

Mitigating Bring Your Own Device Risks by Static Analysis
Empowered by Knowledge Graphs from
Open Web Application Security Project

by
Suzanna Schmeelk

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

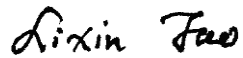
at

Seidenberg School of Computer Science and Information Systems

Pace University

May 2020

We hereby certify that this dissertation, submitted by Suzanna Schmeelk, satisfies the dissertation requirements for the degree of *Doctor of Professional Studies in Computing* and has been approved.



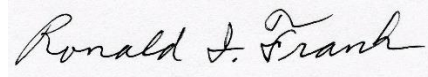
Dr. Lixin Tao
Chairperson of Dissertation Committee

- 4/27/2020
Date



Dr. Charles Tappert
Dissertation Committee Member

- 4/27/2020
Date



Dr. Ron Frank
Dissertation Committee Member

- 4/27/2020
Date

School of Computer Science and Information Systems
Pace University 2020

Abstract

Mitigating Bring Your Own Device Risks by Static Analysis Empowered by Knowledge Graphs from Open Web Application Security Project

by
Suzanna Schmeelk

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

May 2020

Many organizations, to save costs, are moving to Bring Your Own Mobile Device (BYOD). In these scenarios, organizations have a Mobile Device Management (MDM) system in place. MDM solutions lower cyber security risks by providing remote wipe procedures, geo-location fencing, among others. However, MDM systems are not yet focused on application-level security with a fine-level of granularity. MDM systems currently may not monitor for data loss prevention (DLP), or even standard web-application vulnerabilities that a penetration tester would examine. In addition, organizations around the world are adopting applications built by third-parties at an unprecedented rate.

This research contributes an examination of mobile application security through the construction of a knowledge graph for the OWASP Top 10 2014 and 2016 threats. The knowledge graph contribution links threats from the different years to show changes in time and to help determine security changes over time. Currently, only the National Institute of Standards and Technology (NIST) Bug Framework has built any such graph representation to inform analysis. This high-level graphic shows potential vulnerabilities such as the insecure storage of sensitive data and insufficient cryptography, which depending on how the code is utilized can occur heavy fines for the mismanagement of sensitive information.

This research then contributes how specific mobile device source code, specifically Android in this research, can be useful to inform static analysis. In this research we focus on source code analysis; however, the knowledge-graph can be connected to byte code or entirely other mobile device application languages such as Swift, JavaScript, and C/C++.

We then make a contribution to analyze over 200 healthcare Android applications source code from GitHub to learn what, if any, security concerns are being deployed to improve

secure source code development for sensitive data. Some of the applications analyzed collect highly sensitive information such as body weight, body signals (e.g. blood pressure, temperature), obstetrics/gynecology measurements, mental health measurements, among others. Specifically, in this research, we analyzed applications for components of the constructed-knowledge graphs, specifically, components of the confidentiality and integrity of their sensitive information.

As our world moves more-and-more to the edges with the Internet of Things and mobile application development, security concerns and the storage and transmission of sensitive data is becoming a serious concern. In fact, recent regulatory changes are occurring at unprecedented rates with adoptions of new laws at local, national and international levels. Having a clearer picture of security on our mobile devices is now an industry necessity.

Acknowledgements

This dissertation has been a crescendo on my work for cybersecurity software assurance of mobile applications. This work began with deep mobile application software assurance research during my fulltime career work with a former Bell Laboratories subsidiary, later involved mobile application cybersecurity analysis for top New York City hospital(s) and most recently with my industry (pro bono) work as a fulltime Assistant Professor of Cybersecurity. Throughout my fulltime work with these organizations, I observed a lack of cybersecurity completeness in research and practice, which was deeply disturbing. The faculty at Pace University have helped me address many of these observed cybersecurity gaps for the benefit of the research, industry, and general public communities-at-large.

I would like to first thank Dr. Lixin Tao, my adviser, for all his time, support and advice during the years working on this dissertation thesis. Dr. Tao, Pace University Seidenberg School of CSIS Professor and IEEE Senior Member, introduced me to the domain of the Semantic Web Knowledge Graphs, which was a perfect cybersecurity fit for adapting and building more complete representations and adaptable knowledge sets for cybersecurity software assurance. Dr. Tao's contributions to my own personal growth have been consistently domain-enhancing, consistently encouraging, and will never be forgotten. Thank you so much, Dr. Tao!

Thank you to my dissertation committee, especially Dr. Charles Tappert and Dr. Ron Frank, who have been so generous with their technical experiences and teachings. Dr. Tappert, Pace University Seidenberg School of CSIS Clinical Professor, has contributed to every class we took in Pace University. He regularly facilitated discussions based on his outstanding and long career at IBM and earlier professional experience at West Point Military Academy. His contributions to his students' learning are endless and will never be forgotten. In fact, I recall an episode at an International Machine Learning conference, where Dr. Tappert was generously helping all students from Pace University prepare and present their research. He came to all their sessions and made sure session track chairs were on schedule. Thank you, Dr. Tappert!

Dr. Ron Frank, Pace University Seidenberg School of CSIS Associate Professor, has been so generous with his time and sharing his industry expertise. Dr. Frank spent his first career working in industry at IBM before coming to Pace University. He has phenomenal experience and an outstanding repertoire. His stories are extremely authentic and geared to enlightening all his students to prepare them for the next steps in their careers. I recall an episode where Dr. Frank taught all his students about maintaining research memoirs in notebooks which could endure in intellectual property court disputes. This is one example of many of his extremely forward-thinking teaching moments to prepare students to think in advance about preserving their own intellectual property. Thank you, Dr. Frank!

In addition, all the faculty at Pace University have been extremely supportive--especially of their advice, time and industry expertise. I would like to thank our DPS classroom faculty: Dr. Tilak Agerwala, Paul Dantzig, Dr. Sung-Hyuk Cha, Dr. Li-Chiou Chen, Dr. Yegin Genc, Rinaldo DiGiorgio, and Istvan Barabasi. In addition, the Pace faculty and staff, especially the Dean Dr. Jonathan Hill, Dr. Susan Feather-Gannon, Dr. Christelle Scharff, Dr. Miguel Mosteiro, Dr. Thomas Schmidt, Michelle Lang, Jose Cueto, and Jill Olimpieri have also been generous of their time, advice, and expertise. Finally, a big *thank you* to Fred Grossman, Charles Tappert, Joe Bergin, Susan M. Merritt, Allen Stix, Judith E. Sullivan, and David A. Sachs for working so hard to imagine and establish a research doctoral program for fulltime working professionals in the first place.

Thank you to the other DPS Students, who are all fulltime working professionals and/or industry veterans. First, I would like to thank my NYPD Cyber Detective industry veteran colleague and peer, Denise Dragos, for our many research undertakings together. Second, I would like to thank my amazing DPS Team 2 Cohort: Lisa R. Ellrodt, Tonya L. Fields, Ion C. Freeman, and Ashley J. Haigler (including the founding group members Lynne Larkin and Ronald Williams.) Thank you also to our entire DPS 2020 Cohort for sharing their wealth of industry knowledge on our long Friday nights together and all day Saturdays together. Our DPS 2020 Cohort consists of: Miguel Zhindon, Philip Ricciardi, Joseph Porter, Edmund Miller, Paxton Louis, David Lasecki, Rajesh Khemraj, and Binu Jacob. Lastly, many thanks to Dr. Ning Jiang for his work with me on Pace Protegee.

Finally, thank you to my family who have been extremely supportive especially when I have missed family events due to my studies.

As the proverb says, "It takes a village to raise a child." We have learned so much about the computing industry, collaborated with our (invited) industry leading peers, and expanded our own computing industry experience. We have spent many days together both through in-person and virtual communications and conferences. Overall, the learning experience has been extremely rewarding, enlightening, positive, and entirely unforgettable.

Table of Contents

Abstract	iv
List of Tables	viii
List of Figures	x
Chapter 1 Introduction.....	1
1.1 Opportunities and Challenges in Mobile Security Analyses	2
1.2 Problem Statement	3
1.3 Expected Contributions.....	6
1.4 Approach Validation.....	7
1.5 Research Outline.....	7
Chapter 2 Review of Literature	9
2.1 Cybersecurity Ontologies.....	10
2.1.1 Ontologies for HIPAA/HITECH Data Breaches	11
2.1.2 Ontologies for Incident Response.....	11
2.2 Vulnerability and Security Discourse	13
2.2.1 NIST’s Bugs Framework (BF).....	14
2.2.2 MITRE’s Common Weakness Enumeration (CWE).....	15
2.2.3 Malicious Application Detection (MITRE’s CAPEC)	16
2.3 Static Analysis of Mobile Applications	16
2.3.1 Confidentiality Techniques.....	17
2.3.2 Integrity Techniques	19
2.3.3 Availability Techniques	20
2.3.4 Generalizable Techniques.....	20
2.3.5 Other Polyhedral Techniques.....	21
2.3.6 Graphs of Domain Coverage Findings	22
2.4 Ontologies and Knowledge Graphs	26

2.4.1	Ontologies for Software Development	27
2.4.2	Knowledge Graphs for Software Development	28
Chapter 3	Mobile Application Client-Side Structure	29
3.1	Android Linux Sandbox Structure	31
3.2	Android Application Structure.....	36
3.3	Standard Android Application Security Management Coding Patterns	40
3.3.1	Android API.....	40
3.3.2	Vulnerable Libraries	43
3.3.3	Limitations	43
Chapter 4	Ontology of Mobile Application Security Threats	44
4.1	OWASP 2014 Mobile Threats Coding Patterns	44
4.1.1	Mobile 2014 Threat 1: Weak Server-Side Controls	45
4.1.2	Mobile 2014 Threat 2: Insecure Data Storage	46
4.1.3	Mobile 2014 Threat 3: Insufficient Transport Layer Protection.....	53
4.1.4	Mobile 2014 Threat 4: Unintended Data Leakage.....	56
4.1.5	Mobile 2014 Threat 5: Poor Authorization and Authentication	61
4.1.6	Mobile 2014 Threat 6: Broken Cryptography	64
4.1.7	Mobile 2014 Threat 7: Client-side Injection.....	65
4.1.8	Mobile 2014 Threat 8: Security Decisions Via Untrusted Inputs.....	67
4.1.9	Mobile 2014 Threat 9: Improper Session Handling	69
4.1.10	Mobile 2014 Threat 10: Lack of Binary Protections	72
4.2	OWASP 2016 Mobile Threats Coding Patterns	74
4.2.1	Mobile 2016 Threat 1: Improper Platform Usage.....	74
4.2.2	Mobile 2016 Threat 2: Insecure Data Storage	76
4.2.3	Mobile 2016 Threat 3: Insecure Communication	77
4.2.4	Mobile 2016 Threat 4: Insecure Authentication	78
4.2.5	Mobile 2016 Threat 5: Insufficient Cryptography	78

4.2.6	Mobile 2016 Threat 6: Insecure Authorization.....	82
4.2.7	Mobile 2016 Threat 7: Poor Code Quality	83
4.2.8	Mobile 2016 Threat 8: Code Tampering	83
4.2.9	Mobile 2016 Threat 9: Reverse Engineering	84
4.2.10	Mobile 2016 Threat 10: Extraneous Functionality	84
4.3	Summary of OWASP Threat/Risk Findings.....	85
Chapter 5	Detectable Security Management Coding Patterns.....	87
5.1	Analysis of Android Health Source Code for OWASP Top Security Issues ...	87
5.2	Analysis 1 Summary: OWASP 2016 Threat 5 Insufficient Cryptography	88
5.3	Analysis 2 Summary: OWASP 2016 Threat 2 Insecure Data Storage	91
Chapter 6	Conclusion	103
6.1	Contribution Summary.....	103
6.1.1	Contribution #1: Android Mobile Application Knowledge Graph.....	103
6.1.2	Contribution #2: Evaluate Knowledge-Graph.	105
6.1.3	Contribution #3: Identify Unresearched Mobile Vulnerabilities	106
6.1.4	Contribution #4: Analyze Software Assurances in 200+ Applications	106
6.2	Future Research	106
Appendix A	A List of Android Healthcare Mobile Applications Analyzed	109
Appendix B	Glossary of Terms	124
Reference	128

List of Tables

Table 1: CAPEC “Mechanisms of Attack”	22
Table 2: Android Java Code for File Storage [49].....	49
Table 3: Android Java Code for Shared Preferences [53]	50
Table 4: Android Java Code for SQL Lite Database [55]	51
Table 5: Valid Android HTTPS Request [58]	54
Table 6: Android adding a Certifying Authority source code [58].....	55
Table 7: Android Certificate Hostname Verifier source code [58].....	55
Table 8: Android Static Certificate Pinning in manifest source code [59].....	56
Table 9: Android Dynamic Certificate Pinning source code [59]	56
Table 10: Android custom keyboard source code [61].....	58
Table 11: Android URL caching source code [62].....	59
Table 12: Android caching source code [63].....	59
Table 13: Android copy/paste from clipboard source code [64]	60
Table 14: Android screenshot source code [65]	61
Table 15: Android OAuth source code [66]	62
Table 16: Android credential source code [67].....	62
Table 17: Android biometric authentication source code [63]	63
Table 18 Nikolay Elenkov Example [71] [72].....	65
Table 19: Android running JavaScript [63]	66
Table 20: Android client-side injection source code [74].....	67
Table 21: Permissions between two+ co-owned applications source code [63].....	69
Table 22: Android disallow export of Content Providers source code [63]	69
Table 23: Android API Cookie Flags [76].....	71
Table 24: Android signing remnants source code [80].....	74
Table 25: Android Implementation CheckServerTrusted source code [82]	77
Table 26 Nikolay Elenkov Example [71] [72].....	81

Table 27: Relations Introduces in OWASP Mobile Threat Knowledge-Graph	85
Table 28: Analysis 1 OWASP 2016 Mobile Threat 5 Insufficient Cryptography.....	88
Table 29: Analysis 2 OWASP 2016 Mobile Threat 2 Insecure Data Storage (IDS).....	94
Table 30: NIST SAMATE BF - Buffer Overflow (BOF) [93].....	104
Table 31: Full List of Android Healthcare Mobile Applications Analyzed	109

List of Figures

Figure 1: HIPAA Breach Ontology from Kafali et al. [4, p. 532]	10
Figure 2: Ontology of healthcare users Kafali et al. [4, p. 532]	11
Figure 3: NIST BF - Buffer Overflow (BOF) Class [11]	15
Figure 4: Fraction of CAPEC Categories Researched [14]	23
Figure 5: The fraction of NIST BF categories researched [15]	25
Figure 6: Protégé Interface [28]	26
Figure 7: ART vs DVM [34]	30
Figure 8: Android Operating System [40]	31
Figure 9: Android Mobile Device Software Stack [41]	33
Figure 10: File format comparison - APK vs. Jar [43].	37
Figure 11: Android Application Basic Lifecycle [45].	39
Figure 12: Android Cipher API [46]	41
Figure 13: OWASP Mobile Top Ten Threats of 2014	45
Figure 14: OWASP Mobile 2014 Threat 1: Weak Server-Side Controls	45
Figure 15: OWASP Mobile 2014 Threat 2: Insecure Data Storage	47
Figure 16: OWASP Mobile 2014 Threat 3: Insufficient Transport Layer Protection	53
Figure 17: OWASP Mobile 2014 Threat 4: Unintended Data Leakage	57
Figure 18: OWASP Mobile 2014 Threat 5: Poor Authorization and Authentication	61
Figure 19: OWASP Mobile 2014 Threat 6: Broken Cryptography	64
Figure 20: OWASP Mobile 2014 Threat 7: Client-side Injection	66
Figure 21: OWASP Mobile 2014 Threat 8: Security Decisions Via Untrusted Inputs	68
Figure 22; OWASP Mobile 2014 Threat 9: Improper Session Handling	70
Figure 23: OWASP Mobile 2014 Threat 10: Lack of Binary Protections	72
Figure 24: Android Application Signing [80]	73
Figure 25: OWASP Mobile Top Ten Threats of 2016	74
Figure 26: OWASP Mobile 2016 Threat 1: Improper Platform Usage	75

Figure 27: OWASP Mobile 2016 Threat 2: Insecure Data Storage.....	76
Figure 28: OWASP Mobile 2016 Threat 3: Insecure Communication.....	77
Figure 29: OWASP Mobile 2016 Threat 4: Insecure Authentication.....	78
Figure 30: OWASP Mobile 2016 Threat 5: Insufficient Cryptography	79
Figure 31: OWASP Mobile 2016 Threat 6: Insecure Authorization	82
Figure 32: OWASP Mobile 2016 Threat 7: Poor Code Quality	83
Figure 33: OWASP Mobile 2016 Threat 8: Code Tampering	83
Figure 34: OWASP Mobile 2016 Threat 9: Reverse Engineering	84
Figure 35: OWASP Mobile 2016 Threat 10: Extraneous Functionality.....	85
Figure 36: Application seeking permission to be entirely stored on external storage	92
Figure 37: The Google documentation guidance for install location	93

Chapter 1

Introduction

Many organizations, to save costs, are moving to Bring Your Own Mobile Device (BYOD). In these scenarios, many organizations have a Mobile Device Management (MDM) system in place. MDM solutions lower cyber security risks by providing remote wipe procedures, geo-location fencing, among others. However, MDM systems are not yet focused on application-level security with a fine-level of granularity. MDM systems currently may not monitor for data loss prevention (DLP), or typically web-application vulnerabilities that a penetration tester would examine.

In addition, the BYOD paradigm supports users installing personal, and professional applications on the same work device. In such scenarios, penetration tests typically perform tests on applications specific to the organization; however, they have no way of knowing in advance which applications will be run on the device in addition to the work-related application. Penetration tests of mobile devices traditionally may include both static and dynamic application tests. Since mobile computing is relatively new, many penetration testers and application developers have no way of knowing how best to analyze the applications. Currently, penetration tests are ad hoc and can differ between organizations. This dissertation research is significant as it introduces knowledge graphs

as a methodology to examine how mobile security is covered by static analysis and identifies issues which have not been examined in academics. The OWASP knowledge graphs can be used standalone, by penetration testers, by developers, by quality assurance testing teams, by security community researchers (e.g. in industry and by government standardizing organizations such as the National Institute of Standards and Technology (NIST)), and by static analysis tool producers to further enhance their tools and coverage representations. In this dissertation, we show how the knowledge graphs can be used during static analysis to improve the overall mobile security code during development. Over 200 Android Healthcare applications were reviewed from GitHub to improve their mobile cybersecurity risks in accordance with OWASP.

1.1 Opportunities and Challenges in Mobile Security Analyses

Analysis of mobile applications for security issues is extremely relevant to many organizations; however, it comes with challenges. Specifically, typically tools developed to test for security concerns have four types of findings: true positive, false positive, true negative and false negative. A true positive security issue detection means that a tool properly identified a security concern. A false positive security issue identified by the tool has been improperly characterized by the tool as a security issue when in fact it is not a security issue. The lower the false positive rate the more sound a tool is considered. A true negative is when a tool properly labels an issue as a non-issue. A false negative, one of the most serious cases, is when a tool improperly identifies a security issue as a non-issue. The lower the false negatives the more complete a tool is considered to be.

Currently very little, if any research is properly working to identify false negative statistics for security analysis tools. Generally standardizing bodies such as NIST work to help develop benchmarks to help properly classify and analyze security analysis tools.

Two popular security analysis tools are built around static analysis or dynamic analysis or the hybrid of the two. Static analysis examines source, intermediate, or machine code to identify issues. Traditional static analysis applications are within compiler tools. Dynamic analysis tools examine code during execution. There are many methodologies of dynamic analysis such as code coverage, code behavior analysis and sandboxes. Finally, research tools can combine methodologies such as in malware analysis to statically remove timers or other obfuscating code before actually executing the code to examine the behavior.

As discussed in Chapter 2 Review of Literature the research community, especially in mobile static analysis, is not systematically examining software issues. As explained in the chapter literature, systematic analysis is transpiring over certain well-known security issues leaving others entirely unresearched increasing security risks.

1.2 Problem Statement

Organizations around the world are adopting mobile technology and applications built by third parties at an unprecedented rate. This research examines software assurance methodologies specifically through source code analysis to improve mobile application cybersecurity risks in accordance with OWASP best practices. Security analysis coverage of static analysis in Android can be further implanted for malware prevention, mitigation,

and detection. This research examines current static analysis research of Android mobile application to discover trends in analysis. Recent academic publications present the coverage and distribution of current application of static analysis on Android detection, mitigation, and prevention challenges. This research then develops a new knowledge-graph for Android security based on OWASP best practices as contribution. This research links the knowledge graphs with Android source code as empirical evidence that static analysis techniques can be a useful methodology for threat detection, mitigation, and prevention. The dissertation then presents compiles new vulnerabilities useful for detection by static analysis methodologies and derives conclusions about the research trend, which subareas are suitable for static analysis and needs more research. Finally, we then make a contribution to analyze over 200 healthcare Android applications source code from GitHub to learn what, if any, security concerns are being deployed to improve secure source code development for sensitive data. Some of the applications analyzed collect highly sensitive information such a body weight, body signals (e.g. blood pressure, temperature), obstetrics/gynecology measurements, mental health measurements, among others. Specifically, in this research, we analyzed applications for components of the constructed-knowledge graphs, specifically, components of the confidentiality and integrity of their sensitive information.

Methodology Solution Statement

Neither a mobile cybersecurity knowledge graph nor its respective cybersecurity static analysis knowledge graph currently exists. As such, there is considerable effort not only

in building a knowledge graph but also in graph justification. As a result, we have started with the industry standard OWASP Top 10 Mobile Threats to construct initial components of a security knowledge graph for one particular mobile platform, Android. The two most current OWASP Top 10 Mobile Threats published on their main page are from 2014 and 2016. The OWASP frameworks are highly utilized by industry professionals to analyze their code for vulnerabilities and to learn how to solve penetration test remediation problems. We explore both years of OWASP Top 10 Mobile Threats in more detail to see the differences and similarities between the years to work towards building a theoretically complete knowledge-graph to capture all known threats at a point in time.

This research examines mobile application security from the perspective of static analysis to develop new security-focused software analysis methodologies by making three contributions. First, this research contributes knowledge graphs for security threats in mobile applications. These knowledge graphs can be employed by industry to detect software weaknesses during software analysis (e.g. static, dynamic, penetration testing, legal, etc.). Second, the knowledge graphs are connected directly with mobile Android software code to provide real-world empirical evidence to the context of the analysis. Third, the research contributes what static analysis techniques could be employed to capture different categories of mobile application security threats that have previously not been researched or identified in research. Specifically, the source code for 200+ Android healthcare mobile applications are statically examined to improve their mobile cybersecurity risks with OWASP. The field at large can apply and extend any of the contributions to further guide their mobile application cybersecurity analysis.

Specifically, the solution methodology are as follows:

- Review the literature on current cybersecurity static analysis research trends
- Develop knowledge-graphs for Android mobile security based on OWASP
- Evaluate the knowledge-graph with Android source code to improve static analysis techniques capable of detecting cybersecurity risks
- Analyze the mobile source code for 200+ open source Android healthcare applications hosted on GitHub to improve their mobile cybersecurity risks with OWASP

1.3 Expected Contributions

Little systematic research exists for mobile cybersecurity software assurance. As such, industry and academic experts find it difficult to compare and contrast assurance analyses. In fact, penetration tests at one organization may not be equivalent to a penetration test at another organization. Our expected contribution is to improve systematic mobile software security research and to improve their mobile cybersecurity risks. Specifically, the contributions are as follows:

- Develop knowledge-graphs for Android mobile application security based on both of the current OWASP's top ten mobile threats to identify similarities, identify differences and discover longitudinal changes to the OWASP threats for more complete analyses.
- Evaluate knowledge-graph with Android source code connections to elicit types of static analysis techniques capable of detecting

- Identify un-researched mobile software vulnerabilities detectable via static analysis.
- Analyze the software assurances in 200+ open source Android healthcare applications hosted on GitHub to improve their cybersecurity risks with respect to OWASP.

1.4 Approach Validation

As empirical evidence we will examine both the developed OWASP 2014 and 2016 knowledge-graphs through the lens of Android mobile source code by examining 200+ open source healthcare applications which store inherently sensitive information for two of the top ten threats. Future research can extend the developed OWASP knowledge-graphs as known security issues develop.

1.5 Research Outline

This dissertation research examines through literature, building knowledge graphs, and analyzing Android source code from GitHub, new classes of issues to be considered with static analysis.

Chapter 2 reviews the related literature on mobile cybersecurity software assurance with respect to static analysis. The chapter examines research use cases for cybersecurity ontologies. It then examines methodologies to systematically identify and communicate vulnerabilities. The next section explores current literature on mobile application security analyses. Finally, the chapter explores current knowledge graph techniques and literature.

Chapter 3 introduces the Android mobile application structure focusing on the client-side applications structure—the scope of this dissertation research. This section includes the Android Linux sandbox structure, accessing mobile resources from within an application and discusses additional client-side application features such as including external packages and native code. In addition, this section includes an overview discussion on mobile client-server architecture will be discussed to inform on client-side research scope.

Chapter 4 identifies current applications of static analysis to various mobile threats, and argues/describes, based on the knowledge graph dependencies among the threats and among the mobile application infrastructure components, what additional application of static analysis could be used to other related threats.

Chapter 5 examines detectable security management coding patterns to improve their mobile cybersecurity risks with the constructed OWASP knowledge-graph. The Android source code for 200+ Android healthcare mobile applications is statically analyzed in accordance with multiple OWASP Top 10 Threats to improve their risks. The applications store and handle extremely sensitive healthcare information which have higher impact if exposed through loss or theft.

The last chapter, Chapter 6, concludes the research with a summary of findings, limitations and future work directions.

Chapter 2

Review of Literature

The mobile security field contains disjoint security concerns and best practices, which originate from many geographic sources, many perspectives, and do not conform to a given standardized language. One way to unify disjoint topics is through building ontologies and knowledge graphs. Ontologies, or semantic models, have been successful in helping people to assemble their knowledge to understand “their world by forming an abstract description that hides certain details while illuminating others [1, p. 15]” Liyang Yu [2] defines an ontology as, “An ontology formally defines a common set of terms that are used to describe and represent a domain ... An ontology defines the terms used to describe and represent an area of knowledge. [3, p. 151]” Three traditional ways, according to Allemang and Hendler [1, p. 15] that models assist with “understanding is through: (1) communication, (2) explain and make predictions, and (3) mediate multiple viewpoints. Ontologies traditionally are based on hierarchical relationships (e.g. “is a”, taxonomy). Knowledge Graphs, however, support other relationships beyond simply hierarchical (e.g., *partOf*, *comprisedOf*, etc.). Figure 1 below shows an ontology reported by Kafali et al. [4,

p. 532]. As can be seen in this particular ontology, there are two relations, is-a and has-a, and twelve nodes in the ontology.

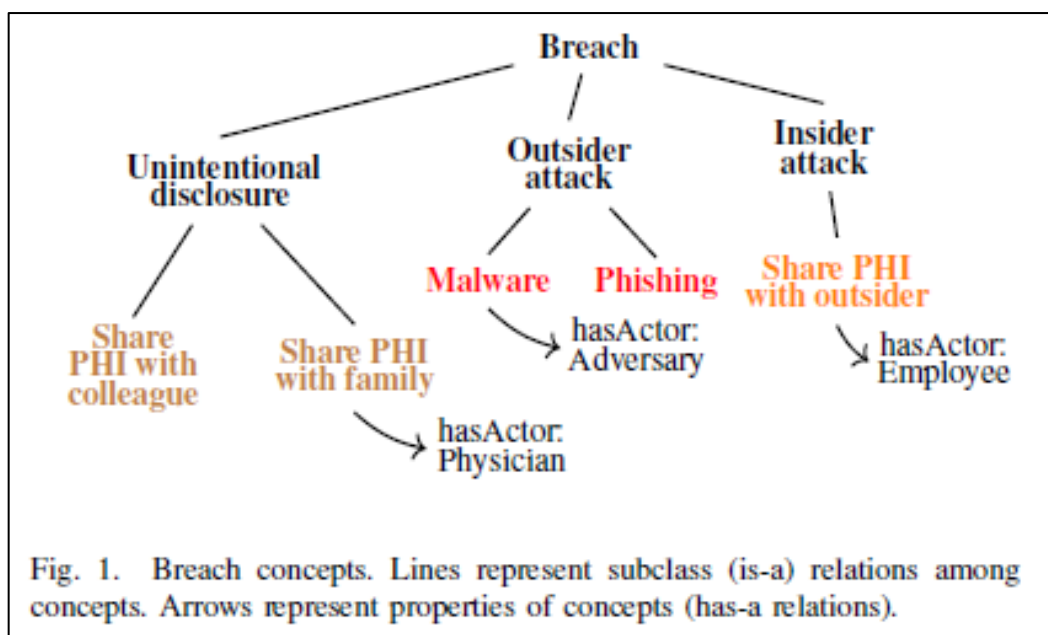


Figure 1: HIPAA Breach Ontology from Kafali et al. [4, p. 532]

2.1 Cybersecurity Ontologies

Developing ontologies for cybersecurity has been done for specific issues such as incident response and analyzing data breaches. Elçi [5] asked the security community “Isn’t the time ripe for a standard ontology on security of information networks?” Elçi reviewed the field certifications and literature and found a startling lack of cybersecurity standardizations. This lack of standardizations—and arguably the entire understanding of cybersecurity—leaves the field without a unified knowledge of how to secure assets. One could argue that cybersecurity, without a standardization of knowledge, remains an art rather than a science.

2.1.1 Ontologies for HIPAA/HITECH Data Breaches

Kafali et al. [4] reviewed known data breaches at entities covered under the Health Insurance Portability and Accountability Act (HIPAA). Their research goal was to help measure the gaps between security policies and reported breaches. They developed a systematic process based on semantic reasoning and proposed a framework, known as SEMAVER, for determining coverage of breaches by policies via comparison of individual policy clauses and breach descriptions. They developed the ontology shown in Figure 1 as one of their research contribution; and, they developed the ontology show Figure 2 as a second research contribution. After developing the ontologies, they worked with the SEMAVER framework to create breach similarity and policy clause coverage scores from data reported to the United States Department for Health and Human Services Office of Civil Rights (US HHS OCR), who maintains a breach portal of open cases.

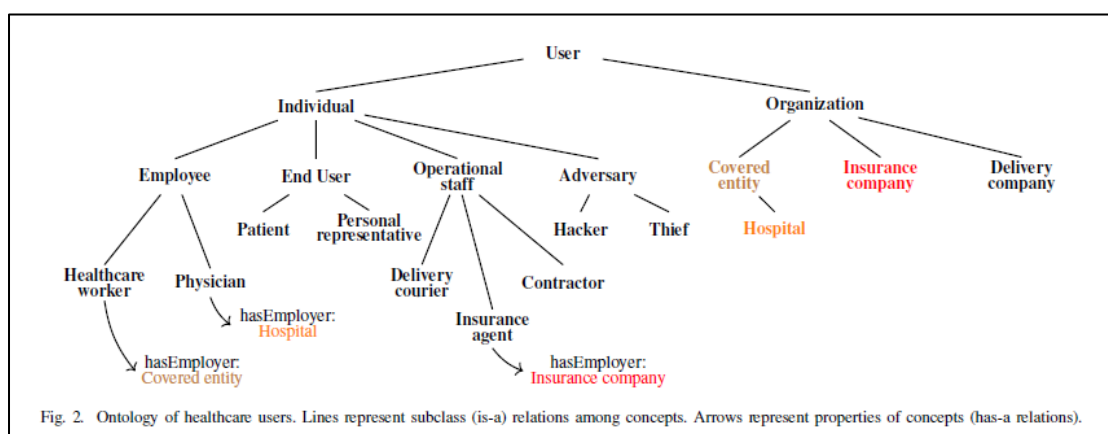


Figure 2: Ontology of healthcare users Kafali et al. [4, p. 532]

2.1.2 Ontologies for Incident Response

Clive Blackwell [6] introduced a security ontology for incident analysis. Clive's ontology extended the work of Howard and Longstaff [7]. Howard and Longstaff followed the

model that “the attacker uses a tool to perform an action that exploits a vulnerability on a target causing an unauthorized result that meets its objectives.” Thus, Blackwell reports the following categories from Howard and Longstaff: attacker, tool, vulnerability, action, target, unauthorized result and objectives. Blackwell, then, extends the taxonomy to include: (1) social and physical aspects of systems using an architectural model, (2) accidental incidents (therefore changing term to perpetrator from attacker), and (3) defensive aspects.

Static analysis is a method to analyze programs without executing them (e.g., Aho et al. [8]). It captures a different view of the code security than other analyses (e.g. embedded credentials, leakage of data, input validation, leakage of resources, etc.). Static analysis can be performed on either the source code or the object code. The more issues discovered before software is released, the more secure the users. When code has already been released, static analysis discovers other issues which cannot be easily addressed using other analyses (e.g. dynamic, emulation, simulation, etc.). Static analysis can be done with a variety of techniques (and tools) such as model checking, data-flow analysis, abstract interpretation, among others. It can be designed to be automated and may require limited resources. This paper examines the methods currently used in security research to characterize what methodologies have been used over the past five years of research in this domain to inform Android application developers, researchers and security analysts.

In order for analysis to be complete with respect to Android application security, all threat vectors and attack surfaces must be analyzed. The National Institute of Standards

and Technology (NIST) National Vulnerability Database (NVD) [9] which is used to inform technology users of vulnerabilities within their devices. A simple search in the NVD on the term Android uncovers hundreds of matching entries indicating that people are finding vulnerabilities in off-the-shelf Android devices. The breadth of devices, firmware, vendors and Android operating system versions used over diverse geographic locations makes detecting exact malware quantification and exploitation statistics difficult.

In order to map the threat vectors and attack surfaces to the current state-of-the-art analysis techniques, we introduce the use of the Bug Framework (BF), Common Weakness Enumeration (CWE), and the Common Attack Pattern Enumeration and Classification (CAPEC) to aid in the discovery of analysis completeness. The intention is to provide a wider overall analysis coverage and understanding. For example, if we only consider Android permissions, we will miss other large threat vectors such as developers failing to remove the storage of cryptographic keys in plain text from applications before they are published and used at large. From a security perspective, researchers need to ensure that their research not only contributes to better techniques in well-known security domains, but they also need to ensure that their research is broadly considering many different categories of security issues.

2.2 Vulnerability and Security Discourse

Industry relies on three pivotal vulnerability discourse frameworks developed independently by MITRE and the National Institute of Standards and Technology (NIST). These frameworks attempt to unify industry into standardizing vulnerability language;

however, industry and research communities can still disagree with vulnerability specifics such as actual ‘fault’ origins and problems (e.g. libraries can disagree on exact interpretation of requirements). Many industry tools rely on these frameworks to convey vulnerability details to developers, for example during static analyses. These frameworks are either aimed at describing malware techniques or they are aimed at describing developer techniques.

2.2.1 NIST’s Bugs Framework (BF)

The United States National Institute of Standards and Technology’s (NIST) Software Assurance Metrics And Tool Evaluation (SAMATE) group introduced the Bugs Framework (BF) a few years ago in Bojanova et al. [10] and Bojanova et al. [11]. NIST SAMATE recently added three new classes of bugs to the BF: encryption bugs (ENC), verification bugs (VRF), and key management bugs (KMN). The BF currently breaks down bugs into four main elements: causes, attribute, consequences and sites of bugs. By distinguishing bugs using this methodology provides more insight into how bugs occur and what effect result from the bug. Currently, the BF employs a knowledge-graph; however, the BF is not entirely mobile application security focused. The BF is referred to as a Periodic Table for bugs. It tries to capture the essence of bugs rather than classify similar patterns separately..

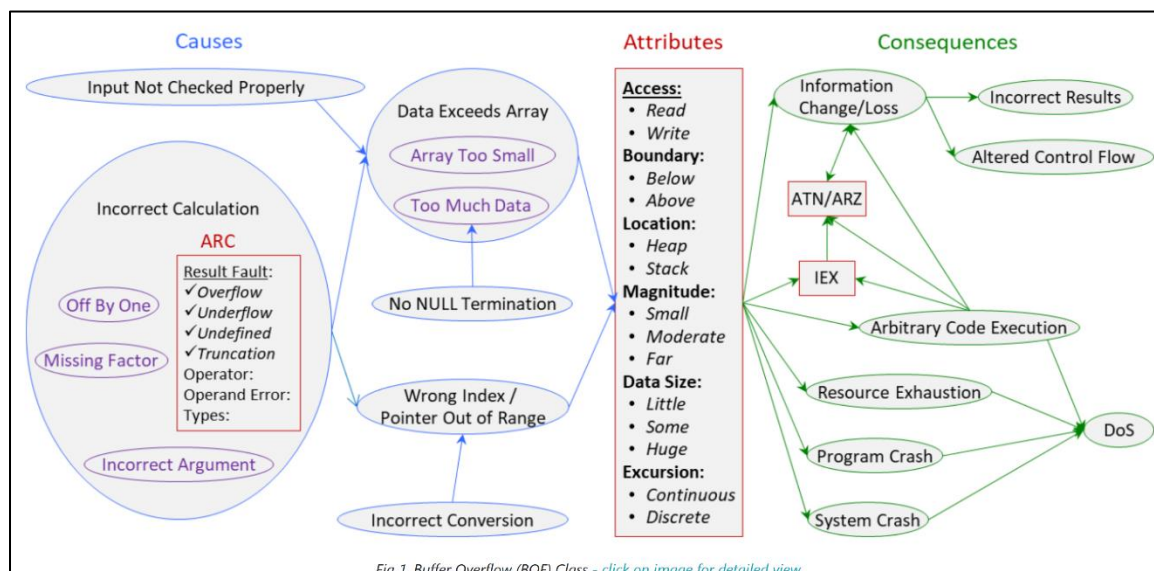


Figure 3: NIST BF - Buffer Overflow (BOF) Class [11]

2.2.2 MITRE's Common Weakness Enumeration (CWE)

MITRE developed the CWE [12] to enable standardized discourse in this domain. CWE's list includes "flaws, faults, bugs, vulnerabilities, and other errors in software code, design, architecture, or implementation that if left untreated could result in systems and networks being vulnerable to attacks. [12]" CWE is a broad list that captures vulnerable software paradigms across languages and operating systems for software assurance efforts. The MITRE CWE taxonomy is useful for assessing which vulnerabilities have been considered in analysis research enabling more security analyses.

The CWE is sponsored by the United States Computer Emergency Readiness Team (US-CERT) in the office of Cybersecurity and Communications at the U.S. Department of Homeland Security. It is a large online repository of software weaknesses known to aid the code security assessment industry. Specifically, it is used for scoring in the U.S. NIST National Vulnerability Database [9].

2.2.3 *Malicious Application Detection (MITRE's CAPEC)*

Static analysis of live code can be beneficial to locate malware. Released code, or “in-the-wild” code, can have built-in attacks embedded into the design. In such a case, the software does not exactly contain weaknesses: it contains attack patterns. MITRE [13] developed the Common Attack Pattern Enumeration and Classification (CAPEC) taxonomy to enable standardized discourse in this domain. The MITRE CAPEC taxonomy is useful to assess which attacks have been considered in detection analysis and mitigation analysis research.

The CAPEC is sponsored by US-CERT in the office of Cybersecurity and Communications at the U.S. Department of Homeland Security. One of the artifacts that CAPEC provides is a list of standard attack patterns. CAPEC defines an attack pattern as “an abstraction mechanism for helping describe how an attack against vulnerable systems or networks is executed.” CAPEC defines each pattern with a description and a mitigation recommendation. CAPEC’s motivation is that developing “attack patterns help[s] categorize attacks in a meaningful way in an effort to provide a coherent way of teaching designers and developers how their systems may be attacked and how they can effectively defend them.”

2.3 Static Analysis of Mobile Applications

Static analysis research on mobile applications is adhoc and quite non-systematic. Four major domains of research exist from a utility perspective, as described by Schmeelk [14], Schmeelk [15], Schmeelk, Yang and Aho [16], and Schmeelk and Aho [17]. In

general, research exists for detecting vulnerabilities during development, detecting malware at large, detecting specific application behavior in a sandbox, and sanitizing/re-packing applications. Furthermore, Schmeelk [14], Schmeelk [15], Schmeelk, Yang and Aho [16], and Schmeelk and Aho [17], showed that the mobile static analysis research can be further categorized into five main security domains: confidentiality (C), integrity (I), availability (A), generalizable (e.g. many different items occurring at once), and other polyhedral (i.e. security concerns related to the CIA triad but at a higher-level such as leaving secret keys hard coded in an application, improper certificate validation, etc.). This section describes the five aforementioned papers. Lastly, we connect the current static analysis research to MITRE's CAPEC, MITRE's CWE, and NIST's BF to understand where the research community is looking for bugs, as described in Schmeelk [14] and Schmeelk [15].

2.3.1 Confidentiality Techniques

Confidentiality is the ability to hide information from those people unauthorized to view it. Bishop [18] defines it as the “concealment of information or resources”; and Whitman and Mattord [19] define confidentiality as “the quality or state of information that prevents disclosure or exposure to unauthorized individuals or systems.” The goal of confidentiality, according to Daswani et al. [20] is to “keep the contents of a transient communication or data on temporary or persistent storage secret.” Confidentiality gives rise to questions about privacy and the usage of collected information. Privacy, as defined by Whitman and Mattord [19], is the “state of being free from unauthorized observation” and reflects the user's ability to exercise control over how their collected personal

information is used by third-parties. Bellovin [21] [22] and Androulaki et al. [23] are well known for their research in computing privacy.

Current confidentiality research encompasses applications' leaking information unknowingly to the user. Schmeelk [15] research domain studies applications that both deliberately leak data as well as naively potentially leak data. If an application is malicious, it will purposefully leak private data from the mobile device without receiving adequate approval from the user. If an application does not require permissions to access its exported activities, exported content providers and exported services, it may unknowingly support a malicious application's request to collect and/or leak private data. In addition, Android allows multiple applications, if they are signed by the same certificate, to share the same Linux User ID on an Android device allowing the applications to share information and resources. In such a situation, a combination of applications can be used to collect and/or leak private data unknowingly from the user.

Many static analysis research techniques exist for confidentiality malware detection concerns. This category of attacks falls under the CAPEC category 118, "Gather Information." Schmeelk [15] found that some earlier tools were replaced with newer research using either new techniques or more precise techniques for malware analysis.

Schmeelk [15] also examined research for confidentiality weakness detection. Weakness detection can be used at compile time to identify vulnerabilities before the application is released; or, it can be used by researchers to discover vulnerabilities in released applications. Schmeelk [15] found six research papers in this category falling

mainly in the CWE category 200 “Information Exposure.” Schmeelk [15] analyzed the papers by date of publication starting with the most recent publication first.

2.3.2 *Integrity Techniques*

Integrity is the ability to ensure that processed data is an accurate and an unchanged representation of the original secure information. Bishop [24] defines integrity as the “trustworthiness of data or resources, and it is usually phrased in terms of preventing improper or unauthorized change”; and Whitman and Mattord [19] define integrity as “the quality or state of being whole, complete, and uncorrupted.” The goal of integrity, according to Daswani et al. [20], is to “not [let] a third party ... be able to modify the contents.” In general, static analysis in this domain are either for malware abuse detection or application weakness detection.

Static analysis with respect to malware detection contains category of attacks which fall under the CAPEC category 233 “Privilege Escalation”. In privilege escalation, adversaries exploit a weakness enabling them to elevate their privilege and perform an action that they are not supposed to be authorized to perform. Some of the techniques used in the earlier confidentiality category extend to integrity.

Static analysis research techniques exist for integrity concerns that can be done by a developer during application development (i.e., pre-deployment) or after development to discover if an application is susceptible to attack (e.g., security analysis). These techniques, not aimed at finding malware, are usually geared at “hardening” applications to lower the

susceptibility of their being compromised by an adversary. This category of attacks falls under the CWE category 441, "Unintended Proxy, Intermediary or Confused Deputy."

2.3.3 *Availability Techniques*

Availability is the assurance that information systems are readily accessible to the authorized viewer at all times. Bishop [24] defines availability as the "ability to use the information or resource desired"; and Whitman and Mattord [19] define availability as, "the quality or state of information characterized by being accessible and correctly formatted for use without interference or obstruction." The goal of availability, according to Daswani et al. [20] is to "respond to users' requests in a reasonable timeframe."

Schmeelk [15] was unable to find any entire papers research focusing specifically on Android application static analysis for availability. Schmeelk reported that a few papers mentioned either a particular case where their technique found a particular availability concern or stated they have done an analysis closely related to availability but without a discussion of security. Availability concerns have not been the focus of any one paper to date other than Schmeelk and Aho [17], which focus was at runtime but offered preliminary insights into availability concerns from risky permissions.

2.3.4 *Generalizable Techniques*

Generalizable tools examine an Android application for more than one weakness or malware patterns. Tools that typically fall into this category include sandboxes and well-developed weakness analyses. Current general analysis research includes proprietary static-analysis tools such as Coverity [25], Fortify [26] and Klocwork [27]. Coverity,

Klocwork, and Fortify all include CWE references in reports generated from code analysis. In all cases, the tools have also found faults that are not, yet, labeled by the CWE. Research in this domain is typically either geared for malware detection or weakness detection.

Static analysis for weakness detection covers different aspects of security: malware frequently uses the same APIs, instrumenting applications can aid in the security analysis process to help log data before it is encrypted, machine learning can be used by different algorithms to identify malware on large-scale analysis, among others. Schmeelk [14] and Schmeelk [15] analyzed the papers by date of publication.

2.3.5 *Other Polyhedral Techniques*

According to Daswani et al. [20] there are other security goals than simply the CIA-triad. Authentication, authorization and accountability, among others, fall into this category. After examining the research, typically research fall into two categories: abuse detection and abuse weakness detection. For abuse detection, the category of attacks falls under other CAPEC categories (e.g., CAPEC-225 “Exploitation of Authentication”, CAPEC-232, “Exploitation of Authorization”, CAPEC-262, “Manipulate Resources”, CAPEC-527 “Manipulate System Users”, etc.). Schmeelk [15] found only one tool geared specifically for this category. Many static analysis research techniques exist for detecting weaknesses in other polyhedral security concerns. Schmeelk [15] found four research papers in this domain: CredMiner, PlayDrone, Static Analysis for Extracting Permission Checks of a Large-Scale Framework, and PScout. This category of attacks falls under other CWE categories (e.g., CWE-522 “Insufficiently Protected Credentials”, CWE-272 “Least

Privilege Violation”, etc.). Schmeelk [15] found three tools geared specifically for this category (CredMiner, PlayDrone and PScout) and one paper. In one case, Schmeelk [15] found that the open source tool is great for a static analysis reference; however, the crawler no longer works with Google Play since Google has changed their API.

2.3.6 *Graphs of Domain Coverage Findings*

Understanding the security domains covered by existing static analysis techniques and tools helps identify limitations and openings in attack vector research. Consequently, Schmeelk [14] categorized each security paper using the analysis in the research as determined by the definitions of MITRE’s vulnerability taxonomy and attack taxonomy. MITRE’s taxonomies were created to create a common dialog and number scheme for all known security issues—both weaknesses and actual attacks. Schmeelk [15] extended the work to categorize the research via the NIST’s BF described in the following subsections.

2.3.6.1 MITRE’s CAPEC

The CAPEC view on “Mechanisms of Attack” [13] lists sixteen categories shown in Table 1. These categories are malware attack surfaces. Examining and expanding on the categories of attacks given by CAPEC help developers, researchers and security analysts secure all attack vectors associated with Android applications. Securing some attack surfaces leaving other attack surfaces completely unanalyzed and potentially unsecured is completely unacceptable from a security perspective.

Table 1: CAPEC “Mechanisms of Attack”

Mechanisms of Attack Gather Information (118) Deplete Resources (119)

Injection (152)
Deceptive Interactions (156)
Manipulate Timing and State (172)
Abuse of Functionality (210)
Probabilistic Techniques (223)
Exploitation of Authentication (225)
Exploitation of Authorization (232)
Manipulate Data Structures (255)
Manipulate Resources (262)
Analyze Target (281)
Gain Physical Access (436)
Malicious Code Execution (525)
Alter System Components (526)
Manipulate System Users (527)

Each of these high-level categories has additional sub-categories. Security research in the static analysis domain would benefit with mapping their research to specific malware discoveries as shown in Figure 4. Without a specific mapping of events knowledge of the usefulness of a technique is limited to both ease of use for security analysts and importance. In addition, the lack of both universally maintained benchmarks and open source projects reduces the analysis comparison metrics.

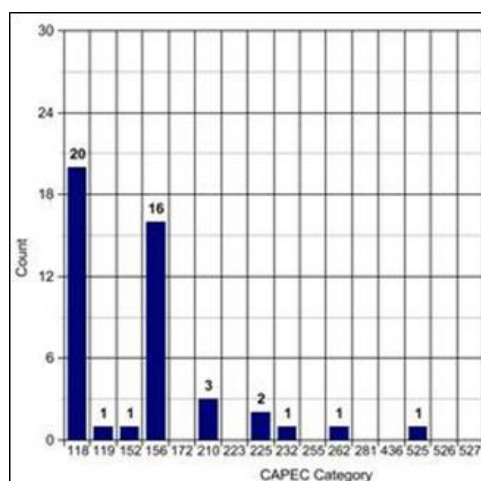


Figure 4: Fraction of CAPEC Categories Researched [14]

Schmeelk [14] found that the majority of malware research fit within two of the above categories with outlying research fitting into eight other categories seen Figure 4 from Schmeelk [14]. Interestingly, most of the malware research is examining CAPEC-118 and CAPEC-156. A few research studies have considered the following: CAPEC-225, CAPEC-210, CAPEC-262, CAPEC-232, CAPEC-119, and CAPEC-526. Examining the research analysis techniques shows that there is limited research of the following static analysis for attack surfaces: CAPEC-172, CAPEC-223, CAPEC-255, CAPEC-281, CAPEC-436, and CAPEC-527. These statistics help inform the security research community to shape the research into attack surface breadth and depth. The community should carefully consider the MITRE attack surface taxonomy when designing analysis tools and techniques to ensure that Android applications have coverage over more attack surfaces.

2.3.6.2 MITRE's Common Weakness Enumeration (CWE)

There are over 1,000 CWE identifiers in existence, according to a MITRE representative. Schmeelk [14] reported that the majority of vulnerability research of the publications we analyzed, fell within eleven of the 1,000+ weakness categories. The eleven weaknesses identified can be seen in Schmeelk [14].

2.3.6.3 NIST's Bugs Framework (BF).

Research has shown that in the Bugs Framework, six current classes capture the static analysis research, as seen in **Error! Reference source not found.** [15]. The static analysis research raises some questions about five findings that may be of the BF four main elements of a bug: causes, attribute, consequences and sites of bugs. The six current classes

captured in the research are: IEX (matching CWE-200: Information Exposure), CRY (matching CWE-310 CATEGORY: Cryptographic Issues), AUT (matching CWE-441: Unintended Proxy or Intermediary), PTR (matching CWE-476: NULL Pointer Dereference), WOP (matching CWE-597: Use of Wrong Operator in String Comparison), and ARG (matching CWE-628: Function Call with Incorrectly Specified Argument). The non-matching CWE comparisons may not directly map into NIST BF as individual classes; they may be another of four main elements. These are: CWE-798: Use of Hard-coded Credentials, CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop'), CWE-500: Public Static Field Not Marked Final, CWE-561: Dead Code, and CWE-272: Least Privilege Violation.

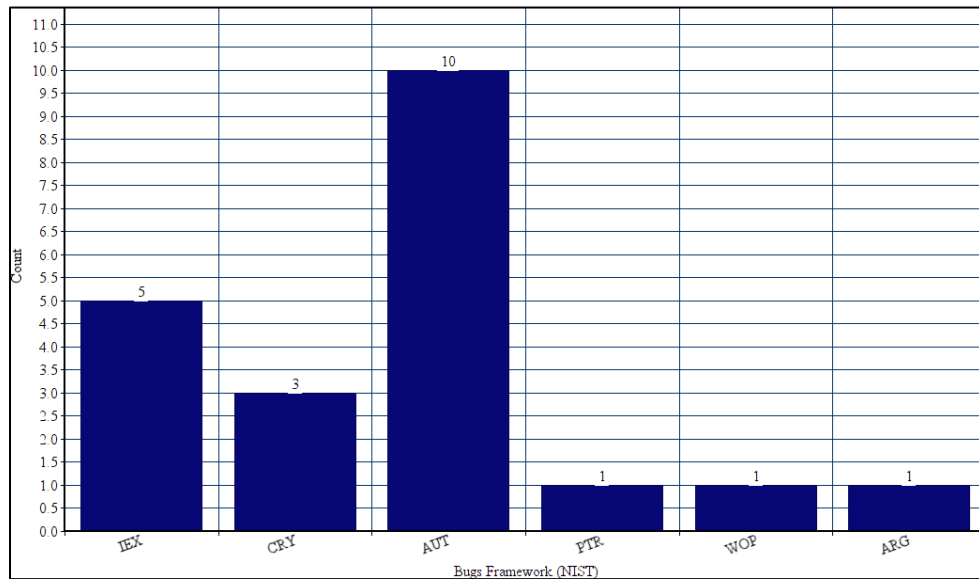


Figure 5: The fraction of NIST BF categories researched [15]

2.4 Ontologies and Knowledge Graphs

Ontologies and knowledge-graphs are essential model representations for communicating system information resources to improve system understandings, usability and durability. According to Allemang and Hendler [1], model representations “help people communicate (p. 15),” “explain and make predictions (p.15),” and “mediate among multiple viewpoints (p.15).” Models can be used “to help us through the mess on the web (p. 16).”

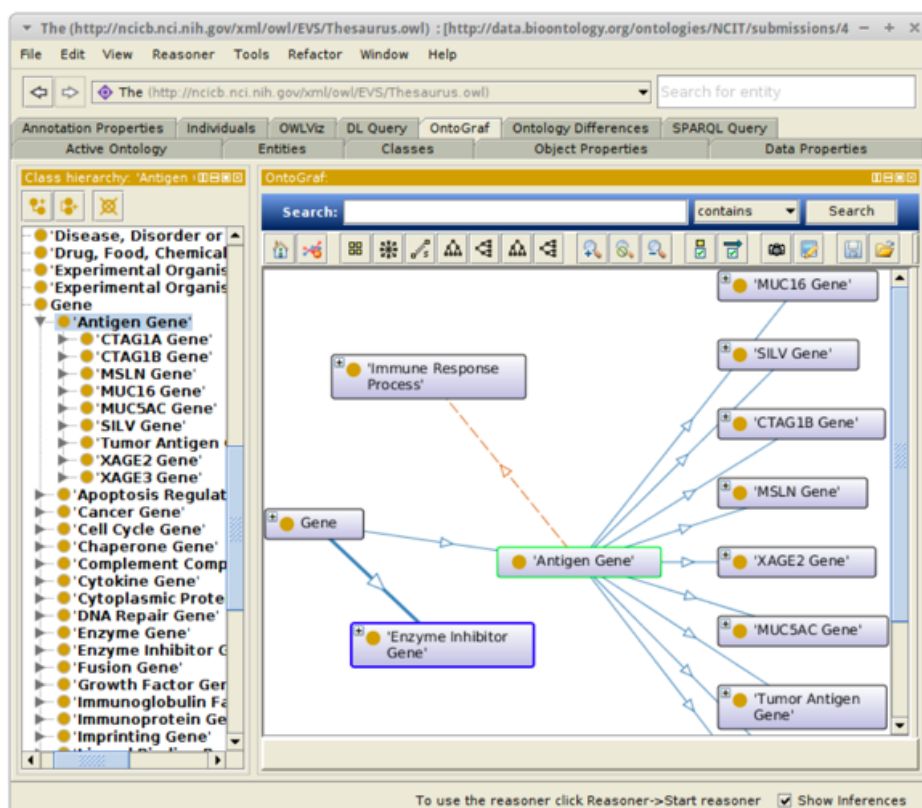


Figure 6: Protégé Interface [28]

2.4.1 Ontologies for Software Development

Ontologies are defined, according to Arda Goknil and Yasemin Topaloglu [29], to be “a formal explicit description of concepts in a domain of discourse, properties of each concept describing various features and attributes of the concept, and restrictions of slots [30].” Lacy [31] state that “ontologies serve a similar function to a database schemas by providing machine-processable semantics of information sources through collections of terms and their relationships. Ontologies can be useful for software development to make clear how systems should operate at a fundamental level [29]. Liyang Yu [2] discusses how ontologies are essential for software development to understand how data should flow and how the software should interact. Yu describes the use of ontologies from the perspective of software development through immense experience programming the transportation systems of Delta Airlines.

There are different ways to describe relationships between entities using ontologies. One methodology is to specify the ontologies through a specific language such as Resource Description Framework Schema (RDFS) or Web Ontology Language (OWL). According to Yu [3] a RDFS “is a language one can use to create a vocabulary (often the created vocabulary is domain-specific), so when distributed RDF documents are created in this domain, terms from this vocabulary can be used. Therefore, everything we say, we have a reason to say it.” A different semantic and syntactical representation is OWL. According to Goknil and Topaloglu [29], “OWL defines and instantiates Web Ontologies.” Other methodologies are to graphically represent the ontologies. In fact, since both RDFS and OWL follow a standardized textual specification, the textual representation can be

transformed into a graphical representations. One industry leading tool for such transformations is Protégé, a tool developed at Stanford University [28]. Protégé is shown in the Figure 6.

2.4.2 Knowledge Graphs for Software Development

Knowledge graphs communicate information with a different representation than an ontology. An ontology formally describes the types, properties and interrelationships between entities. A knowledge graph is a collection of entities where the types and properties have values declared for them, and where the relationships between them are connected. In a knowledge graph, the nodes are the types and properties and the edges are the relationships between nodes.

K. Patel, I. Dube, L. Tao and N. Jiang [32] recently published work in this domain proposing minimal syntax extension to OWL for declaring custom relations with special attributes, and applying them in knowledge representation. Their work they present additions to the OWL API for the declaration, application, and visualization of custom relations. Their research paper outlines revisions and additions to the ontology editor Protégé so its users could visually declare, apply and remove custom relations according to their enriched OWL syntax. Their work describes the modification to the OWLViz plugin for custom relations visualization also known as a knowledge graph.

Chapter 3

Mobile Application Client-Side Structure

In this chapter we introduce the Android operating system with respect to its mobile application structure. This chapter includes the Android Linux sandbox structure, accessing mobile resources from within an application and discusses additional client-side mobile application functionality such as including external packages and native code. A brief discussion on mobile client-server architecture will be discussed to inform on client-side research scope.

Google's Android operating system is a Linux-based variant. The operating system acts as a middleware between phone the hardware-firmware and third-party mobile applications. Mobile applications, in this model, are run on the operating system within a sandboxed environment, the Android Runtime Environment (ART) and predecessor the Dalvik Virtual Machine (DVM), where each mobile application is given its own unique Linux system-level *UID* [33]. Figure 7 shows a comparison of ART and DVM [34].

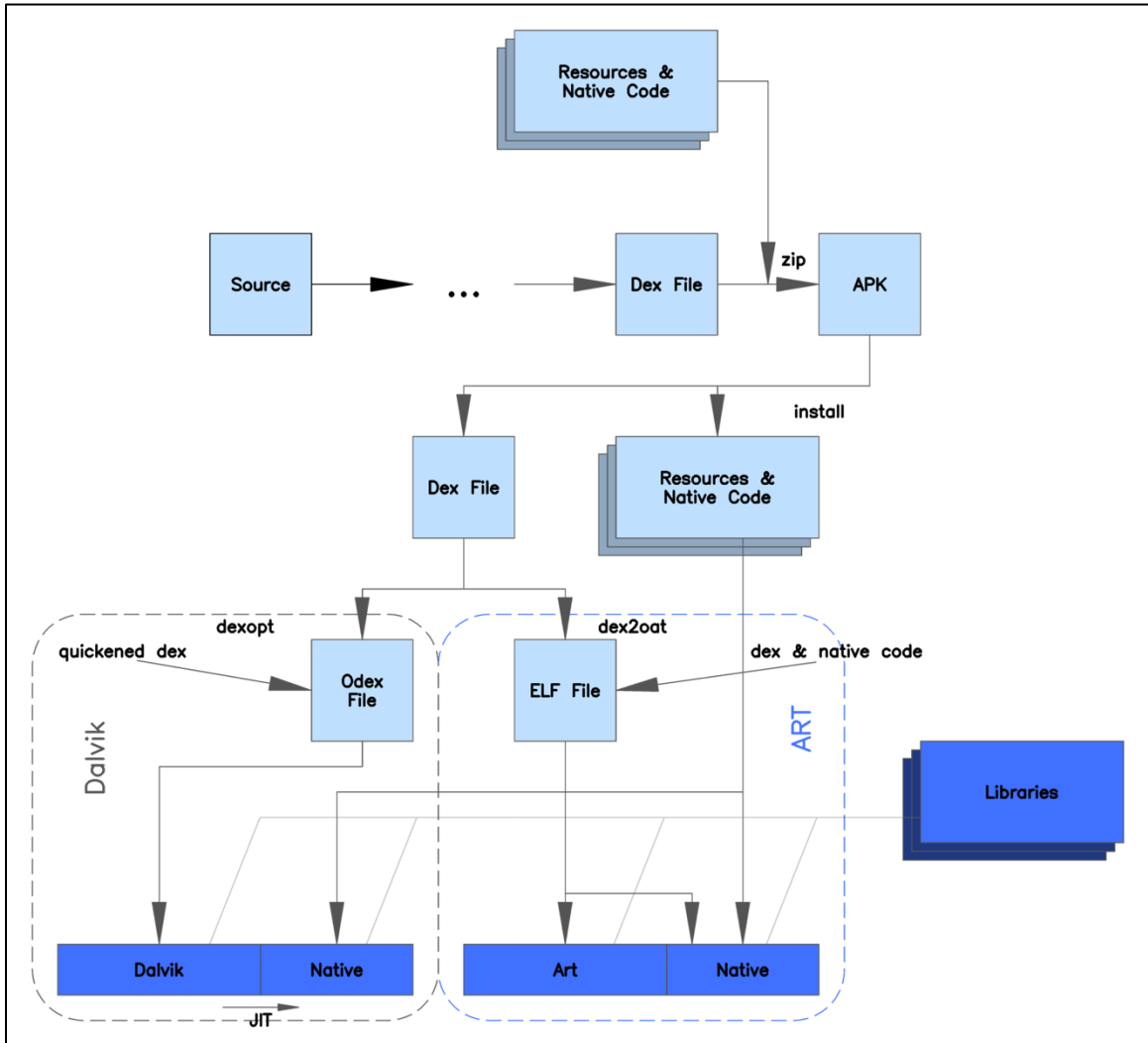


Figure 7: ART vs DVM [34]

The ART is slightly different than Oracle's Java Virtual Machine (JVM) [35]. First, the ART supports 218 opcode instructions [36], [37], [38] versus JVM's 256 instructions [39]. Second, ART uses a register-based architecture as opposed to Oracle's stack-based architecture [36]. Third, ART's SDK library is quite different than Oracle's SDK library. Fourth, the ART interacts with the underlying phone system through a mediated IPC system Binder and the user via call-backs. Fifth, and Android Application package (APK)

is structured differently than a traditional Java archive file (JAR). In the following subsections, we go into addition Android application design details.

3.1 Android Linux Sandbox Structure

The Android Operating System, seen in Figure 8 [40], is designed with a five-layer software model.

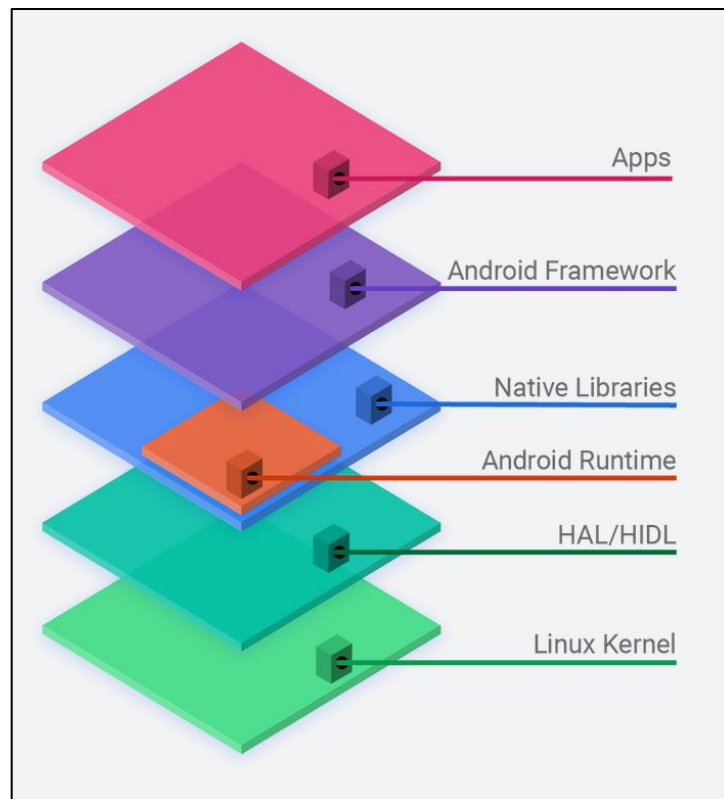


Figure 8: Android Operating System [40]

Layer-1, the Linux Kernel, is where device driver software is integrated. Android employs the Linux kernel with a few special additions such as a memory management system that is more aggressive in preserving memory, a *PowerManager* system service, the Binder IPC driver, and other features important for a mobile embedded platform.

Layer-2, the Hardware Abstraction Layer (HAL), which employs the HAL Interface Definition Language (HIDL), is standard interface for hardware vendors to implement. The HAL specification enables Android to be agnostic about lower-level driver implementations and does not affect higher level system layers.

Layer-3, the Android Runtime and the Native Android Libraries, can be further visualized in [41] also known as Android services (i.e. system services and media services). System services are modular, focused components, which are exposed by the application framework APIs to access the underlying Android functionality (e.g. Window Manager, Search Service, or Notification Manager). Media services include playing and recording media.

Between Layer-3 and Layer-4 are the Binder Inter Process Communications (IPC). The Binder IPC libraries allow the application framework to cross process boundaries and call into the Android system services code, which enables Layer-5 to interact with Android system services. Layer-4, the Figure 9 Android Framework, is the developer APIs, which can map directly to the underlying HAL interfaces.

Layer-5 is the Android Applications themselves.

At the application level, Layer-5, the Android platform follows the Linux user-based protection to identify and isolate each Application resource. The process sandbox thus isolates Applications from each other and somewhat protects Application and the system from malicious Applications.

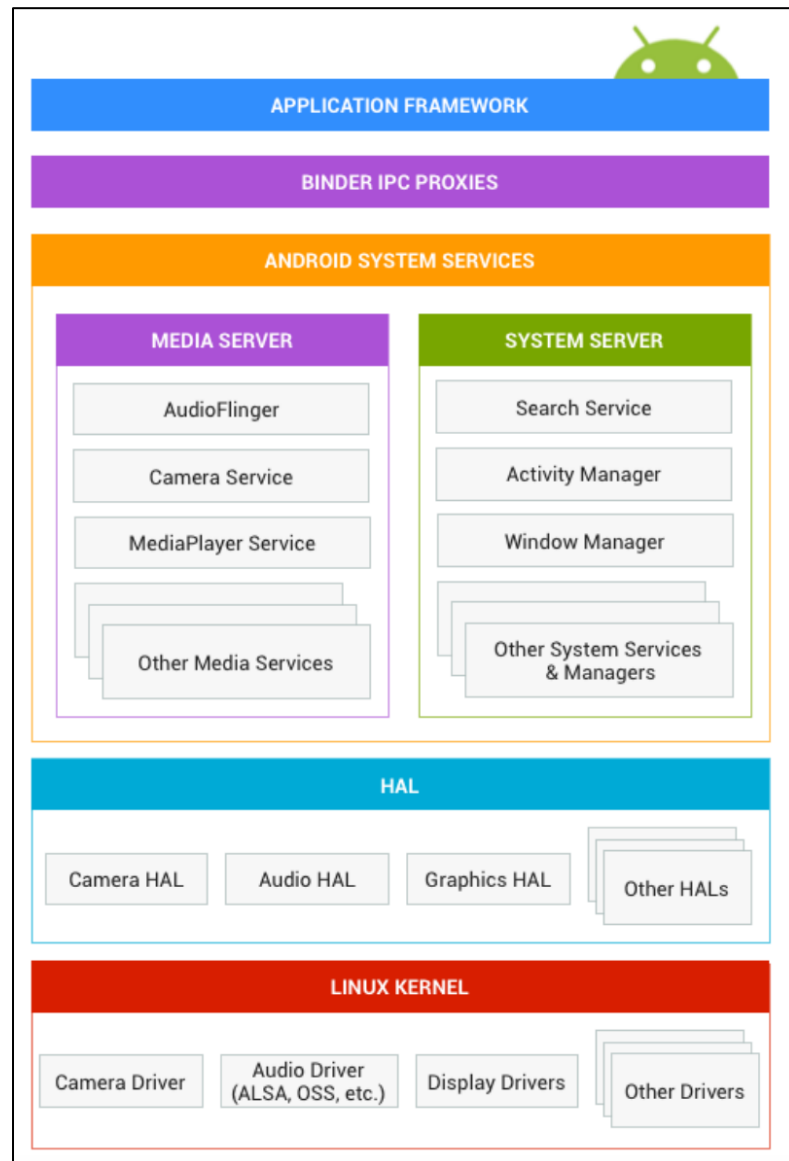


Figure 9: Android Mobile Device Software Stack [41]

To create a sandbox, Android assigns a unique user identifier (UID) to each Android application and runs it in its own process [42]. Android uses the UID to set up a kernel-level Application Sandbox. The kernel enforces security between the mobile applications and the system at the process level through standard Linux facilities such as user and group IDs that are assigned to apps. By default, mobile applications cannot interact with each

other and have limited access to the OS. If a mobile application tries to do something malicious, such as read application B's data or dial the phone without permission, it's prevented from doing so because it doesn't have the appropriate default user privileges. The sandbox is simple, auditable, and based on decades-old UNIX-style user separation of processes and file permissions.

Because the Application Sandbox is in the kernel, this security model extends to both native code and OS applications. All of the software above the kernel, such as OS libraries, application framework, application runtime, and all applications, run within the Application Sandbox. On some platforms, developers are constrained to a specific development framework, set of APIs, or language. On Android, there are no restrictions on how an application can be written that are required to enforce security; in this respect, native code is as sandboxed as interpreted code.

Generally, to break out of the Application Sandbox in a properly configured device, one must compromise the security of the Linux kernel. However, similar to other security features, individual protections enforcing the application sandbox are not invulnerable, so defense-in-depth is important to prevent single vulnerabilities from leading to compromise of the OS or other apps.

Android relies on a number of protections to enforce the application sandbox. These enforcements have been introduced over time and have significantly strengthened the original UID-based Discretionary Access Control (DAC) sandbox.

Android releases included the following protections:

- In Android 11 (API 30 release expected in 2020), this version is planned release in 2020; however, specific details have not, yet, been announced in length.
- In Android 10 (API 29 released in 2019), mobile applications have a limited raw view of the filesystem, with no direct access to paths like */sdcard/DCIM*. However, mobile applications retain full raw access to their package-specific paths, as returned by any applicable methods, such as *Context.getExternalFilesDir()*.
- In Android 9 (API 28 released in 2018), all non-privileged mobile applications with the target software development kit at least version 28 or higher (*targetSdkVersion >= 28*) must run in individual SELinux sandboxes, providing MAC on a per-mobile application basis. This protection improves mobile application separation, prevents overriding safe defaults, and (most significantly) prevents mobile applications from making their data world accessible.
- In Android 8.0 (API 26 released in 2017), all mobile applications were set to run with a *seccomp-bpf* filter that limited the *syscalls* that mobile applications were allowed to use, thus strengthening the app/kernel boundary.
- In Android 6.0 (API 23 released in 2015), the SELinux sandbox was extended to isolate mobile applications across the per-physical-user boundary. In addition, Android also set safer defaults for application data for mobile applications with the target software development kit at least version 24 or higher (*targetSdkVersion >= 24*), default DAC permissions on an mobile application's home directory changed

from 751 to 700. This provided safer default for private mobile application data (although mobile applications may override these defaults).

- In Android 5.0 (API 21 released in 2014), SELinux provided mandatory access control (MAC) separation between the system and apps. However, all third-party mobile applications ran within the same SELinux context so inter-mobile application isolation was primarily enforced by UID DAC.

3.2 Android Application Structure

The application structures of Google's Application package code, file name extensions, inflation, permissions and user interface screens are unique to code running inside the ART. Figure 10 shows some top-level differences between a packaged Android Application Package file (APK) and a packaged Java Archive file (JAR).

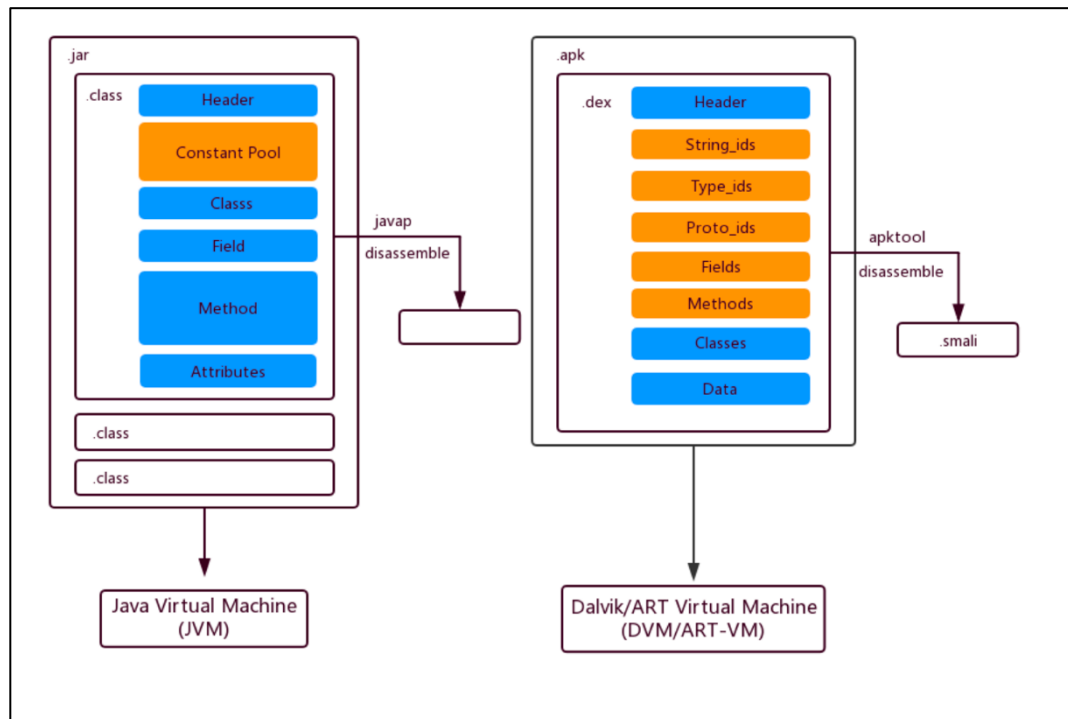


Figure 10: File format comparison - APK vs. Jar [43].

All the building blocks of an Android Application Package can be useful to inform a robust static analysis system. An Android Application is comprised of many meta-elements and its internal code is somewhat complex [44]. The more robust the analysis of the static source/byte code, the more accurate the overall results. An Android Application APK file consists of a manifest file, strings file, main file, R file, resource file, program code and more.

- The *manifest.xml* file, located in the top-level directory, is written in xml and explains what the application consists of, what the main building blocks are, what permissions the Application requires, etc.

- The *strings.xml* file, located in the *res/values* directory, contains all the text that an application uses, including names of buttons, labels, default text and other strings.
- The *main.xml* file, located in the *res/layout/* folder, declares the layout of the mobile user screen.
- The *R.java* file, located in the *gen* folder, transforms Java files with the resource files such as images, video or audio.
- The *resources.ap* file, located at the top-level, is an archive of all the XML resource files encoded in an efficient and easy-to-parse format.
- The Application Java byte code is stored in the *classes* folder and the complete Application Dalvik byte code is stored in the *classes.dex* file.

Accurate static analysis techniques must also consider the life-cycle of an Android Application. Android Applications are unique from regular Java applications in that they support a life-cycle model, as can be seen in Figure 11. Android is unique from other programming paradigms, as it initiates code in an Activity by invoking specific callback methods that correspond to specific stages of its lifecycle rather than invoking a *main()* method. When Activities start up or tear down, there is a sequence of callback methods that take place [45]. In the *Created* state the application system initialization takes place. In the *Started* state the activity becomes visible on the user screen. In the *Paused* state the activity is partially obscured by another activity. “The other activity that is in the foreground is semi-transparent or does not cover the entire screen [45].” In the *Resumed*

state “the activity is in the foreground and the user can interact with it.” A paused activity cannot receive user input nor execute any code. In the *Stopped* state, the activity is completely hidden and not visible to the user; it is considered to be in the background. While stopped, the activity instance and all its state information such as member variables is retained, but it cannot execute any code [45].”

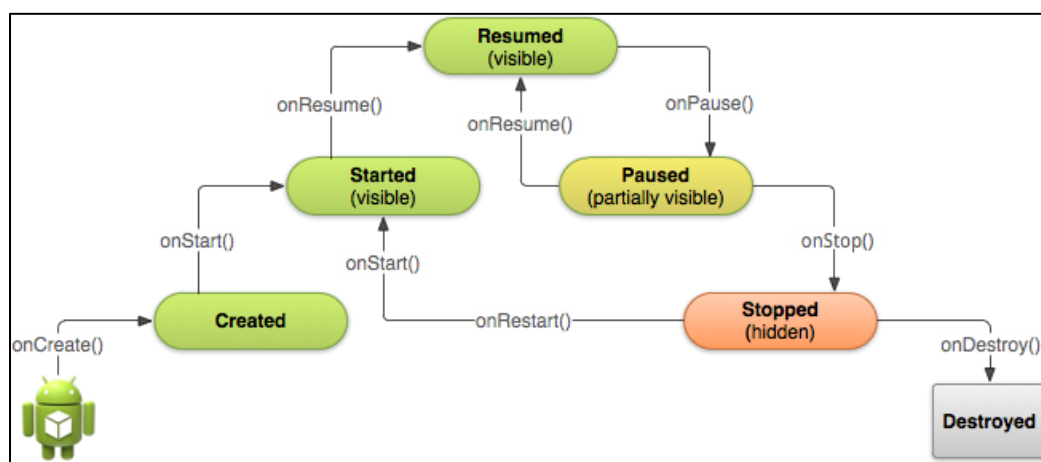


Figure 11: Android Application Basic Lifecycle [45].

Flow in an Android Application is unique and must be correctly modeled for factual internal representation during static analysis. Application components are considered the basic building blocks of an Android mobile application. There are four types of components: *Activities*, *Services*, *Content Providers*, and *Broadcast Receivers*. *Activities* represent a single screen and single user interface. All *Activities* are implemented as subclasses of the SDK library *Activity*. *Services* are long-running operations, typically for remote processes, that are done in the background. All *Services* are implemented as subclasses of the SDK library *Services*. *Content Providers* manage sharing application data such as the *SQLite* database or other persistent memory storage devices on the system. All

Content Providers are required to implement a standard set of API functions for other applications to perform transactions. *Content Providers* are implemented as subclasses of the SDK library *ContentProvider*. *Broadcast Receivers* respond to system-wide broadcast announcements, including battery performance, light sources, picture taken, or screen turned off. *Broadcast Receivers* are implemented as subclasses of the SDK library *BroadcastReceiver*. *Broadcasts* are received as *Intent* objects [45]. All but one component are activated by asynchronous messages called *Intents*. These are *Activities*, *Services*, and *Broadcast Receivers*. *Content Providers* are the exception, as they interface with a *ContentResolver*, which is how they are activated and through which all subsequent transactions transpire. Finally, *Intents* can either be explicit (e.g. specifying component class name) or implicit (e.g. ICC messages, broadcast messages or finding a certain capability via a race) [45].

3.3 Standard Android Application Security Management Coding Patterns

This section discusses standard security management coding patterns which are structured in fixed methodologies by the Android Programming Language. Chapter 4 goes into a detailed discussion of the OWASP top mobile threats with respect to coding patterns.

3.3.1 Android API

Android has many versions. Each new version adds and subtracts features based on Google mobile research. When an application accesses (manages) features related to security through the use of API, then code analysis can detect weak coding patterns.

Algorithm	Modes	Paddings	Supported API Levels	Notes			
AES	CBC CFB CTR CTS ECB OFB	ISO10126Padding NoPadding PKCS5Padding	1+				
	GCM	NoPadding	10+				
AES_128	CBC ECB	NoPadding PKCS5Padding	26+				
	GCM	NoPadding	26+				
AES_256	CBC ECB	NoPadding PKCS5Padding	26+				
	GCM	NoPadding	26+				
ARC4	ECB	NoPadding	10+				
	NONE	NoPadding	28+				
BLOWFISH	CBC CFB CTR CTS ECB OFB	ISO10126Padding NoPadding PKCS5Padding	10+				
	ChaCha20	NONE Poly1305	NoPadding	28+	ChaCha with 20 rounds, 96-bit nonce, and 32-bit counter as described in RFC 7539.		
	DES	CBC CFB CTR CTS ECB OFB	ISO10126Padding NoPadding PKCS5Padding	1+			
		DESede	CBC CFB CTR CTS ECB OFB	ISO10126Padding NoPadding PKCS5Padding	1+		
			RSA	ECB NONE	NoPadding OAEP PKCS1	1+	
					OAEPwithSHA-1andMGF1 OAEPwithSHA-256andMGF1	10+	
				OAEPwithSHA-224andMGF1 OAEPwithSHA-384andMGF1 OAEPwithSHA-512andMGF1	23+		

Figure 12: Android Cipher API [46]

3.3.1.1 Shared Space

Mobile applications can access device shared spaces using the standard mobile Android API. These API calls are discussed in Chapter 4.

3.3.1.2 Inter-Application Communication

Applications can communicate with other applications using the standard mobile Android API. Specifically, communications can occur through: Intents, Android Interface Definition Language (AIDL), and Bound Services. All three of these built in Android features are developed into the Android API.

3.3.1.3 Data-At-Rest: Encryption/Decryption

Applications can apply cryptographic functions using the standard mobile API. Figure 12 shows the cryptographic functions enabled in the Android API.

3.3.1.4 Data-In-Motion: TLS Setup

Applications can specify TLS configurations using the standard mobile API. To enable data-in-motion, Android API handles certificate validation and the instantiation of HTTPS connections, among others.

3.3.1.5 Authentication

Applications can rely on OAuth and other authentication mechanisms (e.g. Biometrics) using the standard mobile API.

3.3.2 Vulnerable Libraries

Applications can include libraries in their code for all system activities. Libraries are more difficult to detect since they may not follow standardized coding methodologies such as the API follows.

3.3.3 Limitations

Applications can include functionality (e.g. dynamic download and execution) which are more challenging to analyze. In addition, obfuscated code and native code are also more difficult (but not impossible) to analyze.

Chapter 4

Ontology of Mobile Application Security Threats

The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software. The non-profit is community-driven with hundreds of local chapters and tens of thousands of members throughout the world. In 2015, OWASP performed a survey from local chapters and members to analyze and re-categorize the OWASP Mobile Top Ten for 2016. The 2016 Mobile Top 10 were more focused on the application rather than the server. OWASP claims the 2016 data collection project goals were to: (1) update the wiki, cross-links to testing guides, and other visualizations (2) collect more data and (3) create a releasable document. OWASP state, “Based on feedback, we have released a Mobile Top Ten 2016 list following a similar approach of collecting data, grouping the data in logical and consistent ways. [47]” To date, the OWASP 2016 remains the most current version hosted in entirety on the OWASP webservers; however, OWASP still links to the 2014 data collection. No newer version is, yet, hosted in entirety on the OWASP portal.

4.1 OWASP 2014 Mobile Threats Coding Patterns

In this section examines top OWASP 2014 mobile threats through ontology and Android source code. This OWASP Top Ten is still one (of two) listed on the main OWASP portal. The community-developed OWASP Mobile Threats of 2014 can be seen in Figure 13.

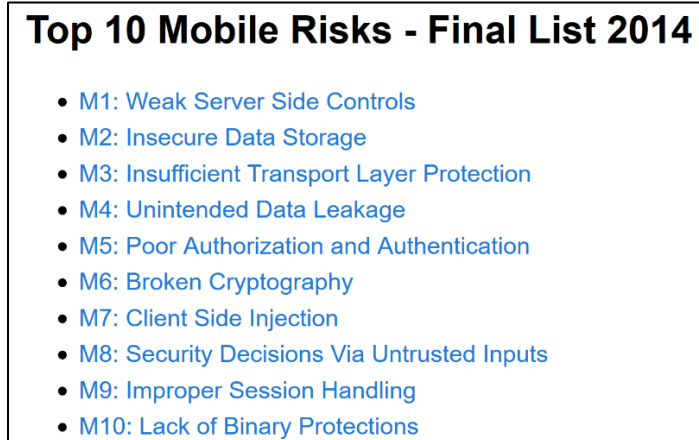


Figure 13: OWASP Mobile Top Ten Threats of 2014

4.1.1 Mobile 2014 Threat 1: Weak Server-Side Controls

The first threat, the OWASP 2014 Mobile Threat 1, is for the category “Weak Server-Side Controls” (M1_Weak_Server_Side_Controls), as seen in Figure 14. Weak Server-Side Controls (WSSC) can potentially lead to the compromise of confidentiality, integrity and availability. Many of the server-side controls cannot be identified within the source code of a client mobile Android application since the exact source code implementation is not present at the client-side. However, the client application may have additional specific code (e.g. cookie flag checking), which does show-up at the client-side source code by default.

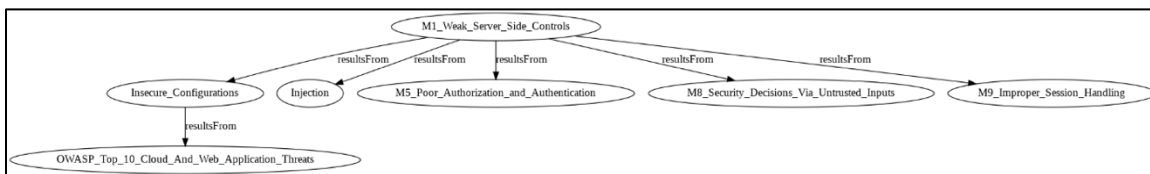


Figure 14: OWASP Mobile 2014 Threat 1: Weak Server-Side Controls

WSSC *resultsFrom* insecure server configuration which changes from different underlying operating systems and web servers. Considering best industry practices,

insecure server configuration resultsFrom the OWASP Top 10 Cloud and Web Application threats, among others.

WSSC *resultsFrom* injection issues along with the OWASP Mobile Threat Security Decision via Untrusted Inputs, explained in Section 4.1.8. Again, many of these vulnerabilities are difficult to detect via the mobile client application source code.

Finally, WSSC *resultsFrom* two other OWASP Mobile Threat categories, M9_Improper_Session_Handling, explained in Section 4.1.9 and, in Section 4.1.5, the M5_Poor_Authorization_and_Authentication.

4.1.2 Mobile 2014 Threat 2: Insecure Data Storage

The OWASP 2014 Mobile Threat 2 is “Insecure Data Storage.” In our knowledge graphs of, seen in Figure 15, we label this threat as *M2_Insecure_Data_Storage*. Insecure data storage (IDS) can potentially lead to data compromise or the propagation of malware [48]. Our knowledge graph is based on industry best practices and industry news reports.

From an application level, storing data insecurely can potentially *resultFrom* storing data in a vulnerable location. The *resultFrom* relation is necessary to describe risk associated with storing data in a vulnerable location. As such, it is not an *isA* relationship, which indicates hierarchy. Risk is traditionally measured using a likelihood and impact model (NIST, 2012). In such a model, the likelihood is the probability of threat being exploited along with the underlying impact, if exploited, to the end users, organization(s) and potential customers.

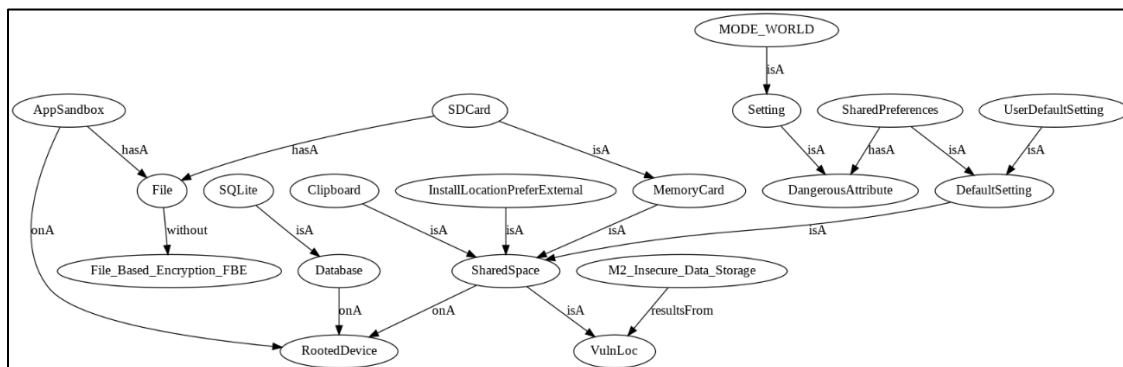


Figure 15: OWASP Mobile 2014 Threat 2: Insecure Data Storage

Android offers at least four locations to store data: internal file storage, external file storage, shared preferences, and databases [49]. Common *VulnerableLocations* on mobile devices are (i.e. *isA*) *Shared Spaces* [50]. All of the android data storage locations are shared spaces under certain conditions. *SharedSpaces* are vulnerable to all CIA concerns since multiple applications have access to information stored in this space by default [50]. Common shared spaces are (i.e. *isA*) *MemoryCards* and are (i.e. *isA*) *DefaultSettings*. On *onA RootedDevice* they are (i.e. *isA*) application *Databases* and (i.e. *isA*) *AppSandbox*. Android has specified that an external storage *MemoryCard isA SDCard* [50].

Table 2 shows the Android Java code for storing a file. Specifically, the method calls to write a file to external file storage can be analyzed by first identifying the external storage home directory using the Android API call *getExternalStoragePublicDirectory*. Additionally, the file can have basic permission set during creation. The developer can calculate the integers directly to set permissions when a file is opened. In addition, the Android API offers fixed standard values stored in the API Context Interface. Four standard API examples are as follows: *MODE_PRIVATE*, *MODE_WORLD_READABLE*, *MODE_APPEND*, and *MODE_WORLD_WRITEABLE*. In either case, static analysis can

detect risky file permissions at different static analysis granularities. As a definition used in the field-at-large, a *complete* static analysis technique guarantees no “*false negatives*,” and, a *sound* static analysis technique guarantees no “*false positives*.”

First, with a source code one-pass text analysis, static analysis can determine if any risky Context API values are used within the application (e.g. locating the external storage home directory (i.e. *getExternalStoragePublicDirectory*), or any public API Context values) [51]. This technique is neither sound nor complete. Second, a complete static analysis technique could be employed by using a context-sensitive control flow analysis examine every potential Android API call to class methods where a file is opened for writing. This technique would identify every risky methodology for writing a file using the Android API. Third, to improve the soundness from the second technique, data flow analysis could be employed to help determine if sensitive data may, in fact, be stored into the file by employing data propagation and taint analysis static analysis techniques.

```

/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}

public File getPublicAlbumStorageDir(String albumName) {

```

```

// Get the directory for the user's public pictures directory.
File file = new File(Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_PICTURES), albumName);
if (!file.mkdirs()) {
    Log.e(LOG_TAG, "Directory not created");
}
return file;
}

//At the file level
String filename = "myfile";
String fileContents = "Hello world!";
FileOutputStream outputStream;

try {
    outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
    outputStream.write(fileContents.getBytes());
    outputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}

```

Table 2: Android Java Code for File Storage [49]

Android specifies at least two types of Default Settings [52]. DefaultSettings are (i.e. *isA*) *User Default Settings* or *Shared Preferences*. *SharedPreferences* are just plain XML files in an application directory on internal storage. Table 2 shows an example of accessing the *Shared Preferences*. Static analysis on the source code can identify the type of *Shared Preferences* employed on the code.

First, with a source code one-pass text analysis, static analysis can determine if any risky context API values are used within the application [51]. This technique is neither sound nor complete. Second, a complete static analysis technique could be employed by using a context-sensitive control flow analysis examine every potential Android API call where *SharedPreferences* methods are accessed for writing [53]. This technique would identify every risky methodology for writing a file using the Android API. Keep in mind that

complete static analysis technique guarantees no “*false negatives*.” Third, to improve the soundness from the second technique, data flow analysis could be employed to help determine if sensitive data may in fact be stored into the file or used by the application for configuration after retrieving information from the *SharedPreferences*.

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
    getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

Table 3: Android Java Code for Shared Preferences [53]

Another *SharedSpace* is a *Database* on a *Rooted* device in the Android API is *SQLite* [54]. This database may have tables, which may not have permissions correctly set to restricted to accessing only the files for which they are privileged [55]. The Android API offers fixed standard values stored in the API context interface [56]. Five standard API permission examples are as follows: *MODE_PRIVATE*, *MODE_WORLD_WRITEABLE*, *MODE_ENABLE_WRITE_AHEAD_LOGGING*, *MODE_WORLD_READABLE* or *MODE_NO_LOCALIZED_COLLATORS*. In addition, on a rooted device the *Database* is entirely exposed to any application on the device.

Table 4 shows Android application code for interacting with the Android *SQLite* database using the Android API [54]. First, with source code one-pass text analysis, static analysis can determine if any risky context API values are used within the application [51]. This technique is neither sound nor complete. Second, a complete static analysis technique could be employed by using a context-sensitive control flow analysis examine every potential Android API call where the database access methods are accessed for writing. This technique would identify every risky methodology for creating a table using the

Android API. Keep in mind that complete static analysis technique guarantees no “false negatives” from an Android API perspective. If a developer were to import database code not native to the Android API, the static analysis would need to be updated accordingly. Third, to improve the soundness from the second technique, data flow analysis could be employed to help determine if sensitive data may in fact be stored into the database or retrieved from the database to be used for application configuration.

```

public class FeedReaderDbHelper extends SQLiteOpenHelper {
    // If you change the database schema, you must increment the database version.
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "FeedReader.db";

    public FeedReaderDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        onUpgrade(db, oldVersion, newVersion);
    }
}
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.openDatabase(File, SQLiteDatabase.OpenParams)
// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle);
// Insert the new row, returning the primary key value of the new row
long newRowId = db.insert(FeedEntry.TABLE_NAME, null, values);

```

Table 4: Android Java Code for SQL Lite Database [55]

Another *SharedSpace* in Android *onA RootedDevice isA ApplicationSandbox* [52]. The relationship *onA* is needed to show that the *ApplicationSandbox* becomes a *SharedSpace*

once a device is rooted and not before. A rooted device is not in itself a *SharedSpace*. Thus, the relationship is not hierarchical. Both the *ApplicationSandbox* and *SharedStorage* may have (i.e. *hasA*) file(s) on them. In either case, files stored in the location are vulnerable if they are without proper *FileBasedEncryption (FBE)*. The *without* relation is necessary as *FBE* is not (i.e. *isA*) a hierarchical relationship. If this *File* is without *File Based Encryption (FBE)*, it is then vulnerable to use by other Android applications as it is in a default common space [50]. The relationship *without* is important to show a missing property for secure data storage.

In addition to the above, application developers can choose to specify where they prefer that their application can be installed. Android currently permits developers to specify applications to *PreferInstallation* entirely onto external storage. This is a popular choice when applications are extremely large. Applications can be stored on external storage, a *SharedSpace*, by specifying the string *android:installLocation* attribute in the application manifest [57]. A static analysis of the application manifest can detect this storage choice in simply a string analysis of the manifest. Applications stored entirely in *SharedSpace* are subject to more concerns than other applications as their *ApplicationSandbox* is also on *SharedSpace*. The device can further be rooted; however, rooting is an entirely separate concern.

In summary, we have identified at least six sub-areas where the static analysis of developer Android application source code can be examined for standard risky Android API calls which are subjects to the risk of OWASP Mobile Threat 2 is “Insecure Data Storage”

(MD_IDS). The threat of insecure data storage (IDS) can potentially lead to data CIA compromise or the propagation of malware. The standard Android API calls include reading and writing from the internal file storage, external file storage, shared preferences, prefer installLocation, databases, and the underlying stored files themselves. As our cybersecurity knowledge expands, so further can our knowledge-graph. This dynamic representation provides a standardized methodology to reason about mobile software assurance.

4.1.3 Mobile 2014 Threat 3: Insufficient Transport Layer Protection

The OWASP 2014 Mobile Threat 3 is “Insufficient Transport Layer Protection” (M3_Insufficient_Transport_Layer_Protection), as seen in Figure 16. Insufficient Transport Layer Protection (ITLP) can potentially lead to unauthorized compromise of the service confidentiality, integrity, and availability. Two main drivers of the OWASP Mobile Threat 3 are improper certificate validation and weak negotiated transport protection, as seen in the knowledge-graph below.

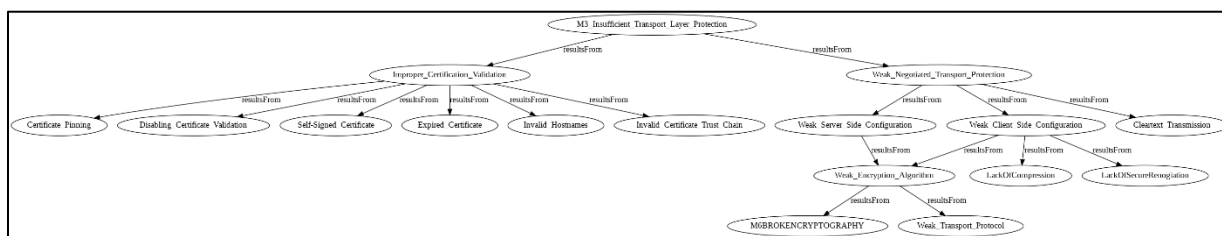


Figure 16: OWASP Mobile 2014 Threat 3: Insufficient Transport Layer Protection

Improper certificate validation can occur at the mobile client when it receives the certificate from the server for which the client would like to negotiate transport layer security. The certificate houses a server’s public key; and, the certificate is signed by a

certifying authority (CA). Improper certificate validation *resultsFrom* three closely related client-side weaknesses: disability certificate validation, not validating certificate expiration, and not validating the domains for which the certificate is valid. Valid Android mobile application code for making HTTPS request is show in Table 5.

```
URL url = new URL("https://wikipedia.org");
URLConnection urlConnection = url.openConnection();
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);
```

Table 5: Valid Android HTTPS Request [58]

Valid Android mobile application code [58] for checking the domain is shown in Table 6.

```
// Load CAs from an InputStream
// (could be from a resource or ByteArrayInputStream or ...)
CertificateFactory cf = CertificateFactory.getInstance("X.509");
// From https://www.washington.edu/itconnect/security/ca/load-der.crt
InputStream caInput = new BufferedInputStream(new FileInputStream("load-
der.crt"));
Certificate ca;
try {
    ca = cf.generateCertificate(caInput);
    System.out.println("ca=" + ((X509Certificate) ca).getSubjectDN());
} finally {
    caInput.close();
}

// Create a KeyStore containing our trusted CAs
String keyStoreType = KeyStore.getDefaultType();
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);

// Create a TrustManager that trusts the CAs in our KeyStore
String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);

// Create an SSLContext that uses our TrustManager
SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);

// Tell the URLConnection to use a SocketFactory from our SSLContext
```

```

URL = new URL("https://certs.cac.washington.edu/CAtest/");
HttpsURLConnection urlConnection =
    (HttpsURLConnection)url.openConnection();
urlConnection.setSSLSocketFactory(context.getSocketFactory());
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);

```

Table 6: Android adding a Certifying Authority source code [58]

Valid Android mobile application code [58] for trusting a certifying checking the domain as shown in Table 7.

```

// Create an HostnameVerifier that hardwires the expected hostname.
// Note that is different than the URL's hostname:
// example.com versus example.org
HostnameVerifier hostnameVerifier = new HostnameVerifier() {
    @Override
    public boolean verify(String hostname, SSLSession session) {
        HostnameVerifier hv =
            HttpsURLConnection.getDefaultHostnameVerifier();
        return hv.verify("example.com", session);
    }
};

// Tell the URLConnection to use our HostnameVerifier
URL url = new URL("https://example.org/");
HttpsURLConnection urlConnection =
    (HttpsURLConnection)url.openConnection();
urlConnection.setHostnameVerifier(hostnameVerifier);
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);

```

Table 7: Android Certificate Hostname Verifier source code [58]

There is conflicting security advice on certificate pinning. In pinning, the application is restricted to only allowing the application trusted or pinned certifying authorities. If an invalid certificate is pinned, then, the application is vulnerable to interception; however, the inclusion is to allow the mobile application to specify which certifying authority to use. Valid Android mobile application code for certificate pinning is shown in Table 8.

```

<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config>
    <domain includeSubdomains="true">appmattus.com</domain>
    <pin-set>
      <pin digest="SHA-256">***</pin>
      <pin digest="SHA-256">***</pin>
    </pin-set>
  </domain-config>
</network-security-config>

```

Table 8: Android Static Certificate Pinning in manifest source code [59]

```

TrustManagerFactory trustManagerFactory =
    TrustManagerFactory.getInstance(
        TrustManagerFactory.getDefaultAlgorithm());
trustManagerFactory.init((KeyStore) null);
// Find first X509TrustManager in the TrustManagerFactory
X509TrustManager x509TrustManager = null;
for (TrustManager trustManager : trustManagerFactory.getTrustManagers()) {
    if (trustManager instanceof X509TrustManager) {
        x509TrustManager = (X509TrustManager) trustManager;
        break;
    }
}

X509TrustManagerExtensions trustManagerExt =
    new X509TrustManagerExtensions(x509TrustManager);
...
URL url = new URL("https://www.appmattus.com/");
HttpsURLConnection urlConnection =
    (HttpsURLConnection) url.openConnection();
urlConnection.connect();

Set<String> validPins = Collections.singleton
    ("***");
validatePinning(trustManagerExt, urlConnection, validPins);

```

Table 9: Android Dynamic Certificate Pinning source code [59]

4.1.4 Mobile 2014 Threat 4: Unintended Data Leakage

The OWASP 2014 Mobile Threat 4 is “Unintended Data Leakage” (M4_Unintended_Data_Leakage), as in Figure 17. Unintended Data Leakage (UDL) can potentially lead to data compromise of sensitive information [48]. From the application level, the leaking of sensitive data can potentially *resultFrom*, storing data insecurely, using

untrusted application plugins, insecure logging application activities, and allowing screenshots can potentially *resultFrom* storing data in a vulnerable location.

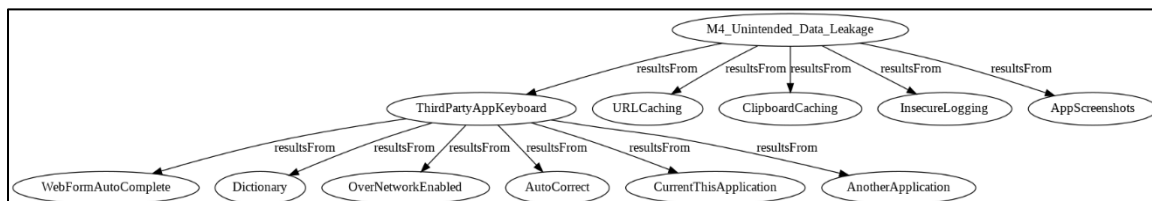


Figure 17: OWASP Mobile 2014 Threat 4: Unintended Data Leakage

Android application can accept the usage of third-party keyboards. These third-party keyboards could contain any set of threats. An April 2019 article lists some top Android keyboards as: Gboard, SwiftKey, Chroma, and Fleksy [60]. The Android custom keyboard source code [61] can be seen in Table 10.

Additional issues with third-party application keyboards *resultsFrom* storing sensitive information into the shared dictionary, enables networked connected keyboards, storing sensitive information into the shared autocorrect tables, storing sensitive information into the shared autocomplete tables. In addition, one or more Android applications may be susceptible to a vulnerable/malicious keyboard depending on how it is installed on the client-side.

```

public class MyInputMethodService extends InputMethodService implements
KeyboardView.OnKeyboardActionListener {
    private KeyboardView ;
    private Keyboard keyboard;
    private boolean caps = false;

    @Override
    public View onCreateInputView() {
        keyboardView =
            (KeyboardView)getLayoutInflater().inflate(R.layout.keyboard_view, null);
        keyboard = new Keyboard(this, R.xml.keys_layout);
        keyboardView.setKeyboard(keyboard);
        keyboardView.setOnKeyboardActionListener(this);
        return keyboardView;}

    ...

    @Override
    public void onKey(int primaryCode, int[] keyCodes) {
        InputConnection inputConnection = getCurrentInputConnection();
        if (inputConnection != null) {
            switch(primaryCode) {
                case Keyboard.KEYCODE_DELETE :
                    CharSequence selectedText = inputConnection.getSelectedText(0);
                    if (TextUtils.isEmpty(selectedText)) {
                        inputConnection.deleteSurroundingText(1, 0);
                    } else {
                        inputConnection.commitText("", 1);
                    }
                case Keyboard.KEYCODE_SHIFT:
                    caps = !caps;
                    ...
                    break;
                case Keyboard.KEYCODE_DONE:
                    inputConnection.sendKeyEvent(new KeyEvent(KeyEvent.ACTION_DOWN,
                    KeyEvent.KEYCODE_ENTER));
                    break;
                default :
                    char code = (char) primaryCode;
                    ...
            }
        }
    }
}

```

Table 10: Android custom keyboard source code [61]

Another form of caching which can result in unintended data leakage *resultsFrom* URL caching. In this caching, certain URL parameters may be cached. An Android source code can be seen in Table 11 [62].


```

public class MainActivity extends ActionBarActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ImageView imageView =
            (ImageView)findViewById(R.id.imageView);
        // Configure image loader
        ImageLoaderConfiguration config = new
            ImageLoaderConfiguration.Builder(getApplicationContext())
            // Thread priority
            .threadPriority(Thread.NORM_PRIORITY)
            // Deny cache multiple image sizes on memory
            .denyCacheImageMultipleSizesInMemory()
            // Processing order like a stack (last in, first out)
            .tasksProcessingOrder(QueueProcessingType.LIFO)
            // Max image size to cache on memory
            .memoryCacheSize(1*1024*2014)
            // Max image size to cache on disc
            .diskCacheSize(2*1024*1024)
            // Write log messages
            .writeDebugLogs()
            .build();

        ImageLoader.getInstance().init(config);
        // Get ImageLoader instance
        ImageLoader imageLoader=ImageLoader.getInstance();
        // Define image display options
        DisplayImageOptions options = new DisplayImageOptions.Builder()
            // Cache loaded image in memory and disc
            .cacheOnDisk(true)
            .cacheInMemory(true)
            // Show Android icon while loading
            .showImageOnLoading(R.drawable.ic_launcher)
            .build();
        String imgUrl="http://www.***.test.jpg";
        imageLoader.displayImage(imgUrl, imageView, options);}}

```

Table 11: Android URL caching source code [62]

Another source of potential unintended data leakage *ResultsFrom* caching. Caching in general should only be used for non-sensitive data. An example of caching a file can be seen in Table 12.

```

File cacheDir = getCacheDir();
File fileToCache = new File(myDownloadedFileUri);
String fileToCacheName = fileToCache.getName();
File cacheFile = new File(cacheDir.getPath(), fileToCacheName);

```

Table 12: Android caching source code [63]

Another source of potential unintended data leakage *ResultsFrom* clipboard caching. In this caching, certain sensitive data may be cached. An Android source code [64] can be seen in Table 13.

```
//Get System Clipboard
//if the user selects copy
case R.id.menu_copy:
// Gets a handle to the clipboard service.
ClipboardManager clipboard = (ClipboardManager)
    getSystemService(Context.CLIPBOARD_SERVICE);
//Copy the data to a new ClipData object
//text
// Creates a new text clip to put on the clipboard
ClipData clip = ClipData.newPlainText("simple text", "Hello, World!");
//URI
// Creates a Uri based on a base Uri and a record ID based on the contact's last
name
// Declares the base URI string
private static final String CONTACTS = "content://com.example.contacts";
// Declares a path string for URIs that you use to copy data
private static final String COPY_PATH = "/copy";
// Declares the Uri to paste to the clipboard
Uri copyUri = Uri.parse(CONTACTS + COPY_PATH + "/" + lastName);
// Creates a new URI clip object. The system uses the anonymous
getContentResolver() object to
// get MIME types from provider. The clip object's label is "URI", and its data is
// the Uri previously created.
ClipData clip = ClipData.newUri(getContentResolver(), "URI", copyUri);
//Intent
// Creates the Intent
Intent appIntent = new Intent(this, com.example.demo.myapplication.class);
// Creates a clip object with the Intent in it. Its label is "Intent" and its data is
// the Intent object created previously
ClipData clip = ClipData.newIntent("Intent", appIntent);
//Put Clip Object on the Clipboard
// Set the clipboard's primary clip.
clipboard.setPrimaryClip(clip);
//Paste from clipboard
ClipboardManager clipboard = (ClipboardManager)
getSystemService(Context.CLIPBOARD_SERVICE);
String pasteData = "";
```

Table 13: Android copy/paste from clipboard source code [64]

Another source of unintended data leakages potentially results from application screenshots. Screenshots can be taken from other running application or even by the user.

Screens containing sensitive identifiers should not permit screenshots. Android source code for executing a screenshot from an application [65] shown in Table 14.

```

public class ScreenShotActivity extends Activity{
private RelativeLayout relativeLayout;
private Bitmap myBitmap;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
relativeLayout = (RelativeLayout)findViewById(R.id.relative1);
relativeLayout.post(new Runnable() {
public void run() {
//take screenshot
myBitmap = captureScreen(relativeLayout);
Toast.makeText(getApplicationContext(), "Screenshot captured..!",
Toast.LENGTH_LONG).show();
try {
if(myBitmap!=null){
//save image to SD card
saveImage(myBitmap);
}
Toast.makeText(getApplicationContext(), "Screenshot saved..!",
Toast.LENGTH_LONG).show();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}}});
}
}

```

Table 14: Android screenshot source code [65]

4.1.5 Mobile 2014 Threat 5: Poor Authorization and Authentication

The OWASP 2014 Mobile Threat 5 is “Poor Authorization and Authentication” (M4_Unintended_Data_Leakage), as seen in Figure 18. Poor Authorization and Authentication (PAA) can potentially lead to the CIA triad compromise.

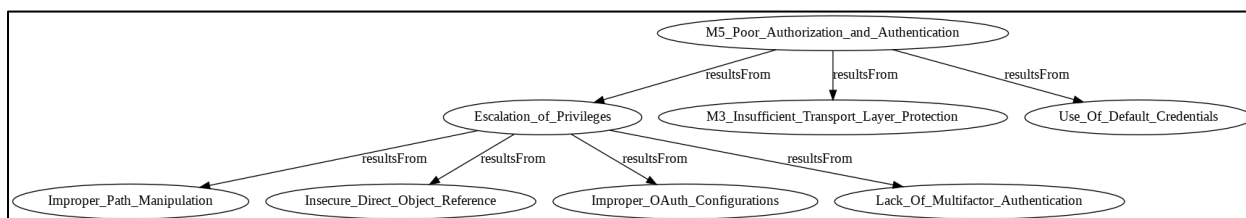


Figure 18: OWASP Mobile 2014 Threat 5: Poor Authorization and Authentication

Typically, web applications are susceptible to escalation of privileges at the server-side. Server-side privilege escalation can *resultFrom* improper path manipulations, secure direct object references and robust error authentication error messages, A lack of multi-factor authentication can also result in escalation in privilege vulnerabilities. Finally, at both the client-side and server-side improperly configured authentication (e.g. OAuth) can result in an escalation of privileges. Android OAuth code [66] can be identified as seen Table 15. Other examples of authentication can be seen in Table 16 and Table 17.

```

AccountManager am = AccountManager.get(this);
Bundle options = new Bundle();
am.getAuthToken(
    ...
    options,           // Authenticator-specific options
    this,             // Your activity
    new OnTokenAcquired(), // Callback called when a token is
    successfully acquired
    new Handler(new OnError())); // Callback called if an error occurs

```

Table 15: Android OAuth source code [66]

Poor authorization and authentication can also occur from interceptions, which occur during Mobile 2014 Threat 3, which is the Insufficient Transport Layer Protections, discussed in Section 4.1.3.

```

// Retrieve stored credentials with Auth.CredentialsApi.request()
Auth.CredentialsApi.request(mCredentialsClient,
mCredentialRequest).setResultCallback(
    new ResultCallback() {
        ... });

```

Table 16: Android credential source code [67]

Finally, poor authorization and authentication can *resultFrom* the use of default or hardcoded credentials [67].

```

private Handler handler = new Handler();
private Executor executor = new Executor() {
    @Override
    public void execute(Runnable command) {
        handler.post(command);
    }
};
@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    // Prompt appears when user clicks "Log in"
    Button biometricLoginButton = findViewById(R.id.biometric_login);
    biometricLoginButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            showBiometricPrompt();
        }
    });
private void showBiometricPrompt() {
    BiometricPrompt.PromptInfo promptInfo =
        new BiometricPrompt.PromptInfo.Builder()
            .setTitle("Biometric login for my app")
            .setSubtitle("Log in using your biometric credential")
            .setNegativeButtonText("Cancel")
            .build();
    BiometricPrompt biometricPrompt = new BiometricPrompt(MainActivity.this,
        executor, new BiometricPrompt.AuthenticationCallback() {
        @Override
        public void onAuthenticationError(int errorCode,
            @NonNull CharSequence errString) {
            super.onAuthenticationError(errorCode, errString);
            Toast.makeText(getApplicationContext(),
                "Authentication error: " + errString, Toast.LENGTH_SHORT)
                .show();
        }
        @Override
        public void onAuthenticationSucceeded(
            @NonNull BiometricPrompt.AuthenticationResult result) {
            super.onAuthenticationSucceeded(result);
            BiometricPrompt.CryptoObject authenticatedCryptoObject =
                result.getCryptoObject();
            // User has verified the signature, cipher, or message
            // authentication code (MAC) associated with the crypto object, so
            // you can use it in your mobile applicaiton's crypto-driven workflows.
        }
    });
    @Override
    public void onAuthenticationFailed() {
        super.onAuthenticationFailed();
        Toast.makeText(getApplicationContext(), "Authentication failed",
            Toast.LENGTH_SHORT)
            .show();
    }
};
// Displays the "log in" prompt.
biometricPrompt.authenticate(promptInfo);

```

Table 17: Android biometric authentication source code [63]


```

int iterationCount = 1000;
int saltLength = 32; // bytes; should be the same size
    as the output (256 / 8 = 32)
int keyLength = 256; // 256-bits for AES-256, 128-bits for AES-128, etc
byte[] salt; // Should be of saltLength

/* When first creating the key, obtain a salt with this: */
SecureRandom random = new SecureRandom();
byte[] salt = new byte[saltLength];
random.nextBytes(salt);

/* Use this to derive the key from the password: */
KeySpec keySpec= new PBEKeySpec(password.toCharArray(), salt,
    iterationCount, keyLength);
SecretKeyFactory keyFactory = SecretKeyFactory
    .getInstance("PBKDF2WithHmacSHA1");
byte[] keyBytes = keyFactory.generateSecret(keySpec).getEncoded();
SecretKey key = new SecretKeySpec(keyBytes, "AES");

```

Table 18 Nikolay Elenkov Example [71] [72]

Weak keys are well known to break cryptography. Weak keys are the *resultFrom* at least three predominate causes. First, keys may not be stored correctly and therefore they are not properly guarded, perhaps by the Android KeyStore [73]. An examples of such a scenario are when the key is stored in a shared space next to the encrypted data. Second, the key derivation function may not be properly coded. In such a scenario, keys may not be randomly generated. Third, the key length changes with industry best practices based on computational power. In such cases, older code relying on shorter key lengths again increases the risk around real time brute force attacks [69].

4.1.7 Mobile 2014 Threat 7: Client-side Injection

The OWASP 2014 Mobile Threat 7 is “Client-side Injection” (M7_Client_Side_Injection), as seen in Figure 20. Client-side Injection can potentially lead to data compromise of sensitive information.

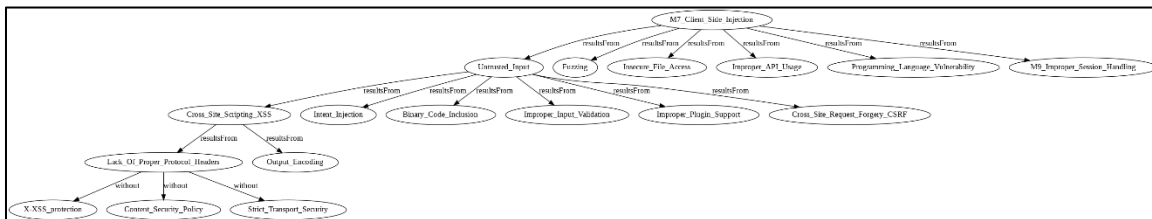


Figure 20: OWASP Mobile 2014 Threat 7: Client-side Injection

A traditional client-side injection *resultsFrom* allowing scripts to run locally at the client. If an application must use the JavaScript interface (Android 6.0+), best practice is to use HTML message channels instead of *evaluateJavascript()* to communicate between client and server, as shown below [63].

```

WebView myWebView = (WebView) findViewById(R.id.webview);
// messagePorts[0] and messagePorts[1] represent the two ports.
// They are already tangled to each other and have been started.
WebMessagePort[] channel = myWebView.createWebMessageChannel();
// Create handler for channel[0] to receive messages.
channel[0].setWebMessageCallback(new
WebMessagePort.WebMessageCallback() {
    @Override
    public void onMessage(WebMessagePort port, WebMessage message) {
        Log.d(TAG, "On port " + port + ", received this message: " + message);
    }
});
// Send a message from channel[1] to channel[0].
channel[1].postMessage(new WebMessage("My secure message"));

```

Table 19: Android running JavaScript [63]

Another traditional source of client-side injection *resultsFrom* an application blindly trusting user input. In the below code, we see where data from the client is directly used to make logical decisions in the code [74].


```

//This defines a one-element String array to contain the selection argument.
String[] selectionArgs = {""};
// Gets a word from the UI
searchString = searchWord.getText().toString();
// Remember to insert code here to check for invalid or malicious input.
// If the word is the empty string, gets everything
if (TextUtils.isEmpty(searchString)) {
    // Setting the selection clause to null will return all words
    selectionClause = null;
    selectionArgs[0] = "";
} else {
    // Constructs a selection clause that matches the word that the user entered.
    selectionClause = UserDictionary.Words.WORD + " = ?";
    // Moves the user's input string to the selection arguments.
    selectionArgs[0] = searchString;
}
// Does a query against the table and returns a Cursor object
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // The content URI of the words table
    projection, // The columns to return for each row
    selectionClause, // Either null, or the word the user entered
    selectionArgs, // Either empty, or the string the user entered
    sortOrder); // The sort order for the returned rows
// Some providers return null if an error occurs, others throw an exception
if (null == mCursor) {
    /* Insert code here to handle the error. Be sure not to use the cursor! You may want to
    * call android.util.Log.e() to log this error.*/
}
// If the Cursor is empty, the provider found no matches
} else if (mCursor.getCount() < 1) {
    /*
    * Insert code here to notify the user that the search was unsuccessful.
    * This isn't necessarily an error. You may want to offer the user the option to
    * insert a new row, or re-type the search term.
    */
} else {
    // Insert code here to do something with the results
}

```

Table 20: Android client-side injection source code [74].

4.1.8 Mobile 2014 Threat 8: Security Decisions Via Untrusted Inputs

The OWASP 2014 Mobile Threat 8 is “Security Decisions Via Untrusted Inputs” (M8_Security_Decisions_Via_Untrusted_Inputs), as seen in Figure 21. Security Decisions Via Untrusted Inputs can potentially lead to data compromise of the CIA triad.

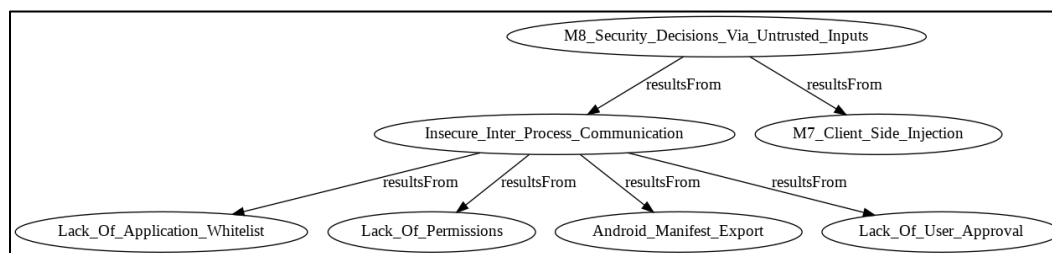


Figure 21: OWASP Mobile 2014 Threat 8: Security Decisions Via Untrusted Inputs

In Threat 8 “Security Decisions via Untrusted Inputs” *resultsFrom* two main categories: Insecure Inter-Process Communication (IPC) and Mobile Threat 7 Client-side Injection.

Insecure Inter-Process Communication (IPC) *resultsFrom* not properly putting information into an allow list. For example, it is recommended that applications using WebView should only load whitelisted content into WebView objects to disallow users from navigating away to sites that are outside of the application control [63].

Insecure Inter-Process Communication (IPC) *resultsFrom* not properly password protecting information passed between applications which are developed by the same developer with the intent to share data between application. In this scenario when sharing data between two mobile applications that the developer owns or controls, the application should be written to use signature-based permissions. These permissions don't require user confirmation and instead check that the mobile applications accessing the data are signed using the same signing key. Therefore, these permissions offer a more streamlined, secure user experience.

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapplication">
  <permission android:name="my_custom_permission_name"
    android:protectionLevel="signature" />

```

Table 21: Permissions between two+ co-owned applications source code [63]

Insecure Inter-Process Communication (IPC) *resultsFrom* exporting services which are not designed to be shared. For example, *ContentProviders* may not be written to trust requests from outside parties. *ContentProviders* which are not intended to be shared should specifically disallow access to the application content providers in the Android Manifest as seen below [63].

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapplication">
  <application ... >
    <provider
      android:name="android.support.v4.content.FileProvider"
      android:authorities="com.example.myapplication.fileprovider"
      ...
      android:exported="false">
      <!-- Place child elements of <provider> here. -->
    </provider>
    ...
  </application>
</manifest>

```

Table 22: Android disallow export of Content Providers source code [63]

4.1.9 Mobile 2014 Threat 9: Improper Session Handling

The OWASP 2014 Mobile Threat 9 is “Improper Session Handling” (M9_Improper_Session_Handling), as seen in the figure below. Improper Session Handling can potentially lead to the data compromise of sensitive information as unauthorized entities can take advantage of weaknesses to either manipulate the confidentiality, integrity or availability of data and system resources. From an application level, improper session handling can potentially *resultFrom* three main software

development errors: improper session tokens, improper session timeout, and improper session termination. The knowledge graph for the threat of Improper Session Handling is seen in Figure 22.

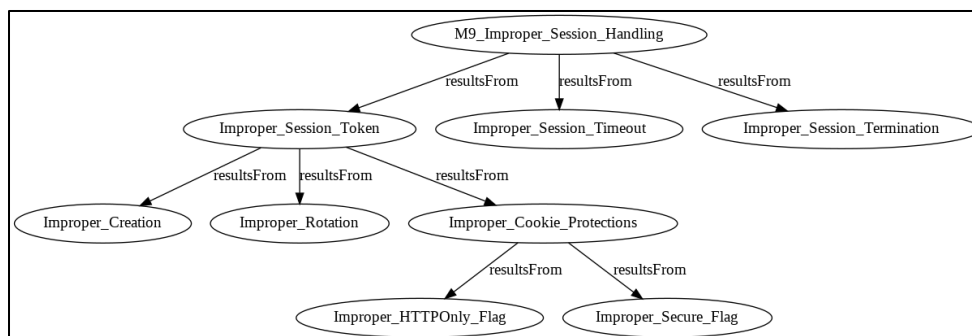


Figure 22; OWASP Mobile 2014 Threat 9: Improper Session Handling

In the session-based authentication, a server will create a session for the client after login. During the session creation, the server creates a token which is subsequently passed to the client. During all subsequent client-server correspondence, the session token is almost always sent. The server evaluates the token seen from the client against the information stored at the server for client authenticity. As such, best practice requires session tokens to be created at the server-side. Improper creation occurs when the token is improperly created such as using a counter or any easy-to-guess value. Best practice require the value to be nearly random such as using an industry best practice hashing algorithm on the current time and random nonce. Improper session token rotation can result in session fixation so that an unauthorized entity can improperly access the web application.

Session tokens can be further protected by setting specific flags. The *HTTPOnly* cookie flag is designed to mitigate the risk of a client-side script accessing the protected

cookie. The HTTPOnly cookie flag in Android [75] can be checked on a *HttpCookie* cookie type by calling *isHttpOnly()*. The Secure cookie flag requires the client to not sent the cookie in plaintext. The Secure flag can be checked in the Android application with an API call to *getSecure()* [75].

```

@Override
public NewCookie fromString(final String value) {
    if (value == null || value.isEmpty()) {
        return null;
    }

    final List<HttpCookie> httpCookies = HttpCookie.parse(value);
    final HttpCookie httpCookie = httpCookies.get(0);
    return new NewCookie(
        httpCookie.getName(),
        httpCookie.getValue(),
        httpCookie.getPath(),
        httpCookie.getDomain(),
        httpCookie.getVersion(),
        httpCookie.getComment(),
        (int) httpCookie.getMaxAge(),
        null,
        httpCookie.getSecure(),
        httpCookie.isHttpOnly());
}

```

Table 23: Android API Cookie Flags [76]

Improper session timeout forces a session timeout when the user has been inactive for a certain amount of time, typically defined by industry best practice. There are two paradigms for developing a timeout: at the client-side and at the server-side. Server-side session timeouts is considered industry best practice since client-side application timeouts are vulnerable to interception. At the server-side, the web application must be specifically built to timeout inactive clients. At the client-side, which would be directly within the installed Android application, there are no built in Android API session timeouts. However, a user on StackOverflow [77] recommends, “use a *CountDownTimer* and bind

it with an interface callback like `onUserInteracted()` and reset the timer whenever it is called.”

Improper session termination is similar into timeout in that it is not fundamentally supported in the Android API. Session termination can occur at either the server-side, which is considered best practice, or termination can occur on the client-side, which is vulnerable to interception. Basically, in a session termination, the web-application will no longer accept the terminated session token as legitimate. In such a scenario, the web-application forces the user to re-authenticate to establish a new session token.

As most of session management is implemented in the server-side, it is difficult to detect them with a static analysis of the Android client application source code unless the developer has written in specific related client-side code.

4.1.10 Mobile 2014 Threat 10: Lack of Binary Protections

The OWASP 2014 Mobile Threat 10 is “Lack of Binary Protections” (M10_Lack_of_Binary_Protections), as seen in Figure 23. Lack of Binary Protections can potentially lead to data compromise of sensitive data.

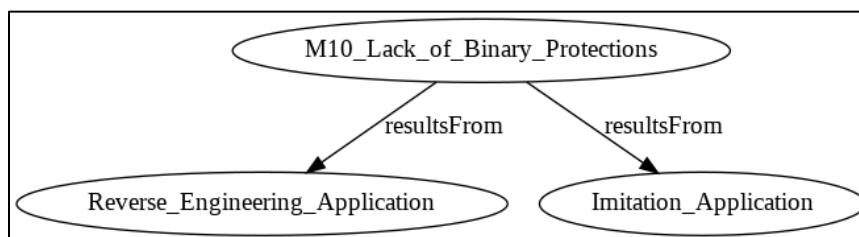


Figure 23: OWASP Mobile 2014 Threat 10: Lack of Binary Protections

Lack of Binary Protections *resultsFrom* mainly two categories: Reverse Engineering the Application and from an Imitation Application.

Reverse engineering of Android applications can be made more difficult by employing some of the Android services. Current industry tools such as ProGuard and DexGuard are Java bytecode optimizers which make reverse engineering applications more difficult [78] [79]. This feature can be statically checked on compiled Android APK files.

Imitation applications can be somewhat difficult to detect. It is advised that developers of Android applications sign their application and develop a check into the source code [80].

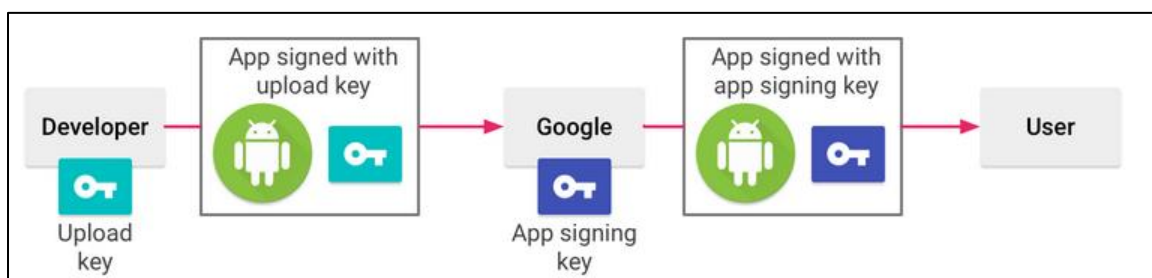


Figure 24: Android Application Signing [80]

However, according to Google [80] when the developer creates a signing configuration, Android Studio “adds [the] signing information in plain text to the module's *build.gradle* files. [...] [developers] should move this sensitive information out of the build files so it is not easily accessible to others.” The *build.gradle* files improved, is seen below:

```

// Create a variable called keystorePropertiesFile, and initialize it to your
// keystore.properties file, in the rootProject folder.
def keystorePropertiesFile = rootProject.file("keystore.properties")

// Initialize a new Properties() object called keystoreProperties.
def keystoreProperties = new Properties()

// Load your keystore.properties file into the keystoreProperties object.
keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
android { ...}

```

Table 24: Android signing remnants source code [80]

4.2 OWASP 2016 Mobile Threats Coding Patterns

This section examines top OWASP 2016 mobile threats through ontology and Android source code. The OWASP Top Ten Mobile Threats are still the main year posted on the OWASP portal. The specific threats can be seen in Figure 25.

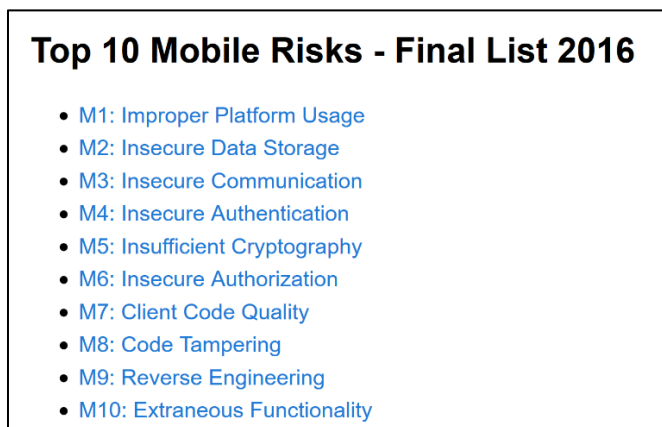


Figure 25: OWASP Mobile Top Ten Threats of 2016

4.2.1 Mobile 2016 Threat 1: Improper Platform Usage

The OWASP 2016 Mobile Threat 1 is “Improper Platform Usage” (M1_Improper_Platform_Usage), as seen in Figure 26. Improper Platform Usage can potentially lead to data compromise of sensitive data or the unauthorized system access.

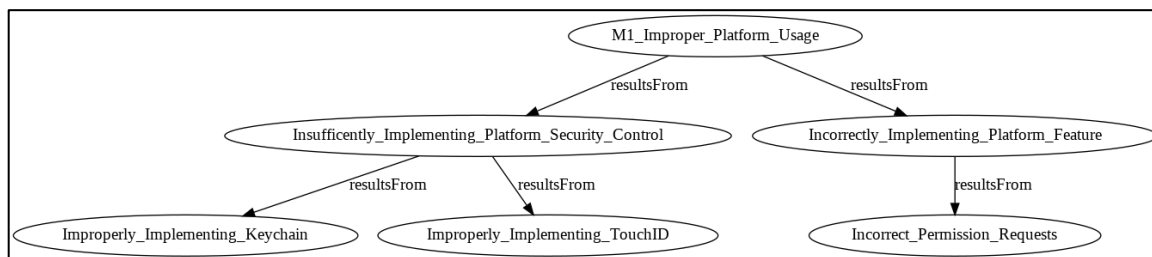


Figure 26: OWASP Mobile 2016 Threat 1: Improper Platform Usage

This threat *resultsFrom* *Insufficiently_Implementing_Platform_Security_Controls* or *resultsFrom* *Incorrectly_Implementing_Platform_Features*.

Insufficiently_Implementing_Platform_Security_Controls *resultsFrom*

Improperly_Implementing_Keychains and *Improperly_Implementing_TouchID*. Google

maintains blogs for best practices of software development of platform security controls which in many cases describes how to properly implement platform features.

Incorrectly_Implementing_Platform_Features *resultFrom*

Incorrect_Permission_Requests. For example, an application may ask for many

unnecessary or incorrect permissions which are not required by the application. As such, the vulnerable code with the improperly allowed permissions could potentially be exploited

by OWASP 2016 Mobile Threat 8 M8_Code_Tampering described in Section 4.2.8.

It is possible to perform a static analysis on Android API class which require certain permissions to ensure that the correct permissions are requested by the application for user approval.

4.2.2 Mobile 2016 Threat 2: Insecure Data Storage

The OWASP 2016 Mobile Threat Two is “Insecure Data Storage” (M2_Insecure_Data_Storage), as seen in Figure 27. IDS can potentially lead to data compromise of sensitive data or the unauthorized data access.

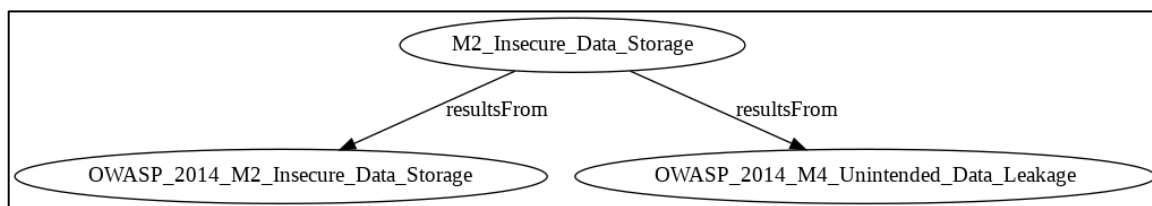


Figure 27: OWASP Mobile 2016 Threat 2: Insecure Data Storage

According to the OWASP 2016 Mobile Threat categories [81], we built the 2016 threat *resultsFrom* either *OWASP_2014_M2_Insecure_Data_Storage* or, since the *M4* category disappeared in 2016, *resultsFrom* *OWASP_2014_M4_Unintended_Data_Leakage*. Therefore, the 2016 category of threats is simply a composite of two former OWASP 2014 threats M2 IDS, Section 4.1.2, and M4, Section 4.1.4, “Unintended Data Leakage.” The beauty of the knowledge graph representation development over time helps build a threat landscape map of the field to show how prior analyses reflect with current cybersecurity analyses and trends. This graphical information is extremely valuable especially if examining how code changes or drifts with time or examining earlier analyses after potential data breach insurance claim reports. Furthermore, the graphical representation can be transformed into many different OWL languages to furthermore enable cybersecurity in the semantic web.

4.2.3 Mobile 2016 Threat 3: Insecure Communication

The OWASP 2016 Mobile Threat 3 is “Insecure Communication” (M3_Insecure_Communication), as seen in Figure 28. Insecure Communication can potentially lead to data compromise of sensitive data or the unauthorized system or data access.

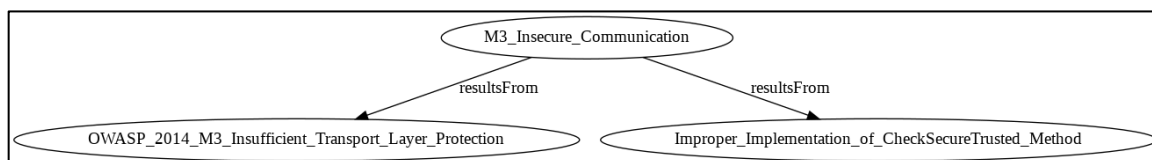


Figure 28: OWASP Mobile 2016 Threat 3: Insecure Communication

According to the OWASP 2016 Mobile Threat categories [81], the 2016 threat *resultsFrom* either *OWASP_2014_M3_Insufficient_Transport_Layer_Protection* or *Improper_Implementation_CheckServerTrusted_Method within the SSLSocketFactory*.

```

public class EasyX509TrustManager implements X509TrustManager {
    private X509TrustManager standardTrustManager = null;
    ...

    public void checkClientTrusted(X509Certificate[] certificates, String authType) throws
    CertificateException {
        standardTrustManager.checkClientTrusted(certificates, authType);
    }
    ...

    public void checkServerTrusted(X509Certificate[] certificates, String authType)
    throws CertificateException {
        if ((certificates != null) && (certificates.length == 1)) {
            certificates[0].checkValidity();
        } else {
            standardTrustManager.checkServerTrusted(certificates, authType);
        }
    }
    ...}
  
```

Table 25: Android Implementation CheckServerTrusted source code [82]

Table 25 shows a properly implementation of the CheckServerTrusted method for SSLSocketFactory connections. The OWASP 2014 threats M3 described in Section 4.1.3.

4.2.4 Mobile 2016 Threat 4: Insecure Authentication

The OWASP 2016 Mobile Threat 4 is “Insecure Authentication” (M4_Insecure_Authentication), as seen in Figure 29. Insecure Authentication can potentially lead to data compromise of sensitive data or the unauthorized system or data access.

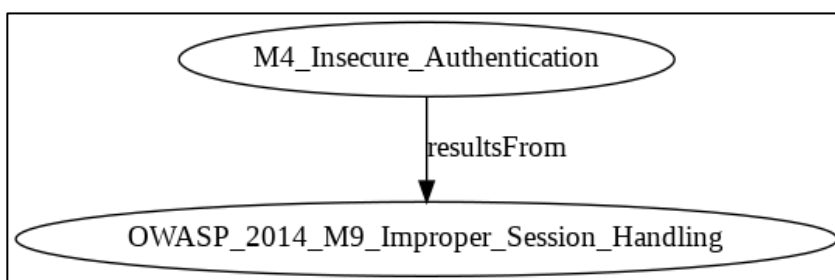


Figure 29: OWASP Mobile 2016 Threat 4: Insecure Authentication

According to the OWASP 2016 Mobile Threat categories [81], the 2016 threat *resultsFrom* *OWASP_2014_M9_Improper_Session_Handling* described in Section 4.1.9.

4.2.5 Mobile 2016 Threat 5: Insufficient Cryptography

The OWASP 2016 Mobile Threat 5 is “Insufficient Cryptography” (M5_Insufficient_Cryptography), as seen in Figure 30. Insufficient Cryptography (IC) can potentially lead to at least data compromise as cryptography is predominantly driven by information confidentiality requirements [83]. From a mobile application level, the application should enforce cryptography is sufficient from its scope. Insufficient cryptography puts information confidentiality at risk during both storage as well as transmission.

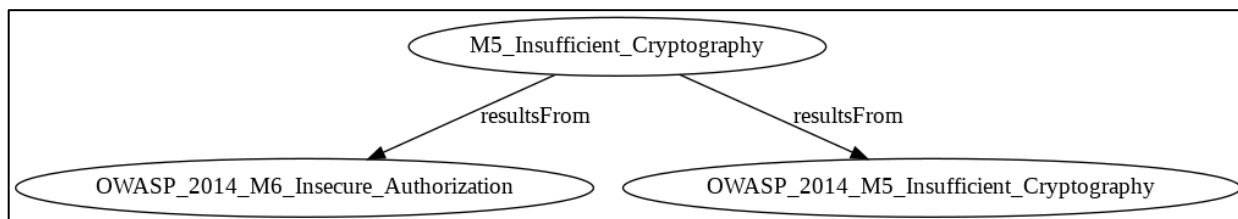


Figure 30: OWASP Mobile 2016 Threat 5: Insufficient Cryptography

From the perspective of an application, there are three main relationships for insufficient cryptography. First, insufficient cryptography can potentially *resultFrom* platform specific issues or using a weak cryptography function. In addition, insufficient cryptography used *onA* certain data (i.e. physical domain) can result in higher risks for the domain information compromise. Third, insufficient cryptography can potentially *resultFrom* improperly applying cryptography techniques.

Platform specific issues include having a device that is unable to perform the proper cryptography as a *resultFrom* from power constraints. As mobile devices can linger on networks for many years [84], their platform may not be able to keep up with modern cryptographic requirements. On average, older cryptographic techniques become deprecated with time for multiple reasons [68]. For example, there could be other platform specific issues, or the algorithm may be weak, preventing the implementation of sufficient cryptography.

Insufficient cryptography can also arise from not properly encoding certain sensitive data (i.e. physical domain). This lack of sufficient cryptography can occur *onA* an endpoint communication channel during transmission. This lack of sufficient cryptography can occur *onA* device without device based encryption (DBE) as DBE is a *featureOf* only

Android 5 [85]. Google has also issued a warning for pre-Android5 devices which have been upgraded, “Caution: Devices upgraded to Android 5.0 and then encrypted may be returned to an unencrypted state by factory data reset.” [85] DBE will be *depricated*In future versions of Android, perhaps due to performance constraints [86]. Google currently has posted, “Caution: Support for full-disk encryption is going away. If you're creating a new device, you should use file-based encryption.” [87] This lack of sufficient cryptography can occur *onA file without* file based encryption (FBE) [88]. Google has already issued OS version specific issues in relation to FBE. For example, the Android Application API currently reads, “Caution: On devices running Android 7.0-8.1, file-based encryption can't be used together with adoptable storage. On devices using FBE, new storage media (such as an SD card) must be used as traditional storage. Devices running Android 9 and higher can use adoptable storage and FBE.” [89]

Insufficient cryptography due to the implementation of a weak cryptography algorithm can *resultFrom* from three main issues. First, a weak initialization vector (IV) will increase the risk of the output cipher text to be easily decoded. Second, a weak cipher algorithm (e.g. 3DES [68]) are either going to be deprecated soon or known to already be non-secure and have already been deprecated by industry and the U.S. federal government. Third, a weak key (e.g. less than 128-bits, non-random, etc.) are also known susceptible to brute force attacks on today's technology [69]. These three predominate issues cause weak cipher text to be output by the cryptography algorithms therefore increasing the risk of information exposure and the loss of confidentiality of the information. Sergio Giro, a Google software engineer blogs [70] the following example from Nikolay Elenkov [71].

As can be seen in the table below, encryption that uses the Android API can be analyzed by employing static analysis techniques (e.g. context sensitive analysis, string analysis, variable propagation, etc.).

```

/* User types in their password: */
String password = "password";

/* Store these things on disk used to derive key later: */
int iterationCount = 1000;
int saltLength = 32; // bytes; should be the same size
    as the output (256 / 8 = 32)
int keyLength = 256; // 256-bits for AES-256, 128-bits for AES-128, etc
byte[] salt; // Should be of saltLength

/* When first creating the key, obtain a salt with this: */
SecureRandom random = new SecureRandom();
byte[] salt = new byte[saltLength];
random.nextBytes(salt);

/* Use this to derive the key from the password: */
KeySpec keySpec = new PBEKeySpec(password.toCharArray(), salt,
    iterationCount, keyLength);
SecretKeyFactory keyFactory = SecretKeyFactory
    .getInstance("PBKDF2WithHmacSHA1");
byte[] keyBytes = keyFactory.generateSecret(keySpec).getEncoded();
SecretKey key = new SecretKeySpec(keyBytes, "AES");

```

Table 26 Nikolay Elenkov Example [71] [72]

Weak keys are known to cause insufficient cryptography. Weak keys are the *resultFrom* at least three predominate causes. First, keys may not be stored correctly and therefore they are not properly guarded, perhaps by the Android KeyStore [73]. An examples of such a scenario are when the key is stored in a shared space next to the encrypted data. Second, the key derivation function may not be properly coded. In such a scenario, keys may not be randomly generated. Third, the key length changes with industry best practices based on computational power. In such cases, older code relying on shorter key lengths again increases the risk around real time brute force attacks [69].

In summary, we have identified at least two sub-areas where the static analysis of developer Android application source code can be examined for standard risky Android API calls which are OWASP Mobile Threat 5 is “Insufficient Cryptography” (IC). The threat of insufficient cryptography potentially results in at least the loss of confidentiality and integrity, which potentially lead to data compromise. The standard Android encryption API calls include creating keys, encrypting, and decrypting, are all detectable using static analysis.

4.2.6 Mobile 2016 Threat 6: Insecure Authorization

The OWASP 2016 Mobile Threat 6 is “Insecure Authorization” (M6_Insecure_Authorization), as seen in Figure 31. Insecure Authorization can potentially lead to data compromise of sensitive data or the unauthorized system or data access.

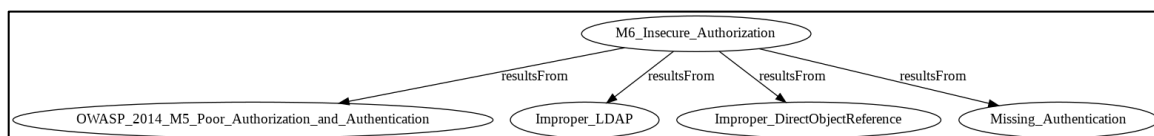


Figure 31: OWASP Mobile 2016 Threat 6: Insecure Authorization

According to the OWASP 2016 Mobile Threat categories [81], the 2016 threat *resultsFrom* either *OWASP_2014_M5_Poor_Authorization_and_Authentication* (described in Section 4.1.5), *Improper_LDAP*, *Improper_DirectObjectReference* or *Missing_Authentication*. These vulnerabilities are mainly implemented on the server-side of an application; however, static analysis on the client-side may expose the potential to test such scenarios.

4.2.7 Mobile 2016 Threat 7: Poor Code Quality

The OWASP 2016 Mobile Threat 7 is “Poor Code Quality” (M3_Poor_Code_Quality), as seen in Figure 32. Poor Code Quality can potentially lead to data compromise of sensitive data or the unauthorized system or data access.

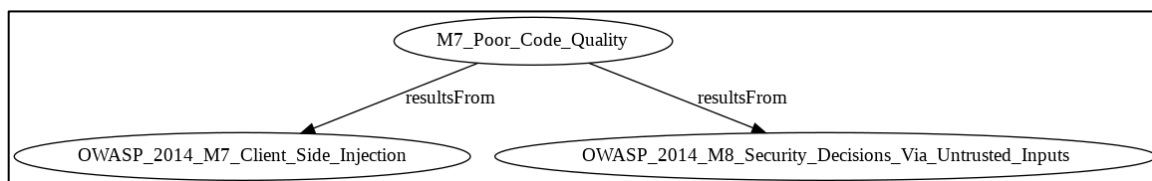


Figure 32: OWASP Mobile 2016 Threat 7: Poor Code Quality

According to the OWASP 2016 Mobile Threat categories [81], the 2016 threat resultsFrom either *OWASP_2014_M7_Client_Side_Injection* (Section 4.1.7) or *OWASP_2014_M8_Security_Decisions_Via_Untrusted_Inputs* (Section 4.1.8).

4.2.8 Mobile 2016 Threat 8: Code Tampering

The OWASP 2016 Mobile Threat 8 is “Code Tampering” (M8_Code_Tampering), as seen in Figure 33. Code Tampering vulnerabilities allow an attacker to modify code delivered and running on client devices. Such code tampering can potentially lead to the compromise of data, application or the entire underlying system resources.

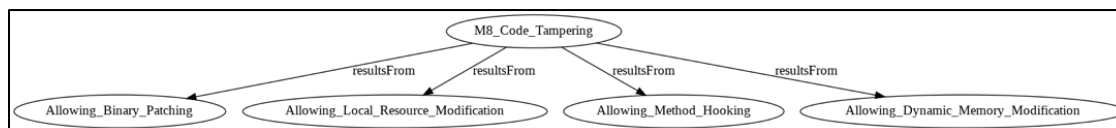


Figure 33: OWASP Mobile 2016 Threat 8: Code Tampering

The 2016 threat M8_Code_Tampering results from all of the following in Android: *Allowing_Binary_Patching*, *Allowing_Local_Resource_Modification*, *Allowing_Method_Hooking*, and *Allowing_Dynamic_Memory_Modification*. In binary patching the

application is completely modified and redeployed to the device [90]. Method hooking, allowed on some Android devices, completely redirect the control flow of Android applications are runtime [91].

4.2.9 Mobile 2016 Threat 9: Reverse Engineering

The OWASP 2016 Mobile Threat 9 is “Reverse Engineering” (M9_Reverse_Engineering), as seen in Figure 34. Reverse Engineering can potentially lead to data compromise of sensitive data or the unauthorized system or data access.

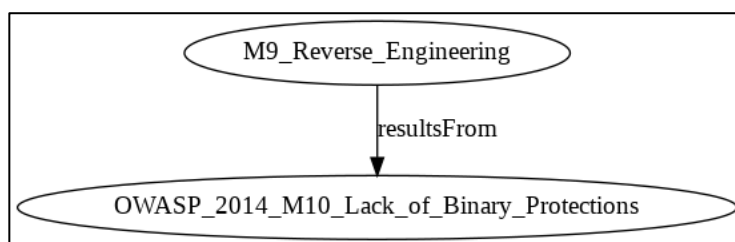


Figure 34: OWASP Mobile 2016 Threat 9: Reverse Engineering

According to the OWASP 2016 Mobile Threat categories [81], the 2016 threat *resultsFrom* *OWASP_2014_M10_Lack_of_Binary_Protections* (Section 4.1.10).

4.2.10 Mobile 2016 Threat 10: Extraneous Functionality

The OWASP 2016 Mobile Threat 10 is “Extraneous Functionality” (M3_Extraneous_Functionality), as seen in Figure 35. Extraneous Functionality can potentially lead to data compromise of sensitive data or the unauthorized system or data access. *M10_Extraneous_Functionality resultsFrom Developer_Debug_Errors* such as hidden backdoors which expose administrative dashboards or leave improperly set debug flags turned on within live production code.

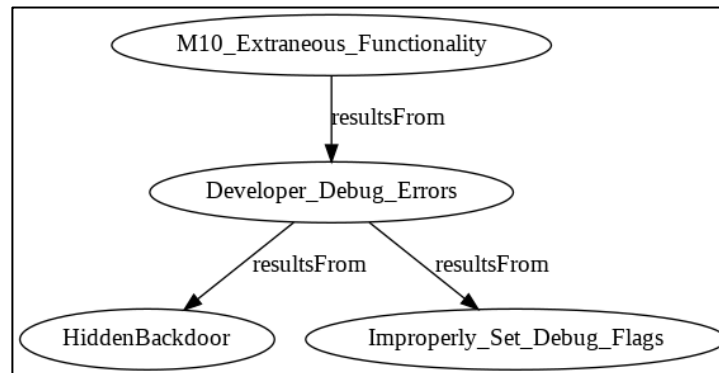


Figure 35: OWASP Mobile 2016 Threat 10: Extraneous Functionality

4.3 Summary of OWASP Threat/Risk Findings

In the Knowledge-Graphs, which we introduced for the OWASP 2014 and the OWASP 2016, we introduced five relationships (i.e. *hasThreat*, *hasA*, *onA*, *resultsFrom*, and *without*) for software assurances and re-applied the *isA* relationship from standard ontologies.

Table 27: Relations Introduces in OWASP Mobile Threat Knowledge-Graph

- *hasThreat* – Specific to certain current cybersecurity threats.
- *isA* – Standard ontology hierarchy
- *hasA* – An optional attribute which could cause further analysis.
- *onA* – The concern is relevant to certain constraints.
- *resultsFrom* – The cause of the concern under investigation.
- *without* – Missing a property related to the security.

With time these, these relationships will be kept constant to provide consistency; however, they may become deprecated as cybersecurity further evolve.

Chapter 5

Detectable Security Management Coding Patterns

In this chapter, we discuss how static analysis methods can contribute to certain kinds of unresearched security threats. Static analysis techniques are archetypal for object-oriented programming but must be enhanced to fit Android. An example archetypal static analysis is type checking, used ubiquitously in standard compilers [8], which dates back to IBMs mid-19th century Fortran and perhaps earlier. In this section, we examine two major OWASP security concerns given from Chapter 4: Mobile 2016 Threat 2: Insecure Data Storage (discussed in Section 4.2.2) and Mobile 2016 Threat 5 Insufficient Cryptography (discussed in Section 4.2.5). We examine the source code of 200+ Android Healthcare applications uploaded to GitHub ranging from full blown grant funded applications to smaller experimental prototype projects. A full listing of the analyzed applications can be seen in Appendix A.

5.1 Analysis of Android Health Source Code for OWASP Top Security Issues

To examine secure coding guidelines being employed utilizing the OWASP 2016 Mobile Threat and OWASP 2014 Mobile Threat knowledge-graph from Chapter 4, over 200 Android healthcare application source codes were reviewed for this research. Note that the full final list of applications analyzed will be copied up onto my GitHub account at www.GitHub.com/schmeelk/pace-dps-2020.

For analysis, we reviewed the 200+ application source code for OWASP 2016 Mobile Threat 2 “Insecure Data Storage” and OWASP 2016 Mobile Threat 5 “Insufficient Cryptography.” Our findings are presented in **Error! Reference source not found.** REF_Ref31450238 \h * MERGEFORMAT **Error! Reference source not found.** shows the following: (1) a row number, (2) the GitHub links to the analyzed Android healthcare application source code repositories, (3) the GitHub application description, (4) if secure coding artifacts are present for OWASP 2016 Mobile Threat 2 Insecure Data Storage discussed in Section 4.2.2 and (5) if secure coding artifacts are present for OWASP 2016 Mobile Threat 5 Insufficient Cryptography.

5.2 Analysis 1 Summary: OWASP 2016 Threat 5 Insufficient Cryptography

We analyzed the source code of 200+ Android healthcare applications hosted on GitHub if secure coding artifacts are present for OWASP 2016 Mobile Threat 5 Insufficient Cryptography based on our knowledge-graph presented in Section 4.2.5 on page 78.

Are results are presented in Table 28. We found with a string analysis that 17 of 203 applications actually employed the *javax.crypto* package from the Java API. The string analysis showed that eight additional applications may employ crypto as lower-level binary inclusions referenced the crypto package (e.g. jar files and other *dex* files). These eight applications are not in scope for this current Java source code level analysis research, but future research should examine the static analysis of binary/lower-level inclusions.

Table 28: Analysis 1 OWASP 2016 Mobile Threat 5 Insufficient Cryptography

#	GitHub Link	Git Hub Application Description	MT_2016_M5_Present_Source	MT_2016_M5_Present_binary_File_eg-jar-dex
1	https://github.com/YahyaOdeh/HealthWatcher	Android Application that can estimate Heart rate, Blood pressure, Respiration rate and Oxygen rate from only the camera of the mobile	X	
2	https://github.com/citiususc/calendula	[UNMAINTAINED] An Android assistant for personal medication management https://citius.usc.es/calendula/	X	
3	https://github.com/Flaque/quirk	A GPL Licensed Cognitive Behavioral Therapy app for iOS and Android https://quirk.fyi	X	
4	https://github.com/Qingbao/HealthCareStepCounter	A step counter on Android platform	X	
5	https://github.com/hashed/HealthCareApp	Health Care Management System in Android	X	
6	https://github.com/openmrs/openmrs-android-client-user-guide	User guide for OpenMRS Android Client	X	
7	https://github.com/krokyze/FitKit	Flutter plugin for reading health and fitness data. Wraps HealthKit on iOS and GoogleFit on Android.	X	
8	https://github.com/ZainMustafaaa/HealthCare-Scan-Nearby-Hospital-Locations	I developed this android application to help beginner developers to know how to use Google Maps API and how to conver...	X	
9	https://github.com/GaneshSinghPapola/rn-samsung-health	React Native package to access Samsung <i>Health</i> data using <i>samsung health kit android sdk</i>	X	

10	https://github.com/gojuukaze/healthgo	a android pedometer app (安卓计步器)	X	
11	https://github.com/brminnick/HealthClinic	An iOS & Android app built in Xamarin.Forms that parses images of food to give nutritional information. Leverages Azure's Cognitive Services. https://docs.microsoft.com/azure?WT.m...	X	
12	https://github.com/ahao/HealthSLife	HealthSLife--an Android APP	X	
13	https://github.com/DeveloperStudentClub-Udaipur/GymBuddy-Official	Android App for Health Zone Gym Udaipur	X	
14	https://github.com/systems/powerup-android	PowerUp is an educational choose-your-own-adventure game that utilizes a users uploaded curriculum to empower pre-adolescents to take charge of their reproductive health. This is the Android version of the game.	X	
15	https://github.com/christianb/iHealth	Mobile Health Support in Hospitals with Android, Arduino and a WebServer.	X	
16	https://github.com/scoote-dich/QuitSmoking	Android app to help smokers to quit smoking. Three fragments organized with tabs: overview, health and diary.	X	
17	https://github.com/medic/medic-android	A native Android container for Medic Mobile's Community Health Worker mobile application	X	

18	https://github.com/chiefg13/SkinHealthChecker	SkinHealthChecker App detects possible melanoma skin cancer using OpenCV and Android camera.		X
19	https://github.com/BaiyuY/AndroidAppPCLink	Android App connect with health measure devices and MySQL		X
20	https://github.com/rjbailey/mystatus	An Android app that provides self-management tools to users with chronic health conditions.		X
21	https://github.com/umaranis/health-book	An open source Android app for helping cancer patients to keep track of their medical history and condition.		X
22	https://github.com/mpatel136/LifePulse	Android Health App		X
23	https://github.com/tarsd/HealthGuru	Android App		X
24	https://github.com/steeppmountain/HealthSync	Android Application that displays data from Samsung S Health		X
25	https://github.com/neil007m/HealthApp	Android app that records a user's symptoms and various information about it.		X

5.3 Analysis 2 Summary: OWASP 2016 Threat 2 Insecure Data Storage

We analyzed the source code of 200+ Android healthcare applications hosted on GitHub to detect what, if any, secure coding artifacts are present for OWASP 2016 Mobile Threat 2 Insecure Data Storage based on our knowledge-graph presented in Section 4.2.2 on page 76 and recently published [92]. In all the healthcare applications, the data collected is considered sensitive or personal identifying information (PII) since it is also can be

attached to certain device specific identifiers such as IMEI or IMSI. The full list of analyzed applications is found in Appendix A.

First, our research found some case specific issues such as the application listed at <https://github.com/doneill123/HealthyHabitsProject/blob/master/app/src/main/AndroidManifest.xml>. This particular application both requests to be entirely installed on external storage (most likely for the removal of space constraint issues) and requests permission to write to external storage, as can be seen in Figure 36. The specific Google guidance on installing applications entirely on the external storage is seen in Figure 37.

Second, we found overall through a string analysis that 70 applications of 203 applications requested permissions to write to external storage, which is unsecured shared space among all the applications on a devices. Of these 70 applications only 9 applications employed either the Android or Oracle crypto packages indicating immediately that 64 applications will qualify for higher risk management methodologies as they are requesting permissions to store data on non-private shared spaces without considering the C and I in the CIA-triad.

```

Branch: master HealthyHabitsProject / app / src / main / AndroidManifest.xml Find file Copy path
doneill123 - google play service classpath updated c815bd6 on Mar 13, 2019
0 contributors
58 lines (53 sloc) 2.16 KB Raw Blame History
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     package="com.example.danieloneill.healthyhabits"
5     android:installLocation="preferExternal">
6
7     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
8     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
9     <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
10    <uses-permission android:name="android.permission.INTERNET" />
11

```

Figure 36: Application seeking permission to be entirely stored on external storage

To allow the system to install your application on the external storage, modify your manifest file to include the `android:installLocation` attribute in the `<manifest>` element, with a value of either `"preferExternal"` or `"auto"`. For example:

Figure 37: The Google documentation guidance for install location

Overall, we found that 99 of the 200+ Android source code applications potentially either directly store information in known vulnerable device shared spaces, or store information in vulnerable locations only if the device is rooted. Rooting a device comes with entirely different risk-levels as not too many end users choose to root their devices since it can potentially violate service plans, among other concerns. Risk related to rooting devices can further be explored during additional research.

Our analysis for IDS can be seen in Table 29. Column W indicates that the string analysis found that the applications requests to write external storage. Column S indicates that a string analysis of the application binary components (e.g. *dex* or *jar* files buried within the application) detected strings preferring external storage. Similarly, Column P indicates that a string analysis of the source code shows evidence that the source code manifest is

requesting the application be preferred to installed entirely on external storage, through the string *android:installLocation="preferExternal*. Once the application is itself installed on external storage, all collected application data is also stored on the external storage as well. Column X shows the applications which access *SharedPreferences*. Our string analysis showed that only one application of 200+ applications accesses *SharedPreferences*, but they were accessed with *Mode_PRIVATE* (e.g. *SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);*) which lowers the risks unless the device is rooted. Column D indicates the applications which access the built in Android SQLite database. Column B indicates the applications binary files may access the built in Android SQLite database. The database is by default private unless the device is rooted.

Table 29: Analysis 2 OWASP 2016 Mobile Threat 2 Insecure Data Storage (IDS)

#	GitHub Link	GitHub Description	W	S	P	X	D	B
2	https://github.com/citiususc/calendula	An Android assistant for personal medication management https://citius.usc.es/calendula/	W				D	
3	https://github.com/Flaque/quirk	A GPL Licensed Cognitive Behavioral Therapy app for iOS and Android https://quirk.fyi	W					
10	https://github.com/gojuukaze/healthgo	a android pedometer app (安卓计步器)	W					
16	https://github.com/scoute-dich/QuitSmoking	Android app to help smokers to quit smoking. Three fragments organized with tabs: overview, health and diary.	W				D	B
17	https://github.com/medic/medic-android	A native Android container for Medic Mobile's Community Health Worker mobile application	W					
19	https://github.com/chiefg13/SkinHealthChecker	SkinHealthChecker App detects possible melanoma skin cancer using OpenCV and Android camera.	W		P		D	B
37	https://github.com/BaiyuY/AndroidAppPCLink	Android App connect with health measure devices and MySQL	W		P			
58	https://github.com/rjbailey/mystatus	An Android app that provides self-management tools to users with chronic health conditions.	W				D	B
65	https://github.com/umaranis/health-book	An open source Android app for helping cancer patients to keep track of their medical history and condition.	W				D	B
20	https://github.com/Get-Siempo/siempo-android-app	Siempo Android Launcher - Smartphone interface for mental health and wellbeing http://getsiempo.com	W				D	
23	https://github.com/nutritionfactsorg/daily-dozen-android	Keep track of the foods that Dr. Greger recommends in his NYT's best-selling	W				D	


		book, How Not to Die with this Android app https://play.google.com/store/apps/de...						
27	https://github.com/bholagabbar/AurumHealthApp	An Android App for Rural Healthcare developed for the RTBI Hackathon Finals https://play.google.com/store/apps/de...	W					B
30	https://github.com/EyeSeeTea/malariapp	Android app to help with health center assessments (development repository)	W				D	
34	https://github.com/cqlzx/health-management	PHMS Android Application	W					
40	https://github.com/sages-health/sagesmobile-common	Android library common to sages-health mobile Android components	W					
45	https://github.com/kudrom/HealthWalk	Aplicación android de sistemas móviles	W				D	
47	https://github.com/Health4TheWorld/Health4TheWorld-android	Android App for Health4TheWorld	W				D	
52	https://github.com/McFlyWYF/HealthManager	Android课设---基于心脏病的健康管理	W				D	
56	https://github.com/ibisTime/xn-health-android	健康e购android	W				D	
59	https://github.com/cupcaketees/PocketFitness	An android application assisting in health and fitness goals and lifestyle	W					
66	https://github.com/psin007/HealthyMaternity	Android application to help rural pregnant women	W					
68	https://github.com/Ethanator/MobileHealth	Data are collected from Google Glass, Moto 360, and Android phone to offer a glimpse into the user's daily activity.	W					
73	https://github.com/norim13/rios-mais-ldso	Website and Android app for river health monitoring and maintenance	W					
74	https://github.com/chennanni/diabetes-control-app	an android app to manage users' health data	W					
77	https://github.com/CMPUT301F18T21/DoctorPlzSaveMe	An Android app for keeping track of health issues	W					

87	https://github.com/simonaMarkova/HealthQuest-android	android	W					
90	https://github.com/aiwac-health-group/HealthRobot	Android project	W				D	
96	https://github.com/mohamedelhadi123/HealthCare1	Framework Android	W				D	
103	https://github.com/ShizuoZ/RUPacer	A health android app using pedometer to count daily and weekly steps. Users can log in via Facebook Account and compete with friends in Leaderboard.	W				D	
109	https://github.com/TobiasReich/HealthTracker	Health Tracker app for Android	W				D	
111	https://github.com/BevoLEt/HealthCare_Application	HealthCare Android Application	W					
117	https://github.com/kitaice/HealthClassifier	android app with decision tree classifier	W					
118	https://github.com/sinanelveren/Smart-Healthband-Bilek-Partner-Bil396-Project	ESP32 based smart healthband and Android application project.	W				D	B
126	https://github.com/malkio/happyfit	an android health app http://maxmenthol.bitbucket.org	W				D	B
127	https://github.com/gameloser/Burning-Fat	Android Project - Health & Fitness	W				D	
133	https://github.com/KourdacheHoussam/HealthContentManager	Application Android de gestion de patients	W				D	B
134	https://github.com/swe-team-c/HealthCareApplication	An android application for health care	W					
136	https://github.com/carlyonmdsol/HealthVaultAndroidExample	Cleaned up Android Health Vault Example	W				D	B
142	https://github.com/CrystalRanita/BabyHelper	Health care tool using OpenCV on android platform	W					
143	https://github.com/timchenggu123/SaveMi	A health monitoring Android app. Winner of Waterloo EngHack 2019	W					

160	https://github.com/siddharthsujir/Health-Buddy	An Android application for health conscious android user	W					D	
161	https://github.com/SayeedAbid/HealthMonitorApp	A personal health monitoring system with android studio and java	W					D	
162	https://github.com/doneill123/HealthyHabitsProject	Mobile app on Android Studio for final year project	W	S					
166	https://github.com/charlesmastin/healthnotifier-android	HealthNotifier Android Client	W						
171	https://github.com/danielCantwell/Fit-Friend	Health & Exercise app for Android	W						B
175	https://github.com/serkansorman/LogMe-Android-App	Smart Health Band Android App https://logmewristband.github.io	W						
179	https://github.com/xiangxianzui/Health-Android-Client	Android client of "HEALTH" app	W					D	
180	https://github.com/Ramonrune/nhs-patient	NFC Health System Patient Android	W					D	B
183	https://github.com/yourSylvia/HealthAssistant	An Android APP for exercise reminder, user activities tracking report, exercise videos and health forum.	W					D	
189	https://github.com/AtifMahmud/HealthWatch	An Android to work in conjunction with a heart rate tracking wearable.	W						
194	https://github.com/MujtabaBinKhalid/lifeline	An android health , fitness application.	W						
196	https://github.com/apoorvsarang10/HealthilyApp	Android health related app which incorporates Fragments, Firebase Authentication, Firebase Firestore, Notifications and Accelerometer.	W						
197	https://github.com/aarjan/Android-apps	A set of android mobile applications	W			X			

202	https://github.com/NgJun/bHealth_Android_SouceCode	Android Code	W					
21	https://github.com/farhan071024/HealthCareApp	Health Care Android Application		S				
91	https://github.com/tarsd/HealthGuru	Android App			P		D	B
147	https://github.com/steepmountain/HealthSync	Android Application that displays data from Samsung S Health			P			B
192	https://github.com/neil007m/HealthApp	Android app that records a user's symptoms and various information about it.			P		D	B
1	https://github.com/YahyaOdeh/HealthWatcher	Android Application that can estimate Heart rate, Blood pressure, Respiration rate and Oxygen rate from only the camera of the mobile					D	
4	https://github.com/Qingbao/HealthCareStepCounter	A step counter on Android platform					D	B
70	https://github.com/mpatel136/LifePulse	Android Health App					D	
18	https://github.com/alexnanrick/health	Basic health app for Android					D	
26	https://github.com/MD4N1/Wireless-E-Health-Monitor	Wireless E-Health Monitor is monitoring medical sensors monitoring using Arduino Duemilanove or similar, USB Host Shield, USB Bluetooth dongle and medical sensors data from Arduino is sent to Wireless E-Health for Android Smartphone/Tablet through USB Bluetooth Dongle that attached in Arduino,					D	
36	https://github.com/vjitendra/PanHealth_Personal_Health_Records	developed by Neha (Android)					D	B
49	https://github.com/manueljeffin/MyHealth	Health Monitoring Android app					D	
55	https://github.com/gudigundla/PersonalHealthCheck	An Android app named Personal HealthCheck. It helps track all your personal					D	B

		medical needs like health care appointments (i.e. Doctor, Dentist, Physio, reoccurring blood work etc). Even recurring events like Prescription Taking reminders. (i.e. Heart medicine every day at 10:00 am, Cholesterol medicine Monday/Wednesday/ Friday 7:00 pm), tasks lik... https://play.google.com/store/apps/de...							
57	https://github.com/ankit1414/Fitvit	Fitvit is an android application focused on the health and fitness of the users.						D	
64	https://github.com/ruifeng2357/FitnessApp	This is an Android & iPhone app for own health state can record, analysis and share using mobile app						D	
71	https://github.com/justiceamoh/AsthmaGuard	Android App for Dartmouth COSC 169: Mobile Health						D	
82	https://github.com/shvmshukla/Healthify-NearByHospitals-	An android application which uses google map api and helps us to find nearby hospitals. Also, it displays detailed information about those hospitals(viz no of doctors,no of beds, contact no etc.)						D	
84	https://github.com/tr016/HealthOut	Android app that allows users to input health metrics; this data is used to compare and graph the user's progress toward's his/her goals.						D	
89	https://github.com/JANGYONGSEONG/healthNotes	android application						D	
94	https://github.com/w771854332/health_android	health_android						D	
98	https://github.com/Alphacoder221/HealthApp-AyurVeda	Android HealthApp						D	
99	https://github.com/KiraSensei13/HealthyGrill	HealthyGrill Android Mobile Application						D	

100	https://github.com/AnkitKiet/HealthCare	Android app with Firebase						D	
101	https://github.com/SRatna/HealthyNepali	Android health related application						D	
108	https://github.com/keelooy/HealthDiagnostics	Android app Patients data storage https://github.com/keelooy						D	
110	https://github.com/KieronMoorcroft/HealthMonitor	An Android Health Monitor App						D	
115	https://github.com/KuznetsovaAnastasiia/HealthyCafe	An <i>android</i> app for the cafe staff						D	
116	https://github.com/verma-ady/HealthLitmus	Android App for www.healthlitmus.com						D	
121	https://github.com/NiallMcCann96/Health-App	An Android Health App						D	
124	https://github.com/PabloPicassoft/MyMediCare	Android Health Measurement App						D	
132	https://github.com/woolver/CYBAWeight	android app for health						D	
138	https://github.com/neelmehta247/Hack4Health	RemindMe is an Android app that helps Alzheimer's patients multitask in their day to day life.						D	
141	https://github.com/chinnatan/Healthy	 Advance app android						D	
152	https://github.com/prabhnoor15/HealthFit	this is the android studio project for "Health Fit"						D	
163	https://github.com/Abdullah-Naveed/HealthChain-Android	Android App for Final Year Project - Health Chain						D	
167	https://github.com/zhning12/Health-Record	Android Individual Project.						D	
169	https://github.com/kenny0202/SimpleHealthPlan	Android App						D	

170	https://github.com/marcgilbert01/ContactsSimpleApp	Android test for "Babylon Health"						D	
187	https://github.com/SuciuCalin/Project_09_HealthyRoutineTracker	Habit Tracker App - Udacity Android Basics Nanodegree by Google						D	
198	https://github.com/nvrocks/MobiDoc	This is a health care android application which determines the disease suffered by the patient on the basis of symptoms entered by him/her.						D	
199	https://github.com/jnoga1996/healthy-eating	Android app for PUM18 course						D	
201	https://github.com/Nolthicha/Health_Care_For_Diabetes	Project Android Silpakorn University						D	
81	https://github.com/kevm66/4thYearProject_Happy	Happy - Mental Health Android App							B
43	https://github.com/engai/FitKit	An Android health app for CSE 110							B
107	https://github.com/rizwan95/HealthChilli	Android application for healthchilli.com							B
119	https://github.com/qianzch/NowSleep	It's time for bed! Now Sleep! [Android App]							B

Chapter 6

Conclusion

This research examined mobile application security through the lens of knowledge graph construction to inform static analysis. As people and organizations around the world are adopting applications written by third-parties at an unprecedented rate, mobile application security analyses are not keeping pace. This research contributes new security vulnerabilities for detection, contributes related mobile security knowledge-graphs and argues via source code review that certain static analysis methodologies can be employed to detect vulnerabilities. We conclude with future research potentials.

6.1 Contribution Summary

This dissertation focused on the software assurance of mobile applications. It presented four main contributions described in the following subsections.

6.1.1 Contribution #1: Android Mobile Application Knowledge Graph

Our research developed a preliminary mobile application security knowledge graph to inform security analysis. A knowledge graph is useful for showing longitudinal changes and inter-dependencies of software.

Based on a large literature review, to date The National Institute of Standards and Technology's Software Assurance Metrics And Tool Evaluation (SAMATE) project team has published multiple articles [10] [11] on the Bug Framework, seen below in Table 30. The Bug Framework is currently the only known open-source taxonomy, which is a subset

of a knowledge graph, aimed at informing application development but at lower levels. The Bug Framework has a different methodology in place. Our Mobile Application program code analysis knowledge graph examines security at a higher-level to further inform nuances in the program application control flow.

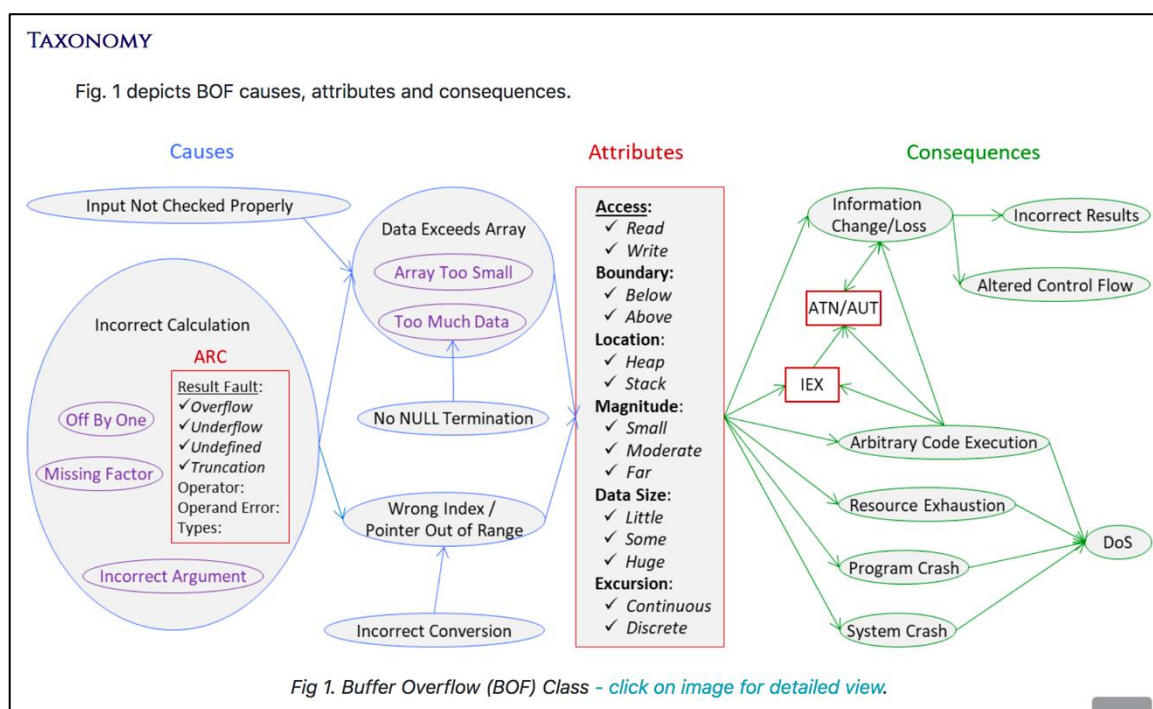


Table 30: NIST SAMATE BF - Buffer Overflow (BOF) [93]

6.1.1.1 New Static Analysis Program Code Security Detection Findings

Our research informs newer static analysis findings such as methodologies to detect when applications may be jeopardizing the confidentiality or integrity of data such as allowing copying and pasting of sensitive information in the clipboard or potentially leading into a shared SQLite database. Currently, little, if any, publications examine source code of all of the security findings brought forth in the knowledge-graph built based on the OWASP Top threats.

6.1.1.2 Analysis of Open Source Mobile Healthcare Applications from GitHub

This work examined 203 open source Android mobile healthcare applications from GitHub. Among the applications analyzed are applications which have won field-specific awards and been grant funded. In fact, some are even promoted by their respective fields at large. Of the applications, some of them measure mental health, body signals such as blood pressure and temperature, obstetrics and gynecology metrics, pictures, and all kinds of other health related data. Since these applications are billing customers, they currently are not under HIPAA; however, the data stored by these applications are in some cases very private. We found that very few applications, if any, being posted to the open source community in healthcare with sensitive data are considered all of the OWASP Mobile Threats applications loss of the confidentiality, integrity and availability of their services and sensitive data.

6.1.2 *Contribution #2: Evaluate Knowledge-Graph.*

Software development which includes software assurances improve security and costs from subsequently mitigating concerns at later states of software development. We showed how our developed OWASP Mobile Threats knowledge-graph can be employed to guide software development practices to identify both risky Android API calls and other risky coding patterns. These patterns can further be integrated into the knowledge graph in future research to enable a more robust risk assessment of software.

6.1.3 Contribution #3: Identify Unresearched Mobile Vulnerabilities

Static analysis research on mobile applications is adhoc and quite non-systematic. Four major domains of research exist from a utility perspective, as described by Schmeelk [14], Schmeelk [15], Schmeelk, Yang and Aho [16], and Schmeelk and Aho [17]. Schmeelk [14] and Schmeelk [15] showed both unsearched and immaturely researched known system vulnerabilities specifically in the Android mobile software ecosystem. The research shows the cybersecurity gaps which can be further investigated for mobile application risk mitigation.

6.1.4 Contribution #4: Analyze Software Assurances in 200+ Applications

We analyzed 200+ Android healthcare applications for the OWASP Top Ten Mobile Threat #2 Insufficient Cryptography and #5 Insecure Data Storage. We found that these applications, which can potentially store private information about people, store data without mitigations for these two particular threats. Future work may be in the following; (1) expanding the knowledge graph to include the specific technology-specific coding patterns and API calls and (2) adding more considerations to the analysis. Cybersecurity will become more valuable as laws are further developed to protect human data. With time, more elaborate analysis mechanisms will need to be developed to help prevent privacy and security breaches.

6.2 Future Research

The knowledge graph created in this research connecting the two most recent years of OWASP Top 10 Mobile Threats for mobile cybersecurity is one of the first public

knowledge graphs. Future research include crowd sourcing the knowledge-graph so that the world can continue to develop out vulnerabilities as they become known. In addition, this knowledge-graph can aid software assurance tool designs, security analysts, and penetration testers in their analysis of 1st party and 3rd party applications which are being integrated daily into facilities around the world (e.g. medical, schools, banks, etc.). Currently research does not adequately map vulnerabilities to a specific graphical representation of cybersecurity issues. The knowledge graph acts as a map guiding overall cybersecurity understandings. In addition, it can expand with time (i.e. longitudinally) to show the complexities and relationships of threats over time.

Other areas for future research are to develop a lower level code-specific knowledge-graph at either machine-code or byte-code levels to inform further program analysis. These type of knowledge graphs would be more similar to the NIST Bug Framework to inform all of the specific coding patterns which can lead to malware indicators and other higher-level Mobile Threats.

Future knowledge-graph construction can be implemented within program analysis tools, Application Stores (e.g. GooglePlay, iTunes, etc.) and Mobile Device Management platforms to detect potentially problematic code. Currently different software assurance tools may detect different threats; thus, unifying the threats in the form of a knowledge graph helps to have are more accurate understanding of the overall system cybersecurity. Our world is moving to mobile and application-dependent services, our security needs to follow these trends especially in regulated sectors.

Further software assurance static analysis methodologies (and other software assurance methodologies) can be built out for all the knowledge graphs created in this research. The knowledge graph can be thus employed to compare, contrast, and improve overall tool soundness and completeness.

The knowledge graphs can also be combined with weights and attributes to further expand the graphs for automated risk assessments and in development of machine learning features. For example, we could add risk measurements and other weights to graph edges, add attributes to graph edges, and map knowledge graphs to: NIST frameworks (e.g. Bug Framework, CAPEC, & NVD) and MITRE frameworks (e.g. CWE). Further risk insights could then be gained directly from the application software.

Finally, the agnostic knowledge-graph representations can be further built out to mobile features such as device *rooting* and to inform other code analysis such as mobile applications running on other platforms such as iPhone which is Swift dependent or even applications developed entirely with front-end languages such as JavaScript, Node.js, etc.

Appendix A

List of Android Healthcare Mobile Applications Analyzed

To examine secure coding guidelines being employed utilizing the OWASP 2016 Mobile Threat and OWASP 2014 Mobile Threat knowledge-graph from Chapter 4, over 200 Android healthcare application source codes were reviewed for this research. Note that the full final list of applications analyzed will be copied up onto my GitHub account a www.GitHub.com/schmeelk/pace-dps-2020.

Table 31: Full List of Android Healthcare Mobile Applications Analyzed

#	GitHub Link	Git Hub Application Description
1	https://github.com/YahyaOdeh/HealthWatcher	Android Application that can estimate Heart rate, Blood pressure, Respiration rate and Oxygen rate from only the camera of the mobile
2	https://github.com/citiususc/calendula	[UNMANTAINED] An Android assistant for personal medication management https://citius.usc.es/calendula/
3	https://github.com/Flaque/quirk	A GPL Licensed Cognitive Behavioral Therapy app for iOS and Android https://quirk.fyi
4	https://github.com/Qingbao/HealthCareStepCounter	A step counter on Android platform
5	https://github.com/hasyed/HealthCareApp	Health Care Management System in Android
6	https://github.com/openmrs/openmrs-android-client-user-guide	User guide for OpenMRS Android Client
7	https://github.com/krokyze/FitKit	Flutter plugin for reading health and fitness data. Wraps HealthKit on iOS and GoogleFit on Android.
8	https://github.com/ZainMustafaaa/HealthCare-Scan-Nearby-Hospital-Locations	I developed this android application to help beginner developers to know how to use Google Maps API and how to conver...

9	https://github.com/GaneshSinghPapola/rn-samsung-health	React Native package to access Samsung <i>Health</i> data using samsung <i>health</i> kit <i>android</i> sdk
10	https://github.com/gojuukaze/healthgo	a android pedometer app (安卓计步器)
11	https://github.com/brminnick/HealthClinic	An iOS & Android app built in Xamarin.Forms that parses images of food to give nutritional information. Leverages Azure's Cognitive Services. https://docs.microsoft.com/azure?WT.m...
12	https://github.com/aghao/HealthSLife	HealthSLife--an Android APP
13	https://github.com/DeveloperStudentClub-Udaipur/GymBuddy_Official	Android App for Health Zone Gym Udaipur
14	https://github.com/systers/powerup-android	PowerUp is an educational choose-your-own-adventure game that utilizes a users uploaded curriculum to empower pre-adolescents to take charge of their reproductive health. This is the Android version of the game.
15	https://github.com/christianb/iHealth	Mobile Health Support in Hospitals with Android, Arduino and a WebServer.
16	https://github.com/scoute-dich/QuitSmoking	Android app to help smokers to quit smoking. Three fragments organized with tabs: overview, health and diary.
17	https://github.com/medic/medic-android	A native Android container for Medic Mobile's Community Health Worker mobile application
18	https://github.com/alexnanrick/health	Basic health app for Android
19	https://github.com/chiefg13/SkinHealthChecker	SkinHealthChecker App detects possible melanoma skin cancer using OpenCV and Android camera.
20	https://github.com/Get-Siempo/siempo-android-app	Siempo Android Launcher - Smartphone interface for mental health and wellbeing http://getsiempo.com
21	https://github.com/farhan071024/HealthCareApp	Health Care Android Application

22	https://github.com/Rafagf/HealthTube	Android health application consisting of video channels which help users to keep a healthy life. An example of how to populate ListViews with YouTube videos playlists and reproduce them in a embedded player
23	https://github.com/nutritionfactsorg/daily-dozen-android	Keep track of the foods that Dr. Greger recommends in his NYT's best-selling book, How Not to Die with this Android app https://play.google.com/store/apps/de...
24	https://github.com/GreenRobots/JPHacks	Android × Health
25	https://github.com/webiansk/Crimson	An android application that can prove to be the ultimate care taker of your eyes. [Built during hackDTU event.]
26	https://github.com/MD4N1/Wireless-E-Health-Monitor	Wireless E-Health Monitor is monitoring medical sensors monitoring using Arduino Duemilanove or similar, USB Host Shield, USB Bluetooth dongle and medical sensors data from Arduino is sent to Wireless E-Health for Android Smartphone/Tablet through USB Bluetooth Dongle that attached in Arduino,
27	https://github.com/bholagabbar/AurumHealthApp	An Android App for Rural Healthcare developed for the RTBI Hackathon Finals https://play.google.com/store/apps/de...
28	https://github.com/strudelauxpomes/FitnessHabits	This Android software tracks fitness activities and health habits such as physical activities, fluids intake, supplements, food intake, sleep durations, and alcohol intake.
29	https://github.com/michaelcarrano/seven_minute_workout_android	An Android application that walks you through the "7 minute workout" that was mentioned in the May/June 2013 issue of ACSM's Health & Fitness Journal.
30	https://github.com/EyeSeeTea/malariapp	Android app to help with health center assessments (development repository)
31	https://github.com/viswesvar/HealthCare-App	Android Application - Diabetes Patients
32	https://github.com/stuardavis/HealthAura	an android app

33	https://github.com/drbohrinkman/HealthyMiamiWatchFace	Miami University themed Android Wear watch face, with step counter.
34	https://github.com/cqlzx/health-management	PHMS Android Application
35	https://github.com/Huangtuzhi/HealthCare	A health supervision device of blood pressure and pulse based on FriendlyARM2440 DB. It can also be manipulated by App on Android phone and send back monitoring data to display on the LCD of Smart Phone.
36	https://github.com/vjitendra/PanHealthPersonalHealthRecords	developed by Neha (Android)
37	https://github.com/BaiyuY/AndroidAppPCLink	Android App connect with health measure devices and MySQL
38	https://github.com/arunrajora/health-bait	an android app for health and fitness
39	https://github.com/liulinru13/Android_healthy_manager	android平台下的健康管理软件
40	https://github.com/sages-health/sagesmobile-common	Android library common to sages-health mobile Android components
41	https://github.com/GlobetrekkerChallenge/cordova-plugin-shealth	Samsung s-health cordova plugin (android only)
42	https://github.com/DevPro7/HealthyU	HealthyU Android application
43	https://github.com/engai/FitKit	An Android health app for CSE 110
44	https://github.com/lakindu95/Healthy-Office-plus	This project is based on a smart IOT product which is used to monitor the sitting pattern of office based employees. In order to detect the presence of the employees at the desk area, two sensors are permanently fixed beneath the work table of the employee. The values generated from these two sensors are logically "and" operated to activate the ... http://healthyoffice.tk

45	https://github.com/kudrom/HealthWalk	Aplicación android de sistemas móviles
46	https://github.com/kouh777/HealthStep	Add android and wear project
47	https://github.com/Health4TheWorld/Health4TheWorld-android	Android App for Health4TheWorld
48	https://github.com/vinitdeodhar/healthnet	A module in openMRS healthcare record management system. It allows patient or healthworker upload data from his or her android device to openMRS server
49	https://github.com/manueljeffin/MyHealth	Health Monitoring Android app
50	https://github.com/gpavlid/eHealth-Android	Android app created using MIT app inventor 2 http://georgepavlidis.info/ehealth-main/
51	https://github.com/brandonscott/pulse	An Android OS app for server health monitor http://www.getcadence.com
52	https://github.com/McFlyWYF/HealthManager	Android课设---基于心脏病的健康管理
53	https://github.com/karahbit/RU_Healthy	Android-based health monitoring software
54	https://github.com/zorosteven/FemaleHealth	a android app for female
55	https://github.com/gudigundla/PersonalHealthCheck	An Android app named Personal HealthCheck. It helps track all your personal medical needs like health care appointments (i.e. Doctor, Dentist, Physio, reoccurring blood work etc). Even recurring events like Prescription Taking reminders. (i.e. Heart medicine every day at 10:00 am, Cholesterol medicine Monday/Wednesday/ Friday 7:00 pm), tasks lik... https://play.google.com/store/apps/details?id=com.gudigundla.personalhealthcheck
56	https://github.com/ibisTime/xn-health-android	健康e购android
57	https://github.com/ankit1414/Fitvit	Fitvit is an android application focused on the health and fitness of the users.

58	https://github.com/rjbailey/mystatus	An Android app that provides self-management tools to users with chronic health conditions.
59	https://github.com/cupcaketees/PocketFitness	An android application assisting in health and fitness goals and lifestyle
60	https://github.com/aungkoman/mm_health	Myanmar Health Android App
61	https://github.com/uurcan/Healthy	Bachelor's graduation project. Health application for Android devices.
62	https://github.com/oah-health/oah-health-android-launcher	Launcher App for Digital Clinic for Android Phones.
63	https://github.com/AkshayMathur92/SyncHealth	SyncHealth is an Android app to move fitness data from Samsung Health to Google Fit.
64	https://github.com/ruifeng2357/FitnessApp	This is an Android & iPhone app for own health state can record, analysis and share using mobile app
65	https://github.com/umaranis/health-book	An open source Android app for helping cancer patients to keep track of their medical history and condition.
66	https://github.com/psin007/HealthyMaturity	Android application to help rural pregnant women
67	https://github.com/Ethanator/MobileHealth	Data are collected from Google Glass, Moto 360, and Android phone to offer a glimpse into the user's daily activity.
68	https://github.com/ogasimli/HealthBarView	Custom health bar like view for Android
69	https://github.com/mpatel136/LifePulse	Android Health App
70	https://github.com/justiceamoh/AsthmaGuard	Android App for Dartmouth COSC 169: Mobile Health
71	https://github.com/beto4812/AirQueue-Android	Air Quality Visualizations and health tracking android application.
72	https://github.com/norim13/rios-mais-ldso	Website and Android app for river health monitoring and maintenance
73	https://github.com/chennanni/diabetes-control-app	an android app to manage users' health data

74	https://github.com/m2y-team/HealthOwl	Android application for healthcare providers to manage patient data
75	https://github.com/medhelpintl/android_hapi_sdk	Medhelp's hAPI health data platform for Android
76	https://github.com/CMPUT301F18T21/DoctorPlzSaveMe	An Android app for keeping track of health issues
77	https://github.com/sydneymainga/Health	firebase android
78	https://github.com/Vktor93/health	android app
79	https://github.com/elijada/CapstoneHealth	Fitness Tracker Android App
80	https://github.com/kevm66/4thYearProject_Happy	Happy - Mental Health Android App
81	https://github.com/shvmshukla/Healthify-NearByHospitals-	An android application which uses google map api and helps us to find nearby hospitals. Also, it displays detailed information about those hospitals(viz no of doctors,no of beds, contact no etc.)
82	https://github.com/Maktm/TALK	Android based application for tracking your mental health
83	https://github.com/rr016/HealthOut	Android app that allows users to input health metrics; this data is used to compare and graph the user's progress toward's his/her goals.
84	https://github.com/SEUNAKINTOLA/HealthMonitor	HealthMonitor android app was built for a final year project that involves automatic monitoring of a patient's health, hereby bringing mobile intelligence to health system
85	https://github.com/alagrin/4tastic-health	Android app leveraging the Knox SDK as well as internal Android APIs to gather data for medical electronic visit verification
86	https://github.com/simonaMarkova/HealthQuest-android	android
87	https://github.com/brunocascio/nodequery-android	An unofficial android native app for https://nodequery.com/

88	https://github.com/JANGYONGSEONG/healthNotes	android application
89	https://github.com/aiwac-health-group/HealthRobot	Android project
90	https://github.com/tarsd/HealthGuru	Android App
91	https://github.com/priv/app-android-iHealth	Android app that connects iHealth devices' data to your Priv
92	https://github.com/sha3rawi33/Sehhetna	Am writing about "OUR" achievement of winning the 2nd place in "E-sus Mobile Applications Competition" in which, competitors creates Mobile applications and research ideas for solving the "SDGs" sustainable development goals of the UN. Our team's submitted application was named after me "Healthy Wealthy"; we worked on the Public Health. The appl...
93	https://github.com/w771854332/health_android	health_android
94	https://github.com/pauloXtr3m/DroidHealth	Android application made in order to help people being healthy following simple advices.
95	https://github.com/mohamedelhadi123/HealthCare1	Framework Android
96	https://github.com/Merdzhanov/HealthyApp	Android calorie counter
97	https://github.com/Alphacoder221/HealthApp-AyurVeda	Android HealthApp
98	https://github.com/KiraSensei13/HealthyGrill	HealthyGrill Android Mobile Application
99	https://github.com/AnkitKiet/HealthCare	Android app with Firebase
100	https://github.com/SRatna/HealthyNepali	Android health related application
101	https://github.com/qinzhig/HealthManager	Android Application for Healthcare

102	https://github.com/ShizuoZ/RUPacer	A health android app using pedometer to count daily and weekly steps. Users can log in via Facebook Account and compete with friends in Leaderboard.
103	https://github.com/DebMaster/health	PHP and Android application that tailors a diet according to user attributes. Has some basic machine learning in there using phpML library.
104	https://github.com/mahadsiyaasi/Health-	This android health app
105	https://github.com/rupalmartin/Health	This Android application calculate Body Mass Index(BMI) based on the height and weight entered by the user. Also it counts the steps taken using sensors.
106	https://github.com/rizwan95/HealthChilli	Android application for healthchilli.com
107	https://github.com/kelooy/HealthDiagnostics	Android app Patients data storage https://github.com/kelooy
108	https://github.com/TobiasReich/HealthTracker	Health Tracker app for Android
109	https://github.com/KieronMoorcroft/HealthMonitor	An Android Health Monitor App
110	https://github.com/BevoLEt/HealthCareApplication	HealthCare Android Application
111	https://github.com/AndrewsJava/HealthFoodConcepts	Interactive Nutritional Android App
112	https://github.com/AndreeaMihalceaN/HealthMonitoring-android	HealthMonitoring-android repository
113	https://github.com/ay3524/HealthPrediction	An android app based on Health Prediction.
114	https://github.com/KuznetsovaAnastasiia/HealthyCafe	An <i>android</i> app for the cafe staff
115	https://github.com/verma-ady/HealthLitmus	Android App for www.healthlitmus.com
116	https://github.com/kitaice/HealthClassifier	android app with decision tree classifier

117	https://github.com/sinanelveren/Smart-Healthband-Bilek-Partner-Bil396-Project	ESP32 based smart healthband and Android application project.
118	https://github.com/qianzch/NowSleep	It's time for bed! Now Sleep! [Android App]
119	https://github.com/tymiles003/ivanlorcaapp	Health & Fitness Android App
120	https://github.com/NiallMcCann96/Health-App	An Android Health App
121	https://github.com/smithevandouglas/Salutem	Health tracking Android application
122	https://github.com/SuitGuy/Health_Application	A Prototype android application to test the HCI considerations for a tele medicine application as part of a software design study course.
123	https://github.com/PabloPicassoft/MyMediCare	Android Health Measurement App
124	https://github.com/ayushkumar0630/VitrixMobileApplication	Vitrix Health Android Application
125	https://github.com/malkio/happyfit	an android health app http://maxmenthol.bitbucket.org
126	https://github.com/gameloser/Burning-Fat	Android Project - Health & Fitness
127	https://github.com/RazorWire13/health-tracker	Health Tracker (Android Mobile)
128	https://github.com/vivi1393/PriorityAndroid	Proyecto Android Priority Health
129	https://github.com/kyang99/healthapp	An Android Health App.
130	https://github.com/Abhishek-karmakar/healthbuzz	Health buzz android app
131	https://github.com/woolver/CYBAWeight	android app for health

132	https://github.com/KourdacheHoussam/HealthContentManager	Application Android de gestion de patients
133	https://github.com/swe-team-c/HealthCareApplication	An android application for health care
134	https://github.com/DiszaJatnika/asyifahealth	Aplikasi Android Asyifa Health
135	https://github.com/carlyonmdsol/HealthVaultAndroidExample	Cleaned up Android Health Vault Example
136	https://github.com/Epamek/HealthyLiving2	Heartrate Monitoring through Android over Bluetooth LE
137	https://github.com/neelmehta247/Hack4Health	RemindMe is an Android app that helps Alzheimer's patients multitask in their day to day life.
138	https://github.com/davitterar/Healthiera	Healthiera android app
139	https://github.com/bitgeeky/healthservice	Android application for Appathon 😂
140	https://github.com/chinnatan/Healthy	Advance app android
141	https://github.com/CrystalRanita/BabyHelper	Health care tool using OpenCV on android platform
142	https://github.com/timchenggu123/SaveMi	A health monitoring Android app. Winner of Waterloo EngHack 2019
143	https://github.com/thomasfn/HealthApp	Android health app for Mobile Application Development coursework 2015.
144	https://github.com/dasilvabalautaro/HealthCheckpoint	Android- Get data of sensors and send to other mobile.
145	https://github.com/Epamek/HealthyLiving	Heart rate monitor for Android using Bluetooth Low Energy
146	https://github.com/steepmountain/HealthSync	Android Application that displays data from Samsung S Health
147	https://github.com/mattdhlee/HealthFacts	An android app that provides users with random health facts.

148	https://github.com/custord/HealthList	[Android] Front-end Application to Browse Nearby Health Care Specialists.
149	https://github.com/adeepbiswas18/HealthyHands	Android app for germs detection in hands using image processing.
150	https://github.com/kurtlewis/know-yourself	DerbyHacks3 Project - An android app for tracking your mental health
151	https://github.com/prabhnoor15/HealthFit	this is the android studio project for "Health Fit"
152	https://github.com/wwwhackcom/HealthCrossplatform	Health and wellness Crossplatform app(Android and iOS) using Xamarin
153	https://github.com/jerryliu3/HealthTracker	https://github.com/jerryliu3/HealthTracker
154	https://github.com/mmarose14/HealthPairs	Android app for looking up Healthy Food combinations https://mattsapps.mobi/health-pairs/
155	https://github.com/jeffborda/health-tracker	Android App
156	https://github.com/DarrinHowell/health-tracker	android app
157	https://github.com/pedfernandes/CarHealth	Aplicativo Android
158	https://github.com/christophergunadi/HealthHack18_Calorie_Camera	Android app designed for Health Hack 2018.
159	https://github.com/siddharthsujir/HealthBuddy	An Android application for health conscious android user
160	https://github.com/SayeedAbid/HealthMonitorApp	A personal health monitoring system with android studio and java
161	https://github.com/doneill123/HealthyHabitsProject	Mobile app on Android Studio for final year project
162	https://github.com/Abdullah-Naveed/HealthChain-Android	Android App for Final Year Project - Health Chain
163	https://github.com/santosh719/HealthMonitoringByGraph	Health monitoring and relevant data storage android app

164	https://github.com/Tianming8585/iHealth	iHealth Android Firebase
165	https://github.com/charlesmastin/healthnotifier-android	HealthNotifier Android Client
166	https://github.com/zhning12/Health-Record	Android Individual Project.
167	https://github.com/Ajayclamour/MyHealthyHost_Android	MyHealthyHost_Android
168	https://github.com/kenny0202/SimpleHealthPlan	Android App
169	https://github.com/marcgilbert01/ContactsSimpleApp	Android test for "Babylon Health"
170	https://github.com/danielCantwell/Fit-Friend	Health & Exercise app for Android
171	https://github.com/samikshay2/HHH-Healthy-Heart-Helper	Android App
172	https://github.com/RavneetDTU/Health-Startup	Android App for Health Startup
173	https://github.com/Touhid7051/BMI-Health-Monitor-Android-app	BMI Health Monitor Android app
174	https://github.com/serkansorman/LogMe-Android-App	Smart Health Band Android App https://logmewristband.github.io
175	https://github.com/AndreasHalim/Afiat-App	Health tracker run on Android
176	https://github.com/ylrodriguez/HealthcareTracking	Aplicación móvil en Android para el control de medicinas y citas para el adulto mayor.
177	https://github.com/kkhwee/hci-assignment	CPR Health Buddy Android Application
178	https://github.com/xiangxianzui/Health-Android-Client	Android client of "HEALTH" app
179	https://github.com/Ramonrune/nhs-patient	NFC Health System Patient Android

180	https://github.com/StevenElberger/HealthMate	An android application that tracks user's health information and uploads it to a database.
181	https://github.com/pratikg11/HealthManager	This is an android App to Calculate Daily Calorie Intake and many other functionalities
182	https://github.com/yourSylvia/HealthAssistant	An Android APP for exercise reminder, user activities tracking report, exercise videos and health forum.
183	https://github.com/ahmetkilinc/FitnessForEnergy	An android app for health https://play.google.com/store/apps/de...
184	https://github.com/cagdasbilecen/health-tracker-app	health tracker app for android
185	https://github.com/samdkershaw/HealthBuddy	An Android application that integrates with the FatSecret platform to provide health guides to end users.
186	https://github.com/SuciuCalin/Project_09_HealthyRoutineTracker	Habit Tracker App - Udacity Android Basics Nanodegree by Google
187	https://github.com/rieekan/QualitativeHealthSystems	Qualitative Health Systems - Android Client
188	https://github.com/AtifMahmud/HealthWatch	An Android to work in conjunction with a heart rate tracking wearable.
189	https://github.com/jocelindarma/HealthNavigator	A fully functioning Android app that allows people to request and give personal hygiene products.
190	https://github.com/sappybawa/MedicationInformationSystem	Hacking Health, Android app UI
191	https://github.com/neil007m/HealthApp	Android app that records a user's symptoms and various information about it.
192	https://github.com/MV1998/HealthStride	An Android Application for health stride which was made during industrial training.
193	https://github.com/MujtabaBinKhalid/lifeline	An android health , fitness application.
194	https://github.com/tanyag330/HealthTotal	This is an Android App used to maintain proper health and diet.

195	https://github.com/apoorvsarang10/HealthilyApp	Android health related app which incorporates Fragments, Firebase Authentication, Firebase Firestore, Notifications and Accelerometer.
196	https://github.com/aarjan/Android-apps	A set of android mobile applications
197	https://github.com/nvrocks/MobiDoc	This is a health care android application which determines the disease suffered by the patient on the basis of symptoms entered by him/her.
198	https://github.com/jnoga1996/healthy-eating	Android app for PUM18 course
199	https://github.com/dimgiatz/healthevents-android-project	University Android Project
200	https://github.com/Nolthicha/Health_Care_For_Diabetes	Project Android Silpakorn University
201	https://github.com/NgJun/bHealth_Android_SouceCode	Android Code
202	https://github.com/SajalGupta/PlanYourHealth	PlanYourHealth Android Application
203	https://github.com/itzatom/HealthSystemApp	Companion App for HealthSystem. Exame project for Mobile Devices. First Java/Android application.

Appendix B

Glossary of Terms

Android Application Package (APK) Files – Applications developed to run on the Google Android platforms.

Android Runtime Environment (ART) – A more modern ahead-of-time (AOH) version of the Google Android runtime platform.

Application Programming Interface (API) – A set of functions and procedures providing the development of applications to access data and operating system features.

Bring Your Own Device (BYOD) - An organizational policy which allows personnel to use their own personal device rather than an official organizational device.

Bug Framework (BF) – A framework that organizes software weaknesses, or bugs, into distinct classes.

Buffer Overflow (BOF) - The software accesses through an array memory location that is outside the boundaries of that array [11].

Common Attack Pattern Enumeration and Classification (CAPEC) - dictionary of known patterns of attack to exploit known weaknesses in technologies.

Common Weakness Enumeration (CWE) – a community-developed common language list of common software and hardware security weaknesses.

Data Loss Prevention (DLP) Technologies – monitors networks and systems to try to detect and prevent potential data breaches and ex-filtration transmissions.

Dalvik Virtual Machine (DVM) – An earlier just-in-time (JIT) version of the Google Android runtime platform.

Encryption/Decryption Bugs (ENC) – Defined by [11] as, “Encryption Bugs: The software does not properly transform sensitive data (plaintext) into unintelligible form (ciphertext) using a cryptographic algorithm and key(s). Decryption Bugs: The software does not properly transform ciphertext into plaintext using a cryptographic algorithm and key(s).”

Health Insurance Portability and Accountability Act (HIPAA) – A federal healthcare law enacted by the 104th United States Congress in 1996 with the primary motivation to modernize the flow of healthcare information.

Inter Process Communications (IPC) – An operating system process mechanism to allow communication and resource synchronization between process.

Java Virtual Machine (JVM) - A virtual machine that enables the execution of Java programs and other languages compiled down to Java bytecode.

Key Management Bugs (KMN) – Defined by [11] as, “The software does not properly generate, store, distribute, use, or destroy cryptographic keys and other keying material.”

Mobile Device Management (MDM) – a technology for the administration of mobile devices to lower organizational risks.

National Institute of Standards and Technology (NIST) – A United States’ physical science laboratory under the U.S. Department of Commerce founded in 1901 to provide a country-wide infrastructure for measurements through standards.

National Vulnerability Database (NVD) – the U.S. government repository of standards-based vulnerability management data.

OAuth Tokens – Authentication tokens following an open standard for access delegation typically employed after the initial system authentication.

Open Web Application Security Project (OWASP) - a nonprofit foundation that works to improve the security of software through community-driven projects (e.g. tools and resources), trainings, and conferences related to cybersecurity.

Penetration Testing - authorized simulated cyberattack on a technology, performed to evaluate the technologies’ security.

Resource Description Framework Schema (RDFS) - Originally produced by the RDF Schema Working Group (1997-2000) as a set of classes with certain properties for the description of ontologies.

Risk Assessment - a process to identify potential threats to weaknesses within a domain for the purpose of analyzing the likelihood and impact to the domain.

Semantic Web - an extension of the “current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. [94]”

Software Assurance Metrics And Tool Evaluation (SAMATE) - The NIST Software Assurance Metrics And Tool Evaluation (SAMATE) project is “dedicated to improving software assurance by developing methods to enable software tool evaluations, measuring the effectiveness of tools and techniques, and identifying gaps in tools and methods. The scope of the SAMATE project is broad: ranging from operating systems to firewalls, SCADA to web applications, source code security analyzers to correct-by-construction methods. [95]”

United States Computer Emergency Readiness Team (US-CERT) - An organization within the Department of Homeland Security’s (DHS) Cyber Security and Infrastructure Security Agency (CISA) responsible for “analyzing and reducing cyber threats, vulnerabilities, disseminating cyber threat warning information, and coordinating incident response activities. [96]”

Unique User Identifier (UID) - A reference associated with a single entity in a system.

Verification Bugs (VRF) - Defined by [11] as, “The software does not properly sign data, check and prove source, or assure data is not altered.”

Web Ontology Language (OWL) - a Semantic Web language designed to represent detailed and complex knowledge about things and their groups and relationships.

Reference

- [1] D. Allemang and J. Hendler, *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*, Morgan Kaufmann Publishers Inc., 2011, p. 384.
- [2] L. Yu, *A Developer's Guide to the Semantic Web*, Heidelberg: Springer, 2015.
- [3] L. Yu, *A Developers Guide to the Semantic Web*, Springer Publishing Company, Incorporated, 2011, p. 608.
- [4] Ö. Kafali, J. Jones, M. Petruso, L. Williams and M. P. Singh, "How good is a security policy against real breaches?: a HIPAA case study," in *Proceedings of the 39th International Conference on Software Engineering*, Buenos Aires, Argentina, 2017.
- [5] A. Elçi, "Isn't the Time Ripe for a Standard Ontology on Security of Information and Networks?," in *Proceedings of the 7th International Conference on Security of Information and Networks*, Glasgow, Scotland, UK, 2014.
- [6] C. Blackwell, "A security ontology for incident analysis," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, Oak Ridge, Tennessee, USA, 2010.
- [7] J. Howard and T. Longstaff, *A common language for computer security incidents*, Albuquerque, NM: Sandia National Laboratories, 1998.
- [8] A. V. Aho, M. S. Lam, R. Sethi and J. D. Ullman, *Compilers: Principles, Techniques, & Tools with Gradience*, USA: Addison-Wesley Publishing Company, 2007.
- [9] National Institute of Standards and Technology (NIST), "National Vulnerability Database (NVD)," 10 July 2019. [Online]. Available: <https://web.nvd.nist.gov>.
- [10] I. Bojanova, P. Black and Y. Yesha, "Cryptography classes in bugs framework (bf): Encryption bugs (enc), verification bugs (vrf), and key management bugs (kmn)," in *In 2017 IEEE 28th Annual Software Technology Conference (STC)*, Gaithersberg, Maryland, 2017.
- [11] I. Bojanova, P. Black, Y. Yesha and Y. Wu, "The bugs framework (bf): A structured approach to express bugs," in *In 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2016.
- [12] MITRE, "Common Weakness Enumeration (cwe).," 10 July 2019. [Online]. Available: <https://cwe.mitre.org>.

- [13] MITRE, "CAPEC View: Mechanisms of attack," 10 July 2019. [Online]. Available: <https://capec.mitre.org/data/definitions/1000.html>.
- [14] S. Schmeelk, "Where are we looking? Understanding android static analysis techniques," in *In 2019 IEEE International Conference on Services Computing*, Milan, Italy, 2019.
- [15] S. Schmeelk, "Where are we looking for security concerns? Understanding Android Security Static Analysis," in *Proceedings of the Future Technologies Conference (FTC) 2019*, San Fransisco, CA, 2019.
- [16] S. Schmeelk, J. Yang and A. Aho, "Android malware static analysis techniques," in *In Proceedings of the 10th Annual Cyber and Information Security Research Conference CISR '15*, New York, NY, USA, 2015.
- [17] S. Schmeelk and A. Aho, "Defending android applications availability," in *2017 IEEE 28th Annual Software Technology Conference (STC)*, Gaithersburg, MD, 2017.
- [18] M. Bishop, Addison-Wesley Professional, Introduction to Computer Security, 2004.
- [19] A. Whitman and M. Mattord, Roadmap to Information Security: For IT and Infosec Managers, 1st edition, Delmar Learning, 2011.
- [20] N. Daswani, C. Kern and A. Kesavan, Foundations of Security: What Every Programmer Needs to Know, Berkely, CA, USA: Apress, 2007.
- [21] S. M. Bellovin, "The puzzle of privacy," *IEEE Security and Privacy*, p. 6(5):88–88, 2008.
- [22] S. M. Bellovin., "Identity and security.," *IEEE Security and Privacy*, p. 8(2):88–88, March 2010.
- [23] E. Androulaki and S. M. Bellovin, "A secure and privacy-preserving targeted ad-system.," in *In Proceedings of the 14th International Conference on Financial Cryptograpy and Data Security, FC'10*, Berlin, 2010.
- [24] M. Bishop, Introduction to Computer Security., Addison-Wesley Professional, 2004.
- [25] A. Bessey, K. Block, B. Cheif, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak and D. Engler, "A few billion lines of code later: Using static analysis to find bugs in the real world.," *Commun. ACM*, p. 53(2):66–75, Feb 2010.
- [26] Hewlett-Packard Development Company, "HP Fortify," 10 July 2019. [Online]. Available: <http://www8.hp.com/us/en/software-solutions/application-security/index.html>.

- [27] Rogue Wave Software, "Klocworkr," 10 July 2019. [Online]. Available: <http://www.klocwork.com>.
- [28] M. A. Musen, "The protégé project: a look back and a look forward," *AI Matters*, p. 4–12, 2015.
- [29] A. Goknil and Y. Topaloglu, "Ontological perspective in metamodeling for model transformations," in *In Proceedings of the 2005 symposia on Metainformatics (MIS '05)*, New York, NY, USA, 2005.
- [30] N. Noy and D. McGuinness, "Ontology development 101: A guide to creating your first ontology.," Technical report at Stanford Knowledge Systems Laboratory, Palo Alto, CA, USA, 2001.
- [31] L. W. Lacy, *OWL: Representing Information Using the Web Ontology Language*, Victoria, BC, Canada: Trafford, 2005.
- [32] I. Patel, I. Dube, L. Tao and N. Jiang, "Extending OWL to Support Custom Relations," in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, New York, NY, USA, 2015.
- [33] Google, "Configure an Android Device," 11 10 2019. [Online]. Available: <https://source.android.com/devices/tech>.
- [34] Wikipedia, "Android Runtime," 12 10 2019. [Online]. Available: https://en.wikipedia.org/wiki/Android_Runtime.
- [35] Oracle, "Java™ Virtual Machine Technology," 12 October 2019. [Online]. Available: <http://docs.oracle.com/javase/7/docs/technotes/guides/vm/>.
- [36] W. Enck, D. Outeau, P. McDaniel and S. Chaudhuri, "A study of android application security," in *In Proceedings of the 20th USENIX Conference on Security, SEC'11*, Berkeley, CA, USA, 2011.
- [37] Google, "Dalvik Executable format," 12 October 2019. [Online]. Available: <https://source.android.com/devices/tech/dalvik/dex-format>.
- [38] C. Mann and A. Starostin, "A framework for static detection of privacy leaks in android applications," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12*, New York, NY, USA, 2012.
- [39] Oracle, "Chapter 7. Opcode Mnemonics by Opcode," 11 October 2019. [Online]. Available: <https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-7.html>.

- [40] Google, "About the Android Open Source Project," 12 October 2019. [Online]. Available: <https://source.android.com/>.
- [41] Google, "Android Architecture," 10 10 2019. [Online]. Available: <https://source.android.com/devices/architecture>.
- [42] Google, "Applicaiton Sandbox," 11 October 2019. [Online]. Available: <https://source.android.com/security/app-sandbox>.
- [43] "Android decompiled process and tools," 11 October 2019. [Online]. Available: <http://www.yelbee.top/index.php/archives/105/>.
- [44] M. Gargenta and N. M., Learning Android: Develop Mobile Apps Using Java and Eclipse, O'Reilly Media, Inc., 2014.
- [45] Google, "Understand the Activity Lifecycle," 12 October 2019. [Online]. Available: <https://developer.android.com/guide/components/activities/activity-lifecycle>.
- [46] Google, "Cipher," 1 April 2020. [Online]. Available: <https://developer.android.com/reference/javax/crypto/Cipher>.
- [47] OWASP, "OWASP Mobile Top 10," 25 March 2020. [Online]. Available: <https://owasp.org/www-project-mobile-top-10/>.
- [48] OWASP, "Mobile Top 10 2016-M2-Insecure Data Storage," 2 May 2018. [Online]. Available: https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage.
- [49] Google, "Data and file storage overview.," 29 November 2018. [Online]. Available: <https://developer.android.com/guide/topics/data/data-storage#db>.
- [50] Google, "Security Tips," 10 December 2018. [Online]. Available: <https://developer.android.com/training/articles/security-tips>.
- [51] Google, "Context.," 15 December 2018. [Online]. Available: [https://developer.android.com/reference/android/content/Context#openFileOutput\(java.lang.String,%20int\)](https://developer.android.com/reference/android/content/Context#openFileOutput(java.lang.String,%20int)).
- [52] A. Rajab, "How to prevent database and shared preferences from being hacked," 09 November 2017. [Online]. Available: <https://stackoverflow.com/questions/47207420/how-to-prevent-database-and-shared-preferences-from-being-hacked>.

- [53] Google, "Save key-value data," 2020. [Online]. Available: <https://developer.android.com/training/data-storage/shared-preferences#java>.
- [54] Google, "Save data using SQLite.," 12 December 2018. [Online]. Available: <https://developer.android.com/training/data-storage/sqlite>.
- [55] Google, "Save data using SQLite," 2020. [Online]. Available: <https://developer.android.com/training/data-storage/sqlite#java>.
- [56] Google, "Context," 2020. [Online]. Available: <https://developer.android.com/reference/android/content/Context>.
- [57] Google, "Save files on device storage.," 10 November 2018. [Online]. Available: <https://developer.android.com/training/data-storage/files#java>.
- [58] Google, "Security with HTTPS and SSL," 27 May 2019. [Online]. Available: <https://developer.android.com/training/articles/security-ssl>.
- [59] M. Dolan, "Android Security: SSL Pinning," 13 January 2017. [Online]. Available: <https://medium.com/@appmattus/android-security-ssl-pinning-1db8acb6621e>.
- [60] A. Wagoner, "Best keyboard apps for Android in 2019," 10 April 2019. [Online]. Available: <https://www.androidcentral.com/best-keyboard-android>.
- [61] A. Sinicki, "Let's build a custom keyboard for Android," 27 January 2018. [Online]. Available: <https://www.androidauthority.com/lets-build-custom-keyboard-android-832362/>.
- [62] Worldbestlearningcenter.com, "Android cache image from url," 27 May 2019. [Online]. Available: <https://www.worldbestlearningcenter.com/tips/Android-cache-image-from-url.htm>.
- [63] Google, "App security best practices," 27 May 2019. [Online]. Available: <https://developer.android.com/topic/security/best-practices>.
- [64] Google, "Copy and Paste," 27 May 2019. [Online]. Available: <https://developer.android.com/guide/topics/text/copy-paste>.
- [65] V. Sutariya, "How to programmatically take a screenshot on Android?," 21 August 2014. [Online]. Available: <https://stackoverflow.com/questions/2661536/how-to-programmatically-take-a-screenshot-on-android>.
- [66] Google, "Authenticate to OAuth2 services," 27 May 2019. [Online]. Available: <https://developer.android.com/training/id-auth/authenticate>.

- [67] Google, "Smart Lock for Passwords on Android," 27 May 2019. [Online]. Available: <https://developers.google.com/identity/smartlock-passwords/android/>.
- [68] J. Henry, "3DES is Officially Being Retired," 3 August 2018. [Online]. Available: <https://www.cryptomathic.com/news-events/blog/3des-is-officially-being-retired>.
- [69] C. Sincerbox, "Security Sessions: Exploring Weak Ciphers," March/April 2014. [Online]. Available: <https://electricenergyonline.com/energy/magazine/779/article/Security-Sessions-Exploring-Weak-Ciphers.htm>.
- [70] S. Giro, "Security "Crypto" provider deprecated in Android N," 9 June 2016. [Online]. Available: <https://android-developers.googleblog.com/2016/06/security-crypto-provider-deprecated-in.html>.
- [71] N. Elenkov, "Using Password-based Encryption on Android," 27 April 2012. [Online]. Available: <https://nelenkov.blogspot.com/2012/04/using-password-based-encryption-on.html>.
- [72] Google, "Using Cryptography to Store Credentials Safely.," 10 December 2018. [Online]. Available: <https://android-developers.googleblog.com/2013/02/using-cryptography-to-store-credentials.html>.
- [73] M. Sabt and J. Traore, "Breaking Into the KeyStore: A Practical Forgery Attack Against Android KeyStore," in *21st European Symposium on Research in Computer Security (ESORICS)*, Heraklion, Greece, 2016.
- [74] Google, "Content provider basics," 27 May 2019. [Online]. Available: <https://developer.android.com/guide/topics/providers/content-provider-basics#java>.
- [75] Google, "HttpCookie," 27 May 2019. [Online]. Available: <https://developer.android.com/reference/java/net/HttpCookie>.
- [76] ProgramCreek, "Java Code Examples for java.net.HttpCookie.isHttpOnly()," 27 May 2019. [Online]. Available: <https://www.programcreek.com/java-api-examples/?class=java.net.HttpCookie&method=isHttpOnly>.
- [77] J. Fernandes, "Implementation of Session Time Outs in Android," 3 February 2015. [Online]. Available: <https://stackoverflow.com/questions/28292390/implementation-of-session-time-outs-in-android>.
- [78] H. Mittal, "How to secure Android App Code from Reverse Engineering," 2 September 2017. [Online]. Available: <https://medium.com/@mittal2810/how-to-safe-app-from-reverse-engineering-d4ca7910d2f>.

- [79] GuardSquare, "Decompiling obfuscated Android applications," 27 May 2019. [Online]. Available: <https://www.guardsquare.com/en/blog/decompiling-obfuscated-android-applications>.
- [80] Google, "Sign your app," 27 May 2019. [Online]. Available: <https://developer.android.com/studio/publish/app-signing>.
- [81] OWASP, "Mobile Top 10 2016-Top 10," 1 August 2019. [Online]. Available: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10.
- [82] ZYS, "an unsafe implementation of the interface X509TrustManager from google," 23 February 2016. [Online]. Available: <https://stackoverflow.com/questions/35545126/an-unsafe-implementation-of-the-interface-x509trustmanager-from-google>.
- [83] OWASP, "Mobile Top 10 2016-M5-Insufficient Cryptography," 13 February 2017. [Online]. Available: https://www.owasp.org/index.php/Mobile_Top_10_2016-M5-Insufficient_Cryptography.
- [84] S. Cole, "New Study Suggests People Are Keeping Their Phones Longer Because There's Not Much Reason to Upgrade," October 30 2018. [Online]. Available: <https://www.wsj.com/articles/upgrade-no-thanks-americans-are-sticking-with-their-old-phones-1540818000>.
- [85] Google, "Full-Disk Encryption," 26 January 2019. [Online]. Available: <https://source.android.com/security/encryption/full-disk>.
- [86] J. Hruska, "Android 6.0 Marshmallow makes full-disk encryption mandatory for most new devices," October 20 2015. [Online]. Available: <https://www.extremetech.com/mobile/216560-android-6-0-marshmallow-makes-full-disk-encryption-mandatory-for-most-new-devices>.
- [87] Google, "Encryption," 26 January 2019. [Online]. Available: <https://source.android.com/security/encryption>.
- [88] Google, "File-Based Encryption," 1 January 2019. [Online]. Available: <https://source.android.com/security/encryption/file-based>.
- [89] Google, "File-Based Encryption," 26 January 2019. [Online]. Available: <https://source.android.com/security/encryption/file-based>.
- [90] Infosec Institute, "Cracking Android App Binaries," 1 August 2019. [Online]. Available: <https://resources.infosecinstitute.com/android-hacking-security-part-17-cracking-android-app-binaries/#gref>.

- [91] F. Brandolini, "Hooking Java methods and native functions to enhance Android applications security," 2016. [Online]. Available: https://amslaurea.unibo.it/12257/1/Brandolini_HookingJavaMethodsAndNativeFunctions.pdf.
- [92] S. Schmeelk and T. Lixin, "Mobile Software Assurance Informed through Knowledge Graph Construction: The OWASP Threat of Insecure Data Storage," *Journal of Computer Science Research*, vol. 2, no. 2, 2020.
- [93] NIST, "Buffer Overflow (BOF) Class," 1 February 2020. [Online]. Available: <https://samate.nist.gov/BF/Classes/BOF.html>.
- [94] T. Berners-Lee, J. Hendler and O. Lassila, *The Semantic Web*, Scientific American, 2001.
- [95] NIST, "Introduction to SAMATE," NIST, [Online]. Available: https://samate.nist.gov/index.php/Introduction_to_SAMATE.html. [Accessed 1 May 2020].
- [96] Wikipedia, "United States Computer Emergency Readiness Team," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/United_States_Computer_Emergency_Readiness_Team. [Accessed 1 May 2020].
- [97] user3898539, "How the SharedPreferences works and is it safe.," 18 August 2014. [Online]. Available: <https://stackoverflow.com/questions/25373145/how-the-sharedpreferences-works-and-is-it-safe>.
- [98] Google, "Save key-value data.," 12 December 2018. [Online]. Available: <https://developer.android.com/training/data-storage/shared-preferences>.
- [99] Google, "SharedPreferences.," 02 December 2018. [Online]. Available: <https://developer.android.com/reference/android/content/SharedPreferences>.
- [10] J. Howard, An Analysis of Security Incidents on the Internet, 1989-1995, PhD Thesis, 0] Pittsburg, PA: Carnegie-Mellon University, 1997.
- [10] National Institute of Standards and Technology (NIST), "Special Publication (SP) 800-30. 1] Guide for Conducting Risk Assessments.," 08 December 2012. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-30r1.pdf>.
- [10] R. Vallée-Rai, P. Co , E. Gagnon and L. Hendre, "Soot - a java bytecode optimization 2] framework.," in *In Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON '99*, 1999.

- [10 dangqingdani, Romangol and MindMac, "SecMobi Wiki," 12 October 2019. [Online].
3] Available: <http://wiki.secmobi.com/tools>.
- [10 A. Bartel, J. Klein, Y. Le Traon and M. Monperrus, "Dexpler: Con-verting android dalvik
4] bytecode to jimple for static analysis with soot.," in *In Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program Analysis, SOAP '12*, New York, New York, USA, 2012.
- [10 F. Wei, S. Roy, X. Ou and R. , "Amandroid: A precise and general inter-component data
5] flow analysis framework for security vetting of android apps," in *2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14*, New York, NY, USA, 2014.
- [10 U. Nikolic´ and F. Spoto, "Reachability analysis of program variables.," *ACM Trans.
6] Program. Lang. Syst.*, 35(4), p. 1–68, 2014.
- [10 Google, "Save files on device storage," 2020. [Online]. Available:
7] <https://developer.android.com/training/data-storage/files#java>.