CSCI 2390 Project Proposal

Archer Wheeler
Brown University

1 Project Idea

My project idea is to build a privacy enforcing dataflow toolkit using a strong statically typed language such as Haskell. My toolkit will seek to emulate a framework like Resin and operate under a weak non malicious threat model. My goal is to build a system that will statically enforce policy based "best practices" and assume that the application programmer actually writes correct and secure policies.

One of the advantages of using a type system in a strongly typed language is that I will not have to deal with any modifications of the runtime. Additionally, statically enforcing privacy has the benefit of less overhead and the challenge of doing something different from what we've read in class.

2 Background

Haskell is a strongly typed language which can make very strong guarantees about the side effects of code. For example, Haskell provides an "IO" type which effectively taints all code capable of running with arbitrary side effects. Every function which does not evaluate to an IO datatype is guaranteed to have no side effects. The type IO is an example of more general, and idiomatically used, typeclass called a Monad. Monads can be used for abstraction, but also to enforce that enclosed datatypes can only be used in specific ways. I plan to create a policy Monad which tracks data and enforces that a corresponding policy is run on it.

I have found at least one similar existing Haskell library called LMonad which provides tools for IFC programming. This library appears to be very low level and focused on giving the tools need for abstract IFC based rules. I do not intend to provide the same IFC based guarantees.

3 Project Goals

I expect the biggest danger of my project is making sure I don't commit to building something significantly more complicated and involved than I realize right now. I do not expect to build a system as robust or feature complete as Resin, but I hope to instead build something small, useful and that could be built upon.

My plan is to build a policy Monad which enforces that a corresponding policy is called on the enclosed data before it is returned to the user. Sensitive data will be originated through some api such as a database query or file system wrapper. At the top level I'll provide a "SafeServer" type which can only accept and return validated policy data. As long as the programer sets up the SafeServer boiler plate and uses it to interact with the outside world, the privacy policy will be enforced. The programmer will of course have to write their own effective policy code depending on the use case.

Since I'm not modifying the runtime, it will be difficult (if not impossible) to guarantee that the programer doesn't make unsafe calls to the database or send unchecked information out of the server through other mechanisms. However, I can provide a setup boiler plate which requires the app conforms to a specifically type. As long as the programer uses this boiler plate they'll be constrained by Haskell's powerful guarantees about side effects.

In addition to writing a small framework, I will use the framework to build a small example application.

4 Proposed Timeline

- Build a lightweight policy datatype which encapsulates sensitive data and enforces use of a specific policy function provided by the application programer.
- Build very simple safe input and output sources. I'd like to avoid dealing with complications arising from using various server frameworks and databases. Because of this, I'll start by only providing a filesystem read as a stand in for a database. Additionally, I could limit outputs to very simple text based GET queries. The user token will be provided directly by the user through a query parameter.

- Build a toy example application. The application should solve a similar problem to that of the Paymaxx example from class.
- Given time, expand the functionality of the framework.

5 Non Goals

- Provide strong guarantees on privacy. Since I'm not baking any code into the runtime, it will certainly be possible for an application developer to "use it wrong." Additionally, I don't want to focus significant work on showing that my core library code is bullet proof. While I obviously intend to write tests and build an effective framework, I don't intend to be able to make strong guarantees about it.
- Provide built in policy code for general use cases. I expect that the core framework will essentially force the application programer to roll their own effective privacy policies. I will of course have to use my own framework to build the toy proof of concept.

6 Additional Thoughts

While Haskell is fairly niche, I do have a good amount of experience with it. Since the framework will probably not end up being very fancy, I'm not all that worried that I won't be able to reach as wide as an audience as a python framework could.

I could also probably use Scala for the framework rather than Haskell if need be. Scala provides similar static type powers, but doesn't have the same nice IO guarantees of Haskell. While I have a lot of Scala experience, this kind of type pattern is far less idiomatic in Scala than it is in Haskell.