Draft NIST Special Publication 800-163
Revision 1

Vetting the Security of Mobile Applications

Michael Ogata
Josh Franklin
Jeffrey Voas
Vincent Sritapan
Stephen Quirolgico

C O M P U T E R S E C U R I T Y

National Institute of Standards and Technology
U.S. Department of Commerce

Draft NIST Special Publication 800-163 Revision 1

Vetting the Security of Mobile Applications

Michael Ogata
Software and Systems Division
Information Technology Laboratory

Josh Franklin
Applied Cybersecurity Division
Information Technology Laboratory

Jeffrey Voas
Computer Security Division
Information Technology Laboratory

Vincent Sritapan
Office of Science and Technology
U.S. Department of Homeland Security

Stephen Quirolgico
Office of the Chief Information Officer
U.S. Department of Homeland Security

July 2018

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Under Secretary of Commerce for Standards and Technology

Authority

This publication has been developed by NIST in accordance with its statutory responsibilities under the Federal Information Security Modernization Act (FISMA) of 2014, 44 U.S.C. § 3551 *et seq.*, Public Law (P.L.) 113-283. NIST is responsible for developing information security standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

National Institute of Standards and Technology Special Publication 800-163 Revision 1 Natl. Inst. Stand. Technol. Spec. Publ. 800-163 Rev. 1, 50 pages (July 2018)
CODEN: NSPUE2

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications.

Public comment period: *July 23, 2018* through *September 6, 2018*

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: nist800-163@nist.gov

All comments are subject to release under the Freedom of Information Act (FOIA).

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems. The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

Abstract

Mobile applications have become an integral part of our everyday personal and professional lives. As both public and private organizations rely more on mobile applications, securing these mobile applications from vulnerabilities and defects becomes more important. This paper outlines and details a mobile application vetting process. This process can be used to ensure that mobile applications conform to an organization's security requirements and are reasonably free from vulnerabilities.

Keywords

app vetting; app vetting system; malware; mobile applications; mobile security; niap; security requirements; software assurance; software vulnerabilities; software testing

Trademark Information

118
119 **Table of Contents**

List of Appendices

List of Figures

List of Tables

1 Introduction

Mobile applications (or *apps*) have had a transformative effect on organizations. Through ever-increasing functionality, ubiquitous connectivity and faster access to mission-critical information, mobile apps continue to provide unprecedented support for facilitating organizational objectives. Despite their utility, these apps can pose serious security risks to an organization and its users due to vulnerabilities that may exist within their software [1].Such vulnerabilities may be exploited to steal information, control a user's device, deplete hardware resources, or result in unexpected app or device behavior.

App vulnerabilities are caused by several factors including design flaws and programming errors, which may have been inserted intentionally or inadvertently. In the app marketplace, apps containing vulnerabilities are prevalent due in part to the submission of apps by developers who may trade security for functionality in order to reduce cost and time to market.

The level of risk related to vulnerabilities varies depending on several factors including the data accessible to an app. For example, apps that access data such as precise and continuous geolocation information, personal health metrics or personally identifiable information (PII) may be considered to be of higher-risk than those that do not access sensitive data. In addition, apps that depend on wireless network technologies (e.g., Wi-Fi, cellular, Bluetooth) for data transmission may also be of high risk since these technologies also can be used as vectors for remote exploits. Even apps considered low risk, however, can have significant impact if exploited. For example, public safety apps that fail due to a vulnerability exploit could potentially result in the loss of life.

To mitigate potential security risks associated with mobile apps, organizations should employ a software assurance process that ensures a level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner [2]. In this document, we define a software assurance process for mobile applications. We refer to this process as an *app vetting process*.

1.1 Purpose

This document defines an app vetting process and provides guidance on (1) planning and implementing an app vetting process, (2) developing security requirements for mobile apps, (3) identifying appropriate tools for testing mobile apps and (4) determining if a mobile app is acceptable for deployment on an organization's mobile devices. An overview of techniques commonly used by software assurance professionals is provided, including methods of testing for discrete software vulnerabilities and misconfigurations related to mobile app software.

[1] A vulnerability is defined as one or more weaknesses that can be accidentally triggered or intentionally exploited and result in a violation of desired system properties [1]

1.2 Scope

Software assurance activities for a mobile application may occur in one or more phases of the mobile application lifecycle: (1) during the development of the app by its developer (i.e., the app development phase), (2) during deployment of the app by the end-user organization (i.e., the app deployment phase) or (3) after receiving a developed app but prior to its deployment by the end-user organization (i.e., the app acquisition phase). These three phases of the mobile application lifecycle are shown in Figure 1.

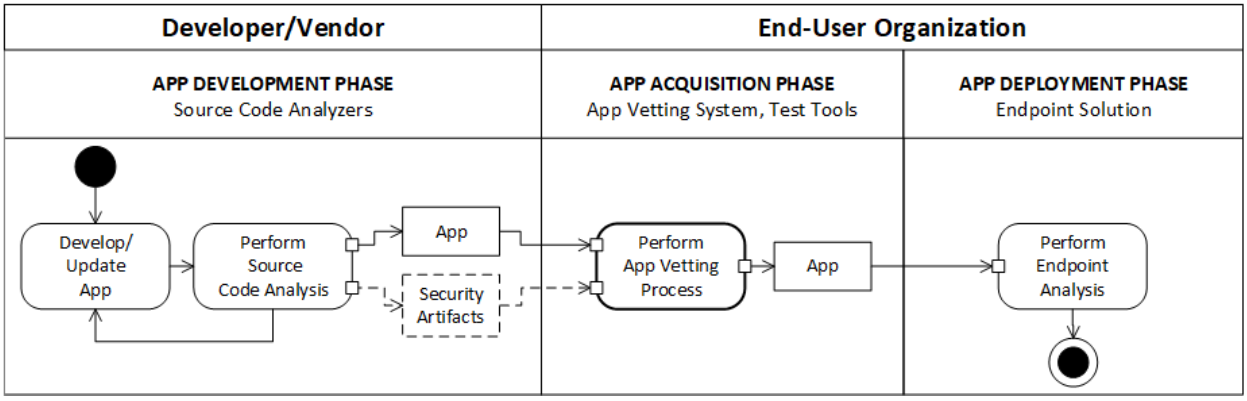

Figure 1 - Software assurance during mobile application lifecycle.

In this document, we focus primarily on the software assurance activities of the app vetting process, which we define as part of the app acquisition phase of the mobile application lifecycle. Thus, software assurance activities performed during the app's development phase (e.g., by source code analyzers) or during the app's deployment phase (e.g., by endpoint solutions) are considered out of scope for this document.

In addition, this document does not address the use of Enterprise Mobility Management (EMM), mobile app management or mobile threat defense systems, although integrations with these systems are briefly examined. Further, this document does not discuss vetting the security of Internet of Things (IoT) apps or address the security of underlying mobile platforms and operating systems. These subjects are addressed in other publications [3]–[5]. Finally, discussion surrounding the security of web services and cloud infrastructures used to support backend processing of apps is also out of scope for this document.

1.3 Intended Audience

This document is intended for public- and private-sector organizations that seek to improve the software assurance of mobile apps deployed on their mobile devices. More specifically, this document is intended for those who are:

- Responsible for establishing an organization's mobile device security posture,
- Responsible for the management and security of mobile devices within an organization,

- Responsible for determining which apps are used within an organization, and
- Interested in understanding what types of assurances the app vetting process provides.

1.4 Document Structure

The remainder of this document is organized into the following sections:

- Section 2—App Security Requirements
- Section 3—App Vetting Process
- Section 4—App Testing Approaches and Vulnerability Classifiers
- Section 5—App Vetting Considerations
- Section 6—App Vetting Systems
- Appendix A—Threats to Mobile Applications
- Appendix B—Android App Vulnerability Types
- Appendix C— iOS App Vulnerability Types
- Appendix D—Acronyms and Abbreviations
- Appendix E—Glossary
- Appendix F—References

1.5 Document Conventions

Applications written specifically for a mobile platform are referred to as “apps” throughout this special publication.

2 App Security Requirements

Before vetting a mobile app for security, an organization must define the security requirements that an app must meet in order to be approved by the organization. In this document, we define two types of app security requirements that organizations should satisfy: *general* and *organization-specific*.

2.1 General Requirements

General app security requirements define the software and behavioral characteristics of an app that should or should not be present in order to ensure the security of the app. These requirements are considered "general" since they can be applied across all mobile applications. General app security requirements may be derived from a number of available standards, best practices, and resources including those specified by NIAP, OWASP, MITRE and NIST[2].

2.1.1 National Information Assurance Partnership (NIAP)

The NIAP Protection Profiles (PPs) specify an implementation-independent set of security requirements for a category of IT products that meet specific consumer needs. Specifically, the NIAP PPs are intended for use in certifying products for use in conjunction with national security systems to meet a defined set of security requirements. Furthermore, the NIAP PPs define in detail the security objectives, requirements and assurance activities that must be met for a product evaluation to be considered ISO/IEC 15408 certified [6]. For application software vetting, including mobile app vetting, NIAP has defined the Protection Profile for Application Software [7].

The requirements defined in the NIAP PP for Application Software are divided into two broad categories:

1) Functional Requirements—Declarations concerning the required existence or absence of particular software behavior or attributes.
2) Assurance Requirements—Declarations concerning actions the evaluator must take or stipulations that must be true for vetting to successfully execute.

Table 1 summarizes the NIAP functional requirements[3].

[2] Additional threats and vulnerabilities can be found in Appendices A, B, and C.

[3] For brevity, many, but not all the functional requirements are listed in Table 1. Some are high-level descriptions of multiple related controls. See NIAP Protection Profile for the full list [7].

Table 1 - NIAP Functional Requirements.

Functional Requirements
Access to Platform Resources
Anti-Exploitation Capabilities
Cryptographic Key Functionality
Cryptographic Operations
Encryption of Sensitive Application Data
HTTPS Protocol
Integrity for Installation and Update
Network Communications
Protection of Data in Transit
Random Bit Generation
Secure by Default Configuration
Software Identification and Versions
Specification of Management Functions
Storage of Credentials
Supported Configuration Mechanism
Transport Layer Security Operations
Use of Supported Services and Application Programming Interfaces
Use of Third-Party Libraries
User Consent for Transmission of Personally Identifiable Information
X.509 Functionality

The Assurance Requirement found in the protection profile can be summarized as follows:

- The application shall be labeled with a unique reference.
- The evaluator shall test a subset of the Target of Evaluation (TOE) security functions (TSF) to confirm that the TSF operates as specified.
- The application shall be suitable for testing (free from obfuscation)
- The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE.

2.1.2 OWASP Mobile Risks, Controls and App Testing Guidance

The Open Web Application Security Project (OWASP) maintains multiple useful resources concerning mobile app testing and security. Their Mobile Application Security Verification Standard (MASVS)[8] is a detailed model for mobile app security that can be used to provide baseline security requirements for an organization. Like the NIAP PP, the MASVS defines a set of declarations concerning the structure and behavior of an app. However, the MASVS also defines three verification levels:

- Standard Security (Level 1)
- Defines in Depth (Level 2)

- Resilience against Reverse Engineering and Threats (Level 3).

Each level's control lists are divided into the categories listed below, with the object described for each control depending on the desired verification level:

- Architecture, Design, and Threat Modeling Requirements
- Data Storage and Privacy Requirements
- Cryptography Requirements
- Authentication and Session Management Requirements
- Network Communication Requirements
- Platform Integration Requirements
- Code Quality and Build-Setting Requirements
- Resilience Requirements

The OWASP Mobile Security Testing Guide (MSTG) [9] is a manual for testing the security of mobile apps. It describes the technical processes for verifying the requirements listed in the MASVS.

2.1.3 MITRE App Evaluation Criteria

In 2016, the MITRE Corporation (MITRE) performed an analysis of the effectiveness of mobile app security vetting solutions for helping enterprises automate portions of their vetting process. To perform the analysis, MITRE developed solution criteria based on NIAP's Protection Profile for Application Software as well as additional criteria to address broader app vetting solution capabilities, threats against the app vetting solution itself, and other common mobile app vulnerabilities and malicious behaviors.

Using its criteria, MITRE developed or obtained multiple vulnerable and malicious-appearing apps for use in assessing mobile app vetting solutions. MITRE used the apps to test the capabilities of mobile app vetting solutions.

MITRE published a technical report [10] describing their methodology, evaluation criteria, test applications and overall results from analyzing then-available solutions. The report and test applications are available on MITRE's GitHub site

2.1.4 NIST SP 800-53

NIST Special Publication 800-53 [5] provides an exhaustive catalog of security and privacy controls designed for federal information systems. In addition, the document defines a process for selecting controls to defend IT systems, individuals and other organizational assets from a variety of threats, such as hostile cyber-attacks, natural disasters, structural failures and human errors. The controls can be customized to an organization-specific process to manage information security and privacy risk. The controls can support a diverse set of security and

privacy requirements across an organization's required policies, standards, and/or business needs. A set of three security control baseline are provided based on high, medium and low impact. Going further, the publication also describes how to develop specialized sets of controls, also known as control overlays, that can be tailored for unique, or specific types of missions/business functions and technologies. The NIST 800-53 security controls can addresses privacy and security from a functionality perspective (the strength of security functions and mechanisms provided) and an assurance perspective (the measures of confidence in the implemented security capability). Addressing both security functionality and security assurance ensures that information technology products and the information systems built from those products using sound systems and security engineering principles are sufficiently trustworthy.

2.2 Organization-Specific Requirements

Organization-specific security requirements define the policies, regulations and guidance that an organization must follow to ensure the security posture of the organization. Examples include banning social media apps from installation on the organization's mobile devices and apps developed by specific vendors cannot be installed on the organization's mobile devices.

To help develop organization-specific security requirements, it is helpful to identify non-vulnerability-related factors that can impact the security posture of mobile apps. Such factors can be derived by considering the criteria as shown in Table 2.

Table 2 - Organization-specific security criteria.

Criterion	Description	
Policies	The security, privacy and acceptable use policies; social media guidelines; and regulations applicable to the organization.	
Provenance	Identity of the developer, developer's organization, developer's reputation, consumer reviews, etc.	
Data Sensitivity	The sensitivity of data collected, stored, or transmitted by the app.	
App Criticality	The level of importance the app is to the organization's business.	
Target Users	The app's intended set of users from the organization.	
Target Hardware	The intended hardware platform, operating system, and configuration on which the app will be deployed.	
Target Environment	The intended operational environment of the app (e.g., general public use vs. sensitive military environment).	
Digital Signature	Digital signatures applied to the app binaries, libraries, or packages.	
App Documentation	User Guide	When available, the app's user guide assists testing by specifying the expected functionality and expected behaviors. This is simply a statement from the developer describing what they claim their app does and how it does it.
	Test Plans	Reviewing the developer's test plans may help focus app vetting by identifying any areas that have not been tested or were tested inadequately. A developer could opt to submit a test oracle in certain situations to demonstrate its internal test effort.
	Test Results	Code review results and other testing results will indicate which security standards were followed. For example, if an app threat model was created, this standard should be submitted. It will list weaknesses that were identified and should have been addressed during app design and coding.

	Service-Level Agreement	If an app was developed for an organization by a third-party, a Service-Level Agreement (SLA) may have been included as part of the vendor contract. This contract should require the app to be compatible with the organization's security policy.

Some information can be gleaned from app documentation in certain cases, but even if documentation does exist it might lack technical clarity and/or use jargon specific to the circle of users who would normally purchase the app. Since the documentation for different apps will be structured in different ways, it may also be time-consuming to find this information for evaluation. Therefore, a standardized questionnaire might be appropriate for determining the software's purpose and assessing an app developer's efforts to address security weaknesses. Such questionnaires aim to identify software quality issues and security weaknesses by helping developers address questions from end-users/adopters about their software development processes. For example, developers can use the Department of Homeland Security (DHS) Custom Software Questionnaire [11] 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 application programming interface (API)?"* Note that such questionnaires can be used only in certain circumstances such as when source code is available and when developers can answer questions.

Known flaws in app design and coding may be reported in publicly accessible vulnerability databases such as the U.S. National Vulnerability Database (NVD).[4] Before conducting the full vetting process for a publicly available app, analysts should check one or more vulnerability databases to determine if there are known flaws in the corresponding version of the app. If one or more serious flaws already have been discovered, this finding alone might be sufficient grounds to reject the version of the app for organizational use, thus allowing the rest of the vetting process to be skipped. However, in most cases such flaws will not be known and the full vetting process will be needed. This necessity is because there are many forms of vulnerabilities other than known flaws in app design and coding. Identifying these weaknesses necessitates first defining the app requirements, so that deviations from these requirements can be flagged as weaknesses.

In some cases, an organization will have no defined organization-specific requirements. As a result, analysts will evaluate the security posture of the app based solely on reports and risk assessments from test tools.

Note that the satisfaction or violation of an organization-specific requirement is not based on the presence or absence of a software vulnerability and thus cannot typically be determined by test tools. Instead, the satisfaction or violation of organization-specific requirements must be determined manually by an analyst.

[4] Vulnerability databases generally reference vulnerabilities by their Common Vulnerabilities and Exposures (CVE) identifier. For more information about CVE, see [12].

2.3 Risk Tolerance

Risk tolerance is the level of risk or degree of uncertainty that is acceptable to an organization. An organization's risk tolerance level is the amount of data and systems that can be risked to an acceptable level. A defined risk tolerance level identifies the degree to which an organization should be protected against confidentiality, integrity or availability compromise.

Risk tolerance should take into account the following factors:

- Compliance with security regulations, recommendations and best practices
- Privacy risks
- Security threats
- Data and asset value
- Industry and competitive pressure
- Management preferences

Risk tolerance is usually categorized by three levels: High, Moderate and Low. These categories are described in Table 3.

Table 3 - Risk Tolerance Categories.

Criterion	HIGH	MODERATE	LOW
Critical domain or market vertical (e.g., Financial, Government, Health Care)	No	Some	Yes
Security Compliance Requirements	None	Some	Multiple, Strict
Sensitive Data	No	Some	Yes
Customer Expectation of Strong Security Controls Requirements	No	Some	Yes
Priority is innovation or revenue before security	Yes	Some	No
Organization has or uses remote locations	No	Some	Multiple

2.3.1 Tool Report Analysis

One issue related to report and risk analysis stems from the difficulty in collating, normalizing and interpreting different reports and risk assessments due to the wide variety of security-related definitions, semantics, nomenclature and metrics used by different test tools. For example, one

test tool may classify the estimated risk for using an app as low, moderate, high or severe risk, while another may classify the estimated risk as pass, warning or fail. While some standards exist for expressing risk assessment[5] and vulnerability reporting[6] the current adoption of these standards by test tools is low. To the extent possible, it is recommended that an organization use test tools that leverage vulnerability reporting and risk assessment standards. If this approach is not possible, it is recommended that the organization provide sufficient training to analysts on the interpretation of reports and risk assessments generated by test tools.

2.3.2 Compliance versus Certification

For mobile application vetting, two terms are frequently used to demonstrate proof of successful implementation of mobile app security requirements. For a mobile application that has been developed to include security aimed at a particular requirement (e.g. National Information Assurance Partnership – Protection Profile for Mobile App Vetting v.1.2) developers may choose to note that they are compliant or certified. The difference depends on the organizations need for compliance or certification.

Compliance for mobile application security would mean either self-attestation or attestation from an unofficial third party that has validated the mobile app meets such security requirements. For example an enterprise may choose to use their own internally developed mobile application vetting process to validate the security and privacy of a mobile application. By going through their own internal process they are approve the mobile application for use in their organization or on their organization's mobile asset.

On the other hand, certification means successful validation from the authorized validator. For example, for NIAP certification, a formal NIAP validation process must be followed. See https://www.niap-ccevs.org/Ref/Evals.cfm. In this case, vendors may choose from an approved Common Criterial Testing Lab to conduct the product evaluation against an applicable NIAP-approved Protection Profile. Following successful completion of the validation process, a formal certification would be granted and listed on an approved product list.

Note: NIAP lists products on a product-compliant list when a certification has been successfully granted. This is an official list and requires NIAP's official certification.

[5] An example standard, the Common Vulnerability Scoring System CVSS, is discussed in Section 4.2.3

[6] Examples are described in Section 2.1

3 App Vetting Process

An app vetting process is a sequence of activities performed by an organization to determine if a mobile app conforms to the organization's app security requirements[7]. If an app is found to conform to the organization's app security requirements, the app is typically accepted for deployment on the organization's devices. An overview of the app vetting process is shown in Figure 2.

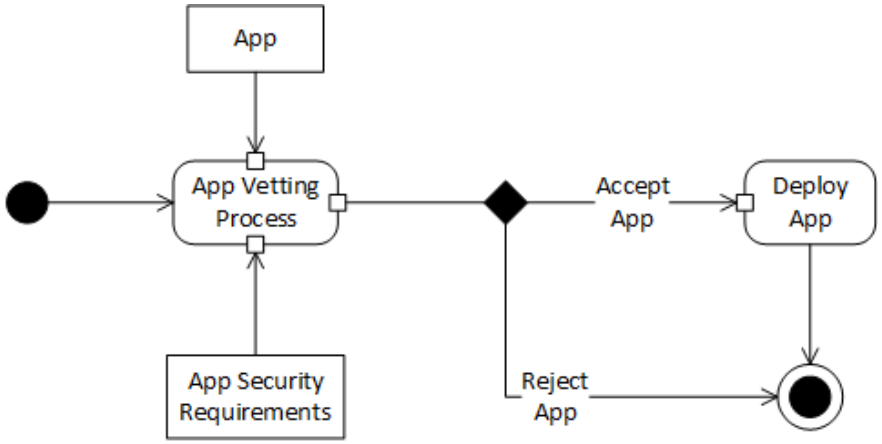

Figure 2 - App vetting process overview.

Although app vetting processes may vary among organizations, each instance of the process should be repeatable, efficient and consistent. The process should also limit errors to the extent possible (e.g., false-positive results). Typically, an app vetting process is performed manually or by an app vetting system that manages and automates all or part of the app vetting activities [13]. As part of an app vetting system, one or more test tools may be used to analyze an app for the existence of software vulnerabilities or malicious behavior consistent with malware.

As shown in Figure 1, organizations perform an app vetting process during the app acquisition phase of a mobile application lifecycle; that is, when the app is received by the organization but prior to the app's deployment on the organization's devices. The rationale for this approach stems from the fact that while developers may perform their own software assurance processes on an app, there is no guarantee the app will conform to an organization's security requirements. Furthermore, because testing of the app by the developer occurs outside the vetting process, an organization must trust the work of these previously-performed assurance activities. Organizations should not assume an app has been fully vetted or conforms to their security requirements simply because it is available through an official app store.

[7] An app vetting process also can be used to assess other issues including reliability, performance and accessibility, but is primarily intended to assess security-related issues.

Performing an app vetting process prior to deployment on a mobile device affords certain benefits including rigorous and comprehensive analysis that can leverage scalable computational resources. Furthermore, since testing occurs before deployment, the vetting process is not limited by timing constraints for remediating discovered threats. However, while this document focuses on the vetting of mobile apps during the organization's app acquisition phase, NIST recommends organizations also perform security analysis during the deployment phase using, for example, an endpoint solution on a mobile device.

An app vetting process comprises four sub-processes: app intake, app testing, app approval/rejection, and results submission processes. These processes are shown in Figure 3.

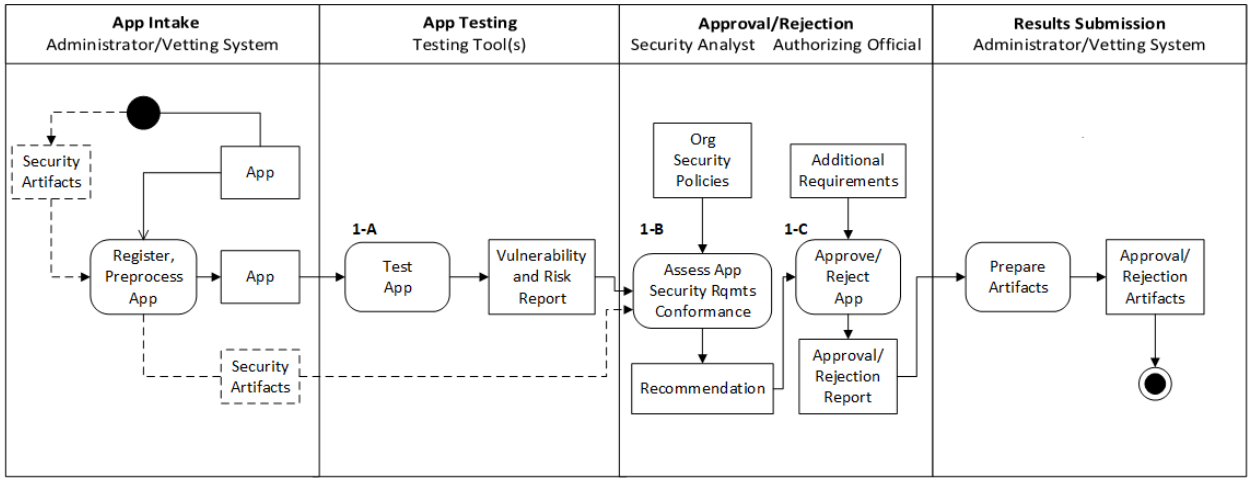

Figure 3 - Four sub-processes of an app vetting process.

3.1 App Intake

The app intake process begins when an app is received for analysis. This process is typically performed manually by an organization administrator or automatically by an app vetting system. The app intake process has two primary inputs: the app under consideration (required) and additional testing artifacts such as reports from previous app vetting results (optional).

After receiving an app, the app may be registered by recording information about the app including developer information, time and data of submission, and any other relevant information needed for the app vetting process. After registration, an app may also be preprocessed. Preprocessing typically involves decoding or decompiling the app to extract required meta-data (e.g., app name, version number) and to confirm that the app can be properly decoded or decompiled since test tools may need to perform this operation prior to performing their analyses.

In addition to the app itself, the app developer may optionally provide software assurance artifacts including previous security analysis reports. It should be noted that organizations

accepting these artifacts must accept the validity and integrity of app quality statements made by the artifacts at the word of the app developer.

3.2 App Testing

The app testing process beings after an app has been registered and preprocessed and is forwarded to one or more test tools. A test tool is a software tool or service that tests an app for the presence of software vulnerabilities[8]. Such testing will involve the use of different analysis methodologies (e.g., static analysis) and may be performed manually or automatically. Note that the tests performed by a test tool may identify software vulnerabilities that are common across different apps and will often satisfy general app security requirements (such as those specified by NIAP).

After testing an app, a test tool will generate a report that identifies any detected software vulnerabilities or potentially harmful behaviors. Additionally, the report typically will include a score that estimates the likelihood that a detected vulnerability or behavior will be exploited and the impact the detected vulnerability may have on the app or its related device or network. Note that a test tool may generate a report that conforms to an existing standard such as NIAP. Further note that some test tools will be able to detect violations of general app security requirements but not violations of organization-specific policies, regulations, etc.

Figure 4 shows the workflow for a typical test tool. When an app is received by a test tool, it is typically saved as a file on the tool vendor's server. If the test tool is static (i.e., the app's code is analyzed), the app is typically decoded, decompiled or decrypted from its binary executable form to an intermediate form that can be analyzed.[9] If the test tool is dynamic (i.e., the run-time behavior of the app is analyzed), the app is typically installed and executed on a device or emulator where the behavior of the app can be analyzed. After the tool analyzes the app, it generates a vulnerability report and risk assessment and submits this report to the app vetting system.

[8] Section 4 describes techniques and approaches used by app vetting tools.

[9] Typically, decoded or decompiled code does not result in source code, but rather an intermediate code that can be analyzed.

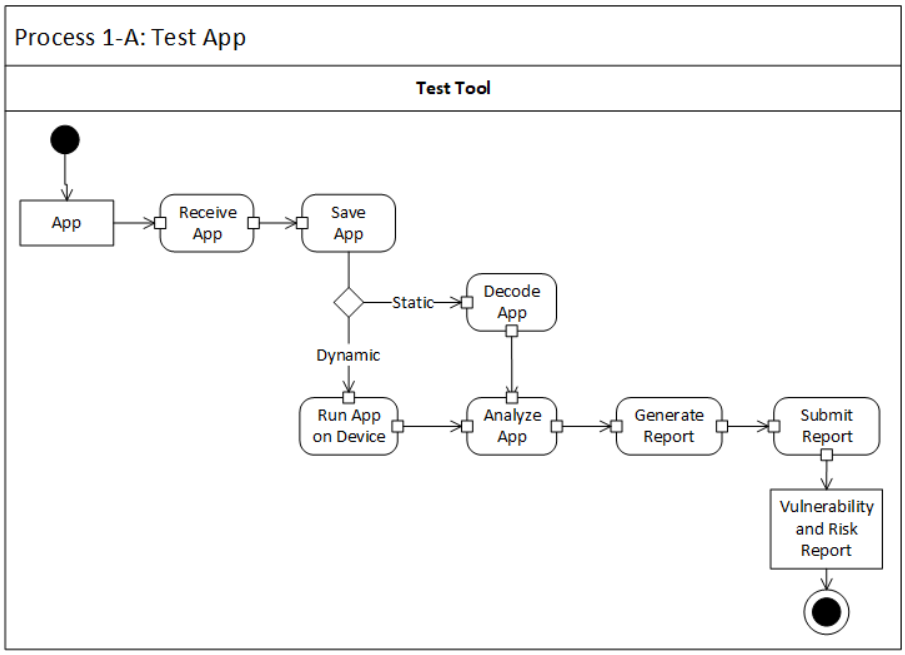

Figure 4 - Test tool workflow.

3.3 App Approval/Rejection

The app approval/rejection process beings after a vulnerability and risk report is generated by a test tool and made available to one or more security analysts. A security analyst (or *analyst*) inspects vulnerability reports and risk assessments from one or more test tools to ensure that an app meets all general app security requirements. An analyst will also evaluate organization-specific app security requirements to determine if an app violates any security policies or regulations. After evaluating all general and organization-specific app security requirements, an analyst will collate this information into a report that specifies a recommendation for approving or rejecting the app for deployment on the organization's mobile devices.

The recommendation report from an analyst is then made available to an authorizing official, who is a senior official of the organization responsible for determining which apps will be deployed on the organization's mobile devices. An authorizing official decides the approval or rejection of an app using the recommendations provided by the analysts and also considers other organization-specific, but non-security related criteria including cost, need, etc. These reports describe the app's security posture as well as possibly other non-security-related requirements. The organization's official approval or rejection is specified in a final approval/rejection report. Figure 5 shows the app approval/rejection process.

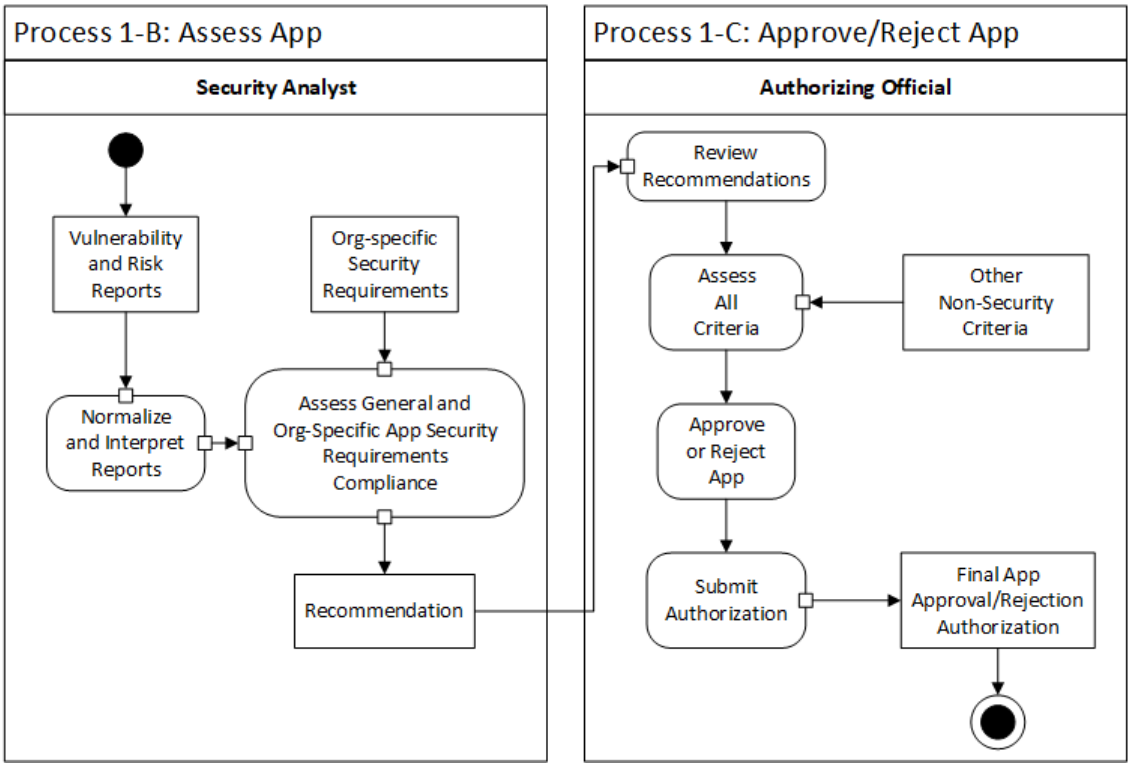

Figure 5 - App approval/rejection process.

3.4 Results Submission

The results submission process begins after the final app approval/rejection report is finalized by the authorizing official and artifacts are prepared for submission to the requesting source. These artifacts may include the final approval/rejection report, test tool reports and possibly a digitally signed version of the app that indicates the app has completed the app vetting process. The use of a digital signature provides source authentication and integrity protection, attesting that the version of the analyzed app is the same as the version that was initially submitted and was not unknowingly modified.

4 App Testing and Vulnerability Classifiers

During the app testing process, test tools are used to test for the existence of app vulnerabilities and malicious behavior. Often, such tools are based on standards such as NIAP and thus, may be used to used determine the satisfaction of general app security requirements. This section covers some of the strategies and approaches used by test tools and services to analyze mobile apps for vulnerabilities. It also describes various classifiers and quantifiers used to describe vulnerabilities.

4.1 Testing Approaches

Test tools employ several different analysis techniques including correctness testing, analysis of source code or binary code, use of static or dynamic analysis, and manual or automatic app testing.

4.1.1 Correctness Testing

One approach for testing an app is *software correctness testing* [14]. Software correctness testing is the process of executing a program to detect errors. Although the objective of software correctness testing is improving quality assurance as well as verifying and validating described functionality or estimating reliability, it also can help reveal potential security vulnerabilities that often can have a negative effect on the quality, functionality and reliability of the software. For example, software that crashes or exhibits unexpected behavior is often indicative of a security flaw. A prime advantage of software correctness testing is that it is traditionally based on specifications of the software to be tested. These specifications can be transformed into requirements that specify how the software is expected to behave while undergoing testing. This is distinguished from security assessment approaches that often require the tester to derive requirements themselves; often such requirements are largely based on security requirements that are common across many different software artifacts and may not test for vulnerabilities that are unique to the software under test. Nonetheless, because of the tight coupling between security and quality, and functionality and reliability, it is recommended that software correctness testing be performed when possible.

4.1.2 Source and Binary Code Testing

A major factor in performing app testing is whether source code is available. Typically, apps downloaded from an app store do not come with access to source code. When source code is available, such as in the case of an open-source app, a variety of tools can be used to analyze it. The goals of a source code review are to find vulnerabilities in the source code and to verify the results of test tools. Even with automated aids, the analysis is labor-intensive. Benefits to using automated static analysis tools include introducing consistency between different reviews and making possible reviews of large codebases. Reviewers should generally use automated static analysis tools whether they are conducting an automated or a manual review and they should express their findings in terms of Common Weakness Enumeration (CWE) identifiers or some other widely accepted nomenclature. Performing a secure code review requires software development and domain-specific knowledge in the area of app security. Organizations should ensure the individuals performing source code reviews have the required skills and expertise.

Organizations that intend to develop apps in-house also should refer to guidance on secure programming techniques and software quality assurance processes to appropriately address the entire software development lifecycle [15], [16].

When an app's source code is not available, its binary code can be analyzed instead. In the context of apps, the term "binary code" can refer to either byte-code or machine code. For example, Android apps are compiled to byte code that is executed on a virtual machine, similar to the Java Virtual Machine (JVM), but they can also come with custom libraries that are provided in the form of machine code, i.e., code executed directly on a mobile device's CPU. Android binary apps include byte-code that can be analyzed without hardware support using emulated and virtual environments.

4.1.3 Static and Dynamic Testing

Analysis tools are often characterized as either static or dynamic.[10] Static analysis examines the app source code and binary code and attempts to reason all possible behaviors that might arise at runtime. It provides a level of assurance that analysis results accurately describe 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 test cases is large, resulting in lengthy processing times. However, methods such as combinatorial testing can reduce the number of dynamic input test case combinations, reducing the amount of time needed to derive analysis results [18]. However, dynamic analysis is unlikely to provide 100 percent code coverage [19]. Organizations should consider the technical tradeoff differences between what static and dynamic tools offer and balance their usage given the organization's software assurance goals.

Static analysis requires that binary code be reverse engineered when source code is not available, which is relatively easy for byte code[11] but can be difficult for machine code. Many commercial static analysis tools already support bytecode as do a number of open-source and academic tools.[12] For machine code, it is especially hard to track the flow of control across many functions and to track data flow through variables, since most variables are stored in anonymous memory locations that can be accessed in different ways. The most common way to reverse engineer machine code is to use a disassembler or a decompiler that attempts to recover the original source code. These techniques are especially useful if the purpose of reverse engineering is to allow humans to examine the code because the outputs are in a form that can be understood by humans with appropriate skills. But even the best disassemblers make mistakes [21] and some of those can be corrected with formal static analysis. If the code is being reverse engineered for static analysis, it is preferable to disassemble the machine code directly to a form that the static analyzer understands rather than creating human-readable code as an intermediate byproduct. A static analysis tool aimed at machine code is likely to automate this process.

---

[10] For mobile devices, there are analysis tools that label themselves as performing behavioral testing. Behavioral testing (also known as behavioral analysis) is a form of static and dynamic testing that attempts to detect malicious or risky behavior such as the oft-cited example of a flashlight app that accesses a contact list [17]. This publication assumes that any mention of static or dynamic testing also includes behavioral testing as a subset of its capabilities.

[11] The ASM framework [20] is a commonly used framework for byte code analysis.

[12] Such as [20]–[23].

In contrast to static analysis, the most important dynamic analysis requirement is to see the workings of the code as it is being executed. There are two primary ways to obtain this information. First, an executing app can be connected to a remote debugger. Second, the code can be run on an emulator that has built-in debugging capabilities. Running the code on the intended mobile device allows the test tool to select the exact characteristics of the device and can provide a more accurate view about how the app will be behave. On the other hand, an emulator provides more control, especially when the emulator is open-source and can be modified by the evaluator to capture whatever information is needed. Although emulators can simulate different devices, they do not simulate all of them and therefore the simulation may not be completely accurate. Note that malware increasingly detects the use of emulators as a testing platform and changes its behavior accordingly to avoid detection. Therefore, it is recommended that test tools use a combination of emulated and physical mobile devices to avoid false-negatives from malware that employs anti-detection techniques.

Useful information can be gleaned by observing an app's behavior even without knowing the purposes of individual functions. For example, a test tool can observe how the app interacts with its external resources, recording the services it requests from the operating system and the permissions it exercises. Although many of the device capabilities used by an app may be inferred by a test tool (e.g., access to a device's camera will be required of a camera app), an app may be permitted access to additional device capabilities that are beyond the scope of its described functionality (e.g., a camera app accessing the device's network). Moreover, if the behavior of the app is observed for specific inputs, the evaluator can ask whether the capabilities being exercised make sense in the context of those particular inputs. For example, a calendar app may legitimately have permission to send calendar data across the network to sync across multiple devices, but if the user merely has asked for a list of the day's appointments and the app sends data that is not part of the handshaking process needed to retrieve data, the test tool might investigate what data is being sent and for what purpose.

4.2 Vulnerability Classifiers and Quantifiers

It is advantageous to use a common language to describe vulnerabilities in mobile apps. The following sections describe some of the more commonly used classifiers and quantifiers used to identify, describe, and measure the severity of vulnerabilities.

4.2.1 Common Weakness Enumeration (CWE)

CWE is a software weakness classification system maintained by the MITRE Corporation [24]. CWE serves as a common language of sorts for software weakness categories. Different programming languages can create language-specific versions of the same software error. CWE ensures terminology exists to refer to the same error across disparate languages and offers mitigation strategies for each. The CWE is used worldwide in industry, government and academia.

4.2.2 Common Vulnerability and Exposures (CVE)

The CVE dictionary is a naming scheme for software vulnerabilities [44] that also is hosted by MITRE. When a vulnerability is identified, it can be reported to a CVE Numbering Authority, which provides a unique, industrywide identifier for the vulnerability. CVEs are reported to the

NVD for scoring and description. The NVD is the U.S. government repository of standards-based vulnerability management data and collects, analyzes and stores data describing specific computer system vulnerabilities. Additionally, the NVD hosts databases of security checklists, security-related software flaws, misconfigurations, product names, and impact metrics. NVD extensively uses the CWE as well as the CVE to accomplish its mission.

4.2.3 Common Vulnerability Scoring System (CVSS)

The Common Vulnerability Scoring System Version (CVSS) is a vulnerability scoring system owned and maintained by the Forum of Incident Response and Security Teams (FIRST) [25]. The CVSS model attempts to ensure repeatable and accurate measurement, while enabling users to view the underlying vulnerability characteristics used to generate numerical scores. This common measurement system can be used by industries, organizations and governments that require accurate and consistent vulnerability exploit and impact scores. The algorithm used to calculate vulnerability scores is open to all and is derived principally by human analyst-provided inputs for three metric categories: base, temporal and environmental. Common uses of CVSS are calculating the severity and prioritization of vulnerability remediation activities. The NVD provides vulnerability scores via the CVSS.

5 App Vetting Considerations

This section describes additional criteria that organizations should consider when establishing their app vetting processes.

5.1 Managed and Unmanaged Apps

Enterprise applications, or third-party applications deployed on enterprise devices (or user's devices used for enterprise tasks), may be managed throughout the deployment lifecycle, from initial deployment and configuration through removal of the app from a device. Administering such managed applications can be performed using enterprise Mobile Application Management (MAM) systems which are designed to enable enterprise control over mobile applications that access enterprise services and/or data. Unmanaged applications are applications that are not administered by MAM (or similar) systems.

One benefit of managing only applications (as opposed to the entire device) is that MAM systems do not require the user/owner to enroll the entire device under enterprise management, nor must the owner accept installation of an enterprise profile on the device. MAM solutions can enable an enterprise to integrate an in-house enterprise applications catalog with a mobile device vendor's App Store (e.g., Apple's App Store, Google Play, or the Microsoft Store) to allow mobile users to easily install an enterprise app. Enterprise system administrators may be able to deploy apps or push out over-the-air app updates to mobile users; they may also be able to restrict app functionalities without affecting the entire device, which may be preferred by Bring Your Own Device (BYOD) users. Some Mobile Device Management (MDM) systems also include MAM functionality, enabling fine grained control over different applications on a single managed device.

An enterprise should consider the tradeoffs between managed and unmanaged apps when designing its mobility solutions, requirements, and policies for managing mobile applications (examples of such security requirements can be found in the DOD memo on "Mobile Application Security Requirements" [26]). Tradeoffs may include the administrative overhead and extra cost versus the security guarantees obtained by allowing only managed apps on mobile devices that access enterprise networks and services.

5.2 App Vetting Limitations

As with any software assurance process, there is no guarantee that even the most thorough vetting process will uncover all potential vulnerabilities or malicious behavior. Organizations should be made aware that although app security assessments should generally improve the security posture of the organization, the degree to which they do so may not be easily or immediately ascertained. Organizations should also be made aware of what the vetting process does and does not provide in terms of security.

Organizations should also be educated on the value of humans in security assessment processes and ensure that their app vetting does not rely solely on automated tests. Security analysis is primarily a human-driven process [15], [27]; automated tools by themselves cannot address many of the contextual and nuanced interdependencies that underlie software security. The most

obvious reason for this is that fully understanding software behavior is one of the classic impossible problems of computer science [28], and in fact current technology has not even reached the limits of what is theoretically possible. Complex, multifaceted software architectures cannot be fully analyzed by automated means.

A further problem is that current software analysis tools do not inherently understand what software has to do to behave in a secure manner in a particular context. For example, failure to encrypt data transmitted to the cloud may not be a security issue if the transmission is tunneled through a virtual private network (VPN). Even if the security requirements for an app have been correctly predicted and are completely understood, there is no current technology for unambiguously translating human-readable requirements into a form that can be understood by machines.

For these reasons, security analysis requires human analysts be in the loop, and by extension the quality of the outcome depends, among other things, on the level of human effort and expertise available for an evaluation. Analysts should be familiar with standard processes and best practices for software security assessment [15], [29]–[31]. In order to be successful, a robust app vetting process should use a toolbox approach where multiple assessment tools and processes, as well as human interaction work together. Reliance on only a single tool, even with human interaction, is a significant risk because of the inherent limitations of each tool.

5.3 Local and Remote Tools and Services

There are many tools and services dedicated to analyzing mobile apps [32], [33]. Depending on the model employed by the tool/service provider, app analysis may occur in different physical locations. For example, an analysis tool may be installed and run within the network of the organization for whom the app is intended. Other vendors may host their test services offsite. Offsite tools may reside on premise of the tool/service provider or may reside in a cloud infrastructure. Each of these scenarios should be understood by an organization prior to employing a vetting tool/service, especially in those cases where the apps may contain sensitive or classified information.

5.4 Automated Approval/Rejection

In some cases, the activities conducted by analysts to derive recommendations for approving or rejecting an app can be automated, particularly if no organization-specific policies, regulation, etc. are required. Here, an app vetting system can be used to support the specification of rules can be configured to automatically approve or reject an app based on risk assessments from multiple tools. For example, an app vetting system could be configured to automatically recommend an app if all test tools deem the app as having “LOW” risk. Similarly, an app vetting system could be configured to automatically enforce organization-specific requirements. For example, using metadata extracted during the preprocessing of an app, an app vetting system could automatically reject an app from a specific vendor.

5.5 Reciprocity

The sharing of an organization's findings for an app can greatly reduce the duplication and cost of app vetting efforts for other organizations. Information sharing within the software assurance

community is vital and can help test tools benefit from the collective efforts of security professionals around the world. The National Vulnerability Database (NVD) [34] is the U.S. government repository of standards-based vulnerability management data represented using the Security Content Automation Protocol (SCAP) [35]. This data enables automation of vulnerability management, security measurement, and compliance. The NVD includes databases of security checklists, security-related software flaws, misconfigurations, product names, and impact metrics. SCAP is a suite of specifications that standardize the format and nomenclature by which security software products communicate software flaw and security configuration information. SCAP is a multipurpose protocol that supports automated vulnerability checking, technical control compliance activities, and security measurement. Goals for the development of SCAP include standardizing system security management, promoting interoperability of security products, and fostering the use of standard expressions of security content. The CWE [24] and Common Attack Pattern Enumeration and Classification (CAPEC) [36] collections can provide a useful list of weaknesses and attack approaches to drive a binary or live system penetration test. Classifying and expressing software vulnerabilities is an ongoing and developing effort in the software assurance community, as is how to prioritize among the various weaknesses that can be in an app [40] so that an organization can know that those that pose the most danger to the app, given its intended use/mission, are addressed by the vetting activity given the difference in the effectiveness and coverage of the various available tools and techniques.

5.6 Budget and Staffing

App software assurance activity costs should be included in project budgets and should not be an afterthought. Such costs may be significant and can include licensing costs for test tools and salaries for analysts, approvers, and administrators. Organizations that hire contractors to develop apps should specify that app assessment costs be included as part of the app development process. Note, however, that for apps developed in-house, attempting to implement app vetting solely at the end of the development effort will lead to increased costs and lengthened project timelines. It is strongly recommended to identify potential vulnerabilities or weaknesses during the development process when they can still be addressed by the original developers. Identifying and fixing errors during the development process is also significantly cheaper than fixing errors once a product is released [37].

To provide an optimal app vetting process implementation, it is critical for the organization to hire personnel with appropriate expertise. For example, organizations should hire analysts experienced in software security and information assurance as well as administrators experienced in mobile security.

6 App Vetting Systems

While an app vetting process may be performed manually, it is typically advantageous to perform an app vetting process in a semi-or full-automated fashion using an app vetting system (e.g., the NIST AppVet system [13]). An app vetting system is a system that manages and automates an app vetting process and may be implemented as a web-based service and is typically part of a larger app vetting ecosystem that comprises test tool/services, app stores, EMMs, and users.

An app vetting system is used by a security analyst (often an enterprise system administrator) to identify app security issues before an app is deployed to a user's mobile device. After the system analyzes the app, the security analyst considers the vetting results within the context of the security posture of the larger enterprise environment's and makes a security recommendation. An authorizing official then decides if to approve the use of the app, given the user's role, the mission need addressed by the app, and the security recommendation of the security analyst. Figure 6 depicts a reference architecture for an app vetting system.

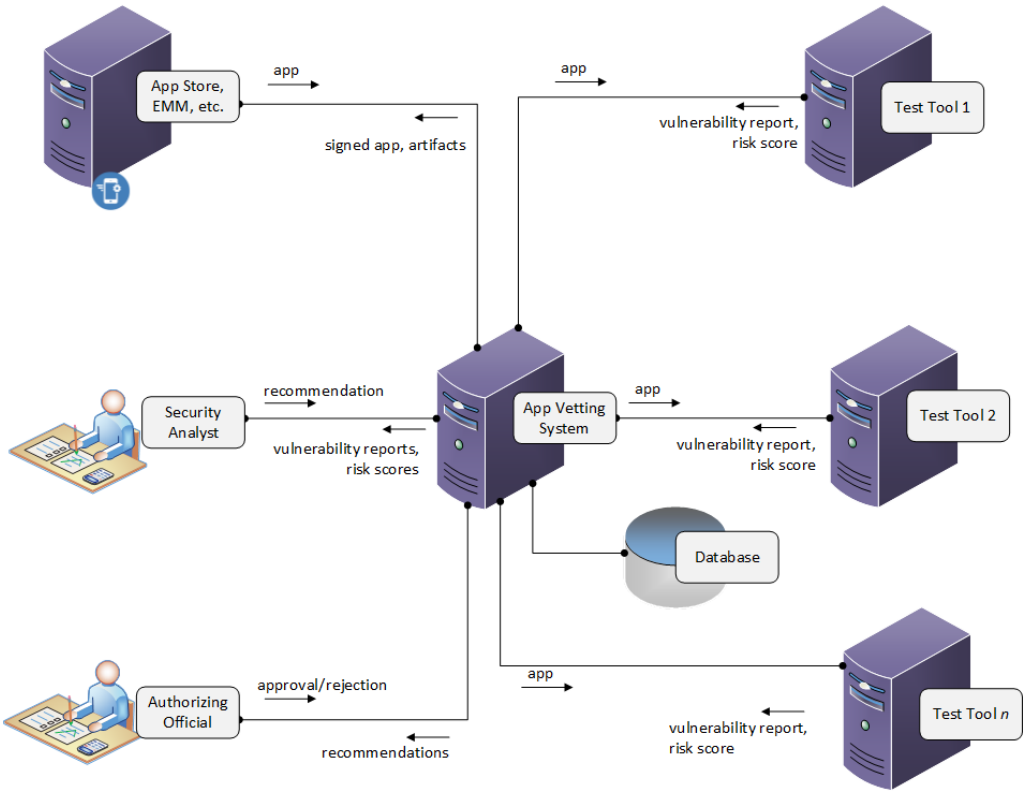

Figure 6 - Example app vetting system architecture.

At the center of the diagram is the app vetting system. This system is the central hub to the larger app vetting ecosystem. The app vetting system coordinates requests and responses among all the

other system components, the security analyst and the authorizing official. A crucial component and function of the vetting system is that it serves as the long-term memory and decision repository for the app vetting process. In the diagram, this is represented by the database symbol connected to the app vetting system. This database should store testing reports as well as the inputs of the security analyst and authorizing official for posterity.

An enterprise mobile device seeking to use an app may do so in several ways. The enterprise may host a specific app store that only contains vetting applications. Alternately, the device may have policy rules enforced by an enterprise mobility management (EMM) system that regulate what apps may be installed from any source. These systems are represented by the box in the upper left corner of the diagram. Information about the requested app (usually app binary code, but sometimes app source code for apps developed "in house") is sent from this system to the app vetting coordination hub to begin the app vetting process

There are many different strategies for examining an app and evaluating its security characteristics. No single algorithm, tool or product offers a complete picture of an app's security characteristics. The reference architecture shows how an organization might take input from multiple (three are shown at right in the figure) test tools to better inform the security analyst. After the request for app vetting is sent from the App Store or EMM system to the vetting hub, the hub contacts each of the three test tools in the diagram. Each tool receives a copy of the information provided about the app (e.g., binary or source code), performs its independent assessment and returns a vulnerability report and some form of risk score.

The vetting hub then gathers the results reported by the various test tools, potentially summarizing those results and offering them to the security analyst in a dashboard view. After reviewing the results of the various tests, the security analyst submits a recommendation, which is recorded by the vetting hub. The authorizing official can then consider the security analyst's recommendation together with mission needs to approve or reject the use of the app by the mobile user. If the app is approved for installation, the vetting hub can provide digitally signed artifacts, including digitally-signed apps, back to the App Store or EMM system to enable the app deployment.

While the figure depicts a locally hosted app vetting system (i.e., the app vetting hub, test tools, database and App Store are shown as residing on hosts), many app vetting systems may be hosted in a cloud environment. In a cloud-hosted scenario, the boxes shown in the diagram would be hosted by a private or public cloud service provider and much of the functionality would be virtualized. The security analyst and authorizing official need not know how the vetting system is implemented. In either type of deployment, users in these roles would interact with the system through a dashboard providing the appropriate services and views. Both types of deployment enable modular extension of the app vetting system to accommodate new vetting test tools as these become available.

An app vetting system uses application programming interfaces (APIs), network protocols and schemas to integrate with distributed third-party test tools as well as clients including app stores. An app vetting system may also include a user interface (UI) dashboard that allows users such as administrators, analysts and authorizing officials to view reports and risk assessments, provide recommendations and approve or reject apps. Figure 6 shows an example of how an app vetting

881 system utilizing APIs and a UI can be used to support integration with all components and users
882 in an app vetting ecosystem.

883

Appendix A—Threats to Mobile Applications

Like all software, mobile apps often contain vulnerabilities (introduced by errors in design or implementation or by malicious intent) that can expose a user, a mobile device and its data or enterprise services or its data to attacks. There are a number of common classes of mobile software errors that can create such vulnerabilities, including errors in the use or implementation of cryptographic primitives and other security services, risky interactions among software components on a mobile device, and risky interactions between the mobile device and systems within its environment. Common errors in using security services or cryptography include weak authentication of users or systems, incorrect implementation of cryptographic primitives, choosing outdated or broken cryptographic algorithms or parameters, or failure to encrypt app traffic between a mobile device and web- or enterprise-hosted services. Risky interactions among software components on a mobile device include the use of data from untrustworthy sources as input to security-sensitive operations, use of vulnerable third-party-provided software libraries, and app code that leaks sensitive data outside of the app (e.g., through logs of app activity). Also, mobile systems may be exposed to malicious code or injections of data through communication with a compromised web or enterprise service.

Vetting mobile apps before deploying them onto a user's mobile device can enable an enterprise system administrator to detect software or configuration flaws that may create vulnerabilities or violate enterprise security or privacy policies. Mobile app vetting systems typically include automated testing and analysis tools and may interact with externally hosted vetting services. This section will discuss different classes of malware that affect mobile devices. Mobile app vetting systems are designed to look for evidence of such malware.

A.1 Ransomware

Ransomware is malware that encrypts data and holds the decryption key hostage for payment [38] In the mobile environment, new ransomware [39]has been observed that not only encrypts the data of users, but also locks them out of their devices by changing the lock screen PIN. Such ransomware has been spreading as a fake software update via compromised websites.

A.2 Spyware

Spyware [40]is malware designed to gather information about an individual or organization without their knowledge and send that information to the attacker's systems. While spyware often has been used to track internet user's movements on the Web, it may also be used to capture SMS messages, photos, phone call logs or sensitive data such as user logins or banking information. Most spyware is installed without a device user's (or the organization's) knowledge, often using deceptive tactics that trick the user into installation. Spyware is generally legal and is often marketed as a tool for parents to monitor their kids or for catching a cheating spouse. Nation-state actors also have used spyware to gather information from mobile users [41].

A.3 Adware

Adware is malware that is embedded within or loaded as part of advertisements and is one of the most common threats to mobile devices worldwide. Mobile ads are instrumental to the current

mobile ecosystem because they provide a source of funding for software developers that offer free mobile apps. Ads may be served from third-party websites and may contain malware (hence "adware") that often is used to capture personal information without a user's permission or knowledge. Recent reports [42]have shown some low-end mobile devices were shipped from the manufacturer with adware pre-installed. Users with affected phones experience popup ads and other annoying problems and because the adware is installed at the firmware level it is incredibly difficult to remove.

A.4 Rooters

A rooter is a software tool that enables a user to root a mobile device. "Rooting" is the process of enabling users to gain privileged (root) access on the device's operating system (OS). Rooting is often performed to overcome restrictions that carriers and device manufacturers often enforce on some mobile devices. Rooting enables alteration or replacement of systems applications and settings, execution of specialized apps requiring administrative privileges, or performance of carrier-prohibited operations. On some mobile platforms (e.g., Android), rooting also can facilitate the complete removal and replacement of the device's OS, e.g., to install a newer version of it. There are two types of rooting [43]

- "Soft rooting" typically is performed via a third-party application that uses a security vulnerability called a "root exploit".
- "Hard rooting" requires flashing binary executables and provides super-user privileges.

A.5 Trojan Horse

A Trojan horse (or a Trojan) is malware that poses as legitimate and often familiar software, thereby tricking a user into running it. For traditional computing platforms, attackers typically hide malware using file names with well-known extensions, such as .doc or .jpg. Users open the Trojan file and the malware begins to execute. In the mobile environment, mobile banking Trojans are a worrisome new trend [44] describes malware that is installed after victims respond to a phishing message that appears to be from their bank. The malware gathers financial information, login credentials and sometimes credit card information.

A.6 Infostealer

An infostealer is a Trojan horse that gathers information, including confidential data, from an infected system and sends it to an attacker's system. The most common types of information stolen include user credentials (e.g., login user name and password) or financial data. Infostealers commonly have affected traditional computing platforms but have more recently begun impacting mobile platforms. Recent reports [45] describe malware that poses as a Google Chrome update for Android devices and disables antivirus applications. The malware can harvest user banking information, call logs, SMS data and browser history, which are sent to remote servers.

A.7 Hostile Downloader

A Hostile Downloader is malware whose primary purpose is to download content, usually from the Internet. Downloaded content may often include other malicious apps (which often are launched by the downloader), configurations or commands for the downloader or for other

software installed on the system, and additional software components to facilitate an attack. For example, in 2017, attackers used a malicious PowerPoint presentation embedded in a spam email to launch a banking Trojan [46]. Opening the PowerPoint file and just hovering the mouse pointer over a displayed hyperlink—no clicking required–caused PowerPoint to execute a malicious script that downloaded a Trojan horse.

A.8 Mobile Billing Fraud

Many mobile service providers allow products or services to be charged to a user's mobile service account, which are paid monthly by the user or account owner. In effect, the mobile account works like a credit card, offering both convenience to the user and paradoxically increased vulnerability to fraud. Users without traditional credit accounts (i.e., "unbanked", often lower-income people) often purchase online content or services through direct carrier billing.

Fraud by carrier companies against users, fraud by users against carriers, and fraud by third-parties against both users and carriers have occurred. The U.S. Federal Trade Commission (FTC) has successfully litigated cases against AT&T [47]Verizon and Sprint [48] for "cramming" customer bills with millions of dollars of unauthorized services. The FTC offers advice [49] to mobile customers about preventing phone bill "cramming." At the same time, mobile carriers are experiencing fraud by customers, similar to that caused by credit card users against banks. Most commonly, users make purchases, deny that they did so and then demand refunds. Finally, third-parties are committing identity theft, using a mobile device user's identity information to take over his/her mobile account to buy new equipment (e.g., smartphones), charge the purchase to the account and resell the equipment for cash [50]. Wireless carriers are working to strengthen authentication of subscribers before allowing new device activations or service changes.

A.9 SMS Fraud

Scams once perpetrated via email now are perpetrated via SMS messaging. Fraudulent business transactions, phishing (called "smishing" when delivered via SMS messages), phony requests for donations, fees to claim lottery prizes and cons originating from dating sites are all SMS scams [51]. Users must be wary of unsolicited texts from strangers or unknown numbers, especially requests for money or personal/sensitive information.

A.10 Call Fraud

Call fraud refers to several malicious and illegal activities. For example, some users of cellular services may receive calls that appear to originate from domestic area codes, but are actually associated with international pay-per-call services. These calls often disconnect after one ring. When the target returns the call he or she is connected to an international line that charges a fee for connecting in addition to significant per-minute fees if the victim stays on the line. These charges usually show up on the victim's cellular bill as premium services.

A.11 Cramming

"Cramming" refers to fraudulent activities that result in charges such as fees for calls or services

to a victim's cellular bill for services that the victim did not order or services with undisclosed fees. These charges often are assessed by dishonest third-parties of data and communication services. Carriers and operators often allow third-parties to bill for services by charging to a user's cellular bill. Other types of call fraud by third-parties against customers often include "PBX dial-through," which can be mounted by placing a call to an enterprise, then requesting to be transferred to "9-0" or some other outside toll number. More information about different fraud activities is available from the FTC [49] and the Communication Fraud Control Association (CFCA).

A.12 Toll Fraud

Toll fraud occurs when a mobile device user makes a call—often using premium services—that is charged to a third-party that did not authorize the call. A common attack with a hacker leasing phone numbers from a web-based service that charges callers for each call and provides a percentage of the profit to the hacker. To make a lucrative fraud-based business, the hacker breaches an independent business's Voice Over IP (VoIP) network to forward calls to the hacker's premium service numbers. The independent company is billed for the calls by the web-based service and the hacker gets a percentage of the profits. To resist these type of attacks, organizations must implement strong network security protections.

Appendix B—Android App Vulnerability Types

This appendix identifies vulnerabilities specific to apps running on Android mobile devices. The scope of this appendix includes app vulnerabilities for Android-based mobile devices running apps written in Java. The scope does not include vulnerabilities in the mobile platform hardware and communications networks. Although some of the vulnerabilities described below are common across mobile device environments, this appendix focuses only on Android-specific vulnerabilities.

The vulnerabilities in this appendix are broken into three hierarchical levels, A, B, and C. The A level is referred to as the vulnerability class and is the broadest description for the vulnerabilities specified under that level. The B level is referred to as the sub-class and attempts to narrow down the scope of the vulnerability class into a smaller, common group of vulnerabilities. The C level specifies the individual vulnerabilities that have been identified. The purpose of this hierarchy is to guide the reader to finding the type of vulnerability they are looking for as quickly as possible.

Table 4 shows the A level general categories of Android app vulnerabilities.

Table 4 - Android Vulnerabilities, A Level.

Type	Description	Negative Consequence
Incorrect Permissions	Permissions allow accessing controlled functionality such as the camera or GPS and are requested in the program. Permissions can be implicitly granted to an app without the user's consent.	An app with too many permissions may perform unintended functions outside the scope of the app's intended functionality. Additionally, the permissions are vulnerable to hijacking by another app. If too few permissions are granted, the app will not be able to perform the functions required.
Exposed Communications	Internal communications protocols are the means by which an app passes messages internally within the device, either to itself or to other apps. External communications allow information to leave the device.	Exposed internal communications allow apps to gather unintended information and inject new information. Exposed external communication (data network, Wi-Fi, Bluetooth, NFC, *etc.*) leave information open to disclosure or man-in-the-middle attacks.
Potentially Dangerous Functionality	Controlled functionality that accesses system-critical resources or the user's personal information. This functionality can be invoked through API calls or hard coded into an app.	Unintended functions could be performed outside the scope of the app's functionality.
App Collusion	Two or more apps passing information to each other in order to increase the capabilities of one or both apps beyond their declared scope.	Collusion can allow apps to obtain data that was unintended such as a gaming app obtaining access to the user's contact list.
Obfuscation	Functionality or control flows that are hidden or obscured from the user. For the purposes of this appendix, obfuscation was defined as three criteria: external library calls, reflection, and native code usage.	1. External libraries can contain unexpected and/or malicious functionality. 2. Reflective calls can obscure the control flow of an app and/or subvert permissions within an app. 3. Native code (code written in languages other than Java in Android) can perform unexpected and/or malicious functionality.

Type	Description	Negative Consequence
Excessive Power Consumption	Excessive functions or unintended apps running on a device which intentionally or unintentionally drain the battery.	Shortened battery life could affect the ability to perform mission-critical functions.
Traditional Software Vulnerabilities	All vulnerabilities associated with traditional Java code including: Authentication and Access Control, Buffer Handling, Control Flow Management, Encryption and Randomness, Error Handling, File Handling, Information Leaks, Initialization and Shutdown, Injection, Malicious Logic, Number Handling, and Pointer and Reference Handling.	Common consequences include unexpected outputs, resource exhaustion, denial of service, *etc.*

1047

1048 Table 5 shows the hierarchy of Android app vulnerabilities from A level to C level.

1049 **Table 5 - Android Vulnerabilities by level.**

Level A	Level B	Level C
Permissions	Over Granting	Over Granting in Code
		Over Granting in API
	Under Granting	Under Granting in Code
		Under Granting in API
	Developer Created Permissions	Developer Created in Code
		Developer Created in API
	Implicit Permission	Granted through API
		Granted through Other Permissions
		Granted through Grandfathering
Exposed Communications	External Communications	Bluetooth
		GPS
		Network/Data Communications
		NFC Access
	Internal Communications	Unprotected Intents
		Unprotected Activities
		Unprotected Services
		Unprotected Content Providers
		Unprotected Broadcast Receivers
		Debug Flag

1050

Potentially Dangerous Functionality	Direct Addressing	Memory Access
		Internet Access
	Potentially Dangerous API	Cost Sensitive APIs
		Personal Information APIs
		Device Management APIs
	Privilege Escalation	Altering File Privileges
		Accessing Super User/Root
App Collusion	Content Provider/Intents	Unprotected Content Providers
		Permission Protected Content Providers
		Pending Intents
	Broadcast Receiver	Broadcast Receiver for Critical Messages
	Data Creation/Changes/Deletion	Creation/Changes/Deletion to File Resources
		Creation/Changes/Deletion to Database Resources
	Number of Services	Excessive Checks for Service State
Obfuscation	Library Calls	Use of Potentially Dangerous Libraries
		Potentially Malicious Libraries Packaged but Not Used
	Native Code Detection	
	Reflection	
	Packed Code	
Excessive Power Consumption	CPU Usage	
	I/O	

1051

Appendix C—iOS App Vulnerability Types

This appendix identifies and defines the various types of vulnerabilities that are specific to apps running on mobile devices utilizing the Apple iOS operating system. The scope does not include vulnerabilities in the mobile platform hardware and communications networks. Although some of the vulnerabilities described below are common across mobile device environments, this appendix focuses on iOS-specific vulnerabilities.

The vulnerabilities in this appendix are broken into three hierarchical levels, A, B, and C. The A level is referred to as the vulnerability class and is the broadest description for the vulnerabilities specified under that level. The B level is referred to as the sub-class and attempts to narrow down the scope of the vulnerability class into a smaller, common group of vulnerabilities. The C level specifies the individual vulnerabilities that have been identified. The purpose of this hierarchy is to guide the reader to finding the type of vulnerability they are looking for as quickly as possible.

Table 6 shows the A level general categories of iOS app vulnerabilities.

Table 6 - iOS Vulnerability Descriptions, A Level.

Type	Description	Negative Consequence
Privacy	Similar to Android Permissions, iOS privacy settings allow for user-controlled app access to sensitive information. This includes: contacts, Calendar information, tasks, reminders, photos, and Bluetooth access.	iOS lacks the ability to create shared information and protect it. All paths of information sharing are controlled by the iOS app framework and may not be extended. Unlike Android, these permissions may be modified later for individual permissions and apps.
Exposed Communication-Internal and External	Internal communications protocols allow apps to process information and communicate with other apps. External communications allow information to leave the device.	Exposed internal communications allow apps to gather unintended information and inject new information. Exposed external communication (data network, Wi-Fi, Bluetooth, *etc.*) leave information open to disclosure or man-in-the-middle attacks.
Potentially Dangerous Functionality	Controlled functionality that accesses system-critical resources or the user's personal information. This functionality can be invoked through API calls or hard coded into an app.	Unintended functions could be performed outside the scope of the app's functionality.
App Collusion	Two or more apps passing information to each other in order to increase the capabilities of one or both apps beyond their declared scope.	Collusion can allow apps to obtain data that was unintended such as a gaming app obtaining access to the user's contact list.
Obfuscation	Functionality or control flow that is hidden or obscured from the user. For the purposes of this appendix, obfuscation was defined as three criteria: external library calls, reflection, and packed code.	1. External libraries can contain unexpected and/or malicious functionality. 2. Reflective calls can obscure the control flow of an app and/or subvert permissions within an app. 3. Packed code prevents code reverse engineering and can be used to hide malware.
Excessive Power Consumption	Excessive functions or unintended apps running on a device which intentionally or unintentionally drain the battery.	Shortened battery life could affect the ability to perform mission-critical functions.

Type	Description	Negative Consequence
Traditional Software Vulnerabilities	All vulnerabilities associated with Objective C and others. This includes: Authentication and Access Control, Buffer Handling, Control Flow Management, Encryption and Randomness, Error Handling, File Handling, Information Leaks, Initialization and Shutdown, Injection, Malicious Logic, Number Handling and Pointer and Reference Handling.	Common consequences include unexpected outputs, resource exhaustion, denial of service, *etc.*

1066

1067 Table 7 shows the hierarchy of iOS app vulnerabilities from A level to C level.

1068 **Table 7 - iOS Vulnerabilities by level.**

Level A	Level B	Level C
Privacy	Sensitive Information	Contacts
		Calendar Information
		Tasks
		Reminders
		Photos
		Bluetooth Access
Exposed Communications	External Communications	Telephony
		Bluetooth
		GPS
		SMS/MMS
		Network/Data Communications
	Internal Communications	Abusing Protocol Handlers
Potentially Dangerous Functionality	Direct Memory Mapping	Memory Access
		File System Access
	Potentially Dangerous API	Cost Sensitive APIs
		Device Management APIs
		Personal Information APIs
App Collusion	Data Change	Changes to Shared File Resources
		Changes to Shared Database Resources
		Changes to Shared Content Providers
	Data Creation/Deletion	Creation/Deletion to Shared File Resources
Obfuscation	Number of Services	Excessive Checks for Service State
	Native Code	Potentially Malicious Libraries Packaged but not Used
		Use of Potentially Dangerous Libraries
		Reflection Identification
		Class Introspection
	Library Calls	Constructor Introspection
		Field Introspection
		Method Introspection

Level A	Level B	Level C
	Packed Code	
Excessive Power Consumption	CPU Usage	
	I/O	

1069

Appendix D—Acronyms

Selected acronyms and abbreviations used in this paper are defined below

API	Application Programming Interface
BYOD	Bring Your Own Device
CAPEC	Common Attack Pattern Enumeration and Classification
CERT	Computer Emergency Response Team
CPU	Central Processing Unit
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DHS	Department of Homeland Security
DoD	Department of Defense
EMM	Enterprise Mobility Management
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IoT	Internet of Things
ISO	International Organization for Standardization
ITL	Information Technology Laboratory
JVM	Java Virtual Machine
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
OMB	Office of Management and Budget
PII	Personally Identifiable Information
PIN	Personal Identification Number

PIV	Personal Identity Verification
SAMATE	Software Assurance Metrics and Tool Evaluation
SCAP	Security Content Automation Protocol
SLA	Service Level Agreement
SP	Special Publication
UI	User Interface
VPN	Virtual Private Network
Wi-Fi	Wireless Fidelity.

1072

Appendix E—Glossary

The definition of selected terms used in this publication are below

Administrator	A member of an organization who is responsible for deploying, maintaining and securing the organization's mobile devices as well as ensuring deployed devices and their installed apps conform to security requirements.
App Security Requirement	A requirement that ensures the security of an app. There are two types of app security requirements: general and organization-specific. General app security requirements define the software and behavioral characteristics of an app that should or should not be present in order to ensure the security of the app. Organization-specific security requirements define the policies, regulations, and guidance that an organization must follow to ensure the security posture of the organization.
Analyst	A member of an organization who inspects reports and risk assessments from one or more test tools as well as organization-specific criteria to verify an app meets the organization's security requirements.
App Vetting Process	A sequence of activities performed by an organization to determine if a mobile app conforms to the organization's security requirements.
App Vetting System	A system for managing and automating an app vetting process.
Authorizing Official	An organization member who decides whether an app is approved or denied for use by the organization.
Dynamic Analysis	Detecting software vulnerabilities by executing an app using a set of input use-cases and analyzing the app's runtime behavior.
Enterprise Mobility Manager	A set of people, processes and technology focused on managing mobile devices, wireless networks and other mobile computing services in a business environment.
Functionality Testing	Verifying an app's user interface content and features perform and display as designed.
Mobile Device Management	The administration of mobile devices such as smartphones, tablet computers, laptops and desktop computers. MDM usually is implemented through a third-party product that has management features for particular vendors of mobile devices.
National Security	Any information system, including any telecommunications system, used or operated by an agency or by a contractor of an agency or other

System	organization on behalf of an agency: The function, operation or use of which-- involves intelligence activities; involves cryptologic activities related to national security; involves command and control of military forces; involves equipment that is an integral part of a weapon or weapons system; or subject to subparagraph (B) is critical to the direct fulfillment of military or intelligence missions; or Is protected at all times by procedures established for information that have been specifically authorized under criteria established by an Executive Order or an Act of Congress to be kept classified in the interest of national defense or foreign policy [52].
Personally Identifiable Information	Information about an individual that can be used by a malicious actor to distinguish or trace the individual's identity and any other information that is linked or linkable to the individual [45].
Risk Assessment	A value that states a test tool's estimated level of security risk when an app is used. Risk assessments typically are based on the likelihood that a detected vulnerability will be exploited and the impact the detected vulnerability may have on the app or its related device or network. Risk assessments typically are represented as categories (e.g., low-, moderate- and high-risk).
Static Analysis	Detecting software vulnerabilities by examining an app's source code and binary and attempting to determine all possible behaviors that might arise at runtime.
Software Assurance	The level of confidence that software is free from vulnerabilities—either intentionally designed into the software or accidentally inserted during its lifecycle—and functions in the intended manner.
Software Correctness Testing	The process of executing a program to finding errors. The purpose of this testing is to improve quality assurance, verify and validate described functionality, or estimate reliability.
Software Vulnerability	A security flaw, glitch or weakness found in software that can be exploited by an attacker.

Test Tool	A tool or service that tests an app to determine if specific software vulnerabilities are present.

1075

Appendix F—References

[1] P. E. Black, L. Badger, B. Guttman, and E. Fong, "Dramatically reducing software vulnerabilities: Report to the White House Office of Science and Technology Policy," National Institute of Standards and Technology, Gaithersburg, MD, NIST IR 8151, Nov. 2016.

[2] R. Kissel, "Glossary of key information security terms," National Institute of Standards and Technology, NIST IR 7298r2, May 2013.

[3] M. Souppaya and K. Scarfone, "Guidelines for Managing the Security of Mobile Devices in the Enterprise," National Institute of Standards and Technology, NIST SP 800-124r1, Jun. 2013.

[4] "Protection Profile for Mobile Device Fundamentals," p. 183.

[5] Joint Task Force Transformation Initiative, "Security and Privacy Controls for Federal Information Systems and Organizations," National Institute of Standards and Technology, NIST SP 800-53r4, Apr. 2013.

[6] ISO/IEC, "Information technology -- Security techniques -- Evaluation criteria for IT security," ISO/IEC 15408-1:2009.

[7] "Requirements for Vetting Mobile Apps from the Protection Profile for Application Software." [Online]. Available: https://www.niap-ccevs.org/MMO/PP/394.R/pp_app_v1.2_table-reqs.htm. [Accessed: 22-Jun-2018].

[8] OWASP, "Mobile Application Security Verification Standard," v0.9.4.

[9] "OWASP Mobile Security Testing Guide - OWASP." [Online]. Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide. [Accessed: 22-Jun-2018].

[10] M. Peck and C. Northern, "Analyzing the Effectiveness of App Vetting Tools in the Enterprise," p. 46, Aug. 2016.

[11] "Build Security In | US-CERT." [Online]. Available: https://www.us-cert.gov/bsi#ques. [Accessed: 22-Jun-2018].

[12] "CVE - Common Vulnerabilities and Exposures (CVE)." [Online]. Available: https://cve.mitre.org/. [Accessed: 22-Jun-2018].

[13] I. T. L. Computer Security Division, "AppVet | CSRC." [Online]. Available: https://csrc.nist.gov/projects/appvet/. [Accessed: 22-Jun-2018].

[14] M. Pezzè and M. Young, *Software Testing and Analysis: Process, Principles and Techniques*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2008.

[15] G. McGraw, *Software security: building security in*. Upper Saddle River, NJ: Addison-Wesley, 2006.

[16] G. G. Schulmeyer, *Handbook of Software Quality Assurance, Fourth Edition*. Norwood, Massachusetts: Artech House, Inc., 2008.

[17] B. B. Agarwal, S. P. Tayal, and M. Gupta, *Software engineering & testing: an introduction*. Sudbury, Mass: Jones and Bartlett, 2010.

[18] J. R. Maximoff, M. D. Trela, D. R. Kuhn, and R. Kacker, "A method for analyzing system state-space coverage within a t-wise testing framework," 2010, pp. 598–603.

[19] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The art of software testing*, 2nd ed. Hoboken, N.J: John Wiley & Sons, 2004.

[20] H. CHEN, T. ZOU, and D. WANG, "Data-flow Based Vulnerability Analysis and Java Bytecode," *7th WSEAS Int. Conf. Appl. Comput. Sci. Venice Italy Novemb. 21-23 2007*,

p. 7.
[21] "FindBugs™ - Find Bugs in Java Programs." [Online]. Available: http://findbugs.sourceforge.net/. [Accessed: 22-Jun-2018].
[22] R. Shah, "Vulnerability Assessment of Java Bytecode," Auburn University, 2005.
[23] WAIM (Conference) *et al.*, *The Ninth International Conference on Web-Age Information Management: WAIM 2008*. Piscataway, N.J.: IEEE, 2008.
[24] "CWE - Common Weakness Enumeration." [Online]. Available: http://cwe.mitre.org/. [Accessed: 22-Jun-2018].
[25] FIRST.org, Inc, "CVSS v3.0 Specification," p. 21.
[26] "Mobile Application Security Requirments." Department of Defense, 06-Oct-2017.
[27] M. Dowd, J. McDonald, and J. Schuh, *The art of software security assessment: identifying and preventing software vulnerabilities*. Indianapolis, Ind: Addison-Wesley, 2007.
[28] H. G. Rice, "Classes of recursively enumerable sets and their decision problems," *Trans. Am. Math. Soc.*, vol. 74, no. 2, pp. 358–358, Feb. 1953.
[29] J. H. Allen, Ed., *Software security engineering: a guide for project managers*. Upper Saddle River, NJ: Addison-Wesley, 2008.
[30] Archiveddocs, "The STRIDE Threat Model." [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v%3dcs.20). [Accessed: 22-Jun-2018].
[31] "Trike: Trike." [Online]. Available: http://www.octotrike.org/home.shtml. [Accessed: 22-Jun-2018].
[32] "Tool Survey - SAMATE." [Online]. Available: https://samate.nist.gov/index.php/Tool_Survey.html. [Accessed: 22-Jun-2018].
[33] M. A. Ogata, "An overview of mobile application vetting services for public safety," National Institute of Standards and Technology, Gaithersburg, MD, NIST IR 8136, Jan. 2017.
[34] "NVD - Home." [Online]. Available: https://nvd.nist.gov/. [Accessed: 22-Jun-2018].
[35] I. T. L. Computer Security Division, "Security Content Automation Protocol | CSRC." [Online]. Available: https://csrc.nist.gov/projects/security-content-automation-protocol/. [Accessed: 22-Jun-2018].
[36] "CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC)." [Online]. Available: https://capec.mitre.org/. [Accessed: 22-Jun-2018].
[37] J. M. Stecklein, B. Dick, B. Haskins, R. Lovell, and G. Moroney, "Error Cost Escalation Through the Project Life Cycle," presented at the 14th Annual International Symposium, Toulouse; France, 2004.
[38] M. Bartock, J. Cichonski, M. Souppaya, M. Smith, G. Witte, and K. Scarfone, "Guide for cybersecurity event recovery," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-184, Dec. 2016.
[39] S. Khandelwal, "New Ransomware Not Just Encrypts Your Android But Also Changes PIN Lock," *The Hacker News*. [Online]. Available: https://thehackernews.com/2017/10/android-ransomware-pin.html. [Accessed: 22-Jun-2018].
[40] W. A. Jansen, T. Winograd, and K. Scarfone, "Guidelines on Active Content and Mobile Code," p. 62.
[41] "When 'Grandma-Proof' Android Spyware Is Good Enough For International Espionage." [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2018/05/15/apple-

iphone-spouseware-used-in-pakistan-government-attacks/#74f2e2515668. [Accessed: 22-Jun-2018].

[42] S. Dent, "Report finds Android malware pre-installed on hundreds of phones," *Engadget*, 24-May-2018. [Online]. Available: https://www.engadget.com/2018/05/24/report-finds-android-malware-pre-installed-on-hundreds-of-phones/. [Accessed: 22-Jun-2018].

[43] H. Zhang, D. She, and Z. Qian, "Android Root and its Providers: A Double-Edged Sword," 2015, pp. 1093–1104.

[44] J. Mello Jr, "Marcher Malware Poses Triple Threat to Android Users | Malware | TechNewsWorld," *Tech News Workd*, 07-Nov-2017. [Online]. Available: https://www.technewsworld.com/story/84936.html. [Accessed: 22-Jun-2018].

[45] D. Palmer, "Irremovable bank data-stealing Android malware poses as Google Chrome update," *ZDNet*. [Online]. Available: https://www.zdnet.com/article/irremovable-bank-detail-stealing-android-malware-poses-as-google-chrome-update/. [Accessed: 22-Jun-2018].

[46] M. Moon, "Malware downloader infects your PC without a mouse click," *Engadget*. [Online]. Available: https://www.engadget.com/2017/06/11/malware-downloader-infects-your-pc-without-a-mouse-click/. [Accessed: 22-Jun-2018].

[47] L. Whitney, "AT&T to pay $105 million to settle charges over mobile billing," *CNET*, 08-Oct-2014. [Online]. Available: https://www.cnet.com/news/at-t-to-pay-105-million-to-settle-fraudulent-mobile-bill-charges/. [Accessed: 22-Jun-2018].

[48] J. Brodkin, "Verizon and Sprint pay $158 million in fines for fraudulent phone charges," *Ars Technica*, 12-May-2015. [Online]. Available: https://arstechnica.com/tech-policy/2015/05/verizon-and-sprint-pay-158-million-in-fines-for-fraudulent-phone-charges/. [Accessed: 22-Jun-2018].

[49] "Mystery Phone Charges," *Consumer Information*, 16-Dec-2013. [Online]. Available: https://www.consumer.ftc.gov/articles/0183-mystery-phone-charges. [Accessed: 22-Jun-2018].

[50] H. Weisbaum, "Fraud Alert: ID Thieves Hijack Mobile Phone Accounts," *NBC News*. [Online]. Available: https://www.nbcnews.com/tech/tech-news/fraud-alert-id-thieves-hijack-mobile-phone-accounts-n599761. [Accessed: 22-Jun-2018].

[51] L. Spector, "5 common SMS text scams, and how to avoid them | PCWorld," *PC World*, 01-Mar-2016. [Online]. Available: https://www.pcworld.com/article/3034696/mobile/5-common-sms-text-scams-and-how-to-avoid-them.html. [Accessed: 22-Jun-2018].

[52] *Federal Information Security Modernization Act of 2014*. 2014.