APrICoT Project Proposal
Inferring Policies in Buggy Code

For more information and biweekly updates, see apricot-soda.herokuapp.com

Project Description

I will be working with Professor Jean Yang of CMU's Principles of Programming group, and her Ph.D. student, Allison Kao. We plan to make a static analysis tool that will automatically repair code that has privacy leaks in the program's information flow. This tool should work with existing code that may contain bugs, and it should not add any extra runtime cost (except for the repaired policies). We hope to accomplish this by first developing inference techniques to infer missing policies in potentially buggy code, and then repairing them automatically using program synthesis tools.

Approach

Jean has already produced a solution for this problem that works dynamically. However, the cost associated with its overhead is enough to make it inviable for a real production environment. Since then, she has started to approach this problem from a static viewpoint. To that end, she (and her collaborators) created Lifty, a programming language that uses liquid types to enforce policies automatically at compile-time. A type error caused by leaky policies is fixed by a program synthesis tool named Synquid, which fills in the missing checks.

However, a new language only shows that the problem *can* be solved, but is not necessarily the solution to the problem. APrICoT aims to extend the techniques developed in the design of Lifty so that it can work with existing code. While programmers have to explicitly annotate their sensitive data values in Lifty, we hope that APrICoT will be able to infer which data values are sensitive, and the policies that protect them. We will begin by writing case studies in LiquidHaskell, including a conference management system for SigBovik, to establish target benchmarks for correct implementations. LiquidHaskell's type system is very strong, so it provides a good starting point for working on the inference techniques. We will also be writing buggy implementations to test the policy inference.

The next step is to adapt the current inference techniques in Lifty to LiquidHaskell by extending the LiquidHaskell compiler. After this, we will begin work on inferring which pieces of data may be sensitive, even if they are not explicitly marked. Finally, we will design a *probabilistic type system* for inferring policy checks from possibly buggy code.

Impact

Our high level goal is to create a world where programmers do not have to worry about the security and privacy of their users. In a large amount of cases, this is not even on the programmer's mind when they are developing their applications. Privacy leaks in information flow is also especially difficult to reason about, as it requires knowing everywhere any piece of data can travel throughout the lifecycle of an application. As well, there are sometimes implicit

leaks that can reveal information to a clever observer. By creating a tool that will automatically insert policy checks for the programmer, we can make the machine responsible for enforcing privacy. As well, a tool that works with existing code has the possibility of repairing code that is already being widely used.

Goals

At the very least, we aim to have a solid set of benchmarks for which to aim, and ideas of what does and does not work for policy inference. If things go according to plan, we hope to have the solid set of benchmarks in addition to working policy inference for LiquidHaskell. If we are extremely productive, we hope to have the solid set of benchmarks, working inference, and a plan for picking policies from a distribution for buggy code. We will be able to measure our success by (1) how accurately we can infer sensitive data values, (2) how accurately we can infer policies, and (3) how much added resource usage the program repair introduces.

Resources Needed

For this project, I'll need to install Haskell, the LiquidHaskell compiler, and Yesod, a web framework for Haskell. I will also need to install Synquid. All of these resources are available at no cost, and no further resources will be needed.

Literature

I expect this list to grow over time, but to start I plan to read the following papers:

- General Background
    - A. Sabelfeld and A. C. Myers. Language-based information-flow security. IEEE Journal on Selected Areas in Communications, 21(1), 2003.
    - F. Pottier and V. Simonet. Information flow inference for ML. ACM Transactions on Programming Languages and Systems, 25(1), Jan. 2003.
- Refinement Types & Information Flow
    - http://research.microsoft.com/~nswamy/papers/fine-esop.pdf
- Policy Agnostic Programming
    - T. H. Austin, J. Yang, C. Flanagan, and A. Solar-Lezama. Faceted execution of policy-agnostic programs. Proceedings of the Eighth ACM SIGPLAN workshop on Programming languages and analysis for security - PLAS '13, 2013
    - J. Yang, K. Yessenov, and A. Solar-Lezama. A language for automatically enforcing privacy policies, 2012.
- Retrofitting Security Policies
    - Benjamin Livshits, Aditya V. Nori, Sriram K. Rajamani, and Anindya Banerjee. Merlin: Specification inference for explicit information flow problems. In PLDI, 2009.
    - Vinod Ganapathy, Trent Jaeger, and Somesh Jha. Retrofitting legacy code for authorization policy enforcement. In Oakland, May 2006.
    - Vinod Ganapathy, Trent Jaeger, and Somesh Jha. Towards automated authorization policy enforcement. In Proceedings of the 2006 Security-Enhanced Linux Workshop, pages 7–11, March 2006.

Timeline

11/8/2016	● Finish project proposal ● Set up website to track my progress/view my progress reports ● Go over LIFTY slides and paper once more
11/15/2016	● Finish Haskell Tutorial ● Read: ○ Specification inference for explicit information flow problems ○ Retrofitting legacy code for authorization policy enforcement ○ Towards automated authorization policy enforcement
11/22/2016	● Set up HotCRP and get comfortable with it ● Learn a Haskell Web Framework (Yesod) ● At this point, I should consider if designing a web Framework for LiquidHaskell is something I might want to do. ● Reach out to Ranjit about frontend work. ● Begin working on first case study (Conference management system)
11/29/2016	● Become familiar with Synquid ● Learn LiquidHaskell ● Keep working on first case study
	● Become comfortable with Lifty
End of semester	● Finish SigBovik conference management system ● Begin thinking about possible incorrect implementations
1/30/2017	● Read probabilistic type system papers ● Write buggy implementations for case studies
2/13/2017	● Develop mechanism to identify sensitive values (needed before being able to identify policies)
2/27/2017	● Work on extending refinement type system with probabilities
3/20/2017	● Preliminary implementation of inference algorithms for buggy programs
4/3/2017	● Extend examples and iterate
4/17/2017	● Preliminary implementation of probabilistic inference
5/1/2017	● Characterize guarantees and performance of the system