
3 **Vetting the Security of**
4 **Mobile Applications**

7
8
9 Michael Ogata
10 Josh Franklin
11 Jeffrey Voas
12 Vincent Sritapan
13 Stephen Quirolgico
14
15
16
17
18
19

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

23 **Draft NIST Special Publication 800-163**
24 **Revision 1**

25 **Vetting the Security of**
26 **Mobile Applications**

27
28 Michael Ogata
29 *Software and Systems Division*
30 *Information Technology Laboratory*

31
32 Josh Franklin
33 *Applied Cybersecurity Division*
34 *Information Technology Laboratory*

35
36 Jeffrey Voas
37 *Computer Security Division*
38 *Information Technology Laboratory*

39
40 Vincent Sritapan
41 *Office of Science and Technology*
42 *U.S. Department of Homeland Security*

43
44 Stephen Quirolgico
45 *Office of the Chief Information Officer*
46 *U.S. Department of Homeland Security*

47
48 July 2018
49



50
51
52 U.S. Department of Commerce
53 *Wilbur L. Ross, Jr., Secretary*

54
55 National Institute of Standards and Technology
56 *Walter Copan, NIST Director and Under Secretary of Commerce for Standards and Technology*

57

Authority

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

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

71 National Institute of Standards and Technology Special Publication 800-163 Revision
72 1 Natl. Inst. Stand. Technol. Spec. Publ. 800-163 Rev. 1, 50 pages (July 2018)
73 CODEN: NSPUE2

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

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

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

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

88 National Institute of Standards and Technology
89 Attn: Computer Security Division, Information Technology Laboratory
90 100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
91 Email: nist800-163@nist.gov

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

93

Reports on Computer Systems Technology

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

104

Abstract

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

111

Keywords

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

114

115

Trademark Information

116

All registered trademarks belong to their respective organizations.

117

118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150

Table of Contents

1 Introduction 1

 1.1 Purpose 1

 1.2 Scope..... 2

 1.3 Intended Audience 2

 1.4 Document Structure 3

 1.5 Document Conventions..... 3

2 App Security Requirements 4

 2.1 General Requirements..... 4

 2.1.1 National Information Assurance Partnership (NIAP)..... 4

 2.1.2 OWASP Mobile Risks, Controls and App Testing Guidance 5

 2.1.3 MITRE App Evaluation Criteria..... 6

 2.1.4 NIST SP 800-53 6

 2.2 Organization-Specific Requirements..... 7

 2.3 Risk Tolerance 9

 2.3.1 Tool Report Analysis 9

 2.3.2 Compliance versus Certification 10

3 App Vetting Process 11

 3.1 App Intake..... 12

 3.2 App Testing..... 13

 3.3 App Approval/Rejection 14

 3.4 Results Submission 15

4 App Testing and Vulnerability Classifiers 16

 4.1 Testing Approaches 16

 4.1.1 Correctness Testing 16

 4.1.2 Source and Binary Code Testing..... 16

 4.1.3 Static and Dynamic Testing..... 17

 4.2 Vulnerability Classifiers and Quantifiers..... 18

 4.2.1 Common Weakness Enumeration (CWE) 18

 4.2.2 Common Vulnerability and Exposures (CVE)..... 18

 4.2.3 Common Vulnerability Scoring System (CVSS) 19

5 App Vetting Considerations 20

151 5.1 Managed and Unmanaged Apps 20

152 5.2 App Vetting Limitations 20

153 5.3 Local and Remote Tools and Services 21

154 5.4 Automated Approval/Rejection 21

155 5.5 Reciprocity 21

156 5.6 Budget and Staffing 22

157 **6 App Vetting Systems 23**

158

159 **List of Appendices**

160 **Appendix A— Threats to Mobile Applications 26**

161 A.1 Ransomware..... 26

162 A.2 Spyware..... 26

163 A.3 Adware..... 26

164 A.4 Rooters 27

165 A.5 Trojan Horse 27

166 A.6 Infostealer 27

167 A.7 Hostile Downloader..... 27

168 A.8 Mobile Billing Fraud 28

169 A.9 SMS Fraud..... 28

170 A.10 Call Fraud 28

171 A.11 Cramming 28

172 A.12 Toll Fraud..... 29

173 **Appendix B— Android App Vulnerability Types 30**

174 **Appendix C— iOS App Vulnerability Types 33**

175 **Appendix D— Acronyms 36**

176 **Appendix E— Glossary..... 38**

177 **Appendix F— References 41**

178

179 **List of Figures**

180 Figure 1 - Software assurance during mobile application lifecycle. 2

181 Figure 2 - App vetting process overview. 11

182 Figure 3 - Four sub-processes of an app vetting process. 12

183 Figure 4 - Test tool workflow. 14

184 Figure 5 - App approval/rejection process..... 15
185 Figure 6 - Example app vetting system architecture..... 23

186

187

List of Tables

188 Table 1 - NIAP Functional Requirements..... 5

189 Table 2 - Organization-specific security criteria..... 7

190 Table 3 - Risk Tolerance Categories..... 9

191 Table 4 - Android Vulnerabilities, A Level..... 30

192 Table 5 - Android Vulnerabilities by level. 31

193 Table 6 - iOS Vulnerability Descriptions, A Level. 33

194 Table 7 - iOS Vulnerabilities by level..... 34

195

196 **1 Introduction**

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

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

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

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

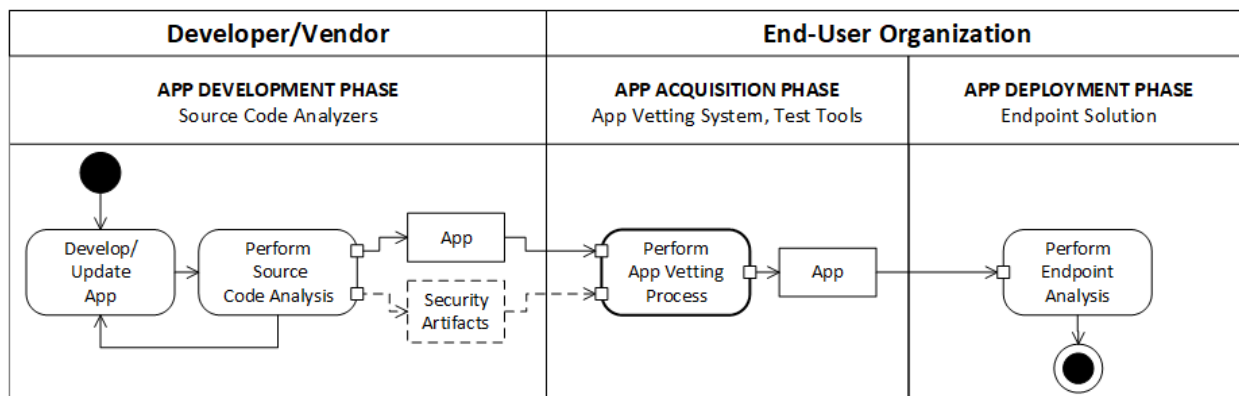
223 **1.1 Purpose**

224 This document defines an app vetting process and provides guidance on (1) planning and
225 implementing an app vetting process, (2) developing security requirements for mobile apps, (3)
226 identifying appropriate tools for testing mobile apps and (4) determining if a mobile app is
227 acceptable for deployment on an organization’s mobile devices. An overview of techniques
228 commonly used by software assurance professionals is provided, including methods of testing
229 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]

230 **1.2 Scope**

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



237
 238 **Figure 1 - Software assurance during mobile application lifecycle.**

239
 240 In this document, we focus primarily on the software assurance activities of the app vetting
 241 process, which we define as part of the app acquisition phase of the mobile application lifecycle.
 242 Thus, software assurance activities performed during the app’s development phase (e.g., by
 243 source code analyzers) or during the app’s deployment phase (e.g., by endpoint solutions) are
 244 considered out of scope for this document.

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

252 **1.3 Intended Audience**

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

- 256 • Responsible for establishing an organization’s mobile device security posture,
- 257 • Responsible for the management and security of mobile devices within an organization,

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

260 **1.4 Document Structure**

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

- 262 • Section 2—App Security Requirements
- 263 • Section 3—App Vetting Process
- 264 • Section 4—App Testing Approaches and Vulnerability Classifiers
- 265 • Section 5—App Vetting Considerations
- 266 • Section 6—App Vetting Systems
- 267 • Appendix A—Threats to Mobile Applications
- 268 • Appendix B—Android App Vulnerability Types
- 269 • Appendix C— iOS App Vulnerability Types
- 270 • Appendix D—Acronyms and Abbreviations
- 271 • Appendix E—Glossary
- 272 • Appendix F—References
- 273

274 **1.5 Document Conventions**

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

277 **2 App Security Requirements**

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

282 **2.1 General Requirements**

283 General app security requirements define the software and behavioral characteristics of an app
284 that should or should not be present in order to ensure the security of the app. These
285 requirements are considered “general” since they can be applied across all mobile applications.
286 General app security requirements may be derived from a number of available standards, best
287 practices, and resources including those specified by NIAP, OWASP, MITRE and NIST².

288 **2.1.1 National Information Assurance Partnership (NIAP)**

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

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

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

304 Table 1 summarizes the NIAP functional requirements³.

305

306

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].

307

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

308

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

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

316 **2.1.2 OWASP Mobile Risks, Controls and App Testing Guidance**

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

- 323 • Standard Security (Level 1)
- 324 • Defines in Depth (Level 2)

- 325 • Resilience against Reverse Engineering and Threats (Level 3).
326

327 Each level's control lists are divided into the categories listed below, with the object described
328 for each control depending on the desired verification level:
329

- 330 • Architecture, Design, and Threat Modeling Requirements
331 • Data Storage and Privacy Requirements
332 • Cryptography Requirements
333 • Authentication and Session Management Requirements
334 • Network Communication Requirements
335 • Platform Integration Requirements
336 • Code Quality and Build-Setting Requirements
337 • Resilience Requirements

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

341 **2.1.3 MITRE App Evaluation Criteria**

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

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

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

354 **2.1.4 NIST SP 800-53**

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

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

371 **2.2 Organization-Specific Requirements**

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

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

379 **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.
--	-------------------------	---

380

381 Some information can be gleaned from app documentation in certain cases, but even if
 382 documentation does exist it might lack technical clarity and/or use jargon specific to the circle of
 383 users who would normally purchase the app. Since the documentation for different apps will be
 384 structured in different ways, it may also be time-consuming to find this information for
 385 evaluation. Therefore, a standardized questionnaire might be appropriate for determining the
 386 software's purpose and assessing an app developer's efforts to address security weaknesses.
 387 Such questionnaires aim to identify software quality issues and security weaknesses by helping
 388 developers address questions from end-users/adopters about their software development
 389 processes. For example, developers can use the Department of Homeland Security (DHS)
 390 Custom Software Questionnaire [11] to answer questions such as "*Does your software validate*
 391 *inputs from untrusted resources?*" and "*What threat assumptions were made when designing*
 392 *protections for your software?*" Another useful question, not included in the DHS questionnaire,
 393 is: "*Does your app access a network application programming interface (API)?*" Note that such
 394 questionnaires can be used only in certain circumstances such as when source code is available
 395 and when developers can answer questions.

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

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

410 Note that the satisfaction or violation of an organization-specific requirement is not based on the
 411 presence or absence of a software vulnerability and thus cannot typically be determined by test
 412 tools. Instead, the satisfaction or violation of organization-specific requirements must be
 413 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].

414 **2.3 Risk Tolerance**

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

419 Risk tolerance should take into account the following factors:

- 420 • Compliance with security regulations, recommendations and best practices
- 421 • Privacy risks
- 422 • Security threats
- 423 • Data and asset value
- 424 • Industry and competitive pressure
- 425 • Management preferences

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

428 **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

429

430 **2.3.1 Tool Report Analysis**

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

434 test tool may classify the estimated risk for using an app as low, moderate, high or severe risk,
435 while another may classify the estimated risk as pass, warning or fail. While some standards
436 exist for expressing risk assessment⁵ and vulnerability reporting⁶ the current adoption of these
437 standards by test tools is low. To the extent possible, it is recommended that an organization use
438 test tools that leverage vulnerability reporting and risk assessment standards. If this approach is
439 not possible, it is recommended that the organization provide sufficient training to analysts on
440 the interpretation of reports and risk assessments generated by test tools.

441 **2.3.2 Compliance versus Certification**

442 For mobile application vetting, two terms are frequently used to demonstrate proof of successful
443 implementation of mobile app security requirements. For a mobile application that has been
444 developed to include security aimed at a particular requirement (e.g. National Information
445 Assurance Partnership – Protection Profile for Mobile App Vetting v.1.1.2) developers may choose
446 to note that they are compliant or certified. The difference depends on the organizations need for
447 compliance or certification.

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

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

460 Note: NIAP lists products on a product-compliant list when a certification has been successfully
461 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 requirements⁷. 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.

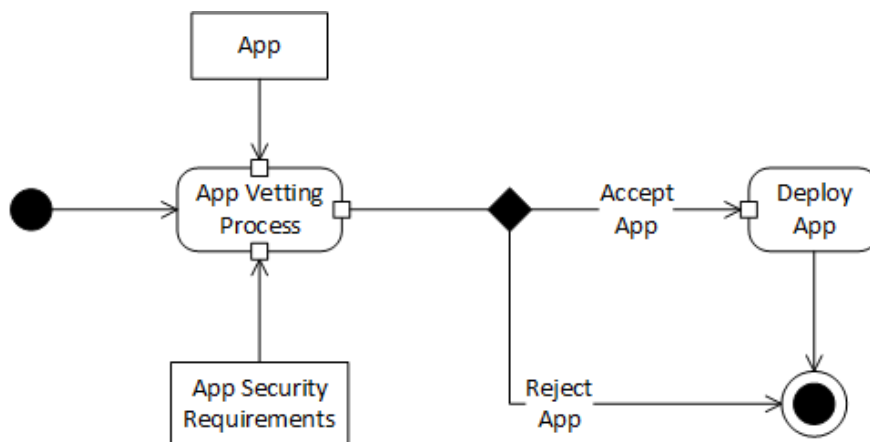


Figure 2 - App vetting process overview.

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.

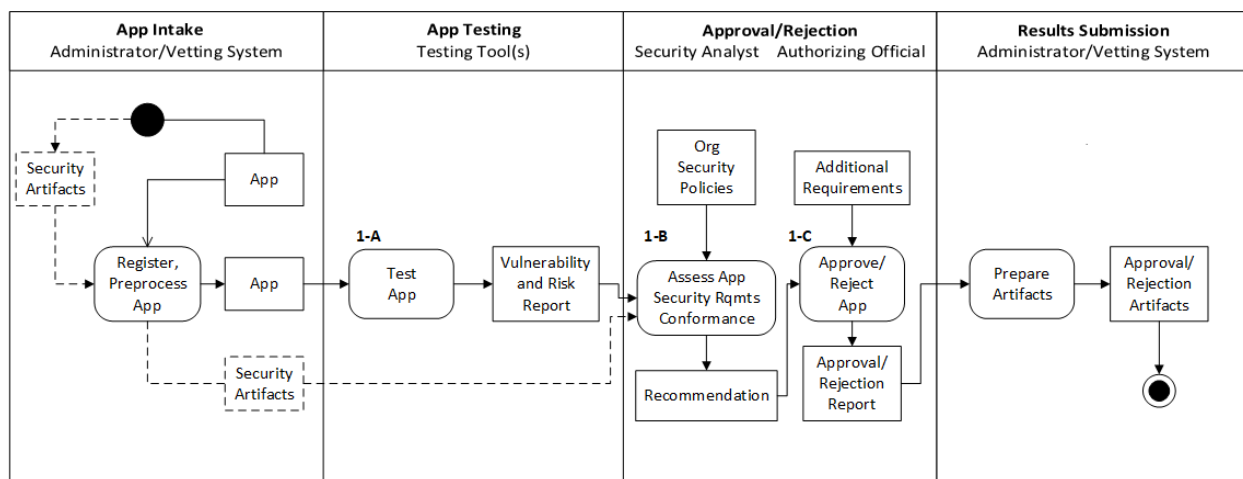
As shown in Figure 1, organizations perform an app vetting process during the app acquisition phase of a mobile application lifecycle; that is, when the app is received by the organization but prior to the app's deployment on the organization's devices. The rationale for this approach stems from the 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.

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

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

496



497

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

498

499

500 3.1 App Intake

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

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

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

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

516 **3.2 App Testing**

517 The app testing process begins after an app has been registered and preprocessed and is
518 forwarded to one or more test tools. A test tool is a software tool or service that tests an app for
519 the presence of software vulnerabilities⁸. Such testing will involve the use of different analysis
520 methodologies (e.g., static analysis) and may be performed manually or automatically. Note that
521 the tests performed by a test tool may identify software vulnerabilities that are common across
522 different apps and will often satisfy general app security requirements (such as those specified by
523 NIAP).

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

531 Figure 4 shows the workflow for a typical test tool. When an app is received by a test tool, it is
532 typically saved as a file on the tool vendor's server. If the test tool is static (i.e., the app's code is
533 analyzed), the app is typically decoded, decompiled or decrypted from its binary executable form
534 to an intermediate form that can be analyzed.⁹ If the test tool is dynamic (i.e., the run-time
535 behavior of the app is analyzed), the app is typically installed and executed on a device or
536 emulator where the behavior of the app can be analyzed. After the tool analyzes the app, it
537 generates a vulnerability report and risk assessment and submits this report to the app vetting
538 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.

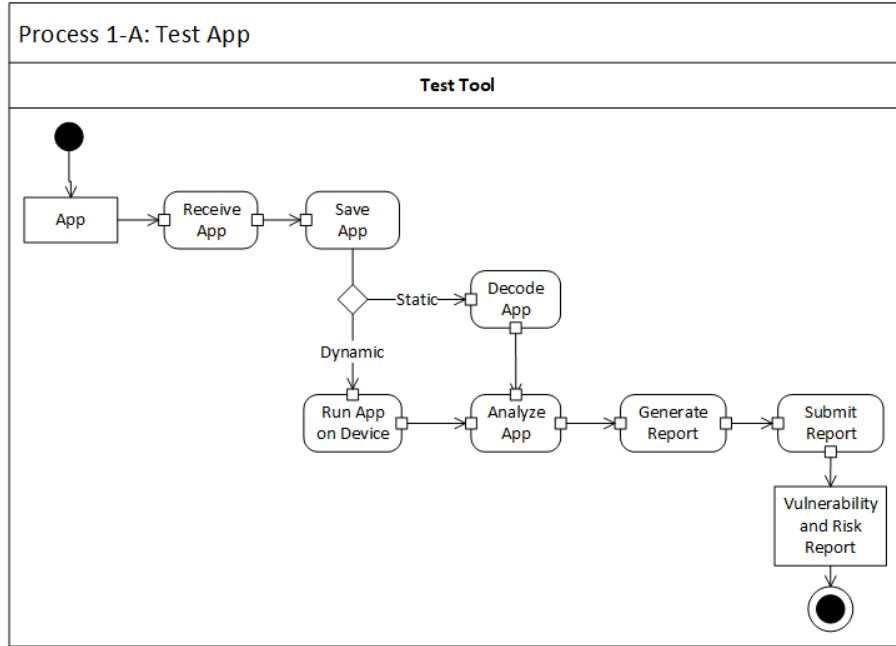


Figure 4 - Test tool workflow.

539

540

541

542 3.3 App Approval/Rejection

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

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

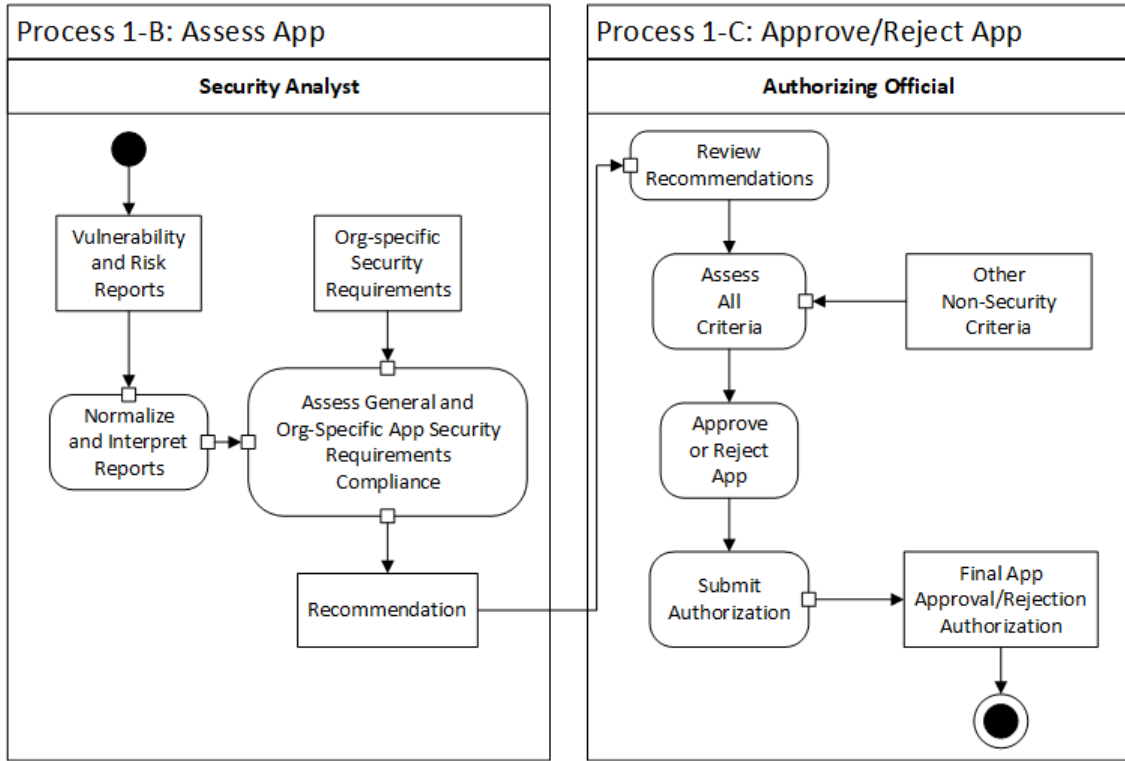


Figure 5 - App approval/rejection process.

559

560

561

562 **3.4 Results Submission**

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

570

571 **4 App Testing and Vulnerability Classifiers**

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

578 **4.1 Testing Approaches**

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

582 **4.1.1 Correctness Testing**

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

598 **4.1.2 Source and Binary Code Testing**

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

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

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

621 **4.1.3 Static and Dynamic Testing**

622 Analysis tools are often characterized as either static or dynamic.¹⁰ Static analysis examines the
623 app source code and binary code and attempts to reason all possible behaviors that might arise at
624 runtime. It provides a level of assurance that analysis results accurately describe the program's
625 behavior regardless of the input or execution environment. Dynamic analysis operates by
626 executing a program using a set of input use-cases and analyzing the program's runtime
627 behavior. In some cases, the enumeration of input test cases is large, resulting in lengthy
628 processing times. However, methods such as combinatorial testing can reduce the number of
629 dynamic input test case combinations, reducing the amount of time needed to derive analysis
630 results [18]. However, dynamic analysis is unlikely to provide 100 percent code coverage [19].
631 Organizations should consider the technical tradeoff differences between what static and
632 dynamic tools offer and balance their usage given the organization's software assurance goals.

633 Static analysis requires that binary code be reverse engineered when source code is not available,
634 which is relatively easy for byte code¹¹ but can be difficult for machine code. Many commercial
635 static analysis tools already support bytecode as do a number of open-source and academic
636 tools.¹² For machine code, it is especially hard to track the flow of control across many functions
637 and to track data flow through variables, since most variables are stored in anonymous memory
638 locations that can be accessed in different ways. The most common way to reverse engineer
639 machine code is to use a disassembler or a decompiler that attempts to recover the original
640 source code. These techniques are especially useful if the purpose of reverse engineering is to
641 allow humans to examine the code because the outputs are in a form that can be understood by
642 humans with appropriate skills. But even the best disassemblers make mistakes [21] and some of
643 those can be corrected with formal static analysis. If the code is being reverse engineered for
644 static analysis, it is preferable to disassemble the machine code directly to a form that the static
645 analyzer understands rather than creating human-readable code as an intermediate byproduct. A
646 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].

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

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

673 **4.2 Vulnerability Classifiers and Quantifiers**

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

677 **4.2.1 Common Weakness Enumeration (CWE)**

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

684 **4.2.2 Common Vulnerability and Exposures (CVE)**

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

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

693 **4.2.3 Common Vulnerability Scoring System (CVSS)**

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

704

705 **5 App Vetting Considerations**

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

708 **5.1 Managed and Unmanaged Apps**

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

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

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

733 **5.2 App Vetting Limitations**

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

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

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

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

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

762 **5.3 Local and Remote Tools and Services**

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

771 **5.4 Automated Approval/Rejection**

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

781 **5.5 Reciprocity**

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

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

803 **5.6 Budget and Staffing**

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

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

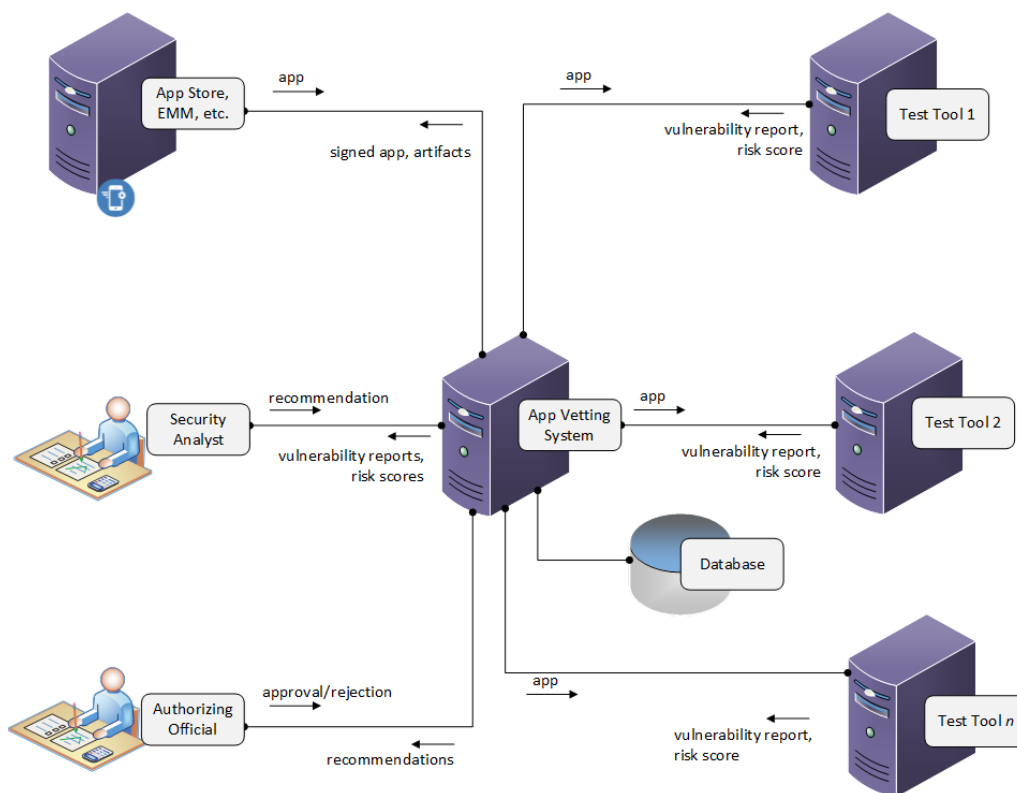
818

819 **6 App Vetting Systems**

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

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

833



834

835

Figure 6 - Example app vetting system architecture.

836

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

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

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

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

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

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

876 An app vetting system uses application programming interfaces (APIs), network protocols and
877 schemas to integrate with distributed third-party test tools as well as clients including app stores.
878 An app vetting system may also include a user interface (UI) dashboard that allows users such as
879 administrators, analysts and authorizing officials to view reports and risk assessments, provide
880 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

884 **Appendix A—Threats to Mobile Applications**

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

900
901 Vetting mobile apps before deploying them onto a user's mobile device can enable an enterprise
902 system administrator to detect software or configuration flaws that may create vulnerabilities or
903 violate enterprise security or privacy policies. Mobile app vetting systems typically include
904 automated testing and analysis tools and may interact with externally hosted vetting services.
905 This section will discuss different classes of malware that affect mobile devices. Mobile app
906 vetting systems are designed to look for evidence of such malware.

907 **A.1 Ransomware**

908
909 Ransomware is malware that encrypts data and holds the decryption key hostage for payment
910 [38] In the mobile environment, new ransomware [39] has been observed that not only encrypts
911 the data of users, but also locks them out of their devices by changing the lock screen PIN. Such
912 ransomware has been spreading as a fake software update via compromised websites.

913 **A.2 Spyware**

914
915 Spyware [40] is malware designed to gather information about an individual or organization
916 without their knowledge and send that information to the attacker's systems. While spyware
917 often has been used to track internet user's movements on the Web, it may also be used to
918 capture SMS messages, photos, phone call logs or sensitive data such as user logins or banking
919 information. Most spyware is installed without a device user's (or the organization's) knowledge,
920 often using deceptive tactics that trick the user into installation. Spyware is generally legal and is
921 often marketed as a tool for parents to monitor their kids or for catching a cheating spouse.
922 Nation-state actors also have used spyware to gather information from mobile users [41].

923 **A.3 Adware**

924
925 Adware is malware that is embedded within or loaded as part of advertisements and is one of the
926 most common threats to mobile devices worldwide. Mobile ads are instrumental to the current

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

934
935

A.4 Rooters

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

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

947
948

A.5 Trojan Horse

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

956
957

A.6 Infostealer

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

965
966

A.7 Hostile Downloader

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

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

975

976 **A.8 Mobile Billing Fraud**

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

983

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

995

996 **A.9 SMS Fraud**

997 Scams once perpetrated via email now are perpetrated via SMS messaging. Fraudulent business
998 transactions, phishing (called “smishing” when delivered via SMS messages), phony requests for
999 donations, fees to claim lottery prizes and cons originating from dating sites are all SMS scams
1000 [51]. Users must be wary of unsolicited texts from strangers or unknown numbers, especially
1001 requests for money or personal/sensitive information.

1002

1003 **A.10 Call Fraud**

1004 Call fraud refers to several malicious and illegal activities. For example, some users of cellular
1005 services may receive calls that appear to originate from domestic area codes, but are actually
1006 associated with international pay-per-call services. These calls often disconnect after one ring.
1007 When the target returns the call he or she is connected to an international line that charges a fee
1008 for connecting in addition to significant per-minute fees if the victim stays on the line. These
1009 charges usually show up on the victim’s cellular bill as premium services.

1010

1011 **A.11 Cramming**

1012 “Cramming” refers to fraudulent activities that result in charges such as fees for calls or services

1013 to a victim’s cellular bill for services that the victim did not order or services with undisclosed
1014 fees. These charges often are assessed by dishonest third-parties of data and communication
1015 services. Carriers and operators often allow third-parties to bill for services by charging to a
1016 user’s cellular bill. Other types of call fraud by third-parties against customers often include
1017 “PBX dial-through,” which can be mounted by placing a call to an enterprise, then requesting to
1018 be transferred to "9-0" or some other outside toll number. More information about different fraud
1019 activities is available from the FTC [49] and the Communication Fraud Control Association
1020 (CFCA).

1021

1022 **A.12 Toll Fraud**

1023 Toll fraud occurs when a mobile device user makes a call—often using premium services—that
1024 is charged to a third-party that did not authorize the call. A common attack with a hacker leasing
1025 phone numbers from a web-based service that charges callers for each call and provides a
1026 percentage of the profit to the hacker. To make a lucrative fraud-based business, the hacker
1027 breaches an independent business’s Voice Over IP (VoIP) network to forward calls to the
1028 hacker’s premium service numbers. The independent company is billed for the calls by the web-
1029 based service and the hacker gets a percentage of the profits. To resist these type of attacks,
1030 organizations must implement strong network security protections.

1031

1032 **Appendix B—Android App Vulnerability Types**

1033 This appendix identifies vulnerabilities specific to apps running on Android mobile devices. The
 1034 scope of this appendix includes app vulnerabilities for Android-based mobile devices running
 1035 apps written in Java. The scope does not include vulnerabilities in the mobile platform hardware
 1036 and communications networks. Although some of the vulnerabilities described below are
 1037 common across mobile device environments, this appendix focuses only on Android-specific
 1038 vulnerabilities.

1039 The vulnerabilities in this appendix are broken into three hierarchical levels, A, B, and C. The A
 1040 level is referred to as the vulnerability class and is the broadest description for the vulnerabilities
 1041 specified under that level. The B level is referred to as the sub-class and attempts to narrow down
 1042 the scope of the vulnerability class into a smaller, common group of vulnerabilities. The C level
 1043 specifies the individual vulnerabilities that have been identified. The purpose of this hierarchy is
 1044 to guide the reader to finding the type of vulnerability they are looking for as quickly as possible.

1045 Table 4 shows the A level general categories of Android app vulnerabilities.

1046 **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.	<ol style="list-style-type: none"> 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

1052 Appendix C—iOS App Vulnerability Types

1053 This appendix identifies and defines the various types of vulnerabilities that are specific to apps
 1054 running on mobile devices utilizing the Apple iOS operating system. The scope does not include
 1055 vulnerabilities in the mobile platform hardware and communications networks. Although some
 1056 of the vulnerabilities described below are common across mobile device environments, this
 1057 appendix focuses on iOS-specific vulnerabilities.

1058 The vulnerabilities in this appendix are broken into three hierarchical levels, A, B, and C. The A
 1059 level is referred to as the vulnerability class and is the broadest description for the vulnerabilities
 1060 specified under that level. The B level is referred to as the sub-class and attempts to narrow down
 1061 the scope of the vulnerability class into a smaller, common group of vulnerabilities. The C level
 1062 specifies the individual vulnerabilities that have been identified. The purpose of this hierarchy is
 1063 to guide the reader to finding the type of vulnerability they are looking for as quickly as possible.

1064 Table 6 shows the A level general categories of iOS app vulnerabilities.

1065 **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, <i>etc.</i>) 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.	<ol style="list-style-type: none"> 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
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

1070 Appendix D—Acronyms

1071 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

1073 **Appendix E—Glossary**

1074 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	<p>organization on behalf of an agency:</p> <p>The function, operation or use of which--</p> <p>involves intelligence activities;</p> <p>involves cryptologic activities related to national security;</p> <p>involves command and control of military forces;</p> <p>involves equipment that is an integral part of a weapon or weapons system; or</p> <p>subject to subparagraph (B) is critical to the direct fulfillment of military or intelligence missions; or</p> <p>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].</p>
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

1076

Appendix F—References

- 1077 [1] P. E. Black, L. Badger, B. Guttman, and E. Fong, “Dramatically reducing software
1078 vulnerabilities: Report to the White House Office of Science and Technology Policy,”
1079 National Institute of Standards and Technology, Gaithersburg, MD, NIST IR 8151, Nov.
1080 2016.
- 1081 [2] R. Kissel, “Glossary of key information security terms,” National Institute of Standards and
1082 Technology, NIST IR 7298r2, May 2013.
- 1083 [3] M. Souppaya and K. Scarfone, “Guidelines for Managing the Security of Mobile Devices
1084 in the Enterprise,” National Institute of Standards and Technology, NIST SP 800-124r1,
1085 Jun. 2013.
- 1086 [4] “Protection Profile for Mobile Device Fundamentals,” p. 183.
- 1087 [5] Joint Task Force Transformation Initiative, “Security and Privacy Controls for Federal
1088 Information Systems and Organizations,” National Institute of Standards and
1089 Technology, NIST SP 800-53r4, Apr. 2013.
- 1090 [6] ISO/IEC, “Information technology -- Security techniques -- Evaluation criteria for IT
1091 security,” ISO/IEC 15408-1:2009.
- 1092 [7] “Requirements for Vetting Mobile Apps from the Protection Profile for Application
1093 Software.” [Online]. Available: [https://www.niap-](https://www.niap-ccevs.org/MMO/PP/394.R/pp_app_v1.2_table-reqs.htm)
1094 [ccevs.org/MMO/PP/394.R/pp_app_v1.2_table-reqs.htm](https://www.niap-ccevs.org/MMO/PP/394.R/pp_app_v1.2_table-reqs.htm). [Accessed: 22-Jun-2018].
- 1095 [8] OWASP, “Mobile Application Security Verification Standard,” v0.9.4.
- 1096 [9] “OWASP Mobile Security Testing Guide - OWASP.” [Online]. Available:
1097 https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide. [Accessed:
1098 22-Jun-2018].
- 1099 [10] M. Peck and C. Northern, “Analyzing the Effectiveness of App Vetting Tools in the
1100 Enterprise,” p. 46, Aug. 2016.
- 1101 [11] “Build Security In | US-CERT.” [Online]. Available: <https://www.us-cert.gov/bsi#ques>.
1102 [Accessed: 22-Jun-2018].
- 1103 [12] “CVE - Common Vulnerabilities and Exposures (CVE).” [Online]. Available:
1104 <https://cve.mitre.org/>. [Accessed: 22-Jun-2018].
- 1105 [13] I. T. L. Computer Security Division, “AppVet | CSRC.” [Online]. Available:
1106 <https://csrc.nist.gov/projects/appvet/>. [Accessed: 22-Jun-2018].
- 1107 [14] M. Pezzè and M. Young, *Software Testing and Analysis: Process, Principles and*
1108 *Techniques*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2008.
- 1109 [15] G. McGraw, *Software security: building security in*. Upper Saddle River, NJ: Addison-
1110 Wesley, 2006.
- 1111 [16] G. G. Schulmeyer, *Handbook of Software Quality Assurance, Fourth Edition*. Norwood,
1112 Massachusetts: Artech House, Inc., 2008.
- 1113 [17] B. B. Agarwal, S. P. Tayal, and M. Gupta, *Software engineering & testing: an introduction*.
1114 Sudbury, Mass: Jones and Bartlett, 2010.
- 1115 [18] J. R. Maximoff, M. D. Trela, D. R. Kuhn, and R. Kacker, “A method for analyzing system
1116 state-space coverage within a t-wise testing framework,” 2010, pp. 598–603.
- 1117 [19] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The art of software testing*, 2nd ed.
1118 Hoboken, N.J: John Wiley & Sons, 2004.
- 1119 [20] H. CHEN, T. ZOU, and D. WANG, “Data-flow Based Vulnerability Analysis and Java
1120 Bytecode,” *7th WSEAS Int. Conf. Appl. Comput. Sci. Venice Italy Novemb. 21-23 2007*,

- 1121 p. 7.
- 1122 [21] “FindBugs™ - Find Bugs in Java Programs.” [Online]. Available:
1123 <http://findbugs.sourceforge.net/>. [Accessed: 22-Jun-2018].
- 1124 [22] R. Shah, “Vulnerability Assessment of Java Bytecode,” Auburn University, 2005.
- 1125 [23] WAIM (Conference) *et al.*, *The Ninth International Conference on Web-Age Information*
1126 *Management: WAIM 2008*. Piscataway, N.J.: IEEE, 2008.
- 1127 [24] “CWE - Common Weakness Enumeration.” [Online]. Available: <http://cwe.mitre.org/>.
1128 [Accessed: 22-Jun-2018].
- 1129 [25] FIRST.org, Inc, “CVSS v3.0 Specification,” p. 21.
- 1130 [26] “Mobile Application Security Requirments.” Department of Defense, 06-Oct-2017.
- 1131 [27] M. Dowd, J. McDonald, and J. Schuh, *The art of software security assessment: identifying*
1132 *and preventing software vulnerabilities*. Indianapolis, Ind: Addison-Wesley, 2007.
- 1133 [28] H. G. Rice, “Classes of recursively enumerable sets and their decision problems,” *Trans.*
1134 *Am. Math. Soc.*, vol. 74, no. 2, pp. 358–358, Feb. 1953.
- 1135 [29] J. H. Allen, Ed., *Software security engineering: a guide for project managers*. Upper
1136 Saddle River, NJ: Addison-Wesley, 2008.
- 1137 [30] Archiveddocs, “The STRIDE Threat Model.” [Online]. Available:
1138 [https://docs.microsoft.com/en-us/previous-versions/commerce-](https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v%3dcs.20))
1139 [server/ee823878\(v%3dcs.20\)](https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v%3dcs.20)). [Accessed: 22-Jun-2018].
- 1140 [31] “Trike: Trike.” [Online]. Available: <http://www.octotrike.org/home.shtml>. [Accessed: 22-
1141 Jun-2018].
- 1142 [32] “Tool Survey - SAMATE.” [Online]. Available:
1143 https://samate.nist.gov/index.php/Tool_Survey.html. [Accessed: 22-Jun-2018].
- 1144 [33] M. A. Ogata, “An overview of mobile application vetting services for public safety,”
1145 National Institute of Standards and Technology, Gaithersburg, MD, NIST IR 8136, Jan.
1146 2017.
- 1147 [34] “NVD - Home.” [Online]. Available: <https://nvd.nist.gov/>. [Accessed: 22-Jun-2018].
- 1148 [35] I. T. L. Computer Security Division, “Security Content Automation Protocol | CSRC.”
1149 [Online]. Available: <https://csrc.nist.gov/projects/security-content-automation-protocol/>.
1150 [Accessed: 22-Jun-2018].
- 1151 [36] “CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC).” [Online].
1152 Available: <https://capec.mitre.org/>. [Accessed: 22-Jun-2018].
- 1153 [37] J. M. Stecklein, B. Dick, B. Haskins, R. Lovell, and G. Moroney, “Error Cost Escalation
1154 Through the Project Life Cycle,” presented at the 14th Annual International Symposium,
1155 Toulouse; France, 2004.
- 1156 [38] M. Bartock, J. Cichonski, M. Souppaya, M. Smith, G. Witte, and K. Scarfone, “Guide for
1157 cybersecurity event recovery,” National Institute of Standards and Technology,
1158 Gaithersburg, MD, NIST SP 800-184, Dec. 2016.
- 1159 [39] S. Khandelwal, “New Ransomware Not Just Encrypts Your Android But Also Changes
1160 PIN Lock,” *The Hacker News*. [Online]. Available:
1161 <https://thehackernews.com/2017/10/android-ransomware-pin.html>. [Accessed: 22-Jun-
1162 2018].
- 1163 [40] W. A. Jansen, T. Winograd, and K. Scarfone, “Guidelines on Active Content and Mobile
1164 Code,” p. 62.
- 1165 [41] “When ‘Grandma-Proof’ Android Spyware Is Good Enough For International Espionage.”
1166 [Online]. Available: <https://www.forbes.com/sites/thomasbrewster/2018/05/15/apple->

- 1167 iphone-spouseware-used-in-pakistan-government-attacks/#74f2e2515668. [Accessed: 22-
1168 Jun-2018].
- 1169 [42] S. Dent, “Report finds Android malware pre-installed on hundreds of phones,” *Engadget*,
1170 24-May-2018. [Online]. Available: [https://www.engadget.com/2018/05/24/report-finds-](https://www.engadget.com/2018/05/24/report-finds-android-malware-pre-installed-on-hundreds-of-phones/)
1171 android-malware-pre-installed-on-hundreds-of-phones/. [Accessed: 22-Jun-2018].
- 1172 [43] H. Zhang, D. She, and Z. Qian, “Android Root and its Providers: A Double-Edged Sword,”
1173 2015, pp. 1093–1104.
- 1174 [44] J. Mello Jr, “Marcher Malware Poses Triple Threat to Android Users | Malware |
1175 TechNewsWorld,” *Tech News Workd*, 07-Nov-2017. [Online]. Available:
1176 <https://www.technewsworld.com/story/84936.html>. [Accessed: 22-Jun-2018].
- 1177 [45] D. Palmer, “Irremovable bank data-stealing Android malware poses as Google Chrome
1178 update,” *ZDNet*. [Online]. Available: [https://www.zdnet.com/article/irremovable-bank-](https://www.zdnet.com/article/irremovable-bank-detail-stealing-android-malware-poses-as-google-chrome-update/)
1179 detail-stealing-android-malware-poses-as-google-chrome-update/. [Accessed: 22-Jun-
1180 2018].
- 1181 [46] M. Moon, “Malware downloader infects your PC without a mouse click,” *Engadget*.
1182 [Online]. Available: [https://www.engadget.com/2017/06/11/malware-downloader-](https://www.engadget.com/2017/06/11/malware-downloader-infects-your-pc-without-a-mouse-click/)
1183 infects-your-pc-without-a-mouse-click/. [Accessed: 22-Jun-2018].
- 1184 [47] L. Whitney, “AT&T to pay \$105 million to settle charges over mobile billing,” *CNET*, 08-
1185 Oct-2014. [Online]. Available: [https://www.cnet.com/news/at-t-to-pay-105-million-to-](https://www.cnet.com/news/at-t-to-pay-105-million-to-settle-fraudulent-mobile-bill-charges/)
1186 settle-fraudulent-mobile-bill-charges/. [Accessed: 22-Jun-2018].
- 1187 [48] J. Brodtkin, “Verizon and Sprint pay \$158 million in fines for fraudulent phone charges,”
1188 *Ars Technica*, 12-May-2015. [Online]. Available: [https://arstechnica.com/tech-](https://arstechnica.com/tech-policy/2015/05/verizon-and-sprint-pay-158-million-in-fines-for-fraudulent-phone-charges/)
1189 policy/2015/05/verizon-and-sprint-pay-158-million-in-fines-for-fraudulent-phone-
1190 charges/. [Accessed: 22-Jun-2018].
- 1191 [49] “Mystery Phone Charges,” *Consumer Information*, 16-Dec-2013. [Online]. Available:
1192 <https://www.consumer.ftc.gov/articles/0183-mystery-phone-charges>. [Accessed: 22-Jun-
1193 2018].
- 1194 [50] H. Weisbaum, “Fraud Alert: ID Thieves Hijack Mobile Phone Accounts,” *NBC News*.
1195 [Online]. Available: [https://www.nbcnews.com/tech/tech-news/fraud-alert-id-thieves-](https://www.nbcnews.com/tech/tech-news/fraud-alert-id-thieves-hijack-mobile-phone-accounts-n599761)
1196 hijack-mobile-phone-accounts-n599761. [Accessed: 22-Jun-2018].
- 1197 [51] L. Spector, “5 common SMS text scams, and how to avoid them | PCWorld,” *PC World*,
1198 01-Mar-2016. [Online]. Available: [https://www.pcworld.com/article/3034696/mobile/5-](https://www.pcworld.com/article/3034696/mobile/5-common-sms-text-scams-and-how-to-avoid-them.html)
1199 common-sms-text-scams-and-how-to-avoid-them.html. [Accessed: 22-Jun-2018].
- 1200 [52] *Federal Information Security Modernization Act of 2014*. 2014.
- 1201