A Study on the Testing of Android Security Patches

Christopher D. Brant
University of Florida
cdbrant@ufl.edu

Tuba Yavuz
University of Florida
tuba@ece.ufl.edu

***Abstract*—Android controls the majority of the global OS market. Android Open Source Project (AOSP) is a very complex system with many layers including the apps, the Application Framework, the middle-ware, the customized Linux kernel, and the trusted components. Although security is implemented in every layer, the Application Framework forms an important of the attack surface due to managing the user interface and permissions. Android security has evolved over the years. The security flaws that have been found in the Application Framework led to a redesign of Android permissions. Part of this evolution includes fixes to the vulnerabilities that are publicly released in the monthly Android security bulletins. In this study, we analyze the CVEs listed in the Android security bulletin within the last 6 years. We focus on the Android application framework and investigate several research questions relating to 1) the security relevant components, 2) the type and amount of testing information for the security patches, and 3) the adequacy of the tests designed to test these patches. Our findings indicate that Android security testing practices can be further improved by designing security bulletin update specific tests, and by improving code coverage of patched files.**

I. Introduction

Android security and how it has been enforced have evolved over the years. A formal report [1] defines the security model as consisting of multi-party consent, open ecosystem access, treating compatibility as a security requirement, secure factory resets, and treating applications as the security principals. Android platform's defense-in-depth aims to prevent adversaries from bypassing the security model through four main strategies [1]: isolation and containment, exploit mitigation, integrity, and patching/updates.

Timely patching of vulnerabilities is critical for a popular platform such as Android. Therefore, Google has established the monthly security bulletins starting in 2015. However, reflecting the patches to various versions/customizations of Android has proved to be challenging [2], [3]. Several solutions have been utilized to improve the security patching process. These include 90-day patching requirement for Original Equipment Manufacturers (OEMs), revising the software architecture and reducing the attack surface by restricting the access to the hardware through the Hardware Abstraction Layer (HAL), and allowing some system core components to be updated through Google Play Store.

An important aspect of Android security is the secure implementation of methods within the framework layer. A vital component for securing the Android framework layer is the permission mechanism and its implementation, which is critical in realizing the multi-party consent aspect of the security model. So, an important line of research has investigated Android permission modeling and the related flaws in its design and implementation [4], [5], [6], [7], [8], [9].

A recent work [9] on automated analysis of custom permissions shows that previous fixes to prevent the privilege escalation attacks reported in [8] did not completely eliminate other attack scenarios that exercise alternative execution paths that bypass the fix.

In this paper, we investigate the Android testing practices for the security patches with a focus on the Android application framework. Our goal is to identify some of the characteristics of security relevant components in the Android application framework and evaluate the adequacy of tests developed for the security patches involving such components. We have analyzed Android monthly security bulletins to compile a set of 188 CVEs that involve the Android Framework from the last 6 years. We have analyzed the patches to identify changed components, the changed tests (if any), and how to execute the relevant tests. Java classes and packages and whether the thrown or caught security exceptions are indicative of security relevance of a Java class in the Android framework. We intend to investigate classes and packages that contain functions relating to security exceptions due to their obvious relevance to security. Additionally, we intend to observe how often these classes are updated and how thoroughly these updated classes are tested, as there is potentially a lot of room for improvement and automation across the Android testing suite. We have formulated seven research questions (RQs) as shown in Table I. The first four questions **RQ1-RQ4** are about understanding the security relevant Java classes and packages. Research questions **RQ5-RQ6** investigate the amount and kinds of testing information available in the security patches. Finally, research question **RQ7** quantifies the adequacy of the tests on a selected sample of tests using Androids automated testing framework.

Our findings indicate that:

1) The Android framework Java classes that contain security exception functions should be given higher priority for extensive testing.
2) The majority of the listed tests are included for regression testing and not to verify the new and correct functionality of the security bulletin updates.
3) The basic block coverage is higher for cases when the security bulletin updates include updated test files specific to the patch. However, across the board, basic block

TABLE I
THE RESEARCH QUESTIONS INVESTIGATED IN THIS STUDY.

RQ1: What percentage of Java classes in the Android framework throw and/or catch security exceptions?
RQ2: What are the most frequently updated Java classes and packages in the Android framework related to the security bulletin updates?
RQ3: What percentage of Java classes updated in relation to security bulletins throw and/or catch security exceptions?
RQ4: What percentage of Java classes that throw and/or catch security exceptions are updated in relation to security bulletins?
RQ5: Which security bulletin updates list testing information?
RQ6: What kind of tests are listed in security bulletin updates to verify patch functionality?
RQ7: When listed, how much of a patch is covered by the given tests?

coverage needs to be improved.

This paper is organized as follows. We discuss related work in Section II. We explain our methodology in Section III. We present the results in Section IV. We discuss our findings in Section V. Finally, we conclude in Section VI with directions for future work.

II. Related Work

a) Android Security Patches: Previous work on Android vulnerability patches focused either on the patch propagation process [10], [11], [2], [3] or patch code size and patterns [12]. While in [10], [11], [2] the focus is on the whole ecosystem, in [3] the focus is on the Android kernel updates only. While these works investigate the delays in patching vulnerabilities using different methods, our approach focuses on the testing aspect of the patches for the Android Application Framework.

The empirical study in [13] reports that the developers update vulnerable native code in their apps at a 10 times slower rate compared to patching other security vulnerabilities in their apps.

b) Android Automated Testing: While majority of automated testing research in the context of Android focuses on app testing, e.g., [14], [15], [16], [17], [18], there are few approaches [15], [9] that involve the Android Application Framework.

In [15] Android Framework SDK is instrumented to capture the registration and de-registration of callbacks for system events for random testing in combination with UI event and input selection for the Android apps. Our instrumentation of the Android Framework is guided by the security patches and we use the instrumentation to evaluate the code coverage of the existing or security relevant tests. A black-box fuzz testing approach is presented in [9] with the goal of detecting privilege escalation attacks. Our work evaluates the adequacy of tests developed for the Android framework and shows some of their shortcomings within the context of security patches, which can inform approaches like [9].

III. Methodology

Our overall methodology for data aggregation began with searching through the Android source files from Google Git [19]. To decide where to begin in the repository to investigate, we utilized the Android Security Bulletin postings [20] over the last 6 years for which files have been modified in security framework updates. Specifically, the range over which we collected data included every month of every year from 2017 through 2021, and each month of 2022 up to and including the month of March. For each public Android Security Bulletin framework update posted over the chosen time period, we collected the following pieces of data:

- CVE Number
- Reference Number and URLs
- Update Type and Severity
- Packages and Classes Updated
- Month and Year of Update
- Any Bug IDs mentioned in the description

From this base data set we created a list of the Java classes and packages that were updated the most frequently, as well as which unique pairs of classes were updated together.

A. Running Tests

Within Android there are a handful of different methods of running tests on the framework. Our research led us to need to run tests in a subset of these possibilities, which are Atest and manually testing using Android Debug Bridge (ADB). For our purposes we instrument every java class file that was patched in each Android security bulletin update over the last 6 years with log instructions, such that it could be observed whether or not the patched updates were being thoroughly tested for the correct functionality. Log statements were specifically added at the entry and exit of each basic block of code listed as changed within the diff file of each Android security bulletin update covered in our work.

Once instrumented, we rebuilt the AOSP source code with the make command so that the instrumented framework classes would all be updated in the image run by the android emulator. With the emulator running our custom instrumented image, we collected the coverage data for the basic block entry and exit statements.

1) Testing with Atest: Atest is a command line tool provided by Google that is designed to allow users to build, install, and run Android tests locally, for the purpose of speeding up tests without requiring knowledge of the Trade Federation test harness [21]. Most of our tests were run utilizing Atest, as the majority of tests listed in the security bulletin updates associated with the most frequently updated files were Atest tests. To test with Atest, all that is required is the name of the class, package, or module.

2) Testing with ADB: ADB is a tool that can be used to run instrumented tests on an Android virtual device from the command line. For our purposes, ADB was necessary for running a small group of tests that Atest was unable to find so

that we could complete our testing of Atest modules manually. To run a test using ADB, first the module or package must be compiled into its corresponding APK file, and then installed before running the given test.

A package containing files for replication will be released at https://github.com/sysrel/AndroidSecurityPatchTestingStudy.

IV. Results

Prior to experimenting with instrumented tests on AOSP, we collected and aggregated a large amount of data about Android security bulletin updates and the changes made in each update. From data collected via the scrapers described above in Section III, we were able to visualize or quantify answers to many of our research questions.

In an effort to understand the security relevant classes in the Android framework, we leveraged the updates in the Android security bulletins as well as security relevant exceptions. The Java SDK provides a `SecurityException` class to represent security related exceptions. The Android framework uses the `SecurityException` class in various parts of the code base. So, when we refer to a security exception we refer the `SecurityException` class[1]. Through **RQ1-RQ4**, we investigate the classes that use the security exceptions, the classes that are most frequently updated as part of the Android security bulletin patches, and the relationship between classes that use security exceptions and those that are updated most frequently as part of security patches. Sections IV-A-IV-D present the results of **RQ1-RQ4**.

Sections IV-E-IV-F present the results on whether any testing, old or new, is performed for each CVE to answer **RQ5-RQ6**.

Finally, Section IV-G presents the results for **RQ7** on the adequacy of tests measured in terms of code coverage for the patches that involved the most frequently updated classes.

A. RQ1: What percentage of Java classes throw and/or catch security exceptions?

In Figures 1, 2, and 3, we categorize the Java classes in the Android framework based on two dimensions: 1) with respect to security exceptions (**SE**) (just catches (**C**) a security exception, just throws (**T**) a security exception, or both catches and throws (**T & C**) a security exception) and 2) with respect to being updated as part of a security patch (updated or not updated). Figures 1, 2, and 3, show us that across Android versions 9, 10, and 11, of all the Java classes containing security exception functions, an average of 65.0% contain functions that throw security exceptions, 45.3% contain functions that catch security exceptions, and 10.3% contain functions that both throw and catch security exceptions. This suggests that a large amount of Java classes throughout the Android framework are high priority classes from a security perspective.

[1]Although other types of exceptions may also be related to security, we think that the SecurityException class has been explicitly designed for security related cases and, therefore, we assume that its usage must be intended to be security relevant.

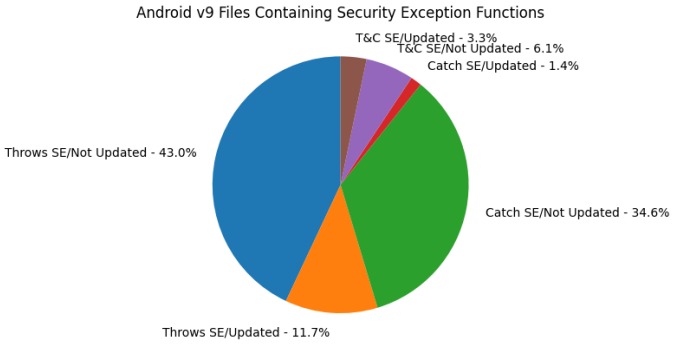

Fig. 1. Distribution of AOSP Files with Security Exception Handling in Android 9

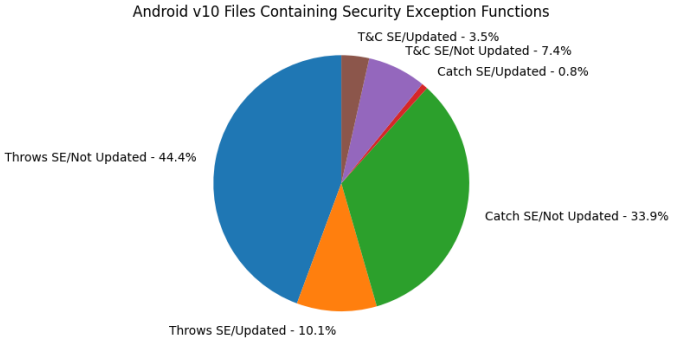

Fig. 2. Distribution of AOSP Files with Security Exception Handling in Android 10

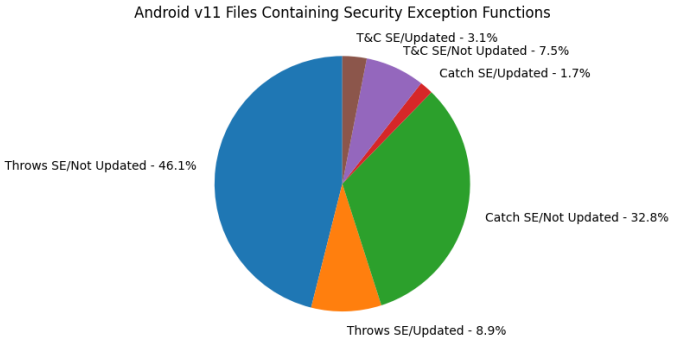

Fig. 3. Distribution of AOSP Files with Security Exception Handling in Android 11

B. RQ2: What are the most frequently updated Java classes and packages related to the security bulletin updates?

Any Java class or package that is frequently mentioned in security bulletin updates is likely to be intriguing. Our python data scraping scripts allowed us to visualize the most frequently updated Java classes and packages in Tables II and III, respectively. From these tables, we observe that package management, activity management, window management, and permission management related services and packages are by far the most commonly updated Java classes and packages. This

suggests that the above listed packages and classes are also of high priority from a security perspective.

TABLE II
FREQUENCY OF UPDATES TO JAVA CLASSES LISTED IN SECURITY BULLETINS

Java Class Name	Instances	T/C SE
PackageManagerService.java	8	Yes
ActivityManagerService.java	7	Yes
WindowManagerService.java	5	Yes
PermissionManagerService.java	4	Yes
ActivityStarter.java	3	No
Notification.java	3	No
SQLiteSecurityTest.java	3	No
PackageInstallerService.java	3	Yes
BasePermission.java	3	Yes
DevicePolicyManagerService.java	3	Yes
LockTaskController.java	2	Yes
RootWindowContainer.java	2	No
SliceProvider.java	2	Yes
LockTaskControllerTest.java	2	No
InstallStart.java	2	No
TextClassification.java	2	No
PermissionManagerServiceInternal.java	2	No
ActivityManagerShellCommand.java	2	No
GrantPermissionActivity.java	2	No
GrantCredentialsPermissionActivity.java	2	No
AppOpsService.java	2	Yes
Layout.java	2	No
PhoneWindowManager.java	2	No
WallpaperManagerService.java	2	Yes
ExternalStorageProvider.java	2	Yes
AccountManagerService.java	2	Yes

TABLE III
FREQUENCY OF UPDATES TO PACKAGES LISTED IN SECURITY BULLETINS

Package Name	Instances
/android/server/wm	35
/android/server/am	17
/android/server/pm	16
/android/view/textclassifier	13
/android/server/pm/permission	10
/android/providers/downloads	8
/android/view	8
/android/os	8
/android/app	6
/android/accounts	6
/android/settings/slices	5
/android/server/devicepolicy	4
/android/packageinstaller	4
/android/database/sqlite/cts	3
/android/server/wallpaper	3
/android/text	3
/android/provider	3
/android/widget	3
/android/server/usage	3
/android/server/autofill	3

C. RQ3: What percentage of Java classes updated in relation to security bulletins throw and/or catch security exceptions?

In our data set of security bulletin updates posted over the last 6 years, there are exactly 200 total Java classes updated, amongst those only 35 contain functions that throw and/or catch security exceptions for AOSP version 9. For AOSP version 10, there are 37 Java classes that contain such functions, and 40 in AOSP version 11. Thus there are only 17.5% of updated Java classes that also contain security exception related functions for AOSP version 9, 18.5% for AOSP version 10, and 20% for AOSP version 11. In addition, of those Java classes contained in our list of frequently updated files shown in Table II, 50% of those contain functions that throw and/or catch security exceptions. Table II also includes a column illustrating which of the most frequently updated Java classes contain security exception functions. All of the top four frequently updated Java classes contain security exception functions. This supports our earlier suggestions that the most frequently updated classes within security bulletin updates are very high priority from a security perspective as they both contain security exceptions and are the most frequently updated classes.

D. RQ4: What percentage of Java classes that throw and/or catch security exceptions are updated in relation to security bulletins?

As seen across Figures 1, 2, and 3, only a relatively small subset of Java classes containing security exception functions are updated to patch an Android Framework related CVE over the last 6 years. For files containing security exception functions, across each of the three AOSP versions, version 9, version 10, and version 11, we see an average of 12.3% of updated files that contain functions that throw security exceptions, 7.3% of updated files that contain functions that catch security exceptions, and 3% of updated files that contain both functions that throw and catch security exceptions. Additionally, across each of the three AOSP versions, an average of only 16.6% of the total number of Java classes containing security exception related functions are updated over the last 6 years. This illustrates that of the total Android framework classes that contain security exception related functions, a very small contingency of that group is updated across the last 6 years of security bulletin updates. This suggests that there is a relatively small subset of classes containing security exception related functions that are being targeted or contain the majority of recognized problems.

E. RQ5: Which security bulletin updates list testing information?

Over the last 6 years, a total of 188 security bulletin updates regarding the Android Framework have been posted, and 171 of those contain testing information in their patch descriptions. Table IV illustrates the categorical breakdown of the those 171 security bulletin updates with included testing information.

When categorizing all of the testing information, we found a grouping of 6 different distinct categories. Those categories being Atest testing, Manual testing, PoC testing, Build/Make testing, Tradefed testing, GTS (Google Mobile Services Testing Suite) [22] testing, and Generic/Nondescript testing. For Atest testing, any testing information that contained the ”atest” or ”Atest” keywords or contained an Atest module was designated in the Atest category. Any testing information that was listed as manual testing was designated into the Manual

TABLE IV
TEST CATEGORY SUMMARY

Test Information Category	# of Unique CVEs
Atest	73
Manual	49
PoC	9
Build/Make	12
Tradefed	13
GTS	1
Generic/Nondescript	14
Total	171

category. Any testing information that contained instructions on using a PoC application or creating a custom PoC script was designated to the PoC category. Any testing information where the functionality test was to simply build or make the code was designated to the Build/Make category. Any testing information that contained commands or instructions for testing with any Trade Federation testing harness, such as cts-tradefed or sts-tradefed, were designated to the Tradefed category. The singular set of tests specifically from the GTS testing suite were categorized separately as GTS. The remaining testing information was simply too vague or generic and was categorized separately as Generic/Nondescript, with "Existing CTS (Compatibility Test Suite) [22] Tests" being a key example of testing information we considered generic or nondescript.

From the totals listed in Table IV we see the vast majority of testing information instructs the usage of Atest or manual testing, with a handful of instances each of testing using PoC applications, building or remaking the code, tradefed testing, and a singular instance of GTS tests. As a result of this data, we chose to test using Atest as much as possible, because of its ability to be easily reproducible with very few steps.

For our purposes, we cross referenced the list of security bulletin updates containing testing information with our list of security bulletin updates containing the most frequently updated Java classes to create a list of the most frequently updated Java classes with testing information included. From that list, we extracted the subset of security bulletin updates that specifically use Atest testing for our testing purposes, as the vast majority of the rest of the testing information is simply instructing the reader that the code must be tested manually. Table V contains a summary of this set of 22 security bulletin updates and their respective AOSP versions covered. For clarification, in Table V # CFs stands for the number of common files from the list of frequently updated files in that CVE. The security bulletin updates in Table V with an asterisk are those specifically containing updates to test files, and are broken down further in Table VI. The importance of categorizing these is that, for our analysis, it is necessary to determine which tests are feasible to be executed for quantitative analysis, as well as to determine which designated tests are not applicable to our study.

TABLE V
SUMMARY OF SECURITY BULLETIN UPDATES CONTAINING ATEST INFORMATION

AV(s)	Severity	CVE Reference	# CFs
10, 11	High	CVE-2021-0486	3
9, 10, 11	High	CVE-2020-0439	1
9, 10	High	CVE-2020-0115	1
11	High	CVE-2021-0321	1
9, 10	High	CVE-2020-0098*	2
9	High	CVE-2018-9492	1
9, 10	High	CVE-2020-0099	1
9, 10, 11	High	CVE-2021-0595*	1
10	High	CVE-2019-2200	1
9, 10, 11	High	CVE-2021-0317	1
11, 12	High	CVE-2021-39619*	1
9, 10, 11	High	CVE-2021-0472*	2
9	High	CVE-2019-2122*	2
11	High	CVE-2021-0645	1
9, 10, 11	High	CVE-2021-0337	1
10	High	CVE-2020-0017	1
9	High	CVE-2019-2003*	1
12	High	CVE-2021-39693	1
10	High	CVE-2020-0121	1
9, 10, 11	Critical	CVE-2021-0687	1
9, 10, 11	High	CVE-2021-0708	1
9, 10, 11	High	CVE-2021-0683	1
9	High	CVE-2018-9582	1

F. RQ6: What kind of tests are listed in security bulletin updates to verify patch functionality?

As seen in Table V, there is a wide variety of modules to run with Atest, however a trend we see of the majority of these is that they are either Android CTS tests or frameworks specific test. The only tests listed in Table V that are neither Android CTS tests or frameworks tests are the tests for CVE-2021-0486, CVE-2021-39693, CVE-2018-9582, and CVE-2019-2091. In the case of both CVE-2021-0486 and CVE-2021-39693, their listed tests are Kotlin based tests and not regular Java tests, and therefore were unable to be run via Atest. Similarly, for both CVE-2018-9582 and CVE-2019-2091, they list tests that are part of GTS, or the GMS Testing Suite, and are also unable to run via Atest with the default setup that we use.

Aside from the aforementioned tests that are not Android CTS tests or frameworks tests, we believe we have identified that very few of the tests listed are designed to stress the correctness of their associated patch. The distinguishing factor that allows us to identify this is that the majority of listed tests are unchanged by their associated patch, thus they do not include any new unit tests specifically designed to verify the particular patch update. While 25 out of our 188 security bulletin updates contain test patches, only those 6 seen in Table VI fall within our criteria to test. It would then seem that the majority of the testing listed in security bulletin updates is not included for verifying the functionality of the patch itself, but rather for verifying the functionality of the system for regression testing purposes only.

TABLE VI
PER VERSION STATISTICS FOR SECURITY BULLETIN UPDATES WITH ATEST TESTING INFORMATION AND PATCHES TO TEST FILES

CVE Reference	TFP	AV	CBE%	CBX%
CVE-2020-0098	Activity StackTests.java, Activity TestsBase.java	9	0%	0%
		10	60%	60%
CVE-2021-0595	RootWindow ContainerTests.java	11	100%	100%
CVE-2021-39619	UserUsage StatsService Test.java	12	63.6%	54.5%
CVE-2021-0472	LockTask ControllerTest.java	9	100%	75%
		10	100%	75%
		11	100%	75%
CVE-2019-2122	LockTask ControllerTest.java	9	100%	66.6%
CVE-2019-2003	TextClassification ManagerTest.java, TextView ActivityTest.java	9	0%	0%

TABLE VII
TEST COVERAGE STATISTICS FOR CVE-2020-0098

AV	File Name	TBE	TBX	CBE	CBX	%BE	%BX
9	Activity Manager Service.java	1	1	0	0	0%	0%
10	Activity Stack.java	3	3	2	2	66.6%	66.6%
	Activity Starter.java	1	1	1	1	100%	100%
	Activity Manager Service.java	1	1	0	0	0%	0%

TABLE VIII
TEST COVERAGE STATISTICS FOR CVE-2021-0595

AV	File Name	TBE	TBX	CBE	CBX	%BE	%BX
11	Root Window Container .java	2	2	2	2	100%	100%

G. RQ7: When listed, how much of a patch is covered by the given tests?

Tables VII, VIII, IX, X, XI, XII, and XIII contain the breakdowns of coverage statistics per file across all their tested Android versions for each relevant CVE. Tables XIV, XV, XVI, and XVII contain the summaries of test coverage information across all security bulletin updates for each version tested. In the tables presented in this section, **PF**, **AV**, **CBE** (**TBE**), **CBX** (**TBX**), and **%BE** (**%BX**) denote the number of patched files, AOSP version, the covered (total) basic block entry log statements, the covered (total) basic block exit log statements, the percentage of covered basic block entry (exit) log statements, respectively. For our purposes, we tested the latest AOSP version associated with each security bulletin update, and for security bulletin updates pertaining to AOSP versions 9, 10, 11, all three versions were tested for the applicable security bulletin updates.

1) Observations from CVE-2020-0098: For the security bulletin updates referenced in Table VI, we found for CVE-2020-0098 that the test files ActivityStackTests.java and ActivityTestsBase.java are edited to include new tests specifically aimed at verifying the functionality of the changes to ActivityStack.java and ActivityStarter.java. We see in Table VII that even with these updates to the test files, not all of the changes in the patch are covered by the tests. CVE-2020-0098 is tested in AOSP versions 9 and 10, but in version 9 the patch only applied to a singular file, and as seen in Table VII, that singularly patched file is not covered by the associated tests. For CVE-2020-0098 in AOSP version 10, the same file, ActivityManagerService.java, is still not covered by the associated tests. Yet there are two more files that are patched in the security bulletin, and those two files, ActivityStack.java and ActivityStarter.java, garner 66.6% and 100% coverage by the associated tests for their individual file coverage, as seen in Table VII.

2) Observations from CVE-2021-0595: In the instance of CVE-2021-0595, while it lists AOSP versions 9, 10, and 11, only AOSP version 11 contains the actual updates to all of the listed files in the patch, including the updated test file. From Table VIII, we see that with the updated additions to RootWindowContainerTests.java, all of the patch updates in RootWindowContainer.java are covered giving a 100% entrance coverage rate and a 100% exit coverage rate.

3) Observations from CVE-2021-39619: CVE-2021-39619 lists both AOSP versions 11 and 12, and while UserUsageStatsServiceTest.java is updated in both versions, there are obviously some additional changes to either the UsageStatsTests module or to UserUsageStatsServiceTests.java between versions 11 and 12 based on the difference in coverage results in Table IX. Those two tables illustrate that while the patches to UserUsageStatsServiceTest.java in the security bulletin update do cause a majority of the file patches to be covered, for AOSP version 11 only 62.5% of patch entrances and exits are covered for UsageStatsService.java, but AOSP version 12 has 87.5% of patch entrances and exits covered for UsageStatsService.java. However, there are no other differences between the results for AOSP versions 11 and 12 for CVE-2021-39619.

4) Observations from CVE-2021-0472 and CVE-2019-2122: In both cases for CVE-2021-0472 and CVE-2019-2122, LockTaskControllerTest.java was updated to verify functionality of patches made to LockTaskController.java. For CVE-2021-0472, Table X shows consistency across all three AOSP versions this security bulletin update was tested on. Specifically, we see for each version, LockTaskController.java has 100% entrance coverage percentage and 75% exit coverage percentage. For CVE-2019-2122, as the updates to LockTaskController.java are different, as there are fewer patched blocks, similar to CVE-2021-0472 we still observe all patched entrances are covered and one patched exit is missing. The difference is seen in Table XI where there is still a 100% entrance coverage

TABLE IX
TEST COVERAGE STATISTICS FOR CVE-2021-39619

AV	File Name	TBE	TBX	CBE	CBX	%BE	%BX
11	Device Policy Manager Service .java	1	1	1	0	100%	0%
	Usage Stats Service .java	8	8	5	5	62.5%	62.5%
	UserUsage Stats Service .java	2	2	1	1	50%	50%
12	Device Policy Manager Service .java	1	1	1	0	100%	0%
	Usage Stats Service .java	8	8	7	7	87.5%	87.5%
	UserUsage Stats Service .java	2	2	1	1	50%	50%

TABLE X
TEST COVERAGE STATISTICS FOR CVE-2021-0472

AV	File Name	TBE	TBX	CBE	CBX	%BE	%BX
9	LockTask Controller .java	3	4	3	3	100%	75%
10	LockTask Controller .java	3	4	3	3	100%	75%
11	LockTask Controller .java	3	4	3	3	100%	75%

percentage, but only a 66.6% exit coverage percentage.

TABLE XI
TEST COVERAGE STATISTICS FOR CVE-2019-2122

AV	File Name	TBE	TBX	CBE	CBX	%BE	%BX
9	LockTask Controller .java	2	3	2	2	100%	66.6%

5) Observations from CVE-2019-2003: The results for CVE-2019-2003 are especially intriguing, as we see in Table XII that while TextClassificationManagerTest.java and TextViewActivityTest.java are both updated to contain changes seemingly designed specifically to test at least one of the cases, yet none of the three patched files have any coverage by the suite of tests designated for this security bulletin update. This particular security bulletin update is very similar to CVE-2020-0017, which we see in Table XIII which runs two of the same tests and sees TextClassification.java have 80% coverage of both patched entrances and exits, but it is tested on AOSP version 10.

TABLE XII
TEST COVERAGE STATISTICS FOR CVE-2019-2003

AV	File Name	TBE	TBX	CBE	CBX	%BE	%BX
9	Linkify.java	6	6	0	0	0%	0%
	Text Links Params .java	1	1	0	0	0%	0%
	Selection Action Mode Helper .java	2	2	0	0	0%	0%

TABLE XIII
TEST COVERAGE STATISTICS FOR CVE-2020-0017

AV	File Name	TBE	TBX	CBE	CBX	%BE	%BX
10	Actions Model Params Supplier .java	1	1	1	1	100%	100%
	Selection Event.java	8	8	4	4	50%	50%
	Text Classifi cation.java	5	5	4	4	80%	80%
	Text Classif ication Context.java	5	5	1	1	20%	20%
	Text Language .java	5	5	4	4	80%	80%
	Text Links.java	5	5	4	4	80%	80%
	Text Selection .java	5	5	4	4	80%	80%
	Text View.java	3	3	3	3	100%	100%
	Conver sation Actions.java	5	5	0	0	0%	0%
	System Text Classifier .java	10	10	0	0	0%	0%
	Text Classifier Event.java	1	1	0	0	0%	0%
	Text Classifi cation Manager Service .java	13	13	0	0	0%	0%

6) Observations from Additional CVEs: Amongst the remaining 16 security bulletin updates that fit our criteria to test, across each AOSP version we test, we see there is a wide range of coverage by the given tests. Table XIV illustrates the average coverage for all security bulletin updates associated with AOSP version 9. We see that for half of the associated security bulletin updates the coverage percentage of patched blocks entered and exited is 0%. However, for the other half of the updates the

percentage of patched blocks entered and exited varies from 50% up to 100%.

In contrast, the data in Tables XV, XVI, and XVII for AOSP versions 10, 11, and 12, respectively, illustrates that the majority of security bulletin updates have a higher percentage of coverage for each version in comparison to AOSP version 9. In AOSP version 10, only 5 security bulletin updates contain 0% coverage, and the remaining 8 have a coverage percentage variation of 16.6% to 100%. For AOSP version 11, only 5 security bulletin updates contain 0% coverage, and the remaining 6 have a coverage percentage variation of 50% up to 100%. Lastly, for AOSP version 12, all security bulletin updates contain some coverage, with the coverage percentage varying from 72.7% to 100% over the two updates.

7) Observations from RQ7 Results: We observed that for the 6 security bulletin updates tested which contain patches to test files, they have an average of 69.2% entrance coverage and 56.2% exit coverage. The remaining 16 security bulletin updates have an average of 32.2% entrance coverage and 28.3% exit coverage. Then across all 22 of the security bulletin updates fitting our criteria that we tested, we observe an average of 42.8% entrance coverage, and 36.6% exit coverage. These observed averages are computed as averages of the already computed coverage percentages per security bulletin update, over each version tested.

For those security bulletin updates that were tested over multiple different versions of AOSP, the majority had consistent coverage results across all versions. However, a few security bulletin updates have discrepancies across AOSP versions. Specifically CVE-2020-0098, CVE-2020-0439, CVE-2021-0317, and CVE-2021-39619. While CVE-2020-0098 and CVE-2021-39619 are discussed above, the differences in coverage results for both CVE-2020-0439 and CVE-2021-0317 is seen to be one more patched block entered and exited between AOSP versions 10 and 11.

These observations suggest that, on average, less than half of the entry points and less than half of the exit points in a given CVE are being stressed or verified by their associated tests. More specifically, even for the 6 security bulletin updates that contain patches to test files, less than 70% of entry points and less than 60% of exit points are being stressed or verified by their associated tests. This suggests a relatively significant problem with the currently disclosed testing methods for Android security framework patches.

TABLE XIV
TEST COVERAGE STATISTICS FOR ANDROID 9 SUMMARY

CVE Ref	PF	TBE	TBX	CBE	CBX	%BE	%BX
2021-0687	1	1	1	0	0	0%	0%
2018-9492	1	3	3	2	2	66.6%	66.6%
2020-0099	2	2	2	1	1	50%	50%
2019-2122	1	2	3	2	2	100%	66.6%
2020-0098	1	1	1	0	0	0%	0%
2020-0439	1	6	6	3	3	50%	50%
2021-0472	1	3	4	3	3	100%	75%
2019-2003	3	9	9	0	0	0%	0%
2021-0317	1	2	2	0	0	0%	0%
2021-0337	2	5	5	0	0	0%	0%

TABLE XV
TEST COVERAGE STATISTICS FOR ANDROID 10 SUMMARY

CVE Ref	PF	TBE	TBX	CBE	CBX	%BE	%BX
2021-0683	1	1	1	0	0	0%	0%
2021-0687	1	1	1	0	0	0%	0%
2020-0099	2	2	2	1	1	50%	50%
2020-0098	3	5	5	3	3	60%	60%
2020-0439	1	6	6	3	3	50%	50%
2021-0472	1	3	4	3	3	100%	75%
2020-0017	12	66	66	25	25	37.8%	37.8%
2021-0708	1	1	1	0	0	0%	0%
2020-0115	1	1	1	1	0	100%	0%
2019-2200	1	6	6	1	1	16.6%	16.6%
2020-0121	1	5	5	5	5	100%	100%
2021-0317	1	2	2	0	0	0%	0%
2021-0337	2	5	5	0	0	0%	0%

TABLE XVI
TEST COVERAGE STATISTICS FOR ANDROID 11 SUMMARY

CVE Ref	PF	TBE	TBX	CBE	CBX	%BE	%BX
2021-0683	1	1	1	0	0	0%	0%
2021-0687	1	1	1	0	0	0%	0%
2021-39619	3	11	11	7	6	63.6%	54.5%
2021-0321	1	1	1	0	0	0%	0%
2021-0595	1	2	2	2	2	100%	100%
2021-0645	1	1	1	1	1	100%	100%
2020-0439	1	6	6	4	4	66.6%	66.6%
2021-0472	1	3	4	3	3	100%	75%
2021-0708	1	1	1	0	0	0%	0%
2021-0317	1	2	2	1	1	50%	50%
2021-0337	2	5	5	0	0	0%	0%

TABLE XVII
TEST COVERAGE STATISTICS FOR ANDROID 12 SUMMARY

CVE Ref	PF	TBE	TBX	CBE	CBX	%BE	%BX
2021-39619	3	11	11	9	8	81.8%	72.7%
2021-39693	1	1	1	1	1	100%	100%

V. DISCUSSION

From our results we are able to gather quite a few very intriguing findings. We are able to identify the most security relevant components by identifying the most frequently updated Java classes amongst our data set. Table II illustrates this as it shows the most frequently updated Java classes, and it also illustrates which of those frequently updated files contains functions that handle security exceptions. Among the three AOSP versions, version 11 contains the maximum number (40) of updated Java classes with security exception functions. Out of these 40 updated Java classes that contain security exception functions, we see in Table II that 13 of those are listed within our most frequently updated files. Therefore, we think that Android framework Java classes containing security exception functions should be given higher priority for extensive testing.

We observed that only 25 out of our 188 total security bulletin updates contain CVE specific tests. This suggests that the majority of listed tests are included for regression testing and not to verify the new and correct functionality of the security bulletin updates.

We were able to analyze 6 out of these 25 security bulletin updates by running their associated tests, and although these contain updates specific to the patch updates we still observe below 100% entrance and exit coverage for multiple instances. As discussed in the penultimate paragraph of Section IV-G, we find that for the 6 security bulletin updates tested which contain patches to test files have an average of 69.2% entrance coverage and 56.2% exit coverage. In contrast, the remaining 16 tested security bulletin updates were observed to have a large drop off comparatively at 32.2% entrance coverage and 28.3% exit coverage. These statistics indicate that the basic block coverage is higher for cases when the security bulletin updates include updated test files specific to the patch. Although this is promising, coverage needs to improved across the board.

VI. Conclusions

In this study, we analyzed the Android framework security bulletin updates over the last 6 years. We identified the security relevant component characteristics, including usage of security exception functions, the most frequently updated Java classes and packages, and the intersection of those two sets. Our investigation reveals that only a 25 of the 188 total security bulletin updates contain patched test files specifically designed for the given update. Our investigation of a subset of these 25 security bulletin updates show that even with specifically designed tests, full coverage is not achieved (69.2% average basic block entry coverage). Across the board the coverage is even worse at an average of 42.8% basic block entry coverage.

Acknowledgements

We would like to thank Guliz Seray Tuncay and the anonymous reviewers for their feedback. This work was funded by the US National Science Foundation under the CNS-1942235 award, as well as by the US National Science Foundation Scholarship for Service (NSF-SFS) Fellowship Award.

References

[1] R. Mayrhofer, J. V. Stoep, C. Brubaker, and N. Kralevich, "The android platform security model," 2019. [Online]. Available: https://arxiv.org/abs/1904.05572

[2] S. Farhang, M. B. Kirdan, A. Laszka, and J. Grossklags, "Hey google, what exactly do your security patches tell us? a large-scale empirical study on android patched vulnerabilities," 2019. [Online]. Available: https://arxiv.org/abs/1905.09352

[3] Z. Zhang, H. Zhang, Z. Qian, and B. Lau, "An investigation of the android kernel patch ecosystem," in 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, Aug. 2021.

[4] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," IEEE Security Privacy, vol. 7, no. 1, pp. 50–57, 2009.

[5] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proceedings of the 18th ACM Conference on Computer and Communications Security, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 627–638.

[6] A. Egners, U. Meyer, and B. Marschollek, "Messing with android's permission model," in 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, 2012, pp. 505–514.

[7] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on android," in 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012. The Internet Society, 2012.

[8] G. S. Tuncay, S. Demetriou, K. Ganju, and C. A. Gunter, "Resolving the predicament of android custom permissions," in 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018. The Internet Society, 2018.

[9] R. Li, W. Diao, Z. Li, J. Du, and S. Guo, "Android custom permissions demystified: From privilege escalation to design shortcomings," in 2021 IEEE Symposium on Security and Privacy (SP), 2021, pp. 70–86.

[10] K. R. Jones, T. Yen, S. C. Sundaramurthy, and A. G. Bardas, "Deploying android security updates: an extensive study involving manufacturers, carriers, and end users," in CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM, 2020, pp. 551–567.

[11] S. Farhang, M. B. Kirdan, A. Laszka, and J. Grossklags, "An empirical study of android security bulletins in different vendors," in WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020, Y. Huang, I. King, T. Liu, and M. van Steen, Eds. ACM / IW3C2, 2020, pp. 3063–3069.

[12] D. Wu, D. Gao, E. K. T. Cheng, Y. Cao, J. Jiang, and R. H. Deng, "Towards understanding android system vulnerabilities: Techniques and insights," in Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019, S. D. Galbraith, G. Russello, W. Susilo, D. Gollmann, E. Kirda, and Z. Liang, Eds. ACM, 2019, pp. 295–306.

[13] S. Almanee, A. Ünal, M. Payer, and J. Garcia, "Too quiet in the library: An empirical study of security updates in android apps' native code," in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021, pp. 1347–1359.

[14] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ser. FSE '12. New York, NY, USA: Association for Computing Machinery, 2012.

[15] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for android apps," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, 2013, p. 224–234.

[16] R. Mahmood, N. Mirzaei, and S. Malek, "Evodroid: Segmented evolutionary testing of android apps," in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 599–609.

[17] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for android applications," ser. ISSTA 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 94–105.

[18] T. Gu, C. Sun, X. Ma, C. Cao, C. Xu, Y. Yao, Q. Zhang, J. Lu, and Z. Su, "Practical gui testing of android applications via model abstraction and refinement," in 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 269–280.

[19] "Git repositories on android." [Online]. Available: https://android.googlesource.com/

[20] "Android security bulletins : Android open source project." [Online]. Available: https://source.android.com/security/bulletin

[21] "Atest : Android open source project." [Online]. Available: https://source.android.com/compatibility/tests/development/atest

[22] "How to obtain google's gms certification for latest android devices?"