Automated Test Generation for Access Control Policies

Evan Martin
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
eemartin@ncsu.edu

Tao Xie
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
xie@csc.ncsu.edu

Abstract

Access control policies are increasingly written in specification languages such as XACML. A dedicated software component called a Policy Decision Point (PDP) receives access requests, evaluates requests against specified policies, and returns responses to inform whether access should be granted. To increase confidence in the correctness of specified policies, policy developers can conduct policy testing to probe the PDP with some typical test inputs (in the form of requests) and check test outputs (in the form of responses) against expected ones. Unfortunately, manual test generation is tedious and manually generated tests are often not sufficient to exercise various policy behaviors. In this paper we present an efficient test generation approach and its supporting tool called Targen. We evaluate the approach on policies collected from various sources in terms of structural coverage and fault-detection capability. Our results show that Targen can effectively generate tests that outperforms the existing random test generation in terms of structural coverage and fault-detection capability.

1. Introduction

Access control mechanisms control which principals such as users and processes have access to which resources in a system. To facilitate managing access control, policy languages such as XACML have been increasingly used to specify access control policies for a system. Assuring the correctness of policy specifications is becoming an important and yet challenging task. Software testing aims at efficiently detecting errors in software through dynamic execution. Errors in policy specifications may also be discovered by leveraging existing techniques for software testing and applying them to policy testing. In policy testing, test inputs are access requests and test outputs are access responses. The execution of test inputs occurs as requests are evaluated by a Policy Decision Point (PDP) against the access control policies under test. Policy authors may then inspect request-response pairs to determine if the policy specification is correct. Access control policies are often tested with manually defined access requests so that policy authors may check the PDP's responses against expected ones. Unfortunately, current policy testing practice tends to be a manual, ad hoc process. With such a process, it is questionable that high confidence can be gained in the correctness of access control policies. To satisfy the need for generating high-coverage tests for complex real-world policies, in this paper we present an automatic test generation approach based on combinatorial coverage [1], implement the approach, and compare it with a random generation technique in terms of structural coverage and fault-detection capability.

2. Target-Driven Request Generation

To automatically generate high-coverage tests for access control policies, we develop a target driven approach (called Targen) that considers each rule in isolation and attempts to satisfy the constraints required for that rule to be applied. The policy under test is modeled as a hierarchy or tree structure. Each leaf in the tree represents a rule and each path from the root to the leaf contains a series of constraints. In particular, each target may have some number of attribute id-value pairs found in the subject, resource, and action sections of the target. We collect these attribute id-value pairs in three sets; one for each section of the target. Once a leaf is reached, we use these sets to form a predicate p out of $s + r + a$ independent clauses where s, r, and a correspond to the number of id-value pairs in the subject, resource, and action set, respectively. Each id-value pair maps to a specific clause in the predicate p. Furthermore, the clauses *within* sets are *or'ed* and each of the predicates formed by the sets are *and'ed*. For example, let the subject, resource, and action set for a particular rule be denoted by $S = \{s_1, s_2, s_3\}$, $R = \{r_1, r_2, r_3\}$, and $A = \{a_1, a_2, a_3\}$. The predicate corresponding to this rule is $p = (s_1 \vee s_2 \vee s_3) \wedge (r_1 \vee r_2 \vee r_3) \wedge (a_1 \vee a_2 \vee a_3)$. A

Table 1. Structural coverage and mutant-killing ratios achieved by tests generated using the random and Targen techniques.

	random				targen			
policy	pol %	rule %	cond %	mut kill %	pol %	rule %	cond %	mut kill %
simple-policy	100	50	-	37.04	100	100	-	44.44
demo-26	100	50	0	42.86	100	100	50	78.57
demo-5	100	100	75	73.68	100	100	75	78.95
demo-11	100	100	75	77.78	100	100	75	77.78
mod-fedora	100	50	50	13.33	100	100	100	56.67
codeA	100	50	-	25.45	100	100	-	36.36
default-2	100	100	75	16.92	100	100	100	50
conference	0	0	-	0	100	100	-	95.45
pluto	0	0	-	0	100	100	-	97.75
continue	100	17	-	10	100	100	-	43.10
average	80	51.7	55	29.71	100	100	80	65.91

request set is generated that satisfies all possible combinations of truth values for each independent clause. Therefore, a predicate with n independent clauses has 2^n possible assignments and so at most 2^n requests are generated for each rule.

3. Evaluation

We evaluate the effectiveness of our new Targen approach by comparing it to an existing random test generation technique [4] in terms of policy structural coverage and fault-detection capability. We use a policy coverage measurement tool [4] to obtain a measure of policy structural coverage and an automated mutation testing framework [3] to obtain a measure of fault-detection capability.

We use 10 XACML policies collected from four different sources[1] [2, 5] in our evaluation. The results summarized in Table 1 show that Targen exhibits better or equivalent performance for each coverage metric across all policies, where pol%, rule%, cond%, and mut kill% denote three structural coverage [4] (policy coverage, rule coverage, condition coverage) and mutant killing ratios [3], respectively. Although some of the simpler policies demonstrate the improvement, the benefits of Targen are most apparent on the more complex policies such as conference, pluto, and continue. The random technique performs worse when the set of attribute values is large, because the probability of randomly generating requests that cover existing rules is much less. In terms of fault-detection capability we observe that Targen has an average mutant-killing ratio of 65.9% whereas the random technique has only an average mutant-killing ratio of 28.71%. The results indicate that Targen can be used to generate tests with higher fault-detection capability than the random technique. Unfortunately the mutant-killing ratios are still low even for the Targen-generated requests. One possible explanation is that the results are skewed because of a large number of equivalent mutants. Our current policy structural coverage corresponds to statement or branch coverage in program testing. We expect that we can achieve higher fault-detection capability when adopting stronger policy structural coverage criteria such as one that corresponds to path coverage in program testing. We plan to explore this direction in future work.

[1] `http://archon.cs.odu.edu/`, `http://www.fedora.info`

References

[1] P. Ammann, J. Offutt, and H. Huang. Coverage criteria for logical expressions. In *Proc. 14th International Symposium on Software Reliability Engineering*, pages 99–107, 2003.

[2] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proc. 27th International Conference on Software Engineering*, pages 196–205, 2005.

[3] E. Martin and T. Xie. Automated mutation testing of access control policies. Technical Report TR-2006-12, Department of Computer Science, North Carolina State University, Raleigh, North Carolina, 2006.

[4] E. Martin, T. Xie, and T. Yu. Defining and measuring policy coverage in testing access control policies. In *Proc. 8th International Conference on Information and Communications Security*, 2006.

[5] N. Zhang, M. Ryan, and D. P. Guelev. Synthesising verified access control systems in XACML. In *Proc. 2004 ACM Workshop on Formal Methods in Security Engineering*, pages 56–65, 2004.