A Lean Formalization of Cedar

Cedar is an open source authorization policy language, developed at Amazon Web Services (AWS). Cedar allows for controlled access to resources via a simple and expressive syntax that supports different authorization paradigms, such as attribute-based access control (ABAC) and role-based access control (RBAC). Cedar policies define who (the principal) can do what (the action) on what target (the resource) when (the context). Policies have a specified effect: either permit or forbid.

Dafny is a verification-aware programming language. Dafny makes use of automated reasoning, allowing programmers to reason about their code formally by making use of specifications. Dafny discharges proof obligations to an SMT solver, Z3, allowing additional pre and post conditions and assertions to assist the solver.

Dafny is used for such a set of policies, and then make an authorization request. The request is either allowed or denied, based on a set of rules. Informally, an authorization request could be of the form “Is Alex allowed to create the admin page Instructions?”, which will be allowed if and only if Alex is in the group admins and not in the group blacklisted_admins.

VERIFICATION GUIDED DEVELOPMENT

Cedar uses a process called verification-guided development, to ensure the correctness of the authorization engine. The authorizer and validator are modeled in Dafny, and using Dafny’s automated reasoning capabilities, a collection of security properties are checked and proved. Via differential random testing (DRT), the production implementation in Cedar is checked for equivalence with the Dafny model. In Cedar, 25 bugs have been found through DRT, and 4 bugs through failed proof attempts.

VALIDATION MODEL

In the existing Dafny formalization, Cedar validation followed a slightly complex model known as permission validator. During the Lean formalization, we decided to change the validation model to be stricter, by simplifying the type system quite a bit. The only non-standard part of the type system is to do with booleans: when we are able to make certain judgements, we type boolean values strongly with tt and ff representing the true and false types, over and above the regular anyBool type which corresponds to the more familiar boolean type.

AN ALTERNATIVE VERIFICATION ENVIRONMENT

Dafny was chosen for its balance between usability and automation for basic properties. However, meta-theoretic properties of Cedar have proved less suitable for Dafny’s automation. The specification suffered from poor proof performance and brittleness, where small changes to the program, or minor updates to Dafny or Z3 caused verification timeouts. Proof brittleness is a well known issue with SMT-based tools such as Dafny. To ensure robust performance and minimal maintenance, highly detailed proofs are better, and this favors the use of an interactive theorem prover over an automated one. We port the Cedar formalization to the interactive theorem prover Lean, and try to answer the question: Can Lean be used for verifying a project at the scale of Cedar, with performance and proof size metrics comparable to the existing formalization in Dafny?

The Lean specification outperforms the Dafny specification – this can be attributed to the use of type classes and higher order functions, as well as Lean’s extensive standard library. Dafny proofs are mostly shorter than Lean proofs – this was expected, and is attributed to Dafny’s ability to automatically solve simple proof obligations via an SMT solver. Both verification time and time per test request for Lean were significantly lower than Dafny, which can be attributed to the difference in underlying compilers. Overall, the Lean specification performs extremely well in all regards and is a significant improvement especially with respect to the differential random testing that Cedar relies on, as more tests can be run in the daily fixed period.

We prove the following theorems for the authorizer:

- If some forbid policy is satisfied, then the request is denied.
- A request is allowed if and only if it is explicitly permitted (i.e., there is at least one permit policy that is satisfied).
- Authorization produces the same result regardless of policy evaluation order or duplicates.

The Lean specification outperforms the Dafny specification – this can be attributed to the use of type classes and higher order functions, as well as Lean’s extensive standard library. Dafny proofs are mostly shorter than Lean proofs – this was expected, and is attributed to Dafny’s ability to automatically solve simple proof obligations via an SMT solver. Both verification time and time per test request for Lean were significantly lower than Dafny, which can be attributed to the difference in underlying compilers. Overall, the Lean specification performs extremely well in all regards and is a significant improvement especially with respect to the differential random testing that Cedar relies on, as more tests can be run in the daily fixed period.

AN ASIDE: MODELING SETS

Cedar makes heavy use of sets, and in Dafny, sets are axiomatized in Lean, this is not the case. We have a type Value where set : Set Value -> Value, that is, we use a constructor that takes in a set of values as a parameter. To define a type representing sets on Value, we would need to define a function and a type in a manually recursive fashion, which is not permitted in Lean. Hence, we had to settle on an alternative definition: a set is a wrapper around a list, but we only deal with well-formed sets, that is, sets where the underlying list is sorted and duplicate free.