Teaching Statement

Deian Stefan

I find teaching and mentoring to be highly rewarding, educational, and closely tied to my research approach. I am interested in making systems and applications more secure by changing how developers write code. A crucial part of this is to teach the next generation of students (and developers) how to build secure systems through more principled approaches. Given my work on applying programming languages to secure systems, I am well-positioned to teach a variety of classes in both programming languages and systems. One natural class for me to teach is introductory programming languages, which I have already co-taught twice at Stanford, but I am also excited to teach courses on security, operating systems, and introductory computer science, as well as more specific graduate courses focusing on my research interests.

Teaching experience. I have spent the last two years co-teaching (with Edward Z. Yang) the Programming Languages course at Stanford. The goal of this class is not to teach students how to write code in a select few languages. Rather, it is to get students to think about concepts in programming languages and the design and implementation trade-offs of different language features. The class covered various topics, including foundations (lambda calculus and a brief introduction to operational semantics), high-order functions, objects, type inference, polymorphism, continuations, and concurrency.

My teaching approach shares some similarities with my research style in being a multi-step process motivated by foundations and practical concerns. In lecture, I try to find ways to explain a concept in its simplest and most fundamental form and then relate it to other concepts and practical uses. Students are far more likely to internalize and retain things they have figured out for themselves. Hence, I motivate problems by laying out clear examples and use cases, but strive to get students to arrive at the core problem statement themselves. Once they have an understanding of the problem, I then guide them through different solutions and real-world language implementations, evaluating their trade-offs along the way. For instance, one of the concepts in the lecture on objects was *dynamic lookup/dispatch*. Rather than simply explain what dynamic lookup is, I used several code snippets to motivate what problem dynamic lookup was solving: calling a method on an object cannot always be determined statically. The solution was obvious. We then discussed the different trade-off of implementing dynamic lookup as done by Smalltalk, Java, and C++.

In addition to discussing concepts in the context of real programming languages, I like to encourage students to further explore their understanding through hands-on prototyping. To this end, I changed the course to include labs in addition to the more theoretical problem sets. Some of the labs we created included a lambda calculus evaluator, type inference, reference counting and tracing garbage collectors, and C++-style dynamic lookup. By providing students with boilerplate code, they focused their efforts towards understanding the core concepts, rather than mundane coding. Indeed, many students have confirmed that the labs were a fun and helpful way to solidify their understanding of the concepts covered in lecture.

Finally, when teaching, I try to convey my passion and enthusiasm for the topic and make the class enjoyable. One example of this was bringing in external speakers to talk about real-world language design and implementation; we hosted talks from core members of the Go, Rust, and Mozilla JavaScript teams that the students found very interesting. Another was to give a lecture on research topics in the field. It was particularly rewarding to see some students get excited about programming languages and approach me to work on research.

Graduate seminars. I am excited to teach graduate seminars on my research interests, including a seminar on language-based security, and one on the intersection of programming languages and operating systems. Language-based security is a well-established area of research, with over a decade of interesting work, and

this seminar would cover key and recent topics on the use of programming language methods to enforce and reason about security. Language-based security grew out of the need for applications to address security concerns that operating systems do not and cannot enforce. But security is not the only case where operating system abstractions have come short; the second seminar will explore such cases (e.g., performance) and how better integration with programming languages address these shortcomings.

Browser design & implementation. There are a number of fundamental concepts in computer science that, traditionally, a course on operating systems conveys, including resource management, concurrency, scheduling, security, and interface design. I am interested in developing a course on browser engines that can complement such courses by providing a platform for exploring these concepts in a novel setting, with application-specific concerns and demands.

This course will cover the major subsystems of the browser, including the network stack, security architecture, JavaScript engine, the DOM, and the renderer. More importantly, it will explore the interaction between these subsystems and how fundamental concepts arise in such a large real-world system. To give students the opportunity to put their understanding into practice, I am interested in designing labs atop the Breach modular browser and select (understandable) parts of Firefox. For example, one lab may entail implementing a simple database system that is then exposed to web sites; the goal of this is to get students to think about interface design, resource management, and security.

By experiencing core concepts in multiple contexts (e.g., an OS and browser) students are more likely to get a deeper understanding for fundamentals. This course can serve that role. Moreover, it can help to lower the barrier to entry for exploring research ideas, such as COWL, in the context of the browser.

Mentoring experience. Over the last three years I have also been advising several students who worked with me on my research projects, including Masters and junior Ph.D. students, early undergraduate students, and a high-school student. Of the more senior students, I mentored Kyle Brogle (now at Apple Security) on finding a cryptographic encoding for DCLabels, the logic used to specify policies in Hails and COWL, and Stefan Heule on developing the formal semantics for a generalized model of COWL. I am currently working closely with Annie Liu (a junior Ph.D. student at Princeton) on ESpectro and Devon Rifkin on addressing user privacy in the presence of malicious and vulnerable browser extensions.

As a mentor, I hope to foster a friendly and collaborative environment. To this end, I plan to collaborate with students and encourage them to work together. My goal is to guide students towards fruitful and high-quality research. In their early years, this may involve low-level feedback. But, beyond this, I wish to create an atmosphere where students pursue projects they find most interesting. I believe that the key role of advisor is to guide students towards problems that match their interests and strengths—working with students that are independent and passionate has been both intellectually stimulating and immensely rewarding.