The CyberWire
621 East Pratt Street, suite 300
Baltimore MD 21202–3140
editor@thecyberwire.com
www.thecyberwire.com
@thecyberwire

Jailbreak Security Summit 2015

May 8, 2015 Laurel, MD

To view the video coverage of this event visit:
http://www.thecyberwire.com/events/jailbreak-security-summit-2015.html

Apple Security Talks and Craft Beer

To our knowledge, this is the first security summit hosted by a production brewery, the Jailbreak Brewing Company of Laurel, Maryland. Jailbreak assembled a group of technical experts to discuss the not always well-understood world of Apple security.

Corporate co-sponsors of the event included CyberPoint International, Booz Allen Hamilton, FireEye, ClearShark, Novetta, Blackpoint Technologies, Endgame, and Synack.

Exploitation of Memory Corruption Bugs Utilizing the Objective-C Runtime on Mac OS X

First up was Nemo, a.k.a. Neil Archibald, a vulnerability researcher at Accuvant Labs. He discussed the exploitation of memory corruption bugs with Objective-C runtime on Mac OS X. Objective-C is language of choice on Mas OS X and iOS, and it's likely to be around for awhile.

It's rare to find a standalone information leak, and Nemo outlined ways of achieving remote execution without such a leak. (A video of his presentation, with full details, will be posted shortly.) Some of the exploit techniques he outlined included avenues newly opened by vendor upgrades.

He closed by suggesting to fellow white hats that, when they find a non-exploitable bug, to report it swiftly to the vendors…so we can achieve more stable exploits. (Alright — he was joking, but he does recommend reporting bugs.)

How to Practically Create Elegant, Bad@ss OS X Malware

Patrick Wardle of Synack began his presentation by noting that there are lots of unknown unknowns with respect to OS X malware. Mac's increasing market share is sure to draw attention on the malware black market, and to drive development of appropriate malware.

Even today, Mac malware, despite impressions to the contrary, is very much a reality. We often see nice exploits, but followed up by low-quality malware. The known status quo with respect to Macs remains that most infections require user interaction. The malware tends to hide in plain site: it may show some nice features, but for the most part it's caught by firewalls and similar defenses.

So how could Mac malware get better? Most of it currently depends on social engineering. But in theory, a network-level adversary could execute man-in-the-middle attacks against Mac systems. Apple Gatekeeper would help (a bit), but too many users depend upon Gatekeeper to protect downloads. Can we get around Gatekeeper? Most software (including security products) use http, not https, which offers attackers an opportunity.

Most persistence methods Mac malware exhibits, Wardle believes, are "very lame" — that is, easily detected (and inartistic). He then considered some ways in which Mac malware could be more effectively developed.

One technique available would be binary infection. Fairly stealthy and difficult to disinfect, binary infection may bypass PSPs. OS Loader, for example, looks for LC_Code_Signature, and protections could be bypassed by removing this block.

Another method would be dylib hijacking. There are a few techniques to accomplish dylib hijacking: you would need a vulnerable app. Configured for compatibility, malicious dylib could achieve persistence hosted in a trusted process.

Another technique is plugin persistence. Create a spotlight template, and then match type. This offers the advantage of involving no new process. It would abuse legitimate OS X functionality.

"What," Wardle asked rhetorically, "about malware self-protection? You don't see much in Mac malware (but you do in Windows)." Suppose you were to encrypt the Mach-O binaries (naively supported), create a padding function at the start of the binary, compile it, and lock the padding function at start.

But to do even better, you might consider strongly encrypting your malware for a specific target. "Gauss and Equation used this targeted technique on the Windows side."

"All OS X malware so far is pretty easily deleted," Wardle noted. "How can we make it hard to delete?" There are a variety of techniques available for active defense, and self-monitoring can be used to cue such defense. Well-constructed malware should monitor network communications for signs the malware is being uploaded (and it might then kill box). Or, you might simply use Google AdWords to monitor when someone's inquiring about your malware.

There's also load-time process injection, which is less complex and less easily detected. It's relatively easy to bypass Gatekeeper with this technique (despite the good job it does).

You can also bypass signature-based Xprotect AV tools by recompiling, writing anew, or simply renaming.

The OS X Sandbox can also be escaped: there are at least twenty bugs that permit this. It's possible to sidestep Apple's Rootpipe patch using a logic bug for privilege escalation.

You can even bypass capable behavior-based solutions like LittleSnitch.

Wardle has written (and is sharing) various free tools for security Macs. One is KnockKnock. Another is BlockBlock, which monitors known persistence locations. BlockBlock is particularly nice because of its low false-alsarm rate, and Wardle invited the audience to check back for other tools as he writes them.

In sum, while current OS X malware is "pretty lame," it could easily become a lot more capable.

To a question asking how many of these OS X techniques would work on iOS, Wardle said that in his estimation few of them would, "because iOS is so well locked-down."

What Does a "Secure" iOS App Look Like?

Chris Forant (of Booz Allen Hamilton) spoke to the security of iOS apps. Mobiles, he noted, have highly tailored OSs and "not a lot of 'alive' time," as well as different development paradigms.

iOS is itself different, and highly controlled. Its sandboxing is very good, its API usage controlled. iOS background modes are quite different from those on desktops, as are its code signings and entitlements. Mobile app review is important — "a whole other system is checking your binaries." Finally, trust – "by which I mean," he said, "device pairing"— is highly significant for iOS. Custom URL schemes, app extension (in iOS 8.0), and Airdrop govern sharing among apps.

After a consideration of the developer ecosystem — code signing, controlled API usage, review — Forant described trust, aka device pairing. "With trust, you can get access to sandbox content, and your app can be backed up. Without trust, you're able to charge your device. That's about it."

He moved to a discussion of Swift —"modern, expressive, fast, and safe by default." Swift is also compatible with Objective-C. And while that may bring Objective-C vulnerabilities, Swift handles type safety and mutability quite differently from the way Objective-C does. Swift is thus very constant, and safer. It can still call Objective-C things—Swift, remember, doesn't run on its own—and so developers should be aware: Objective-C is still a big part of many projects.

But much can go wrong, mostly stemming from developer unfamiliarity with security. Forant illustrated with a demonstration of the development of a fictitious secret-storing app he called "Keymaster," in homage to Ghostbusters. Watch the video of his presentation for an illustration of developer missteps and their corrections in testing.

(Note—this section was corrected and clarified on May 13, 2015.)

A Deep-Dive into the Many Flavors of IPC Available on OS X

Ian Beer, a security researcher for Google Project Zero, offered an overview of most IPC mechanisms on iOS/OS X. After a quick look at Mach Message fundamentals, he took a deep-dive into XPC services and the exploitation bugs therein. "We care about IPC because you probably get initial code execution is some sandbox in user space, but some things, like Abode reader, remain unsandboxed. An IPC mechanism stands between sandboxed and unsandboxed processes."

He described privilege escalation models. He also discussed unsafe versions of the XPC Services API. "The compiler won't help you, and neither will the developer—it's up to you to check."

He noted that "lame heap spray" works remarkably well for developing exploits. Preparing a reliable proof-of-concept sandbox escape took about half a day's work. "Dumb generational fuzzing is unlikely to make it that far, but it's trivial to do so manually."

Beer concluded with a call for stronger security—heap spraying shouldn't be that effective—and for the provision of more effective sandboxing.

How OS X Malware is Upping Its Game

CyberPoint's Dale Robson described Mac sandboxes and malware, considering the sandbox problem from the perspective of a reverse engineer. He asked the audience to consider how "basic irritants" interact with the Internet. He also asked them to consider controls during his presentation: privilege separation, signature checks, hardening, behavioral limitation, and containers.

Apple's mandatory access control is in some ways a throwback, he said. The OS will refuse to do what's not on its list. The Mac sandbox was originally called "Seatbelt," and "the idea of a sandbox is that it's a safe, isolated place for code to play."

"But in fact," Robson observed, "a sandbox is more like a jail, where programs live and can't do anything outside their cell." Typical limits include no access beyond the app's sub directory, inability to see other processes, ability to see only files individually selected by the user, restriction from using the network without permission, restriction from using devices, etc.

Some malicious code can ignore the sandbox, if only through additional links in the malware chain, but Robson noted that "Mac sandboxing is actually pretty effectively designed," and went on to describe the Mac sandbox architecture. Sometimes, the sandbox's jail cell feels too small to developers. But that, of course, is by design.

Robson reviewed examples of Mac malware: Flashback (2012), Mac Defender/Mac Protector (2011), Kitmos, Niqtana (2012), and Yontoo (2012). But he's found no public malware that escapes the sandbox. The attack surface is shrinking, and the number of steps in the chain to escape is growing. The reason for the lack of such publicly available sandbox escapes may lie in the economic realities of crimeware: "Sandbox escape may not make economic sense in the criminal marketplace."

He concluded with some observations. The IOKit interface could afford a large attack surface if it can be seen from the sandbox. The Mach message system poorly studied. These issues aside, the sandbox does a good job of keeping you safe. (A hacker taking over your app is in jail, too.)

A Full Disclosure of Masque Attack, Including More than Five Vulnerabilities.

Tao Wei and Zhaofeng Chen, of FireEye, described spearphishing against iOS and the Masque attacks. Spearphishing against iOS is easy, effective, flexible, and persistent, they said. They've found, however, few malware instances for non-jailbroken iOS devices. They outlined the details of four Masque attacks:

- Masque Attack 1: App Masque. Demonstrating iOS malicious upload.
- Masque Attack 2: URL Masque. Masque 2 involved inter-app communication. URL Masques on App Store, whether designed as a feature or inherited as a mistake.
- Masque Attack 3: Extension Masque. "Don't trust" bypassing and extension hijacking.

- Masque Attack 4, from August of last year, with no reported patch yet. Apple doesn't allow security vendors to implement system-level protections. And classic network security can't always protect mobile devices.

The Jailbreak Security Summit concluded appropriately with a happy hour in the host brewery. We'll be adding video and copies of presentations to this page as they become available.

the cyberwire

editor@thecyberwire.com
www.thecyberwire.com

@thecyberwire
+TheCyberWire

About The CyberWire

The CyberWire is supported by CyberPoint International and its community partners. We invite the support of other organizations with a shared commitment to keeping this informative service free and available to organizations and individuals across the globe.

Revised 8.4.15