CS 260r Final Project Proposal: Verifying Information Confidentiality under Optimization in HotCRP

Richard Cho and Dan Fu

I. Introduction

HotCRP is a web-based conference submission and review system [1], [2]. One of its primary features is a strong search capability: program committee members can search for papers by title, authors, decision, and other relevant fields. With such a search capability comes a number of issues with information flow, however. For example, program committee members may themselves submit papers to the conference; in such cases, they should not be allowed to read reviews or see decisions about their paper before de-anonymization.

Such information flow issues are compounded by attempts at query optimization. In particular, information confidentiality is enforced at the level of the PHP server, but it is desirable to move query burden from the PHP server to the database. If done without care, such query optimization can result in information leakage. Consider an example of a user searching for all papers that do not have a positive "accept" decision, for example. The user should receive a list of all papers that are not written by the user and have not been accepted, and all the papers written by the user, *regardless of whether the paper has been accepted or not*. A naive optimization might move the entire query to the SQL layer and return a list of all papers that have not been accepted. By the time a server-level information policy has been applied to this list, it is too late: the user will be able to deduce which of their papers have been accepted by the absence of such papers from the returned list.

In this project, we wish to formally verify that query optimization in the HotCRP system obeys information confidentiality policies. We will model the database, information policy, and user queries of HotCRP in the Coq interactive proof assistant [3] and show that optimized and un-optimized versions of the same user query return the same list of papers, with the same information. We have yet to name our project, but potential candidates include HotCRP+Coq, CoqCRP, and HotCoq.

II. Approach

A. Capstone Theorem

Let DB be a database of papers, Q_S an SQL query on the database, P a policy filter, u a user, and Q_U a user query. Q_S is a function on DB and returns a list of papers. P takes a list of papers and a user, and returns another list, potentially shorter or with information about certain papers scrubbed out. Q_U takes a list of papers and further filters it based on the user's request. The final list of papers that a user gets from a user query is thus

$$Q_U(P(u, Q_S(DB)))$$

Now, we can formally define the theorem we wish to prove. Let $Q^*(\cdot)$ be a SQL query that returns all the papers in DB. Let O be an optimization function that takes a policy, a user, and a user query, and returns a tuple containing an optimized user query and SQL query. For a given policy P, user u, and user query Q_u, let O_S^{P,u,Q_u} denote the optimized SQL query, and O_U^{P,u,Q_u} denote the optimized user query. We wish to prove the following theorem about O, HotCRP's user query optimization function:

$$\begin{aligned}\forall\ u, Q_U, P, DB:\ & Q_U(P(u, Q^*(DB))) = \\ & O_U^{P,u,Q_u}(P(u, O_Q^{P,u,Q_u}(DB)))\end{aligned} \tag{1}$$

In prose, the HotCRP optimization function splits a user query into a user query and SQL query. Eq. (1) states that for any user query, user, policy, and database, the optimized user query and SQL query return the same list of papers as the original user query applied with the Q^* SQL query.

B. Design

The statement of (1) suggests a fairly straightforward architecture: we need models of papers, the database, SQL queries, users, user queries, and information policies. For simplicity, we will model the database as a list of papers. The complexity of user-paper interactions will be captured in the fields given to papers and users: each paper will have an author list (a list of users), as well as a team. A team is an abstraction for various complex user groups relevant to the conference context - it can model a group of people with conflicts of interest, the group of people reviewing a paper, or many other similar groups. A user has a list of teams that she belongs to.

As our project develops, we may add more fields to the definition of team or user; such fields can represent abstractions such as subject, paper tag, or subfield for papers, or special privileges such as editorship or field expertise for users.

C. Stages

We divide our project into stages to make it easier to make progress:

- Stage 1: Basic user queries, no optimization. For this stage, we substantiate all the models in our design. For simplicity, we let user queries be the same as SQL queries, and we define the policy filter as a map that is applied to a list of papers to scrub out sensitive data. Here, the optimization function is the identity.
- Stage 2: Basic optimization. For this stage, we define an optimization function hard-coded for the policy we have written. We prove that this optimization function satisfies (1) (but only for the single policy).
- Stage 3: Policy generalization. For this stage, we define a generalized model of information policies and define a function to convert from the generalized model to a usable map. We construct optimization functions for certain policies and prove those correct.
- Stage 4: General optimization. Finally, we define an optimization function that can optimize any policy and prove that it satisfies (1). We will also prove that this optimization function is not just the identity (potentially via counterexample).

Our general timeline is as follows: we have already completed stage 1. We wish to complete stage two by a week from now (though that will mostly be Richard's work because Dan has a paper deadline). We wish to finish stage 3 by halfway through reading period, and stage 4 by a few days past the end of reading period (in time to turn the final project in).

D. Division of Labor

Dan did most of stage 1, Richard will do most of stage 2. Dan and Richard will design stages 3 and 4 on white board/pen and paper. Dan will code up the Coq definition of general policies for stage 3 and write the optimization functions for that stage; at the same time, Richard will write the generalized optimization function per the plan we came up with. Dan and Richard will help each other figure out how to prove stage 4.

III. Failure Cases

Stages 1 and 2 should go off without a hitch. Stage 3 is currently slated to be started before stage 4, but finished up concurrently. If we end up not being able to do stage 4, we will at least have stage 3 and the examples from stage 3 to fall back on. We suspect that we will be able to finish stage 4, however.

IV. Future Work

Since we are working on a model of the HotCRP system, we are not actually proving that the system that is running in the wild is correct. One future avenue of work is taking our generalized policy definition and optimization function and exporting them to PHP/SQL. The author of HotCRP might then incorporate our exported code into the actual version running in the wild.

Alternatively, someone could write a version of PHP that is correct (or at least passes its test suite), or at least write a formal spec for PHP, and verify that HotCRP preserves information flow under that formal spec of PHP.

References

[1] E. Kohler. HotCRP. https://hotcrp.com/.
[2] E. Kohler. Hot Crap! Published in *WOWCS'08 Proceedings of the conference on Organizing Workshops, Conferences, and Symposia for Computer Systems*, April 2008.
[3] https://coq.inria.fr/.