BILINGUAL PUBLISHING CO. Pioneer of Global Academics Since 1984

Journal of Computer Science Research
http://ojs.bilpublishing.com/index.php/jcsr

ARTICLE

Mobile Software Assurance Informed through Knowledge Graph Construction: The OWASP Threat of Insecure Data Storage

Suzanna Schmeelk[1*] **Lixin Tao**[2]

1. St. John's University, United States
2. Pace University, United States

ARTICLE INFO

Article history
Received: 23 March 2020
Accepted: 9 April 2020
Published Online: 31 May 2020

Keywords:
Cybersecurity
Secure software development
Penetration testing
Risk assessment

ABSTRACT

Many organizations, to save costs, are moving to the Bring Your Own Mobile Device (BYOD) model and adopting applications built by third-parties at an unprecedented rate. Our research examines software assurance methodologies specifically focusing on security analysis coverage of the program analysis for mobile malware detection, mitigation, and prevention. This research focuses on secure software development of Android applications by developing knowledge graphs for threats reported by the Open Web Application Security Project (OWASP). OWASP maintains lists of the top ten security threats to web and mobile applications. We develop knowledge graphs based on the two most recent top ten threat years and show how the knowledge graph relationships can be discovered in mobile application source code. We analyze 200+ healthcare applications from GitHub to gain an understanding of their software assurance of their developed software for one of the OWASP top ten mobile threats, the threat of "Insecure Data Storage." We find that many of the applications are storing personally identifying information (PII) in potentially vulnerable places leaving users exposed to higher risks for the loss of their sensitive data.

1. Introduction

Many organizations, to save costs, are moving to Bring Your Own Mobile Device (BYOD) and adopting applications built by third-parties at an unprecedented rate. In these scenarios, organizations may have a Mobile Device Management (MDM) system in place, which help to lower cyber security risks by providing remote wipe procedures, geo-location fencing, and other security-minded features. 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. Therefore, in mobile application software development design choices, there can remain unmitigated higher risks.

As our world moves more-and-more to the edges with mobile application development and the Internet of Things (IoT), software assurance will have more requirements. Since the storage and transmission of sensitive data is a higher risk concern, it is optimal to detect related security concerns early. 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.

**Corresponding Author:*
Suzanna Schmeelk,
St. John's University, United States;
Email: schmeels@stjohns.edu

1.1 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 designing knowledge graphs to inform security analysis coverage for the prevention, mitigation and detection of mobile threats. This research first examines current Android mobile application static analysis software assurance techniques to discover trends in analysis. Recent academic publications examine the coverage and distribution of current Android static analysis for malware and other threat detection, mitigation, and prevention challenges. Recent research has shown that software assurance methodologies are adhoc and non-systematic. This research works to develop a unified model which can be further extended as new cybersecurity concerns develop and change over time. Finally, we examine 200+ mobile applications source code to understand their cybersecurity software assurance designs with respect to the OWASP Mobile Threat Two for "Insecure Data Storage."

1.2 Review of Literature

Software assurance of mobile applications, specifically informed through knowledge graph construction, requires a literature background grounded in both (1) cybersecurity ontology and knowledge graph construction and (2) mobile software cybersecurity assurance and development. Both domains are highlighted in the following subsections.

1.2.1 Cybersecurity 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)."

Ontologies are defined, according to Goknil and Topaloglu [2], 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 [4]." Lacy [5] states 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 (e.g. [2])can be useful for software development to make clear how systems should operate at a fundamental level. Yu [3] 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.

Kafali et al. [8] 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 (p. 532) as one of their research contribution. As can be seen in this particular ontology, there are two relations, *is-a* (represented by lines) and *has-a* (represented by arrows), and twelve nodes in the ontology. 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), whom maintains a breach portal of open cases.

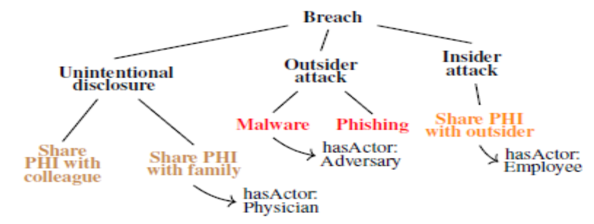


Fig. 1. Breach concepts. Lines represent subclass (is-a) relations among concepts. Arrows represent properties of concepts (has-a relations).

Figure 1. HIPAA Breach Ontology (p. 532) from Kafali et al. [8]

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 [6] recently published work in this domain proposing minimal syntax extension to the OWL Web Ontology Language for declaring custom relations with special attributes and applying them in knowledge representation. Their work presents additions to the OWL API for the declaration, application, and visualization of custom relations. The research paper outlines revisions and additions to the ontology editor Protégé so its users can 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 knowledge graphs.

1.2.2 Cybersecurity Software Assurance

Industry relies on three pivotal vulnerability discourse frameworks developed independently by MITRE and the National Institute of Standards and Technology (NIST). MITRE maintains the developer specific taxonomy, the Common Weakness Enumeration (CWE) [8], and the malware specific taxonomy, the Common Attack Pattern Enumeration and Classification (CAPEC™) [9]. NIST maintains the developer specific condensed taxonomy, the Bug Framework (BF) [10]. These frameworks attempt to unify the industry into standardizing vulnerability language; however, industry and research communities can still disagree with vulnerability specifics such as actual 'fault' problem sources or effects. Similarly, library designers and users can also disagree on exact implementation requirements interpretations. Many industry tools rely on these MITRE and NIST frameworks to convey vulnerability details to software engineers. These frameworks are designed to answer questions pertaining to describing malware or developer techniques.

Current static analysis research on mobile applications is adhoc and quite non-systematic. Four major domains of research exist from a unifying utility perspective, as described by Schmeelk [11], Schmeelk [12], Schmeelk and Aho [13], and Schmeelk, Yang, and Aho [14]. In general, research exists for software assurance methodologies including detecting vulnerabilities during development, detecting malware at large, detecting specific application behavior in a sandbox, and sanitizing/re-packing applications.

Schmeelk [11], Schmeelk [12], Schmeelk and Aho [13], and Schmeelk, Yang, and Aho [14] 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 (confidentiality, integrity, and availability) triad but at a higher-level such as leaving secret keys hard coded in an application, improper certificate validation, etc.).

Schmeelk's [12] research showed that only a few Android software assurance static analysis research papers fit into the NIST BF's six current classes, as seen in Figure 2. The static analysis meta-research also raised questions about five findings that may be of the BF four main elements of a bug: causes, attribute, consequences and sites of bugs. Schmeelk [11,12] showed that some of the meta-research matched in both the MITRE and NIST framework, such as: 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). Schmeelk [12] discussed differences between the MITRE CWE analysis and the NIST BF analysis. The research found that there are currently some non-overlapping categories (i.e. 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) between the two taxonomies. Schmeelk [11,12] noted that the non-matching CWE comparisons may not directly map into NIST BF as individual classes.

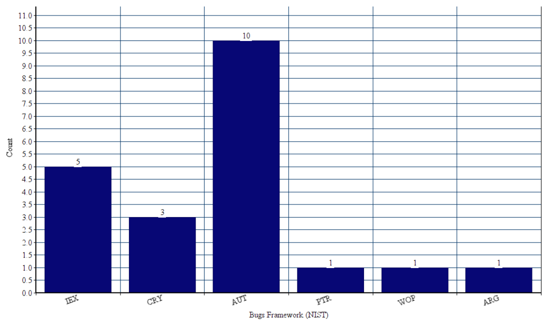


Figure 2. The fraction of NIST BF categories researched in Android analysis across the publications as presented in Schmeelk [12]

2. Methodology/Methods

This research contributes an examination of mobile application security through the construction of a knowledge graph for the current OWASP Top 10 Mobile Threat 2 "Insecure Data Storage." The knowledge graph contribution links threats from the two most recent major OWASP Mobile Top 10 Threat releases, 2014 and 2016, to show changes in time and to help determine security changes over time. Currently, only the NIST BF has built any graphical representation to inform further software assurance analysis. Our developed OWASP specific high-level graphical representation shows potential vulnerabilities such as the insecure storage of sensitive data, which depending on how the mobile software code is utilized, can occur heavy fines for the mismanagement of sensitive personal identification information (PII).

This research then contributes to how specific mobile device source code, specifically Android in this research, can be useful to informing static analysis. In addition, developing such a knowledge graph can be further extended over time as cybersecurity evolves and changes. In this research we focus on source code analysis; however, the knowledge graph can be extended to include byte code patterns or entirely other mobile device application languages, such as Swift, JavaScript, and C/C++.

In our final contribution, we analyze 200+ healthcare Android applications source code from GitHub to learn what, if any, security concerns are being developed into their source code. Static analysis can be employed to ensure proper source code management of personally identifying information and other sensitive data. Some of the applications analyzed collect highly sensitive information such a body weight, body signals (e.g. blood pressure, temperature), mental health measurements, obgyn measurements, among other sensitive information. Specifically, in this research, we analyzed applications for components of the constructed-knowledge graphs, confidentiality and integrity of their sensitive information.

2.1 OWASP 2014 M2 Insecure Data Storage (IDS)

The OWASP 2014 Mobile Threat 2 is "Insecure Data Storage." In our knowledge graphs of, seen in Figure 3, we label this threat as *M2_Insecure_Data_Storage*. Insecure data storage (IDS) can potentially lead to data compromise or the propagation of malware [15]. Our knowledge graph is based on industry best practices and industry news reports.

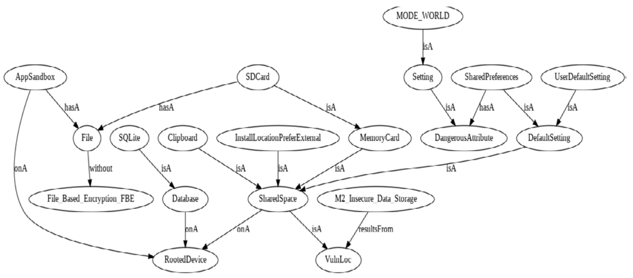


Figure 3. Our Knowledge Graph for OWASP 2014 M2 Insecure Data Storage

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 [16]. 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.

Android offers at least four locations to store data: internal file storage, external file storage, shared preferences, and databases [17]. Common *VulnerableLocations* on mobile devices are (i.e. *isA*) shared spaces [18]. 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 [18]. 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* [18].

Table 1. Android Java Code for File Storage [23]

```
/* 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 = new File(Environment.getExternalStoragePublicDirectory(
   Environment.DIRECTORY_PICTURES), albumName);
 if (!file.mkdirs()) {
  Log.e(LOG_TAG, "Directory not created");
 }
 return file;
} …
```

Table 1 shows the Android Java code for storing a file. Specifically, the method call 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. There are at least two definitions debatably used in the field-at-large for static analysis tools. First, if a tool is *complete*, it will never report false positives; but, it may result in "*false negatives*." Second, if a tool is *sound*, it will never miss any violations; but, it may result in "*false positives.*"

First, with a source code one-pass text analysis, static analysis can determine if any risky Context API values [19] are used within the application (e.g. locating the external storage home directory (i.e. *getExternalStoragePublicDirectory*), or any public API Context values). 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.

Android specifies at least two types of default settings [20]. DefaultSettings are (i.e. *isA*) *User Default Settings* or *Shared Preferences. SharedPreferences* are just plain XML files in an application directory on internal storage [21]. 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 [19] are used within the application. 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. 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*.

Table 2. Android Java Code for *Shared Preferences* [22]

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

Another *SharedSpace isA* Database *onA rooted* device in the Android API is SQLite [24]. This database may have tables, which may not have permissions correctly set to restrict access to only the files for which they are privileged. The Android API offers fixed standard values stored in the API context interface. 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 3 shows Android application code for interacting with the Android *SQLite* database using the Android API [24]. First, with simple source code one-pass text analysis, static analysis can determine if any risky context API values are used within the application. This technique is neither sound nor complete [19]. 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.

Table 3. Android Java Code for SQL Lite *Database* [24]

```
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, SQLiteData-
base.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);
```

Another *SharedSpace* in Android *onA RootedDevice isA ApplicationSandbox* [20]. The relationship *onA* is needed to show that the *ApplicationSandbox* becomes a *SharedSpace* once a devices 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 hierarchal 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 [18]. The relationship *without* is important to show a missing property for secure data storage.

Application developers can choose to specify where they prefer that their application is installed. Android currently permits developers to specific 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 [25]. 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 internal applications. 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 to can our knowledge graph. This dynamic representation provides a standardized methodology to reason about mobile software assurance.

2.2 OWASP 2016 M2 Insecure Data Storage (IDS)

The OWASP 2016 Mobile Threat Two is "Insecure Data Storage" (M2_ Insecure_Data_Storage), as seen in Figure 4. IDS can potentially lead to data compromise of sensitive data or the unauthorized data access.

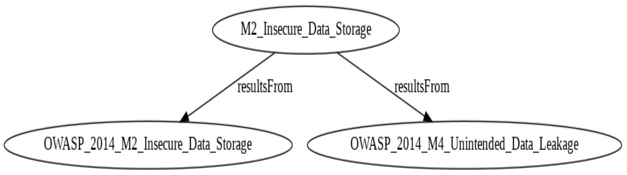
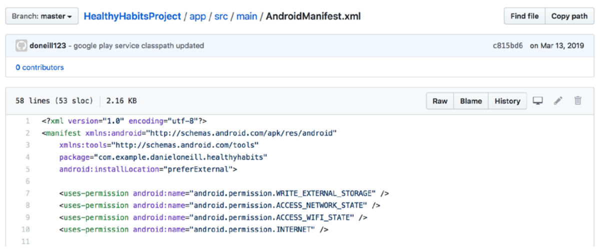

M2_Insecure_Data_Storage
resultsFrom
resultsFrom
OWASP_2014_M2_Insecure_Data_Storage
OWASP_2014_M4_Unintended_Data_Leakage

Figure 4. Our Knowledge Graph for OWASP 2016 M2 Insecure Data Storage

According to the OWASP 2016 Mobile Threat categories [26], we built the 2016 threat *resultsFrom* either *OWASP_2014_M2_Insecure_Data_Storage* or, since the *M4* category disappeared in 2016, *results From OWASP_2014_M4_Unintended_Data_Leakage*. Therefore, the 2016 category of threats is simply a composite of two former OWASP 2014 threats M2 IDS and M4 "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.

3. Results

We analyzed the source code of 200+ Android healthcare applications hosted on GitHub. In our mobile application analysis, we detected what, if any, secure coding artifacts are present for the OWASP 2016 Mobile Threat 2 "Insecure Data Storage." The artifacts chosen for this analysis were based on our Section 2 knowledge graph. In all the healthcare applications, the data collected is considered sensitive or personal identifying information (PII) since it can be attached to certain device specific identifiers such as IMEI or IMSI.


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

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.danieloneill.healthyhabits"
    android:installLocation="preferExternal">

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />

```

Figure 5. Application seeking permission to be stored on external storage

First, our research found some case specific issues such as the application listed at https://github.com/doneill123/HealthyHabitsProject/blob/master/app/src/main/Android-

Manifest.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 5. The specific Google guidance on installing applications entirely on the external storage is seen in Figure 6.

Secondly, 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.

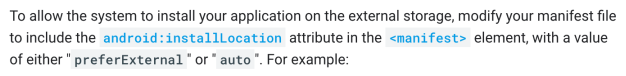

To allow the system to install your application on the external storage, modify your manifest file to include the `android:installLocation` attribute in the `<manifest>` element, with a value of either "`preferExternal`" or "`auto`". For example:

Figure 6. 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 4. 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 be installed entirely on external storage, through the string "*android:installLocation*="preferExternal." Once the application is itself installed on external storage, different application risks are introduced. 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 4. 200+ Healthcare GitHub Application Analysis Results for Potential IDS Risks

#	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/health-go	a android pedometer app (Android pedometer)	W					
16	https://github.com/scoute-dich/Quit-Smoking	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/Skin-HealthChecker	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 book, How Not to Die with this Android app https://play.google.com/store/apps/de…	W				D	
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/malar-iapp	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	Appication android mobile systems	W				D	
47	https://github.com/Health4TheWorld/Health4TheWorld-android	Android App for Health4TheWorld	W				D	
52	https://github.com/McFlyWYF/HealthManager	Android Course Design --- Health Management Based on Heart Disease	W				D	
56	https://github.com/ibisTime/xn-health-android	Health e purchase 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/Healthy-Maternity	Android application to help rural pregnant women	W					
68	https://github.com/Ethanator/Mobile-Health	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/CMPUT-301F18T21/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 healtband 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 apps	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 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…					D	B
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/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.					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/Health-Care	Android app with Firebase					D	
101	https://github.com/SRatna/HealthyNepali	Android health related application					D	
108	https://github.com/kelooy/HealthDi-agnostics	Android app | Patients data storage https://github.com/kelooy					D	
110	https://github.com/KieronMoorcroft/HealthMonitor	An Android Health Monitor App					D	
115	https://github.com/KuznetsovaAnas-tasiia/HealthyCafe	An *android* app for the cafe staff					D	
116	https://github.com/verma-ady/Health-Litmus	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/CYBA-Weight	android app for health					D	
138	https://github.com/neelmehta247/Hack4Health	RemindMe is an Android app that helps Alzheimer's patients multi-task in their day to day life.					D	
141	https://github.com/chinnatan/Healthy	Advanace 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/Sim-pleHealthPlan	Android App					D	
170	https://github.com/marcgilbert01/ContactsSimpleApp	Android test for "Babylon Health"					D	
187	https://github.com/SuciuCalin/Proj-ect_09_HealthyRoutineTracker	Habit Tracker App - Udacity Android Basics Nanodegree by Goo-gle					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/kevm66/4thYear-Project_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/Health-Chilli	Android application for healthchilli.com						B
119	https://github.com/qianzch/NowSleep	It's time for bed! Now Sleep! [Android App]						B

4. Discussion

In this section, we report on how our results reflect the current state of research in the field and make suggestions for future research. First, we summarize the relations we introduced in our knowledge graph discussions and report on how the static analysis of the 200+ Healthcare Applicatons informs the cybersecurity of protecting personal identifying information (PII). Lastly, we dicuss future work.

4.1 Introduced Knowledge Graph Relations

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

Table 5. Relations Introduces in Mobile Threat Knowledge graph

• *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 relationships will be kept constant to provide consistency; however, they may become deprecated as cases further evolve.

4.2 Static Analysis Detection

Static analysis is one methodology to inform secure software development. Even though there are infinite ways to write a computer program, some applications rely on the standard Android API help analysis engines to detect software assurance concerns. Currently cybersecurity best practices rely entirely on the software architect(s) and software developer(s), quality assurance team(s), so penetration tester(s). Most standard compliers do not yet inform on cybersecurity concerns. In fact, compiler designers can argue that cybersecurity is not responsibility of the compiler. Since the cybersecurity domain is changing at an unprecedented rate, it is nearly impossible for any software architect, software developer, quality assurance team member or penetration tester to detect every possible security concern in real-time. Developing knowledge graphs to inform on software assurance patterns within code help to standardize the overall software assurance process and to catch cybersecurity issues at any point of the secure software development lifecycle.

4.3 Future Work

The knowledge graph created in this research connecting the OWASP 2014 and the OWASP 206 mobile threats for mobile security is among the first known public software assurance knowledge graphs. Future research includes crowd sourcing the knowledge graphs so that the world can continue to develop and detect vulnerabilities as they are discovered. In addition, a knowledge graph can aid 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.). The knowledge graphs can indicate risk and other important metrics to inform organizations as they make important technology adoption and redesign decisions.

Another area for future research is to develop a lower level code-specific knowledge graph at either the source code or byte-code levels to inform further program analysis. These knowledge graphs would be similar to the NIST BF to inform all of the specific lower-level coding patterns which can lead to the higher-level mobile threats.

All future knowledge graph construction can be implemented within program analysis tools, application stores (e.g. Google Play, iTunes, etc.) and mobile device management platforms to detect potentially problematic code. Since the people on our planet are moving to mobile and application-dependent services, our security needs to follow their trends especially in legally regulated sectors.

Finally, the agnostic knowledge graph representations can be further built to inform other code analysis such as iOS applications, which are currently mainly developed in Swift or Objective-C, or even applications developed entirely with front-end languages such as JavaScript, Node.js, PHP, Elm, among others.

5. Conclusion

Organizations across the world are moving to mobile devices following BYOD strategies as well as utilizing third-party applications. Currently very little research exists into building overall cybersecurity frameworks to inform software assurance at any level (development, forensics, penetration tests). To date, most research is adhoc leaving risk assessments completely different by different parties as no unifying guidance exists for mobile application software assurance. Our research starts filling the industry and research gap by developing knowledge graphs for the widely-used OWASP threat of Insecure Device Storage. Our knowledge graph reflects the changes to the threats over the years, helping cybersecurity experts to identify changes. After working to develop these knowledge graphs, we show how the knowledge graphs could be used to inform software assurance static analysis by connecting the graphs with Android source code. Lastly, we analyze the source code for 200+ Android applications from GitHub. These applications fall into the domain of healthcare applications; thus, they collect very sensitive personal identifying information (PII). We found that many applications request permissions to write sensitive data to external storage devices and a few applications (perhaps due to their size) are coded to prefer to be stored on external drives entirely. As these known external drives are shared among all the applications on the device, we can see that our source code has now give the application higher risks for keeping the CIA of sensitive data. As more and more laws are developed, we can further develop our knowledge graphs for the community to consistently identify risks of how the source code manages sensitive data.

References

[1] Allemang, D., Hendler, J.. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. Morgan Kaufmann Publishers Inc., 2011.

[2] Goknil, A., Topaloglu, Y.. Ontological perspective in metamodeling for model transformations. In Proceedings of the 2005 symposia on Metainformatics (MIS '05). New York, NY, USA: Association for Computing Machinery, 2005: 7-es.

[3] L. Yu. A Developers Guide to the Semantic Web. Springer Publishing Company, Incorporated, 2015.

[4] Noy, N., McGuinness, D.. Ontology development 101: A guide to creating your first ontology. Palo Alto, CA, USA: Technical report at Stanford Knowledge Systems Laboratory, 2001.

[5] Lacy, L. W.. OWL: Representing Information Using the Web Ontology Language. Victoria, BC, Canada: Trafford, 2005.

[6] Patel, I., Dube, I., Tao, L., & Jiang, N.. Extending OWL to Support Custom Relations. 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing. New York, NY, USA: IEEE, 2015: 494-499.

[7] Kafali, Ö., Jones, J., Petruso, M., Williams, L., Singh, M. P.. How good is a security policy against real breaches?: a HIPAA case study. Proceedings of the 39th International Conference on Software Engineering. Buenos Aires, Argentina: IEEE Press, 2017: 530-540.
DOI: 10.1109/ICSE.2017.55

[8] MITRE. Common Weakness Enumeration (CWE), 2020. Retrieved from:
https://cwe.mitre.org/

[9] MITRE. Common Attack Pattern Enumeration and Classification (CAPEC™), 2020. Retrieved from:
https://capec.mitre.org/about/index.html

[10] NIST. Bug Framework (BF), 2020. Retrieved from:
https://samate.nist.gov/BF/

[11] Schmeelk, S.. Where are we looking for security concerns? Understanding Android Security Static Analysis. Proceedings of the Future Technologies Conference (FTC) 2019. San Francisco, CA: Springer, 2019: 1-9.

[12] Schmeelk, S.. Where are we looking? Understanding android static analysis techniques. In 2019 IEEE International Conference on Services Computing. Milan, Italy: IEEE, 2019.

[13] Schmeelk, S., & Aho, A.. Defending android applications availability. 2017 IEEE 28th Annual Software Technology Conference (STC). Gaithersburg, MD: IEEE, 2017: 1-5.

[14]Schmeelk, S., Yang, J., Aho, A.. Android malware static analysis techniques. In Proceedings of the 10th Annual Cyber and Information Security Research Conference CISR '15. New York, NY, USA: ACM, 2015: 51–58.

[15] OWASP.. Mobile Top 10 2016-M2-Insecure Data Storage, 2018. Retrieved from owasp.org:
https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage

[16] NIST. Guide for Conducting Risk Assessments, 2012. Retrieved from:
https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final

[17] Google. Data and file storage overview. 2020. Retrieved from:
https://developer.android.com/guide/topics/data/data-storage#db

[18] Google. Security Tips. 2020. Retrieved from:
https://developer.android.com/training/articles/security-tips

[19] Google. Context. 2020. Retrieved from:
https://developer.android.com/reference/android/content/Context#openFileOutput(java.lang.String,%20int)

[20] Rajab, A.. How to prevent database and shared preferences from being hacked. 2017. Retrieved from Stack overflow:
https://stackoverflow.com/questions/47207420/how-to-prevent-database-and-shared-preferences-from-being-hacked

[21] User3898539. How the *SharedPreferences* works and is it safe. 2014. Retrieved from Stack overflow:
https://stackoverflow.com/questions/25373145/how-the-sharedpreferences-works-and-is-it-safe

[22] Google.). Save key-value data. 2020. Retrieved from developer.android.com:

https://developer.android.com/training/data-storage/shared-preferences

[23] Google Developers.. Saving Files. 2020. Retrieved from stuff.mit.edu:
https://stuff.mit.edu/afs/sipb/project/android/docs/training/basics/data-storage/files.html

[24] Google.. Save data using SQLite. 2020. Retrieved from developer.android.com:
https://developer.android.com/training/data-storage/sqlite

[25] Google.. Save files on device storage. 2020. Retrieved from developer.android.com:
https://developer.android.com/training/data-storage/files#java

[26] OWASP.. Mobile Top 10 2016-Top 10. 2020. Retrieved from owasp.org:
https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10