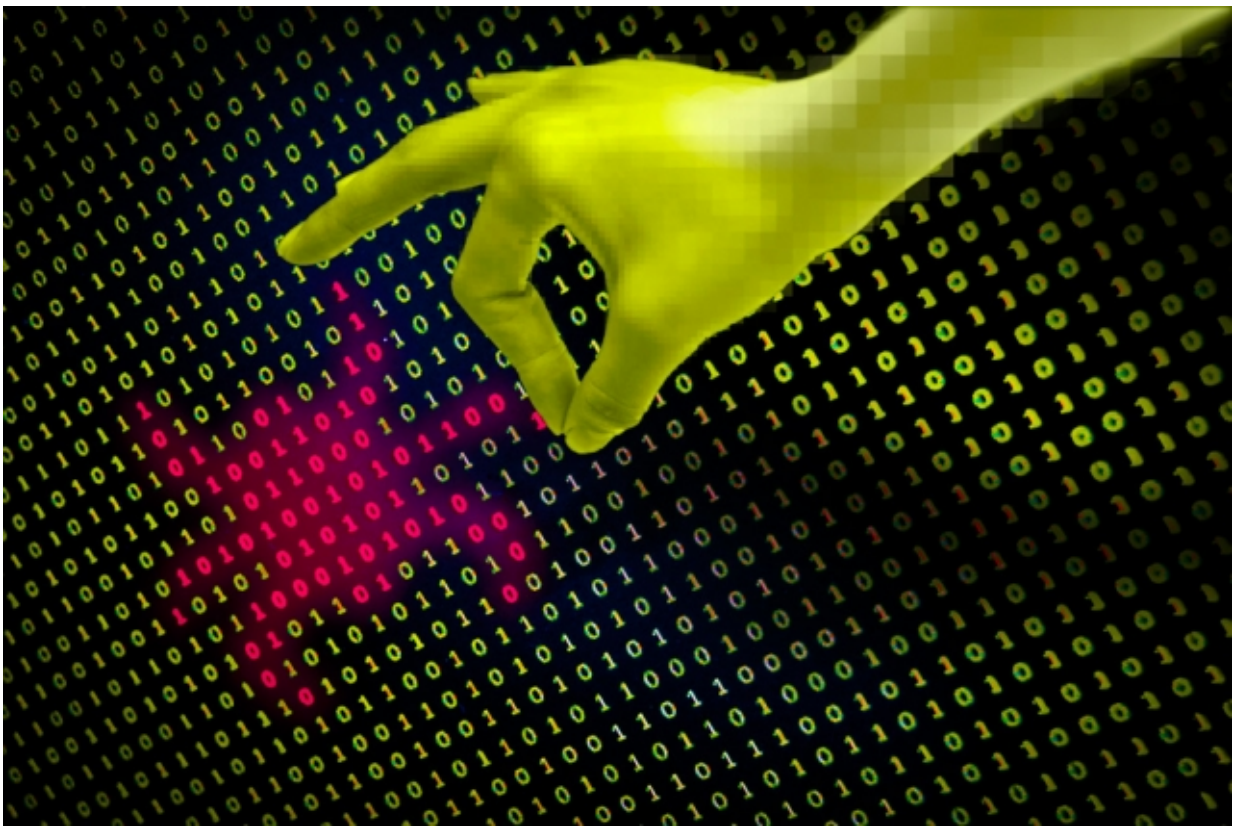
New debugging method found 23 undetected security flaws in 50 popular Web applications in less than an hour

April 15 2016, by Larry Hardesty



In tests on 50 popular Web applications written using Ruby on Rails, a new debugging system found 23 previously undiagnosed security flaws, and it took no more than 64 seconds to analyze any given program. Credit: MIT News

By exploiting some peculiarities of the popular web programming framework Ruby on Rails, MIT researchers have developed a system that can quickly comb through tens of thousands of lines of application code to find security flaws.

In tests on 50 popular web applications written using Ruby on Rails, the system found 23 previously undiagnosed security flaws, and it took no more than 64 seconds to analyze any given program.

The researchers will present their results at the International Conference on Software Engineering, in May.

According to Daniel Jackson, professor in the Department of Electrical Engineering and Computer Science, the new system uses a technique called static analysis, which seeks to describe, in a very general way, how data flows through a program.

"The classic example of this is if you wanted to do an abstract analysis of a program that manipulates integers, you might divide the integers into the positive integers, the negative integers, and zero," Jackson explains. The static analysis would then evaluate every operation in the program according to its effect on integers' signs. Adding two positives yields a positive; adding two negatives yields a negative; multiplying two negatives yields a positive; and so on.

"The problem with this is that it can't be completely accurate, because you lose information," Jackson says. "If you add a positive and a negative integer, you don't know whether the answer will be positive, negative, or zero. Most work on static analysis is focused on trying to make the analysis more scalable and accurate to overcome those sorts of problems."

With web applications, however, the cost of accuracy is prohibitively

high, Jackson says. "The program under analysis is just huge," he says. "Even if you wrote a small program, it sits atop a vast edifice of libraries and plug-ins and frameworks. So when you look at something like a web application written in language like Ruby on Rails, if you try to do a conventional static analysis, you typically find yourself mired in this huge bog. And this makes it really infeasible in practice."

That vast edifice of libraries, however, also gave Jackson and his former student Joseph Near, who graduated from MIT last spring and is now doing a postdoc at the University of California at Berkeley, a way to make to make static analysis of programs written in Ruby on Rails practical.

A library is a compendium of code that programmers tend to use over and over again. Rather than rewriting the same functions for each new program, a programmer can just import them from a library.

Ruby on Rails—or Rails, as it's called for short—has the peculiarity of defining even its most basic operations in libraries. Every addition, every assignment of a particular value to a variable, imports code from a library.

Near rewrote those libraries so that the operations defined in them describe their own behavior in a logical language. That turns the Rails interpreter, which converts high-level Rails programs into machine-readable code, into a static-analysis tool. With Near's libraries, running a Rails program through the interpreter produces a formal, line-by-line description of how the program handles data.

In his PhD work, Near used this general machinery to build three different debuggers for Ruby on Rails applications, each requiring different degrees of programmer involvement. The one described in the new paper, which the researchers call Space, evaluates a program's data

access procedures.

Near identified seven different ways in which web applications typically control access to data. Some data are publicly available, some are available only to users who are currently logged in, some are private to individual users, some users—administrators—have access to select aspects of everyone's data, and so on.

For each of these data-access patterns, Near developed a simple logical model that describes what operations a user can perform on what data, under what circumstances. From the descriptions generated by the hacked libraries, Space can automatically determine whether the program adheres to those models. If it doesn't, there's likely to be a security flaw.

Using Space does require someone with access to the application code to determine which program variables and functions correspond to which aspects of Near's models. But that isn't an onerous requirement: Near was able to map correspondences for all 50 of the applications he evaluated. And that mapping should be even easier for a programmer involved in an application's development from the outset, rather than coming to it from the outside as Near did.

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology