Inferring Access-Control Policy Properties via Machine Learning

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

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

Abstract

To ease the burden of implementing and maintaining access-control aspects in a system, a growing trend among developers is to write access-control policies in a specification language such as XACML and integrate the policies with applications through the use of a Policy Decision Point (PDP). To assure that the specified polices reflect the expected ones, recent research has developed policy verification tools; however, their applications in practice are still limited, being constrained by the limited set of supported policy language features and the unavailability of policy properties. This paper presents a data-mining approach to the problem of verifying that expressed access-control policies reflect the true desires of the policy author. We developed a tool to investigate this approach by automatically generating requests, evaluating those requests to get responses, and applying machine learning on the request-response pairs to infer policy properties. These inferred properties facilitate the inspection of the policy behavior. We applied our tool on an access-control policy of a central grades repository system for a university. Our results show that machine learning algorithms can provide valuable insight into basic policy properties and help identify specific bug-exposing requests.

1. Introduction

Access-control policies are used to govern the various types of access that different entities may have to information. As a result of the complexity introduced by hard coding policies into programs [3], an increasing trend is to define policies in a standardized specification language such as XACML [6] and integrate the policies with applications through the use of a Policy Decision Point (PDP). A PDP is a software component that receives an access request from a Policy Enforcement Point (PEP) and returns a response instructing the PEP as to whether access should be permitted or denied. This approach allows policies to be reasoned about independently from the software that enforces them.

As the number and complexity of access-control policies grow, the effort required for their implementation and maintenance is also growing. Implementing and maintaining policies are issues that must be addressed. Specifying policies in standardized languages as opposed to hard coding them into applications is a step in the right direction. But unfortunately a new question arises: is there a discrepancy between the policy specification and its intended function? Standardized policy specifications are more formal than their natural language equivalents and may be error-prone without sufficient tool support. Furthermore, correct implementation of policies by using applications such as a PDP is based on the premise that the policy specification is correct. As a result, policy specifications must undergo rigorous verification and validation to ensure the expressed policy truly encapsulates the desires of the policy authors.

Our primary objective is to efficiently identify discrepancies between the policy specification and its intended function. We propose a data-mining approach to help users identify specific requests that may illustrate these discrepancies. By probing a policy with requests and observing its behavior, we can use machine learning algorithms to infer properties of the policy. Any request that violates the inferred properties represents a special case in which the PDP receiving the request produces a response that deviates from the policy's normal behavior. We developed a tool that implements this approach through automatic request generation and the use of machine learning algorithms to infer properties of policies from request-response pairs. We applied our tool on an access-control policy of a central grades repository system for a university based on an example used by Fisler et al. [3]. Our preliminary results show that machine learning algorithms can provide valuable insight into basic policy properties and help identify specific bug-exposing requests.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the use

of machine learning to infer statistically true properties of a policy and describes the tool we developed. Section 4 discusses the results of applying our tool and Section 5 concludes with future work.

2. Related Work

Policy verification is used to formally check general properties of access control policies. The verification problem can become intractable as policies become more complex. Recently several tools have been developed to verify specific properties of a given XACML policy [6]. For example, Hughes and Bultan [4] translated XACML policies to the Alloy language [5] and checked their properties using the Alloy Analyzer. Fisler et al. [3] developed a tool called Margrave that verifies user-specified properties and performs change-impact analysis. Zhang et al. [10] developed a model-checking algorithm and tool support to evaluate access control policies written in RW languages, which can be converted to XACML [9]. These approaches support only a subset of the XACML policy specification language because it is challenging to generalize these verification approaches to support full-featured XACML policies with complex conditions. Some of these approaches also require the user to specify a set of properties in some formal language to be verified; however, these formally specified properties often do not exist in practice. The approach to policy property inference proposed in this paper operates solely on request-response pairs and thus may be applied to any type of access control policy. Furthermore, the limitations are determined by the PDP used to evaluate the requests against the policy. The PDP used by our tool supports all the mandatory features of XACML, all of the standard attribute types, functions, and combining algorithms as well as a number of optional features. Policy property inference is complementary to the preceding property verification tools [3, 4, 10] by supplying their inputs previously manually supplied by the user.

Burgess [1] presents an approach to anomaly detection that combines policies and machine learning implemented in a suite of tools known as cfengine[1]. Cfengine is a policy-based monitoring and configuration management system used to configure the files and processes running on networked computers. A policy in cfengine summarizes the expected behavior of the system but the exact behavior is not enforceable. Data mining is just one technique used in concert with several others to derive a statistical model of the exact observed behavior of the system. Anomalies are detected when this model violates the policy. Conversely, in our approach, the expected behavior resides only in the mind of the policy author while the exact behavior is dictated by the policy. We also model exact behavior using a statistical model and use this model to detect anomalies. In our case, anomalies are potential bugs and they occur when the statistical model deviates from the exact behavior.

[1] `http://www.cfengine.org/`

3. Approach

Our primary objective is to efficiently identify discrepancies between the policy specification and the true desires of the policy authors. We help a user identify these discrepancies or bugs by finding specific requests that are likely bug-exposing. We first observe the policy's behavior by probing it with several automatically generated requests. These observations are used as input, in the form of request-response pairs, to a particular class of machine learning algorithms called classification learning. The output of the machine learning algorithms is essentially a summary of the policy in the form of inferred properties that may *not* be true for all requests but are true for *most* requests. We do not wish to recreate the policy in its entirety through these inferred properties but merely capture the general policy behavior in order to help identify special cases. The rationale is that the policy specification is mostly correct and that the bug-exposing requests represent a small percentage. Under this rationale, any request that violates the inferred properties are special cases or requests that result in responses that deviate from the policy's normal behavior. These special cases are identified as being likely bug-exposing and warrant manual inspection. We have integrated Sun's XACML implementation [7] and a collection of machine learning algorithms for data mining tasks [8] into a tool that implements our approach through request generation, request evaluation, and policy property inference.

3.1. Request Generation

Our tool supports two different techniques of supplying requests to the system. The first technique is simply identifying existing XACML request documents via a standard file choosing dialog. The second technique inspects the specified policy and constructs a request factory that generates requests on demand. There are various algorithms that can be devised to generate requests. In our current tool, we have implemented two simple factories called the `AllComboReqFactory` and the `UnifRandomReqFactory`. The former attempts to generate requests for all possible combinations of subjects, resources, and actions while the latter randomly selects elements from the subject, resource, and action set following a uniform distribution. The `AllComboReqFactory` is only possible if the set of subjects, resources, and actions are finite and can be enumerated. The example policy used in Section 4 is a simple role based access control policy used for a central grades repository at a university. The policy

has three subjects (Student, Faculty, TA); two resources (InternalGrades, ExternalGrades); and two actions (Receive, Assign). We generate all possible combinations by incrementing an integer i from 0 to 2^{s+r+a} where s, r, and a are the number of possible subjects, resources, and actions, respectively. To construct a request from the integer i, we first convert i to binary and use the $s + r + a$ least significant bits as a set of boolean flags to indicate the presence or absence of the possible attributes for subject, resource, and action. This approach guarantees that all possible combinations of the available attributes are generated. However, this approach is a simplistic one and it is not realistic for larger policies or policies in which the set of subjects, resources, and actions are not all finite. Furthermore, generating all possible combinations of subjects, resources, and actions does not necessarily result in a valid request. As a result, we find that many of the generated requests evaluate to `Indeterminate` or `NotApplicable`. We plan to develop sophisticated approaches for request generation in future work (Section 5).

3.2. Request Evaluation

Request evaluation or response generation is simply the evaluation of a request against the specified policy. Sun's XACML implementation [7] provides an API to implement a PDP, which receives an access request and returns an access decision. As requests are generated by a request factory, the PDP is used to evaluate the request to produce a response.

3.3. Property Inference

We infer policy properties by applying machine learning on request-response pairs. Our current tool leverages Weka [8], a collection of machine learning algorithms for data mining tasks. Weka contains tools for pre-processing, classification, regression, clustering, association rules, and visualization. In general, data mining is defined as the process of discovering patterns in data such as explicit knowledge structures (i.e., structural descriptions) [8].

In our research context, we are mostly interested in the knowledge structures acquired as a mechanism to infer general and not necessarily universally true properties of the policy. Machine learning techniques are frequently used to gain insight into the structure of their data rather than to make predictions for new cases [8]. We use knowledge structures generated from a genre of machine learning algorithms called *classification learning* to summarize the results of request-response pairs thereby expressing the policy in a different and often more concise way.

As requests are evaluated against the policy, our tool appends relevant information about the request-response pairs

```
1.1 If Faculty = 1
    and (Receive = 1 or Assign = 1)
    and (ExternalGrades = 1 or InternalGrades = 1)
    then Permit
1.2 If Student = 1
    and Receive = 1
    and ExternalGrades = 1 then Permit
1.3 If Student = 1
    and Assign = 1
    and ExternalGrades = 1 then Deny
1.4 If True then Deny
```

Figure 1. Rules in the actual XACML policy.

```
2.1 If Faculty = 1 then Permit
2.2 If Student = 1
    and InternalGrades = 0
    and Receive = 1 then Permit
2.3 If TA = 1 then Deny
2.4 If Student = 1
    and InternalGrades = 1 then Deny
2.5 If Student = 1
    and Receive = 0 then Deny
```

Figure 2. Rules generated by the Prism classification algorithm on the partial data set.

to a data file in a particular format being used as training data for Weka [8]. Weka mines the request-response pairs to find and describe structural patterns. These structural patterns are described in the form of rules or properties that are simple conditional expressions or properties that classify requests into four response types: `Permit`, `Deny`, `NotApplicable`, and `Indeterminate`. These rules are useful for manual inspection and for identifying corner cases. Because the rules produced by the classification learning algorithms are statistically true, it is likely of interest to the user to inspect the requests that violate those rules. If violating requests exist in the training data, then they are identified by Weka as misclassified instances. If no violating request has been generated by the request factory, then it is possible to translate the rule into a property and use an existing property verification tool [3, 4, 10] to generate a request or set of requests that violate the inferred property.

4. Preliminary Results

We applied our tool on an access control policy of a central grades repository system for a university, which was earlier used by Fisler et al. [3] to illustrate policy verification. Because the policy defines small, finite sets of subjects, resources, and actions, we use the `AllComboReqFactory` to generate the entire set of possible requests. With 3 subjects, 2 resources, and 2 actions, the request factory generated $2^7 = 128$ different requests. Unfortunately 56 of the combinations produced invalid requests that resulted in `Indeterminate` responses, 54 evaluated to `NotApplicable` responses, 10 evaluated to `Deny` responses, and 8 evaluated to `Permit` responses.

We used the Prism classification algorithm [2] to generate rule sets using two different sets of training data.

The first set used the output from all 128 requests. In the second set we removed all instances with the response of `Indeterminate` or `NotApplicable`. Although the performance of the partial data set appears similar to that of the full data set, the number of rules is smaller and more relevant for the partial data set. The full data set produces 30 rules whereas the partial data set produces the 5 rules shown in Figure 2.

For comparison purposes, we have translated the rules in the actual XACML policy to the form shown in Figure 1. By comparing Figure 1 and Figure 2, we see that the inferred properties do indeed summarize the policy. However, because there are misclassified instances, we know that the inferred properties are *not* universally true. Note that Rules 1.1 and 1.2 are equivalent to Rules 2.1 and 2.2, respectively. Rules 2.4 and 2.5 are intuitively correct because a student should not have access to internal grades or have assign permissions for any resource.

An error in the policy specification was discovered after we investigated the misclassified requests. Recall that those misclassified requests represent instances in which the policy produces responses that are inconsistent with the responses of similar requests. The rationale is that these special cases are likely bug-exposing requests. The request in question is one in which a `Student` wishes to `Receive` and `Assign` the resource of `ExternalGrades`. The classification model *correctly* classifies the response as `Deny` but the policy evaluates the request to `Permit`. The policy authors did not intend for a student to have permissions to assign their own grade as shown by Rule 1.3. This error is the same discrepancy found by Fisler et al. [3], which is a result of a subtlety of the XACML language. The root cause of the problem is that XACML allows an arbitrary number of values for a given attribute. This example illustrates that the investigation of misclassified requests can lead to the discovery of errors in policy specifications.

This simple example has shown that even with a small number of request-response pairs, machine learning can be a valuable tool for discovering and summarizing the basic properties of a policy. We suspect that its value and power will increase as the complexity and size of the policy grows because the inferred properties can summarize and aggregate the complex rules specified in the expressed policy. Further experimentation with a broader range of classification algorithms on large, complex policies is still required in future work to further assess the approach.

5. Conclusion

We have developed a tool that helps users identify requests that may reveal errors in policy specifications by integrating several components including request generation, request evaluation, and machine learning algorithms to infer policy properties from request-response pairs. We have applied our tool on an access control policy of a central grades repository system for a university. Our results show that applying machine learning algorithms can provide valuable insight into basic policy properties and help identify specific fault-revealing requests.

In future work we plan to perform experiments on much larger and more complex policies to further evaluate the usefulness of our request generation and machine learning techniques to identify discrepancies between a policy's specification and its intended function. We have only scratched the surface of what Weka is capable of and we need to further explore its data processing, classification, and visualization capabilities as well as many parameters and statistical measures used to tune and evaluate various machine learning algorithms. We also plan to investigate alternative techniques of request generation, for example, by using property verification tools such as Margrave. We also plan to use Margrave's counter-example generation feature to provide requests that do not follow the general policy properties inferred by the classification learning algorithms.

References

[1] M. Burgess. Probabilistic anomaly detection in distributed computer networks. *Science of Computer Programming*, 60(1):1–26, 2006.

[2] J. Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.

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

[4] G. Hughes and T. Bultan. Automated verification of access control policies. Technical Report 2004-22, Department of Computer Science, University of California, Santa Barbara, 2004.

[5] D. Jackson, I. Shlyakhter, and M. Sridharan. A micromodularity mechanism. In *Proc. 8th ESEC/FSE*, pages 62–73, 2001.

[6] OASIS. OASIS eXtensible Access Markup Language (XACML). Published Standard, 2005.

[7] Sun Microsystems. Sun's XACML Implementation. Sourceforge, 2005.

[8] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[9] 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.

[10] N. Zhang, M. Ryan, and D. P. Guelev. Evaluating access control policies through model checking. In *Proc. 8th International Conference on Information Security*, pages 446–460, September 2005.