RIK FARROW

musings

rik@usenix.org

WELCOME TO THE NINTH SECURITY edition of *;login:*. As one of the authors in this issue mentions, the existence of a focus issue on security is evidence that security is still a concern. Of course, we all have plenty of first-hand experience, and evidence, that our computer security problems remain unsolved. At best, we have moved the goalposts a bit and redesigned the playing field. But our computer systems appear to be as insecure as ever.

Not that there hasn't been progress. Go back to the 1990s, when UNIX and Linux systems included exploitable services, running as root, out of the box. Over time, Linux in particular has improved in default security, a very good thing indeed (with BSDs way ahead in security by the late 1990s). And open source programs in general may not be bug-free, but their support teams can boast of being many times faster at releasing patches than most of their commercial counterparts.

For reading that at least brings a smile to my face every time, I can recommend the Symantec Internet Security Report [1]. You might think me cynical (and you would be correct) when I write this, but the stats included in their executive summary are incredible. For example, the good ol' USA is number one in a number of areas other than pollution, but Beijing is fast catching up. The U.S. boasts of more attack and malicious activity, phishing servers, and command and control systems, while being both the greatest source and the largest target of spam. Beijing, however, has more bot-infested computers than any other city in the world, and China has 29% of the world's bots as well. So the Chinese have another interesting problem to deal with, beyond the pollution created by their rapidly expanding economy.

The Symantec report is hardly a source of encouraging news, but there are some bright spots. HP has exchanged places with Sun (or is it Java?) as the slowest vendor to release a patch to a vulnerability (112 days). Microsoft was the winner for having the most enterprise-level unpatched vulnerabilities for the first six months of 2007, unchanged from winning this contest in the last half of 2006. Okay, so I am still being really cynical.

Apple does well in the report, with fewer reported vulnerabilities in Safari than IE or Mozilla but more than Opera. Apple can also boast of having the shortest time to patch browser bugs, beating

out the Mozilla Foundation this time, with an average window of exposure of four days (Mozilla's was only two days in the previous report). But speaking of Apple . . .

Sour Apples

By the time you read this, Apple's "bricking" of unlocked iPhones will be old news. You will remember the flurry of hacking activities that followed the release of the iPhone in late June and the discovery that the iPhone (and iPod) actually run Mac OS X. Deeper inspection of the iPhone revealed that all applications *run as root*, evidence either that the iPhone was rushed to market or that Apple programmers are extremely lame. I know the second conjecture is not true, so I must presume the first.

Many people I know have raved about the design of the iPhone, and they seem to be very happy with the cell phone as it exists. But other people I know were quite anxious to extend the capabilities of the iPhone to include shell access, an SSH client, WiFi scanner, and other tools that Apple neglected to include, even after several updates. Update 1.1.1 disabled all added applications, and changes to the firmware of the iPhone disabled the cell phone functionality, "bricking" the phone, of those iPhones that had been unlocked (allowed to work with the SIMs from carriers other than AT&T).

I don't intend to muse about the propriety of Apple destroying the functionality of a device people had purchased (at an inflated price). Instead, consider the implications of update 1.1.1. With this update, Apple, belatedly, added protection to the iPhone's firmware that prevents unlocking. Why wasn't that in the first version? To the OS that runs your cell phone—any cell phone, to my knowledge—the radio looks like a modem. Remember how you commanded modems back in the day, using AT commands? Cell phone radios work similarly. So the cell radio appears to the OS as a serial device, and any application that runs as root can access that serial device and send it AT commands.

Depending on the cell radio, the set of commands that can be issued may include changing the radio frequencies that the phone uses, as well as the power used when transmitting. In the U.S., the FCC (Federal Communications Commission) regulates the use of the radio spectrum, and it takes a dim view of putting this level of flexibility, and control, into the hands of cell phone users. Carriers are also loath to allow their customers to change settings, because in doing so they may disrupt the operations not just of the adjusted phones, but also of other cell phones operating in the area. The cell radio interface must be protected.

This brings us back to the iPhone, where all applications run as root. That implies that any application installed on the iPhone may have access to the cell radio. I like to see well-designed security, particularly when a device must comply with national laws, and the iPhone appears to fall far short of this goal. But with this design, it should be obvious that Apple *had to* prevent third-party applications from running on the iPhone.

Apple has had other security issues this year. The mDNSResponder, a friendly service that broadcasts information about your Mac to any device that can listen, as well as accepting information from any system in range, has had at least two root exploits this year (one that has not been reported). I checked to see if the patched mDNSResponder was still running as root after the patch, and it was (I've since disabled it, so can't be sure if this is still true). But mDNSResponder, which, according to the Darwin source code,

only needs to be able to do *one thing* as root, does everything as root. Notice a pattern here?

Part of what mDNSResponder (a.k.a. Bonjour) does is parse variable-length input broadcast from other systems, and parsing variable-length input happens to be the exact same issue that plagued SNMP and other ASN.1 notation-based systems in the not distant past. I'd like to see Apple change how mDNSResponder works by separating out the code that must run as root, advice that dates back to Matt Bishop's article from 1987 that provides advice about writing set-user-id programs [2]. Although Matt focused on setuid, the notion of partitioning a program into two parts, a privileged part and a larger part that runs without privileges, has become part of accepted security practice today.

Then there was the Airport exploit, during BlackHat 2006. This may appear to be old news, but you can now read what had been hidden from view about this kernel-level exploit [3]. At the time, Apple vehemently denied that the exploit actually worked on Apple MacBooks [4], but Maynor explains that he was perplexed when a nontargeted MacBook crashed several minutes after he had conducted fuzzing using WiFi beacon and probe packets. His article is useful to anyone interested in understanding panic.log files found on Mac OS X systems after kernel crashes.

I much prefer that vendors be honest about the security of their products, even if the news is bad. History, for example that of Windows NT, has shown that no matter how secure you claim your system is, the more people who use it, the more bugs will be discovered. Mac OS X does have some advantages over NT, as it represents a much more mature technology than NT was in 1998. But there are other more disturbing parallels between Microsoft and Apple, such as the lack of any documentation about how security mechanisms work in Mac OS X. Like other security professionals, I prefer to understand how a mechanism works and see how it is implemented, rather than to trust a vendor who simply states, "Don't worry, it's secure." Where have I heard that before?

The Lineup

As a tribute to the advances in exploit technology, we begin this issue with two articles about P2P (peer-to-peer) command and control as used in recent Windows trojans. I wanted to write "bots" here, but Dave Dittrich has convinced me that bots have that name because they are controlled via IRC servers, and I have to agree with him. The first article, by Dave Dittrich and Sven Dietrich, explores how remote-control trojans have evolved over time, moving from clunky handler/agent command and control to P2P. The change to P2P makes trojans more difficult to write, but also much more difficult to detect, and nearly impossible to trace back to the owner of a collection of trojans. The authors do provide some advice about what you can do to detect and traceback trojans in your own networks.

The next article, by Stover et al., dissects two recent trojans that use P2P for command and control. Stover and associates focus on the network communications of both trojans, rather than disassembling them, to reveal how they behave when viewed from outside the box. In the case of Storm, there is one configuration file, used to seed the list of peers when the trojan begins to communicate. With Nugache, the list of initial peers is embedded in the executable, and communications are further hidden through the use of encryption between each peer.

Tal Garfinkel and Andy Warfield entertain us next, with an article describing current and future security uses of virtualization. Being an old-school kind

of guy, I like the idea of isolating services, that is, running a single service per system. Virtualization allows us to do something similar, that is, run one service per guest operating system, allowing much more efficient use of hardware, along with the improvements in system management possible via virtualization. Tal and Andy also point out future uses of VMM—for instance, running anti-virus software and HIPS within the VMM. Currently, malware commonly disables AV and hides itself using rootkit technology, so the notion of moving the protection outside of the guest OS into the VMM makes lots of sense. I liked the idea, but did point out during the review process that if they move the protection into the VMM, that will change the focus to attacking the VMM. Within a week, a vulnerability that allows executing commands as root within domain 0 (the VMM OS) was announced, allowing me to feel like a prophet [5].

Gernot Heiser offers his own view of the future of OS security. Gernot has been working on the L4 microkernel for many years, and I recently learned that seL4, a secure version of L4, was going through the process of being formally proven. That is, the design of seL4 would be verified via theorem provers that it matches its specifications. I had challenged Gernot to prove to me that proving any OS really matters to anyone, and this article is his response. To be honest, I was playing devil's advocate in this case. I would like to see small and verified OSes in the future of servers, desktops, and embedded systems, and seL4 is a great leap forward.

I had lunch with Gary McGraw this summer, while we were at USENIX Annual Tech in Santa Clara. Gary waxed enthusiastic about his new book, *Exploiting Online Games*, and we decided that an interview-style article would be an interesting addition to this issue. I also wanted to bring out some of the topics we had in our own discussions that don't appear in his book or in the summary of the IT he gave at USENIX Security (see the summaries section of this issue, and watch the video, listen to the MP3 of the talk, and view his presentation slides on www..usenix.org/events/sec07/tech/).

Mike Rash, who has a new book out about Linux firewalls, has explained a couple of the tools he has written that augment netfilter: fwsnort, which allows you to add snort-based rules to Linux firewalls, so you can block TCP connections that match these rules, and psad, which performs passive system identification, using the same database as p0f, but using iptables log messages. If you want to learn more about netfilter and iptables, as well as Mike's tools, I recommend reading this article.

David Blank-Edelman has fully joined our conspiracy to discuss security in this issue. David writes about using Perl modules to embed the use of nmap and p0f. I had heard about embedding nmap into Perl scripts, a technique that adds a lot to the usability of nmap in large scans. Using p0f within Perl was a new concept for me.

Dave Josephsen also plays along with the security theme. Dave explains how to analyze netflows to monitor your networks and to improve their security. Steve Romig wrote about the use of netflow logs in security way back in the September 1999 [6] issue of *;login:*, and I'm glad that Dave decided to bring us up to date on this topic.

Heison Chak was invited to the security table, but he chose instead to explain speech recognition software as used in Asterisk. Heison, like most of us, finds little charm in most automated phone systems, and he shows us how to embed speech recognition within Interactive Voice Response (IVR) systems that will, hopefully, be well-designed and not drive callers to acts of anger and frustration.

Robert Ferrell, staying with our security focus, attacks the naming conventions used by AV companies. Not content with boring names such as Win32/DEL.100907.ZZA, Robert has some interesting suggestions for the worms and trojans of the near future.

In the book reviews section, Elizabeth Zwicky starts off with a quick review of the new edition of *The Practice of System and Network Administration* and then covers a book on designing reliable sites. Sam Stover, not content with just writing an article, reviews the tome named *The Art of Software Security Assessment* and comes away very favorably impressed. Finally, Ming Chow reviews the book I mentioned earlier, *Exploiting Online Games*, with unsparing accuracy.

We have many conference reports in this issue, starting with USENIX Security in Boston. I wrote some about cell phone issues in this column. There was a panel, an Invited Talk (about Symbion viruses), and a paper about cell phone security at the symposium, and you will find in-depth summaries about these talks and others here. Five workshops were co-located with USENIX Security, and we have summaries from three of them here: USENIX/ACCURATE Electronic Voting Technology Workshop, First USENIX Workshop on Offensive Technologies, and MetriCon 2.0: Second Workshop on Security Metrics. Finally, Matt Bishop sent us summaries of the 2007 New Security Paradigms Workshop.

Afterthoughts

I just tried imagining what it would be like if we had secure computer systems. Sure, I would be out of a job, but I like to consider the benefits to the wider world. If our computers were secure, we could use online banking with little fear (although the banks could still screw up our accounts). We could communicate with complete assurance that no one could read our email (and then there's the NSA). We would no longer have to pay fees to AV companies whose software is, at best, only partially able to protect systems from malware. We could view the Web without fear of our systems being infected with spyware, adware, or trojans (porn without fear!). And we could trust that our personal data, whether financial or health-related, would remain secure on the servers where it is stored.

Honestly, at this point in time, imagining a future of secure operating systems and software is hard to do. But perhaps, having already used a sports analogy, I can use a car analogy as well. Just imagine what cars were like in the 1920s: Horseless carriage was a good description. Today's cars are both safer and a lot more reliable than cars from just 20 years ago. Perhaps when software is 100 years old, it will reach the level of reliability that we see in cars today.

I just don't want to have to wait that long.

REFERENCES

[1] Symantec Internet Security Report: http://www.symantec.com/business/theme.jsp?themeid=threatreport (see pp. 3–8 for the executive summary).

[2] Matt Bishop's paper on writing setuid programs: nob.cs.ucdavis.edu/bishop/secprog/1987-sproglogin.pdf.

[3] David Maynor's article about how he uncovered the flaw in the Apple Airport driver and created a proof of concept exploit: http://uninformed.org/index.cgi?v=8&a=4&p=2.

[4] "Apple Fixes Mac Wi-Fi Flaws": http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci1217510,00.html.

[5] Bug in pygrub used within Xen: http://bugzilla.xensource.com/bugzilla/show_bug.cgi?id=1068.

[6] S. Romig, M. Fullmer, and S. Ramachandran, "Cisco Flow Logs and Intrusion Detection at the Ohio State University": http://www.usenix.org/publications/login/1999-9/index.html.