Three Projects Proposed by Dr. Rick Skowyra

1. Design of a Provably Safe SDN Controller

**Description:** Software-Defined Networking (SDN) relies on a network controller to manage routing tables on every SDN switch. This controller is a single point of failure in SDNs, as its compromise or faulting would result in subversion by an attacker or denial of service and network collapse. Standard SDN architectures like OpenFlow isolate the controller on a separate control plane to mitigate its attack surface. Unfortunately, in order to handle changing network flows the controller must still process arbitrary dataplane packets forwarded by SDN switches. An attacker can take advantage of this by crafting malformed or corrupted packets which crash the controller, result in remote code execution via vulnerability exploitation, etc.

Current SDN controllers (such as NOX, POX, Floodlight, and Ryu) have at best limited guarantees against crashing or corruption via malformed dataplane packets. This project will investigate the design and engineering of a controller with provable safety and liveness properties regarding packet parsing and handling in the OpenFlow SDN architecture. The design should provide meaningful guarantees while minimally impacting the time taken by the controller to process incoming packets.

**Deliverables:** The project team should provide a rigorous analysis of at least two controller designs with different performance/effort/safety tradeoffs, including justifications of language choice, the degree of manual effort required in implementing the design (e.g. if interactive theorem proving or proof-carrying code is required), the safety and liveness properties which the design provides, and the resulting performance impact.

2. High Performance Memory Safety in C via Hybrid Analysis

**Description:** Memory safety can be obtained for arbitrary C code using dynamic enforcement techniques like Softbound (spatial memory safety) and CETS (temporal memory safety). Unfortunately, these methods have an overhead of 100%-250% and are not suitable for deployment to production systems like webservers or browsers. This project will investigate ways to minimize memory safety overhead by combining runtime dynamic enforcement (which is high-overhead) with source code-based model checking or theorem proving (which is high-effort). Runtime profiling will be used to identify hot paths in a program: those execution paths which are taken on the majority of inputs, and represent the most-used functionality. These paths suffer the most from dynamic enforcement, but usually make up only a small fraction of the actual codebase and thus are more amenable to theorem proving or model checking for memory safety. Conversely, cold paths are good candidates for dynamic enforcement: they are rarely taken and represent uncommon program behaviors.

**Deliverables:** Students working on this project should choose a popular open source program (such as the Nginx webserver). Over the course of the semester, they will have three distinct project phases with an accompanying deliverable:

- Profile the program against a common benchmark and produce an analysis of execution path frequency. Using this analysis, determine a threshold for identifying hot vs cold paths.
- Compile the program with dynamic enforcement of only cold paths. Common enforcement techniques are Softbound+CETS, but other techniques may also be applicable.
- Via theorem proving (e.g., Isabelle/HOL) or model checking (e.g. SPIN or Alloy), attempt to prove the absence of any memory safety violations on the remaining hot paths in the program.
3. Discovering Meaning in Fine-Grained Access Control Policies

Description: Fine-grained access control policies, such as those used by SELinux, permit the principle of least privilege to be enforced with high precision. Unfortunately, the level of detail rapidly becomes difficult for a human to reason about or understand. Firefoxs SELinux policy for filesystem access control (i.e. read/write access to files), for example, can easily span thousands of lines of permissions. This project will investigate ways to support interactive reasoning about such access control policies. Given a policy P as input, the system should be able to answer the following questions:

1. Does P entail another policy P supplied by the user? If not, a counterexample should be provided.

2. Given a set of weighted constraints (representing, for example, the performance cost to implement a policy), what combination of these constraints is both satisfiable and maximizes (or minimizes) the total weight?

Potential tools for performing this analysis include minSAT, maxSAT, SMT solvers, and #SAT, although other approaches are fine as long as they can answer the same questions. The project should also investigate ways to automatically compile a policy (written in a standard access control language) into an appropriate formalism (such as Boolean logic) and back to a human-readable form (such as a visualization or language file).

Deliverables: This project has both a design/analysis and implementation phase. Due to time constraints, a full implementation is not feasible. Instead, students should focus on an end-to-end proof of concept.

- During the design/analysis phase, students will choose an access control language to work with. Given policies written in this language, they should then conduct a rigorous analysis of which formalism and associated tool is most appropriate for use in the system (given its ability to encode the problem and answer queries about entailment and weighted constraint satisfaction). Finally, they should provide an algorithm for translation between the input language and the formalism.

- The implementation phase should aim to support an end-to-end example of a policy being compiled, interactively queried, and the results of that query being translated back to the input language or a visualization. Students should choose a language appropriate to the task of parsing and generating language files and working with formalisms like Boolean logic.