Insecure IT

Vetting Mobile Apps

Steve Quirolgico, Jeffrey Voas, and Rick Kuhn, National Institute of Standards and Technology

Bell Labs conceived the notion of mobile networks in 1947 and Japan launched the first analog mobile network in 1979. The first commercial handheld phone in the US appeared in 1983 and cost over US$8,000 in today's dollars.[1] Rampant growth followed.

By December 2009, the US had 286 million mobile subscribers covering 91 percent of the population.[2] Other nations have been adding mobile devices at an even faster pace. The world contained some 4.6 billion active mobile phones at the start of 2010.[3] As of June 2010, China had 775 million mobile subscribers[4] covering 58 percent of the population. China's overall wireless subscribership was growing at a rate of 15.4 percent per year in the third quarter of 2009.[5] The overall Asian Pacific region houses 39 percent of the broadband market and supports 42 percent of all Internet users. This region also has the largest number of cell phone users in the world.[6] Furthermore, it's rumored that over 50 billion mobile devices will be in use worldwide in 2020.

But what about the apps that provide the functionality of devices? Hundreds of thousands of different apps for mobile devices are available through commercial stores and open source repositories. Billions of copies of these apps have been purchased, and this number is projected to increase dramatically. However, with this growth comes an increase in the spread of potentially dangerous security vulnerabilities. Because of an app's low cost and high proliferation, the threat of these vulnerabilities could be far greater than that of traditional computers.

Mobile App Vulnerabilities

Like traditional applications for desktop and laptop computers, mobile apps suffer from myriad security vulnerabilities. Many of these vulnerabilities are unintentional, caused by poor programming practices. App developers, for example, might fail to validate input from the web allowing adversaries to access protected files or hardcode passwords, allowing unauthorized access to user files by those with access to the app's source code.

Some security vulnerabilities arise from the programming languages with which apps are implemented. For example, Web-based languages including JavaScript and ActiveX have been used to inject client-side scripts into web pages viewed by other users allowing adversaries to bypass access controls. In addition, older programming languages such as C are prone to a host of well-known security vulnerabilities, such as buffer overflows, heap overflows, format string attacks, and integer overflows. Using third-party libraries, regardless of the implementation language, can also open up the potential for vulnerabilities, because the source code for such libraries often isn't available to the developer.

Some security vulnerabilities occur when sensitive data is transmitted to and from remote servers over unencrypted channels. In one recent incident, authentication tokens for users were transmitted wirelessly to mobile apps in plain text, allowing adversaries to steal sensitive user information by gaining access to the users' calendar, contact information, and private Web albums.[7]

Vulnerabilities can also be intentional and malicious—and hidden within a seemingly safe and legitimate app. Such malware often requests more privileges or permissions from the device's operating system than what's consistent with the app's described functionality. For example, a simple paint app with full Internet and GPS capabilities likely isn't consistent with its described functionality. This type of vulnerability has been used to develop rogue apps that aim to disrupt or deny operation, gather information leading to loss of privacy or exploitation, and gain unauthorized access to system resources.

Perhaps the most severe app vulnerabilities are those that exploit lax security of stored data. Often, apps must write sensitive data to storage but don't take measures to secure that data. This was recently the case with Android's Skype app, which left personal data—including name, emails, date of birth, and biographical information—as well as call logs unencrypted and easily accessible.

Vetting Apps

Purchasing organizations or third-party labs need to vet apps before selling them to help ensure their security. However, the vetting process poses several challenges, including specifying security and analysis requirements; identifying appropriate tools, mechanisms, and approaches for analyzing security vulnerabilities; and finding appropriate personnel to manually vet the apps.

Security Requirements

Specifying the security requirements is challenging due to the sheer number and types of potential vulnerabilities. Identifying app vulnerabilities requires understanding the details of known vulnerabilities, such as those described in the Common Weakness Enumeration (http://cwe.mitre.org). The CWE is a community-developed dictionary that provides a unified, measurable set of software weaknesses. It also describes the top 25 most dangerous software errors. Leveraging resources such as the CWE can facilitate the app-vetting process by providing a baseline from which to define a set of *target vulnerabilities*—that is, a specific set of vulnerabilities to look for in an app.

App vetting might also require vetting other components with which the app communicates, such as a third-party library, device, or server. Such vetting is typically challenging, because licensing issues can restrict access to third-party source code, and user privileges can restrict access to an app's associated server.

To foster the availability of only "safe” apps, it's also necessary to vet the app store. Testing security vulnerabilities such as cross-site scripting help to ensure that an attacker can't compromise the app store to modify or remove authorized apps. Such testing can also prevent sensitive information such as customer names, addresses, and credit card numbers from being leaked.

Analysis Requirements

After identifying the app's security requirements, you need to determine what types of analysis to conduct—*static* or *dynamic*—to detect the set of target vulnerabilities. Static analysis examines the app source code and reasons over all possible behaviors that might arise at runtime. It guarantees that analysis results are an accurate description of the program's behavior regardless of the input or execution environment.

Dynamic analysis operates by executing a program using a set of input use cases and analyzing the program's runtime behavior. In some cases, the enumeration of input use cases is large, resulting in a lengthy processing time. However, methods such as combinatorial testing[8] can reduce the number of input use-case combinations, reducing the amount of time needed to derive analysis results.

In addition to static and dynamic analyses, you can also use questionnaires to assess an app's security vulnerabilities. Such questionnaires, such those published by the US Department of Homeland Security (DHS), aim to identify security vulnerabilities by helping vendors address questions about their software. For example, vendors can use the DHS Custom Software Questionnaire[9] to answer questions such as, “Does your software validate inputs from untrusted resources?” and “What threat assumptions were made when designing protections for your software?” Another useful question (not included in the DHS questionnaire) is “Does your app access a network API?”

Analysis Tools

After defining the security and analysis requirements, you need to identify the suite of tools for detecting the app's target vulnerabilities, which is challenging because of the varied capabilities of diverse commercial and open source tools. The state of the art in static analysis today is that tools can find a significant portion of important software security weaknesses.[10] However, no single tool can detect all vulnerabilities and you'll likely need multiple tools.

The state of the art in static analysis today is that tools can find a significant portion of important software security weaknesses.[10] However, there's a great deal of variability among tools in the classes of vulnerabilities found, and no tool detects all weaknesses.

App Submission and Reporting

Before you can analyze an app, you need an infrastructure for testing the app and retrieving the results. In addition, if you're conducting the testing automatically, you must implement mechanisms to invoke test tools when submitting the app to the system.

Developing such an infrastructure will likely require integrating components such as a Web-based system for managing user accounts and apps, a database for storing apps and related analysis data, scripts for automatically invoking test tools, and mechanisms for sharing results with developers.

Human Analysis

Vetting an app using analysis tools will still require human expertise. This is particularly true in vetting false positives. In such cases, a detected vulnerability might be appropriate given a specific context. For example, a static analysis tool might identify a camera app that attempts to access a GPS API as having a security vulnerability. However, if the camera app is intended to log location information for pictures taken by a device, then the use of a GPS API is appropriate.

The challenges include finding human analysts who understand the source code, intended behavior, and operational context of every app they must analyze.

Currently, app stores do not incorporate a vetting process that thoroughly examines potential security vulnerabilities in the apps made available to consumers. The reason for this is likely due in part to the cost and time associated with vetting an app as well as the interactions required with developers to resolve potential vulnerabilities. Given the growing potential for dangerous and widespread vulnerabilities, however, it is becoming increasingly critical to vet apps for such vulnerabilities, but in a cost- and time-efficient manner. Although we have outlined a general approach for vetting mobile apps, more research is needed in determining how to reduce cost and expedite the vetting process.

Acknowledgments

We identify certain software products in this document, but such identification does not imply recommendation by the US National Institute for Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose. No US Government resources were used for this paper. It reflects the authors' personal opinions, not the opinions of the Department of Commerce or NIST.

References

1. "DYNA TAC Cellular Mobile Telephone, Instruction Manual," Motorola Corp., Feb. 1983.
2. "Wireless Industry Indices Report, 1985–2009," CTIA, Jan. 2010.
3. *Measuring the Information Society*, Int'l Telecommunication Union, 2010; www.itu.int/ITU-D/ict/publications/idi/2010.
4. M. Ramsay, "China's Mobile Subs Top 775 Million," *Wireless Week*, 21 June 2010; www.wirelessweek.com/News/2010/06/Carriers-China-Mobile-Subs-775M.
5. L. Collins, "One Billion Chinese Mobile Subscribers in 2014, Says Report," The Institution of Engineering and Technology, 1 Mar. 2010; http://kn.theiet.org/news/mar10/china-1bn.cfm.
6. V. Gray, "Asia-Pacific Telecommunication/ICT Indicators 2008 Broadband in Asia-Pacific: Too Much, Too Little?" Int'l Telecommunication Union, Sept. 2008; www.itu.int/pub/D-IND-AP-2008/en.
7. D. Goodin, "Security Shocker: Android Apps Send Private Data in the Clear," *The Register*, 24 Feb. 2011; www.theregister.co.uk/2011/02/24/android_phone_privacy_shocker.
8. D.R. Kuhn, Y. Lei, R. Kacker, "Practical Combinatorial Testing—Beyond Pairwise Testing," *IT Professional*, vol. 10, no. 3, 2008, pp. 19–23.
9. *Software Assurance (SwA) in Acquisition: Mitigating Risks to the Enterprise*, Appendix D, Dept. of Homeland Security, 2008; https://buildsecurityin.us-cert.gov/swa/acqart.html#ques.
10. V. Okun, A. Delaitre, and P. Black, "Second Static Analysis Tool Exposition," Nat'l Inst. Standards and Technology, June 2010; http://samate.nist.gov/docs/NIST_Special_Publication_500-287.pdf

Steve Quirolgico is a computer scientist at the US National Institute of Standards and Technology. His research interests include information security, software assurance, and policy-based security systems. Quirolgico received his PhD from the University of Maryland Baltimore County. Contact him at steveq@nist.gov..

Jeffrey Voas is a computer scientist at the US National Institute of Standards and Technology (NIST). He also serves as IEEE Division VI Director. Voas is a Fellow of IEEE. Contact him at j.voas@ieee.org.

Rick Kuhn is a computer scientist at the US National Institute of Standards and Technology. His research interests include information security, software assurance, and empirical studies of software failure. Kuhn has an MS in computer science from the University of Maryland College Park and an MBA from William & Mary, Mason School of Business. Contact him at kuhn@nist.gov.

//Digital Library abstract and keywords .//

Billions of apps for mobile devices have been purchased in recent years. With this growth, however, comes an increase in the spread of potentially dangerous security vulnerabilities. Because of an app's low cost and high proliferation, the threat of these vulnerabilities could be far greater than that of traditional computers. Thus, purchasing organizations or third-party labs should vet the apps before selling them, and consumers need to understand the risks of apps and the prospects for ensuring their security.

Keywords: mobile devices, apps, security, information technology