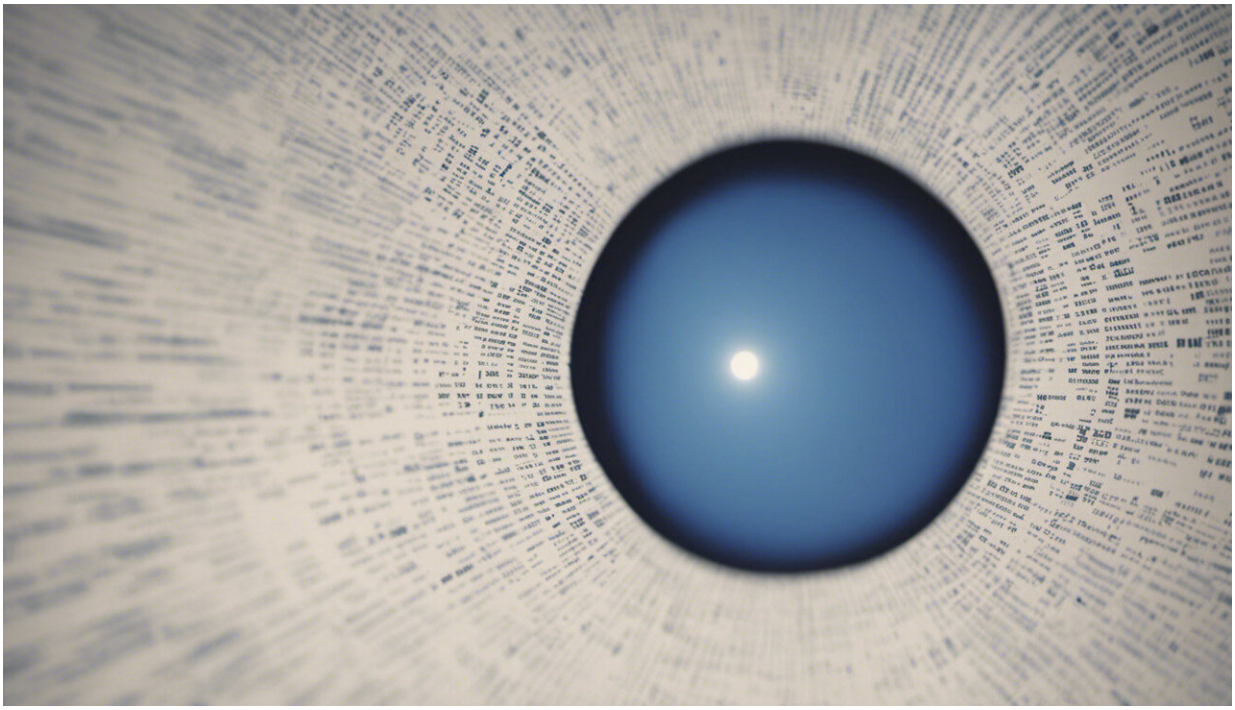
Building privacy right into software code

February 21 2017, by Jean Yang



Credit: AI-generated image (disclaimer)

When I was 15, my parents did not allow me to use AOL Instant Messenger. All of my friends used it, so I had to find a way around this rule. I would be found out if I installed the software on my computer, so I used the web browser version instead. Savvy enough to delete my internet history every time, I thought my chatting was secret.

Then one day my mother confronted me with all the times I had gone on

Instant Messenger in the past week. Whenever I visited the site, it had left a trail of cookies behind. Intended to make my user experience more convenient, cookies saved my login information for repeat visits. Unfortunately, the cookies made my life less convenient: My mother knew how to inspect them to determine when I had been illicitly instant messaging.

Since then, I have been very interested in protecting user privacy. I studied computer science in college and ended up pursuing a career in the field. I became fascinated with programming languages, the construction materials for the information age. Languages shape how programmers think about software, and how they construct it, by making certain tasks easier and others harder. For instance, some languages allow rapid website prototyping, but don't handle large amounts of traffic very well.

Regarding my main interest, I discovered that many of today's most common languages make it difficult for programmers to protect users' privacy and security. It's bad enough that this state of affairs means programmers have lots of opportunities to make privacy-violating errors. Even worse, it means we users have trouble understanding what computer programs are doing with our information – even as we increasingly rely on them in our daily lives.

A history of insecurity

As part of the first generation who came of age on the internet, I enjoyed the benefits of participating in digital life, like instant messaging my friends when I was supposed to be doing homework. I also knew there was the potential for unintended information leaks.

A then-crush once told me that he took advantage of a fleeting Facebook opportunity to discover that I was among his top five stalkers. For a brief

period of time, when a user typed "." into the search bar, the autocompleted searches were the users who most searched for them. I was mortified, and avoided even casual browsing on Facebook for a while.

This small social crisis was the result of a programming problem, a combination of both human programmer error and a shortcoming of the language and environment in which that human worked. And we can't blame the programmer, because the languages Facebook uses were not built with modern security and privacy in mind. They need the programmer to manage everything by hand.

Spreading protections across the program

As those older languages developed into today's programming environments, security and privacy remained as add-ons, rather than built-in automatic functions. Though programmers try to keep instructions for different functions separate, code dedicated to enforcing privacy and security concerns gets mixed in with other code, and spread all throughout the software.

The decentralized nature of information leaks is what allowed my mother to catch me messaging. The web browser I used stored evidence of my secret chatting in more than one place – in both the history of what sites I visited and in the cookie trail I left behind. Clearing only one of them left me vulnerable to my mother's scrutiny.

If the program had been built in such a way that all evidence of my activity was handled together, it could have known that when I deleted the history, I wanted the cookies deleted too. But it wasn't, it didn't and I got caught.

Making programmers do the work

The problem gets even more difficult in modern online systems. Consider what happens when I share my location – let's say Disney World – on Facebook with friends who are nearby. On Facebook, this location will be displayed on my "timeline." But it will also be used for other purposes: Visitors to Disney World's Facebook page can see which of their friends has also been to the amusement park. I can tell Facebook to limit who can see that information about me, so people I don't know can't go to Disney World's page and see "Jean Yang checked in 1 hour ago."

It is the programmer's job to enforce these privacy restrictions. Because privacy-related code is scattered throughout all the programs Facebook uses to run its systems, the programmer must be vigilant everywhere. To make sure nobody finds out where I am unless I want them to, the programmer must tell the system to check my privacy settings everywhere it uses my location value, directly or indirectly.

Every time a programmer writes instructions to refer to my location – when displaying my profile, the Disney World page, the results of queries such as "friends at Disney World" and countless other places – she has to remember to include instructions to check my privacy settings and act accordingly.

This results in a tangle of code connecting the rules and their implementation. It is easy for programmers to make mistakes, and difficult for anybody else to check that the code is doing what it's supposed to do.

Shifting the burden to computers

The best way to avoid these problems is to take the task of privacy protection away from humans and entrust it to the computers themselves. We can – and should – develop programming models that allow us to more easily incorporate security and privacy into software. Prior research in what is called "language-based information flow" looks at how to automatically check programs to ensure that sloppy programming is not inadvertently violating privacy or other data-protection rules.

Even with tools that can check programs, however, the programmer needs to do the heavy lifting of writing programs that do not leak information. This still involves writing those labor-intensive and error-prone privacy checks throughout the program. My work on a new programming model called "policy-agnostic programming" goes one step farther, making sloppy programming impossible. In these systems, programmers attach security and privacy restrictions directly to every data value.

For instance, they could label location as information requiring protection. The program itself would understand that my "Disney World" location should be shown only to my close friends. They could see that not only on my own page, but on Disney World's page.

But people I don't know would be shown a less specific value in both places. Perhaps friends of my friends might see "away from home," and total strangers could only learn that I was "in the United States." Looking at my page, they wouldn't be able to tell exactly where I am. And if they went to the Disney World page, I wouldn't appear there either.

With this type of structure, the humans need no longer write code to repeatedly check which information should be shared; the computer system handles that automatically. That means one less thing for programmers to think about. It also helps users feel more confident that

some element of a complicated piece of software – much less a human error – won't violate their personal privacy settings.

With software programs handling our driving, shopping and even choosing potential dates, we have much bigger problems than our mothers seeing our internet cookies. If our computers can protect our privacy, that would be a huge improvement to our rapidly changing world.

This article was originally published on The Conversation. Read the original article.

Provided by The Conversation

Citation: Building privacy right into software code (2017, February 21) retrieved 27 April 2024 from https://phys.org/news/2017-02-privacy-software-code.html