When a security problem is found, we blame the developer who wrote the vulnerable software. But is this the right place to place the blame? In many cases, the developer used a library which encouraged the writing of insecure code. Attempting to enable SSL certificate authentication in CURL? Better set SSL_VERIFYHOST to '2' and not 'true', lest your connection be vulnerable to a trivial man-in-the-middle attack. [1] Attempting to generate a public key? Better make sure your RNG is properly seeded, or risk your key being factored. [2] When we tell people to write secure code, we are actually asking them to navigate a minefield of vulnerabilities, most of which they don't even know about! This is too much to ask. Any given vulnerability can be patched, but in aggregate, developers continue to make the same mistakes over and over again.

One solution is to fix these APIs so that they cannot be misused. But even when a fix is known (e.g. error-by-default on integer overflow), creating convincing replacements that developers will use has proven to be very difficult. Feature adoption is a social process: disseminating a better API requires an understanding of how developers behave. Until recently, we had very little data in this area. Nowadays, the code for millions of projects is available through social coding websites such as GitHub. This unprecedented corpus of material is largely understudied, though there are some efforts to take advantage of this material. Code clone research looks for copy-pasted code fragments as likely candidates for bugs; “Socio-PLT” [3] seeks to elucidate the sociological behavior of developers. My goal is to generalize this research and apply it to the entire open source world, so that we can design a new generation of APIs and programming language features that average developers are likely to adopt.

I have experience building libraries that remain secure even when misused by developers: I have done research characterizing HTML that stays stable even in the face of unpredictable transformations by client-side browsers [4], and have worked on information flow control systems which enforce security policies even when code is actively malicious. [5] I have expertise in programming languages and compiler development, developed during my time spent at Microsoft Research and Galois. I believe I am well placed to combine this knowledge with empirical findings about API use in the wild, so that we can stop blaming developers for their mistakes and start learning lessons from them.

[1] Georgiev M, et al "The most dangerous code in the world: validating SSL certificates in non-browser software"
[2] Lenstra A, et al "Ron was wrong, Whit is right"
[3] Meyerovich L, et al “Socio-PLT: Sociological Principles for Programming Language Adoption”
[4] “Mutations, TrueHTML and the DOM” Under submission to S&P 2013
[5] “Eliminating Cache-Based Timing Attacks with Instruction-based Scheduling” Under submission to S&P 2013