traditional (W and HSI) and one recent (SPY) FSM-based testing methods were applied on FSM models specifying RBAC policies and the obtained test suites were com-

pared. The generated test suites were evaluated based on the characteristics (number of resets, test case length, and test suite length) and the effectiveness of the test suites using the RBAC fault model.

The main contributions of this study are then twofold: the reanalysis of two investigations [8] [14] by replicating an experiment involving the evaluation of different FSM-based testing methods on RBAC domain, and the experimental protocol which can be used for replicating this study and investigating further stages of RBAC testing process, such as test prioritization and test selection [17].

Our results pointed out that the SPY method can reduce the total cost of test execution to approximately 32% and 46% regarding the number of resets and test suite overall length, respectively. Average test case length can be increased by 78% and positive correlations were found between all test and FSM characteristics. Our results also showed that when the number of users and roles, and consequently of inputs and states of the FSM models, increase, longer test cases tend to be generated, no matter what methods are used, contradicting some of Endo and Simao's results [8]. Additionally, the SPY method was able to significantly decreased the overall test suite length and number of resets without impacts on the effectiveness. As future work, we plan to replicate this study with more RBAC policies, extending the RBAC-BT tool with hierarchical RBAC support, and investigating novel test criteria for test selection and prioritization on the RBAC domain.

9. ACKNOWLEDGMENTS

Carlos Diego Nascimento Damasceno's research is supported by the National Council for Scientific and Technological Development (CNPq), process number 132249/2014-6.

10. REFERENCES

[1] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2008.

[2] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering*, 32(8):608–624, Aug 2006.

[3] ANSI. Role based access control, 2004. ANSI/INCITS 359-2004.

[4] A. Ben Fadhel, D. Bianculli, and L. Briand. A comprehensive modeling framework for role-based access control policies. *J. Syst. Softw.*, 107(C):110–126, Sept. 2015.

[5] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner. *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[6] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, May 1978.

[7] R. Dorofeeva, K. El-Fakih, S. Maag, A. R. Cavalli, and N. Yevtushenko. Fsm-based conformance testing methods: A survey annotated with experimental evaluation. *Information and Software Technology*, 52(12):1286 – 1297, 2010.

[8] A. T. Endo and A. Simao. Evaluating test suite characteristics, cost, and effectiveness of fsm-based testing methods. *Information and Software Technology*, 55(6):1045 – 1062, 2013.

[9] S. C. P. F. Fabbri, M. E. Delamaro, J. C. Maldonado, and P. C. Masiero. Mutation analysis testing for finite state machines. In *Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on*, pages 220–229, Nov 1994.

[10] D. F. Ferraiolo, R. D. Kuhn, and R. Chandramouli. *Role-Based Access Control, Second Edition*. Artech House, Inc., Norwood, MA, USA, 2007.

[11] A. Gill. *Introduction to the Theory of Finite State Machines*. McGraw-Hill, New York, 1962.

[12] J. Jang-Jaccard and S. Nepal. A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, 80(5):973 – 993, 2014. Special Issue on Dependable and Secure Computing The 9th IEEE International Conference on Dependable, Autonomic and Secure Computing.

[13] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, 37(5):649–678, Sept 2011.

[14] A. Masood, R. Bhatti, A. Ghafoor, and A. P. Mathur. Scalable and effective test generation for role-based access control systems. *IEEE Transactions on Software Engineering*, 35(5):654–668, Sept. 2009.

[15] A. Masood, A. Ghafoor, and A. P. Mathur. Fault coverage of constrained random test selection for access control: A formal analysis. *J. Syst. Softw.*, 83(12):2607–2617, Dec. 2010.

[16] M. Masood, A. Ghafoor, and A. Mathur. Conformance testing of temporal role-based access control systems. *IEEE Transactions on Dependable and Secure Computing*, 7(2):144–158, April 2010.

[17] T. Mouelhi, D. E. Kateb, and Y. L. Traon. Chapter five - inroads in testing access control. In A. Memon, editor, *Advances in Computers 99*, volume 99 of *Advances in Computers*, pages 195 – 222. Elsevier, 2015.

[18] A. Petrenko and G. V. Bochmann. Selecting test sequences for partially-specified nondeterministic finite state machines. In G. Luo, editor, *7th IFIP WG 6.1 International Workshop on Protocol Test Systems*, IWPTS '94, pages 95–110, London, UK, UK, 1995. Chapman & Hall, Ltd.

[19] P. Samarati and S. Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. Springer Berlin Heidelberg, 2001.

[20] A. Simao, A. Petrenko, and J. Maldonado. Comparing finite state machine test. *Software, IET*, 3(2):91–105, April 2009.

[21] A. Simão, A. Petrenko, and N. Yevtushenko. Generating reduced tests for fsms with extra states. In M. Nunez, P. Baker, and M. Merayo, editors, *Testing of Software and Communication Systems*, volume 5826 of *Lecture Notes in Computer Science*, pages 129–145. Springer Berlin Heidelberg, 2009.