EDITORIAL

Musings

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

I've often written about how depressing I find computer security is for the December issue, so this year I thought I'd try a different tack. Honestly, there were parts of USENIX Security, particularly the WOOT workshop, that had me laughing out loud.

I really liked the "Fast and Vulnerable" [1] paper for its humorous insights into the state of programming. A widely used product, one that is Internet-connected and can be used to control cars, totally fails at having any security at all. What a laugh! They even included the private SSH key for the root account for the device—and the same key is used on all devices by this manufacturer.

Not that SSH is needed at all: just a simple SMS message to the device can be used to instruct it to download a software update. That's right. All you need is a phone number and to send a text message, and you can "own" someone else's car. And the phone number could be war-dialed. As if this weren't enough, there's also a Web *and* a Telnet interface you can use.

Programmers

I've heard Wietse Venema and D. J. Bernstein's names mentioned many times at USENIX Security as the only people who have a proven track record for writing secure software. That should give you food for thought: if just these two guys have done it right, that implies everyone else is doing it wrong.

And that seems to be about right. Getting the security right is very hard; even the best programmers often make exploitable mistakes, and I think we should assume that the best programmers are a tiny minority. That leaves about six nines, or 99.9999% of programmers among those who are not the best. Then what those statements about Venema/Bernstein really mean is that effectively no one can write software securely.

A large part of the problem has to do with the nature of programming. Someone, hopefully a skilled programmer, gets tasked with creating software that will convert requests into the appropriate responses. In most cases, the programmer writes a list of instructions, does some testing, then keeps working on the software until it appears to work.

Nowhere in that outline of the programming task does the concept of security even appear. If security does come up, the requirement is often something like "must include crypto," and the programmer then adds a function with a hash check or perhaps XORs something so the program includes encryption. Perhaps the programmers are more advanced and decide they will use a library in OpenSSL, but because they don't understand cryptography they use weak keys and repeat the same initialization vector with every request.

I used to laugh at early attempts to port Microsoft MS-DOS programs to UNIX. The authors didn't understand the UNIX security model, so they would run their software as root. All files would be publicly writeable by all, too. For someone coming from the MS-DOS world, where there was no security, this was the equivalent security "ported" to UNIX.

In the research for "Fast and Vulnerable," the programmers/designers for the car-connected device didn't do much better. They did have a root password and a user password (for the user named "user"), but they were trivially cracked. You can query, update, or control the device either locally, over a network connection (via a USB port), or remotely, as the Web and Telnet services appear both locally and remotely. And since this device connects to the OBD-II port on US-built cars, you can play the types of tricks with the car as Miller and Valasek did with the entertainment head found in new Chrysler Jeeps [2].

The Question

Finding vulnerable devices online is nothing new. Companies have created everything from routers to medical devices that appear online, have software with exploitable vulnerabilities, and provide no mechanism for updating the software of these devices, and this has been true for about as long as there has been a public Internet (1991). You might think that we, the computer science community, might try and do something about this sorry state of affairs. And we have, but because security isn't easy, things haven't worked out so well.

Let's take a look at a couple of recent attempts to provide security for applications that assume that programmers shouldn't be expected to do this themselves: Android and iOS. Of the two, iOS has a stricter model, which has worked well until XcodeGhost [3] came along. By adding a Trojan object file to the Xcode development framework, used to write applications for iOS, applications would include the Trojan binary. And Apple accepted these applications into the Apple App Store, as the applications appeared to follow the rules. Ooops.

Android has fared much worse for a couple of reasons. Google never wanted, or could have, the same degree of control over the apps market for Android, and that has opened the field to malware. And second, the model chosen for Android was never intended to be as robust. When I first heard about the Android security model, at a USENIX Security Symposium in Montreal (2009), I heard that the model would partially rely on people noticing and complaining about insecure apps. Also, users would be expected to decide whether or not to allow apps access to their devices, with over 140 different types of access potentially allowed. I made a point of speaking with the program manager, and when I complained to him that they expected way too much from Android users, he told me that it was too late to change the design.

Google has recently updated the Android security design [4], but the user still must make security decisions, and those decisions are still all or nothing. For example, if you want to use the Uber app, you grant Uber complete access to your phone and personal information. I found that very interesting. I also know that lots of people are willing to trust Uber, the corporation, and Uber programmers, with complete access to their phones. Really? Have you done that?

Google has created a new sandbox, without any access permissions, that apps can be run in. Another of my favorite papers at Security, "Boxify: Full-Fledged App Sandboxing for Stock Android" [5], will allow knowledgeable programmers to run other people's apps inside a permissionless sandbox, and have finer-grained control over what personal data we are willing to share and when. What distinguishes the new sandbox from the old one is the ability to run unmodified apps, regardless of what permissions the app programmers have asked for (everything if you're Uber), and decide to grant access to a selected set when executed, instead of at install time.

Microsoft got serious about security in 2001. Just this year, they replaced the default browser, IE, with something better. Apple and Google have always been serious about security, but their results have been mixed so far. Keep in mind that iOS in China, where lots of iPhones are jailbroken, is on a par with Android in the rest of the world. None of this is easy.

To sum up, we have a history of programmers being unable to write programs securely. We have vendors who are unable to provide secure environments in which to run insecure apps. So why are we at all surprised at the lack of security today?

The Lineup

I was so impressed with Ian Foster's WOOT '15 presentation about finding weaknesses in an OBD II device that I asked him and one of his co-authors (Karl Koscher) if they would write about CAN bus for *;login:*. Understanding CAN bus is the key to understanding how modern cars work—and why we see remote hacks of cars that might appear to be magic if we didn't know better.

Roel Verdult and Flavio Garcia explain the problems found in Megamos, a widely used automobile immobilizer. In what could be seen as a reprise to the theme of my editorial, Verdult and Garcia analyzed both the immobilizer itself *and* how it has been used in many brands of automobiles, showing that indeed, programmers cannot program securely and vendors don't understand cryptography.

Simson Garfinkel volunteered an article on digital forensics. Fresh from chairing the DFRWS 2015 conference, Simson provides us with a clear view of how the field of digital forensics has grown both broader and more challenging over time. Both a current researcher and an accomplished writer, Simson takes us through from the early days of cloning drives to modern tools that can analyze the vast amount of data found on our digital devices and networks.

EDITORIAL

Musings

I met John Murray during the USENIX Security '15 poster session, where he was explaining the Menlo Report to anyone who would listen. I'd only that afternoon heard of the Menlo Report, during a panel session by past program committee members discussing how they handle ethical lapses in submitted papers. I asked John if he could tell us more about how the Menlo Report provides a better basis for institutional review boards (IRBs) considering security research proposals.

I've known Rick Forno for many years, from when he worked for the early US Internet name registry. Rick offered to share five years of experience running the Maryland Cyber Challenge, a cyber competition that allows participants at different levels of education and experience to learn together.

Andy Seely has written a summary of his 11 columns on managing system administrators. If you missed any of his past columns, you can review the useful practices that Andy has shared, as well as locate the columns you either missed or find that you now need.

I interviewed Darrell Long (UCSC) about how the actual arrival of a real non-volatile memory (NVM) RAM product will affect the world of computing. The answer? You need to read the interview, but I can tell you here, this is big.

Peter Salus writes his final USENIX history article and discusses the founding of the Software Tools User Group (STUG) and LISA. If you've ever wondered where USENIX came from, who came up with that name, I suggest you read all of Peter's articles. They aren't that long but do put the past of USENIX in perspective.

I also interview Rob Kolstad. Rob was elected to the USENIX Board three times and started the LISA conference. Rob is an amazing guy, and you can find hints of this by reading this interview, as well as another look at where USENIX has come from and what happened in the past to make it the organization that it is today.

Dave Beazley waxes enthusiastic about the new asynchronous I/O (asyncio) module in Python 3.5. As Dave writes, this will blow your mind if you are already familiar with past asyncio Python modules. More practically, you will learn the current direction for handling thousands of threads of activity using coroutines.

David Blank-Edelman took up the challenge I tossed him when he submitted his October column and worked up some Perl script magic for using OAuth2. OAuth2 is more than an authentication protocol, as OAuth2 tokens are used for delegating access to resources on other folks' servers. David's column covers checking all of the Google Calendars shared with you for events that may be bugging you with daily notifications, perhaps while the person involved is off on vacation.

Dave Josephsen discusses monitoring for programmers. While we usually think of monitoring as a task belonging to system administrators and SREs, Dave reminds us that developers need to understand what types of events they should be reporting in their code.

Robert Ferrell uses his fertile imagination to come up with a new form of secure email, Streamailer, and also addresses how difficult change in *any* form is for many people, and ends with a new technique for authentication.

We have book reviews by both Mark Lamourine and myself. Mark reviewed *Python for Data Analysis* and an introductory book on Go, while I reviewed the Donovan and Kernighan Go book.

Restarting

I believe that if people are allowed to do anything, including things harmful to themselves or others, someone will try to do those harmful things. And it appears that humanity in general agrees with this belief, as we have laws and customs that set constraints on behavior. We drive on a particular side of roads, do not pick up apples at a farmer's market and walk off with them without paying, throw stones or shoot at passersby, and so on. We live in a civil society (for the most part).

But in the world of programming, we have few constraints. The "Fast and Vulnerable" [1] paper clearly shows what happens when programmers are set free of the constraints of security to design a product that, when misused, can kill people.

I also believe that people are much more comfortable with constraints that appear natural. We all learn as children that when we jump up, we quickly fall back to earth, if we eat too much, our stomach starts hurting, and so on. These are natural constraints, and they work for the most part (see 72-ounce steak rules [6]), for most people.

That said, I also believe that we need programming environments where writing secure code comes naturally. I believe we can do that with modern languages and a structure of natural constraints that encourage writing small modules that require least privileges, as Venema and Bernstein have been showing us for years.

When we, the culture of computer scientists, began writing programming languages, we needed something better than writing in machine language, and we got FORTRAN and COBOL. We've come a long way since then, but it's painfully obvious we still have a long way to go.

There is another important leg to having an environment where secure programming feels natural, and that is the hard part: hardware. Our programming environment matches our hardware architecture, where even a smartwatch has an architecture

that evolved from 1960 mainframes. We are still building time-sharing systems, even now that we can put tens of cores on a tiny chip. Those cores, properly connected, could form the basis for many small services that work today.

There's even a name for those services that has become quite popular: microservices. Let's build the architecture [7] for running those microservices natively and securely, and design a software architecture that makes programming securely natural.

References

[1] I. Foster, A. Prudhomme, K. Koscher, S. Savage, "Fast and Vulnerable: A Story of Telematic Failures," in the *Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT '15)*, August 2015, Washington, DC.

[2] Andy Greenbaum, "Hackers Remotely Kill a Jeep on the Highway—With Me in It," July 21, 2015: http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[3] http://researchcenter.paloaltonetworks.com/2015/09/novel-malware-xcodeghost-modifies-xcode-infects-apple-ios-apps-and-hits-app-store/.

[4] Security-Enhanced Linux in Android: https://source.android.com/devices/tech/security/selinux/index.html.

[5] Michael Backes, Sven Bugiel, Christian Hammer, Oliver Schranz, and Philipp von Styp-Rekowsky, "Boxify: Full-Fledged App Sandboxing for Stock Android," in the *Proceedings of the 24th USENIX Security Symposium*, August 2015, Washington, DC, pp. 691–706: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/backes.

[6] The 72 oz. Steak Rules: http://bigtexan.com/72oz-steak-rules/.

[7] http://www.rikfarrow.com, Design for Security.