Hey Google, What Exactly Do Your Security Patches Tell Us? A Large-Scale Empirical Study on Android Patched Vulnerabilities

Sadegh Farhang *
Pennsylvania State University
smf5604@psu.edu

Mehmet Bahadir Kirdan *
Technical University of Munich
bahadir.kirdan@tum.de

Aron Laszka
University of Houston
alaszka@uh.edu

Jens Grossklags
Technical University of Munich
jens.grossklags@in.tum.de

Abstract

Android has the largest market share among smartphone platforms worldwide with more than one billion active devices. Like other platforms, security patches play a pivotal role in keeping Android devices safe from the exploitation of known vulnerabilities. Previous research efforts have documented many attacks, vulnerabilities, and defenses in the Android ecosystem. However, no previous work has studied Android vulnerabilities and their implications on consumers, public vulnerability disclosure, and the Android ecosystem together.

In this paper, we perform a comprehensive study of 2,470 patched Android vulnerabilities that we collect from different data sources such as Android security bulletins, CVEDetails, Qualcomm Code Aurora, AOSP Git repository, and Linux Patchwork. In our data analysis, we focus on determining the affected layers, OS versions, severity levels, and common weakness enumerations (CWE) associated with the patched vulnerabilities. Further, we assess the timeline of each vulnerability, including discovery and patch dates.

We find that (i) even though the number of patched vulnerabilities changes considerably from month to month, the relative number of patched vulnerabilities for each severity level remains stable over time, (ii) there is a significant delay in patching vulnerabilities that originate from the Linux community or concern Qualcomm components, even though Linux and Qualcomm provide and release their own patches earlier, (iii) different AOSP versions receive security updates for different periods of time, (iv) for 94% of patched Android vulnerabilities, the date of disclosure in public datasets is not before the patch release date, (v) there exist some inconsistencies among public vulnerability data sources, e.g., some CVE IDs are listed in Android Security bulletins with detailed information, but in CVEDetails they are listed as unknown, (vi) many patched vulnerabilities for newer Android versions likely also affect older versions that do not receive security patches due to end-of-life.

*Sadegh Farhang and Mehmet Bahadir Kirdan equally contributed to this work.

1 Introduction

Modern mobile phones have become an integral part of our lives. They are the central information hub for many users, harboring deeply personal information, and are also the conduit for many economically relevant transactions, such as mobile banking or healthcare. As a result, the focus of cybercriminals and other attackers has also shifted towards this context. Thus, it is essential to keep smartphones secure with particular emphasis given to central key functions such as the operating system (OS), like Android and iOS.

In our work, we focus on Android, the mobile operating system developed by Google and released under open-source licenses as the Android Open Source Project (AOSP). The first commercial Android device was launched in September 2008, and Android now has the largest market share among smartphone platforms worldwide with more than one billion active devices [8].

Similar to other software vendors, Google provides (monthly) security bulletins, which contain the details of patched vulnerabilities affecting multitudes of Android devices. Google has been publishing its Android security bulletins starting from August 2015 to the present [40]. Similar to Google Android security bulletins, some other vendors initiated their own security bulletins. For instance, Samsung and LG started to publish security bulletins in October 2015 and in May 2016, respectively [61, 69].

There has been extensive work to find vulnerabilities in the Android ecosystem, such as [92, 93, 101], in addition to enhancing its security, e.g., [79, 91]. Despite these efforts, we observe an increase in the number of attacks and vulnerabilities. In 2016, the total number of publicly disclosed vulnerabilities for all platforms (i.e., all vendors and product versions) reached 6,447. However, the number increased to 14,714 in 2017 and maintained this upward trend in 2018 by reaching 16,555 [72].

In this paper, we perform the most comprehensive analysis of Android vulnerabilities, to the best of our knowledge. We create a dataset containing 2,470 patched vulnerabilities with

their detailed information by scraping multiple data sources, e.g., Android security bulletins, CVEDetails, Google Gits. This dataset enables us to perform multiple analyses to better understand security in the Android ecosystem. In the following, we summarize our findings.

- We document the expected behavior that Google does not provide updates for all affected OS versions due to end-of-life (EOL), which is the point in time at which a company ceases to create any further updates (including patches) for a given OS version. However, many patched vulnerabilities are likely common among different OS versions. Therefore, OS versions that reached their EOL are vulnerable to many vulnerabilities, which are patched for newer versions. This practice puts consumers of outdated operating systems at risk.
- We find that different OS versions receive security updates for different periods of time. For example, the difference in introduction dates of versions 4.4 and 4.4.4 is about 8 months; however, version 4.4 stopped receiving updates 22 months before version 4.4.4 reached EOL. This difference in duration of receiving security support differentiates among its consumers. Consumers who update their OS are rewarded by receiving security support for a longer period of time.
- Some Android vulnerabilities originate from other resources, e.g., Qualcomm and Linux. We observe a significant delay from the Android security team to provide patches for these vulnerabilities, while Qualcomm and Linux provide and release their own patches earlier. The average delays for Qualcomm and Linux are 307.8 and 324.9 days, respectively.
- We study the patch release dates, fix-commit dates, and public repository disclosure dates for patched vulnerabilities. We find that for the majority, i.e., 94%, the patch release date occurs before or in the same month as the public disclosure date, which can be considered a prudent, secure practice. We discuss whether disclosure that predates fixing and patching places consumers at risk.
- We determine the time difference between the introduction of the first line of code associated with a vulnerability and the publication time of a security bulletin, which is called *maximum vulnerability lifetime*. We find that the average maximum vulnerability lifetime is about 1350 days, which provides an understanding of how good the state of the art security tools are in terms of detecting and fixing Android vulnerabilities.
- Even though the monthly number of patched vulnerabilities for each severity level[1] changes considerably, we find that the relative number of patched vulnerabilities for each severity level has a similar distribution mean each year.
- We observe some inconsistencies between Android security bulletins and CVEDetails. Some CVE IDs are listed in Android Security bulletins with detailed information, while in CVEDetails those CVEs often remain unknown.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. In Section 3, we pose our research questions. We describe our data collection methodology and our dataset in Section 4, which is followed by the presentation of our results in Section 5. We discuss our findings in Section 6 and limitations in Section 7. We offer concluding remarks in Section 8.

2 Related Work

Mobile devices contain lots of sensitive and private information. As a result, security should be an integral part of the Android ecosystem. Significant research efforts have been spent on investigating attack techniques on Android, finding vulnerabilities, and designing a more secure infrastructure in Android [79, 93, 98, 99]. Similar to other platforms, within the Android ecosystem security patches are developed and issued in a regular fashion to maintain device security. Contrary to previous works, we follow a different approach to investigate the overall practices related to vulnerability disclosure and security patching in the Android ecosystem by gathering and analyzing Android security bulletins. While the previous research on Android-related vulnerabilities and their implications is sparse [87], software updates, as well as Android security, have been investigated from different perspectives [77, 83, 84].

2.1 Android-Related Vulnerabilities

Most closely related to our work, Linares-Vásquez et al. [87] conducted an empirical study based on the collection of 660 Android-related vulnerabilities mined from Android security bulletins, CVEDetails and XML feeds provided by NVD. They investigated three issues. First, they studied the CWE hierarchies and the types of vulnerabilities affecting Android. Second, they studied the Android layers affected by vulnerabilities. Third, they investigated the time intervals between the introduction date of the vulnerability and its fix date. Note that they mined the patched vulnerabilities up to November 2016. On the contrary, we collect 2,470 patched vulnerabilities in Android from August 2015 up to January 2019, which is more comprehensive. Moreover, our study not only covers all of their analysis with a more comprehensive dataset, but also studies new issues. These new investigations include, but are not limited to, the duration of security support for different

[1] Google uses the name *severity ratings*. Here, we use severity levels instead of severity ratings.

versions and the delay in patching vulnerabilities originating from Linux and Qualcomm.

2.2 Software Updates

Several works investigate the delivery and installation of patches across devices. Nappa et al. [90] analyzed the life cycle of vulnerabilities in client applications by observing the deployment of patches on users' devices. They used the Worldwide Intelligence Network Environment (WINE) as their data source and found that the patching rates differ among applications. Mathur and Chetty [89] studied the issue of semi-automatic updates in Android mobile devices and the impact of user experiences on update behavior. Taking a similar research focus, Edwards et al. reported that purely automatic updates are subject to failure [78]. Considering the issues related to full security automation, human behavior and vendor policies (in our case, we mostly focus on Google) remain an integral part of the security decision-making process [74, 96].

From the user perspective, there are several works that study the role of humans regarding software updates and upgrades. Vaniea and Rashidi [97] found six stages that users go through during their software updates. These six stages are awareness, deciding to update, preparation, installation, troubleshooting, post-state of the update. Farhang et al. [81] conducted a survey to better understand the relevant factors for upgrade decisions in Microsoft operating systems by recruiting 239 participants. They studied how users perceive privacy issues associated with OS upgrade decisions, and whether security constitutes a significant decision-making factor.

Frei et al. [82] also investigated the life cycle of Microsoft and Apple vulnerabilities by defining a new metric called *0-day patch rate*. They defined it as the number of vulnerability patches that a vendor releases when the vulnerabilities are publicly disclosed. Using this metric, they compared Apple with Microsoft in terms of their security performance and concluded that while Apple shows an ascending trend, Microsoft is more stable than Apple with regard to the average number of unpatched vulnerabilities. Further, Li and Paxson [86] scraped 3,000 vulnerabilities that belong to 682 different open-source software projects and studied the patch development life cycle.

Shahzad et al. [94] conducted an exploratory measurement study of 46,310 vulnerabilities disclosed from 1988 to 2011 concerning different software vendors. Their main goals were to analyze disclosure trends, as well as the evolution of CVSS scores and so-called vector metrics such as confidentiality impact, access complexity, and availability impact. One of their findings was that the annual number of disclosed vulnerabilities has stopped increasing since 2008, which is opposite to what we are observing now in the Android context.

2.3 Economics of Software Updates and Android Security

Arora et al. [76] investigated the relation between software vulnerability disclosure and patch release time. They found that disclosure accelerates patch release, and open source vendors are quicker in releasing the patches. Alhazmi and Malaiya [75] focused on different vulnerability discovery models in operating systems and compared the results of the models with each other by performing statistical tests. One of the limitations of their model was that it did not differentiate among vulnerabilities with different severity levels.

Jo [84] proposed a model for examining competition intensity in the context of patching security vulnerabilities for free-of-charge software products. She found that an increase in market concentration improves vendor response in patching vulnerabilities. To test the model, she focused on the web browser market. Farhang et al. [80] studied the issue of competition in a different domain, i.e., the competition in the Android ecosystem and customization of different vendors' offerings which may result in security issues. To solve that issue, they proposed a regulatory fine model, which incentivizes a certain level of investment in security while decreasing the price of products.

A set of studies also investigates the ecosystem around so-called bug bounty platforms, which host vulnerability discovery programs for different companies and offer (monetary) incentives for white hat hackers to participate [85, 88, 102]. While a number of companies offering IoT and mobile services participate in bug bounty programs, Android is not a primary focus of these platforms.

3 Study Design

To have a better and systematic understanding of Android security vulnerabilities and patches from different perspectives, we focus on the following five research questions.

RQ1: ***How have severity distribution and root causes of patched Android vulnerabilities evolved over time?***

Google has published 42 monthly Android security bulletins since August 2015 until the time of data collection, January 2019. Each of these monthly security bulletins consists of several Android vulnerabilities and their patch details. To have a better understanding of these vulnerabilities, it is essential to investigate their overall distribution over time. To achieve this, we first investigate the severity levels of patched Android vulnerabilities.We also study the trend in the number of patched vulnerabilities. Next, we classify the vulnerabilities based on their Common Weakness Enumerations (CWE). This enables us to recognize what the common causes of vulnerabilities are. These results can help Android developers to identify problematic practices and areas that necessitate heightened attention to prevent such common causes of vulnerabilities.

RQ2: ***Is the duration of security support equal for different AOSP versions?***

Google has 28 Android API levels and has released 63 different AOSP versions since 2008 [1]. Most of these versions are still used by consumers. For example, at the time of writing (January 2019), the market shares of versions Froyo and Ginger are 0.02% and 0.25%, respectively [63]. However, Google stops providing security updates for each version after some time. For instance, Google's policy for Pixel devices is as follows: "Pixel phones get security updates for at least 3 years from when the device first became available on the Google Store, or at least 18 months from when the Google Store last sold the device, whichever is longer. After that, we cannot guarantee more updates" [66]. In this question, we investigate how long each Android version receives security patch updates. As a result of this investigation, we observe that Google has provided security patch updates for some versions for shorter periods of time than for other versions. For example, the difference in introduction dates of version 4.4 and version 4.4.4 is about 8 months. However, version 4.4 stopped receiving updates 22 months earlier. We also investigate the patched AOSP versions and the affected AOSP versions over time. We can demonstrate that the Android security team does not provide security patch updates for all affected versions because older versions have reached their end-of-life. Considering that most AOSP versions are still used by consumers, this practice leaves many devices and consumers unprotected.

RQ3: ***How long does it take for Google to patch vulnerabilities originating from other resources?***

Besides vulnerabilities that are specific to Android, there are vulnerabilities that affect other related key software building blocks. For instance, kernel development is driven by the Linux community, which patches kernel-related vulnerabilities separately. Qualcomm components are used in Android and Qualcomm also patches their Qualcomm-related vulnerabilities by itself. In our Android dataset, we find 1092 Qualcomm-related and 213 kernel-related patched vulnerabilities, and they all have references that point to their corresponding repositories. These repositories provide detailed information about the patch including the fix-commit date. We investigate the time gaps between the last fix-commit date and the published date on Android security bulletins. Thus, we can calculate how long it takes for Google to publish Qualcomm and Linux patched vulnerabilities in Android security bulletins.

RQ4: ***Are vulnerabilities in Android patched before their public disclosure?***

Google publishes the patched vulnerabilities on its monthly security bulletins with an exact bulletin publication date. This date can also be considered as patch release date. For example, in the January 2019 security bulletin, it is remarked that "the Android security bulletin contains details of security vulnerabilities affecting Android devices. Security patch levels of 2019-01-05 or later address all of these issues" [39]. As a result, we use a term called ***patch release date*** that indicates the publishing date of a patched vulnerability on its security bulletin as well as the patch release for that vulnerability. We also use ***public disclosure date*** to specify when vulnerability details are publicly available in public datasets like CVEDetails. Considering that some of the patched vulnerabilities have references that navigate to Git repositories, we also retrieve the last commit dates that fix the vulnerability and we refer to it as ***last commit date*** or short ***commit date***. In an ideal and secure situation, the ordering of these three notions should typically be as follows. The last commit date should be first and the public disclosure time should not be earlier than the patch release date. We investigate the actual ordering of these three notions in Android and their respective frequencies. For example, we can evaluate whether a patch is released no later than the public disclosure date of the corresponding vulnerability.

RQ5: ***How long does it take to patch a vulnerability?***

We use a term called ***vulnerability lifetime*** that indicates the time difference between a vulnerability introduction and its publishing time on the bulletin. The aim of this research question is to find the distribution of vulnerability lifetime for vulnerabilities with different severity levels. This gives us a better implicit understanding of how good the state of the art security tools are in terms of detecting and fixing Android vulnerabilities. Likewise, this analysis provides an indication whether we invest enough resources to reduce the vulnerability lifetime considering the huge market share and impact of Android devices in our daily lives.

4 Data Collection Methodology

In this section, we describe the collection process for the dataset to provide answers to the research questions posed in Section 3. The research questions shape our data collection methodology and indicate what information we need to provide answers.

Our main source of data are the Android security bulletins for the date range from August 2015 to January 2019 [40]. Note that August 2015 is also the start date for publication of the security bulletins. However, the security bulletins do not provide all our required information regarding Android vulnerabilities, like CWE types and public disclosure dates. Therefore, we also crawl the website CVEDetails [50]. In the following, we describe the details of our web scraper.

4.1 Web Scraper

Considering that the mentioned websites do not provide JSON, XML or any exportable formats for the vulnerabilities and their details, it is essential to implement a web scraper. Our scraper is implemented in Python 3.6 [68], with

the help of its libraries called BeautifulSoup [44] and Selenium Browser Automation [70]. During the scraping, there is a possibility that the server blocks the client if the server receives lots of consecutive requests from one particular IP address. We choose Selenium as it opens a web browser and navigates between the pages like a normal user does. Since it provides a small time gap between each consecutive request, it decreases the likelihood of blocking.

4.1.1 Security Bulletins Dataset

Figure 1 shows an example snippet from our JSON document, which illustrates its general structure. At the first level, we use three keys for each security bulletin object. The first key is ***timestamp***, which shows the *Unix timestamp* when a bulletin was published. The second key is ***formatted_date***, which represents *timestamp* in a human-readable time format. The last key is ***cves***, which contains the details of all vulnerabilities and their patches that are published in a particular security bulletin. Under the *cves* key, all vulnerabilities are grouped by their severity levels. There are four different severity levels in Android security bulletins: **low**, **moderate**, **high**, and **critical**. For example, if the severity level of a vulnerability is critical, then this vulnerability is placed under the *critical* key. Android has its own security team, who define their own metrics for severity levels of Android vulnerabilities [41]. As a result, these severity levels are different from commonly used severity levels such as version 2 and version 3 CVSS scores [67]. In our security bulletin dataset, we use the Android security team's severity levels.

We also scrape the following information from Android security bulletins: **CVE**, which is the ID of a vulnerability; **Type**; **Updated AOSP Versions**; **Date reported**; and **References**. Note that these fields might have different names on different security bulletins. For instance, the field *Updated AOSP Versions* has different names, such as *Affected Versions* used in August 2015 [14] and *Updated Versions* used in December 2015 [16]. To avoid confusion, we use the same name ***updated AOSP versions*** throughout this paper.

Similarly, the field *References* also has more than one name. In early security bulletins, some of the following names have been used: "*Bug(s)*" [17], "*Bug(s) with AOSP Links*" [16], and "*Bugs with AOSP links*" [18]. In the latest security bulletins, the term *References* is used. As a result, we use ***references*** throughout the paper for all above terms. Each reference is a link that navigates to a corresponding code repository. Note that there can be multiple references for a patched vulnerability that navigate to different branches of AOSP Git repositories. For instance, `CVE-2017-18159`, published in June 2018, has 9 different references [35]. In the security bulletins, each of them is listed with brackets in a separate row in the tables that consist of patched vulnerabilities. However, in early security bulletins, there are vulnerabilities whose references have multiple rows. Some of these different rows have even different severity levels and updated AOSP versions. For example, `CVE-2015-3873`, published in October 2015 [15], has 14 different separate rows. Except for the last row, all rows have the same severity level and the same updated AOSP versions and still belong to the same patched vulnerability. In such cases, where a vulnerability has more than one row, we check the severity level and the updated AOSP versions for each row. If a row has the same severity level and updated AOSP versions as previous rows, then we only add the new row's references into the existing references. Otherwise, we create a new entry for this particular row and consider this entry as a different patched vulnerability.

Another issue is that some vulnerabilities are patched on multiple security bulletins. For instance, `CVE-2016-2059` is first patched in September 2016, which is followed by another mentioning in October 2016 [24, 25]. Similarly, `CVE-2017-0391` is patched in both January 2017 [26] and June 2017. In such cases, we count them as different patched vulnerabilities in our dataset.

There are 80 vulnerabilities with these two consistency issues. 36 of them have multiple rows, however, with the same severity levels and the same updated AOSP versions with multiple references. We count each of these 36 vulnerabilities only once. In contrast, we count the remaining 44 vulnerabilities multiple times, since they are on multiple different security bulletins or have multiple references with different severity levels and/or updated AOSP versions.

Furthermore, the attributes *Date Reported* and *Type* are not even used in some of the security bulletins, like September 2018 and July 2017 [22, 37]. In particular, for the attribute *Date Reported*, Google stopped publishing it since June 2017 [28]. For this reason, in such cases, we leave the respective data fields empty. Hence, for certain analyses, we can only analyze the patched vulnerabilities that contain the necessary attributes.

We add the attributes that we scrape from CVEDetails to the key named ***details***. Under this key, there are vulnerability details such as *public disclosure date*, *CVSS score*, and *products affected*, which are added to their corresponding key names.

4.1.2 Mining Code Repositories

Since RQ3, RQ4, and RQ5 are related to patching times of the vulnerabilities, we need to find the last fix-commit date of a vulnerability and the log of all changed lines on branches of each patched vulnerability. In such cases where Google does not provide references publicly, some vulnerabilities do not have any references. For the patched vulnerabilities that have references, there are three different reference types, which link to different code repositories. The first one is the AOSP Git repository [5], which is referenced by the majority of patched vulnerabilities. The second one is the Qualcomm Code Aurora page for Qualcomm-related vulnerabilities [45].

```
{
 "timestamp": 1538344800.0,
 "formatted_date" : ISODate("2018-10-01T00:00:
     00.000+0000"),
 "cves": {
  "critical": [
      {
        "id": "CVE-2018-9490",
        "type" : "EoP",
        "updated_aosp_versions" : "7.0, 7.1.1,
            7.1.2, 8.0, 8.1, 9",
        "category" : "Framework",
        "references" : [
          {
            "name" : "A-111274046",
            "link" : "https://android.googlesource
                .com/platform/external/chromium-
                libpac/+/948d..."
          },
          {
            "name" : "2",
            "link" : "https://android.googlesource
                .com/platform/external/v8/+/
                a24543157..."
          }
        "details": {
          "cvss_score" : 0.0,
          "confidentiality_impact" : null,
          ...
        }
      }
      {
        ...
      }
    ]
  "high": [
    ...
   ]
  }
}
```

Figure 1: Example snippet from our security bulletin dataset

The third reference type is the Linux Patchwork page, which is used for kernel-related vulnerabilities [62]. Note that although there are vulnerabilities related to other vendors and third-parties (e.g., *MediaTek* and *libxml*), Google only published references to Qualcomm and Linux.

We scrape the **AOSP Git branches**, **directories of all changed files**, **the last commit dates**, and **the last commit IDs** from AOSP Git repositories. If a vulnerability has more than one reference, we scrape all of them. By collecting the above information, we can find which Android stack layer is affected by which vulnerability. We describe the details of our approach for finding the corresponding layer for each vulnerability in Appendix A.

For patched vulnerabilities that originate from Qualcomm and Linux kernel, we only scrape the last fix-commit dates. However, not all vulnerabilities related to Qualcomm and Linux kernel have Qualcomm Code Aurora or Linux Patchwork pages, respectively. For example, `CVE-2018-10882`, published in January 2019 [39], has a reference that points to a bug tracking tool called *Bugzilla*. In such cases, we use the ticket closing time.

After scraping the Android security bulletins, Git repositories and CVEDetails, we transfer all the retrieved data into a JavaScript Object Notation (JSON) document. JSON is an open-standard format that uses human-readable text to transmit data objects consisting of key-value pairs and array data types [60]. After formatting, we store the data in MongoDB [64]. For querying, displaying, and exporting our collected dataset in a JSON format, we use Studio 3T as a graphical user interface (GUI), which is a technology partner with MongoDB [71]. In total, we collect 2,470 patched vulnerabilities from Android security bulletins that are published from August 2015 to January 2019 as well as their details from CVEDetails and Git repositories. Hereafter, we refer to our collected dataset that contains all Android security bulletins and CVEDetails as **security bulletins**.

5 Results

In this section, we analyze the collected data with a focus on the posed research questions.

5.1 RQ1: Evolution of Severity Distributions and Root Causes

5.1.1 Severity Levels

We first investigate how severity levels of patched vulnerabilities evolve over time, and we examine the similarities of the severity level trend between the years. Note that we use severity levels that are defined by the Android security team [41], which are different from other used severity level calculations, such as V2 and V3 CVSS score. For instance, the Android security team classifies the severity level of a vulnerability as *critical* if one of the following conditions is met: (i) Arbitrary code execution in the Trusted Execution Environment[2], (ii) Remote arbitrary code execution in a privileged process, Bootloader, or the Trusted Computing Base[3], (iii) Remote permanent denial of service (device inoperability: completely permanent or requiring re-flashing the entire operating system). For the severity level of *high*, the criteria are as follows: (i) Local secure Boot bypass, (ii) Remote arbitrary code execution in an unprivileged process, (iii) Local bypass of user interaction requirements on package installation or equivalent behavior.

Only one of the patched vulnerabilities does not have a severity level, `CVE-2016-2842`, which is published in August 2016 with the severity level of *None** [23]. We exclude this

[2]Trusted Execution Environment is a component that is designed to be protected even from a hostile kernel.

[3]Trusted Computing Base is a part of the kernel, and it responsible for loading scripts into a kernel component.

patched vulnerability from our security bulletins dataset only for this analysis. Therefore, for our severity level analysis, we have 2,469 patched vulnerabilities. Table 1 shows the annual number of patched vulnerabilities. In 2015, the number of patched vulnerabilities is for only 5 months. In subsequent years, the number of patched vulnerabilities is at least 7 times higher than 2015.

Year	2015	2016	2017	2018	2019
No. of Vulnerabilities	94	662	939	747	27

Table 1: Annual number of patched vulnerabilities

Figure 2 shows the annual number of patched vulnerabilities for each severity level from 2015 to 2018. Figure 3 also shows the annual percentage of each severity level (as a fraction of all vulnerabilities). Note that we do not consider the patched vulnerabilities in 2019 as we scrape only the January security bulletin. As we see in Figure 2, the number of patched vulnerabilities with high severity level is increasing from 2015 to 2018. Furthermore, they are the majority among all patched vulnerabilities, except for 2015. In 2017, 507 of 939 patched vulnerabilities have a high severity level. In 2018, while the total number of patched vulnerabilities decreases from 939 to 747, the number of patched vulnerabilities with high severity level increases from 507 (54%) to 629 (84%). Similarly, the number of both moderate and critical severity levels rise slightly between 2016 and 2017. However, the number of moderate severity level patched vulnerabilities falls sharply between 2017 and 2018, from 211 to 17. In general, the number of low severity level patched vulnerabilities is close to zero. In 2015 and 2016, there are only 5 and 3 of them, respectively, and their number rises to only 7 in 2017. In 2018, we do not have *any* (reported) patched vulnerabilities with low severity level.

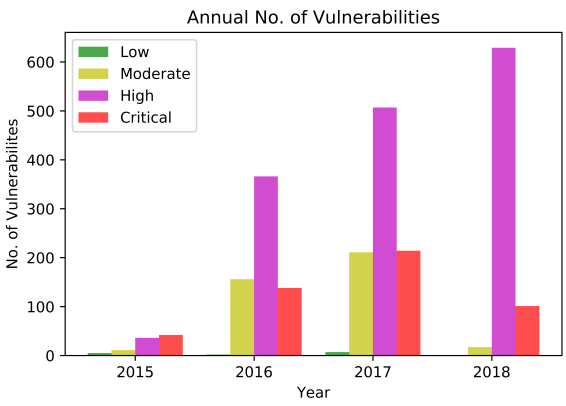

Figure 2: Annual number of patched vulnerabilities by severity levels

Figure 4 depicts the number of patched vulnerabilities for each month from August 2015 to January 2019 grouped by

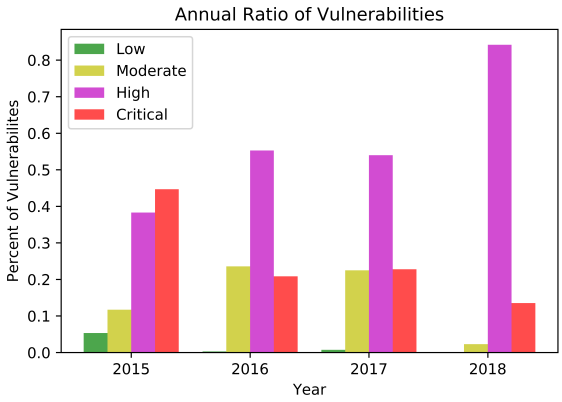

Figure 3: Annual ratio of patched vulnerabilities by severity levels

severity levels. In this figure, we observe that starting from March 2016, the high severity level patched vulnerabilities are always the most frequent. In April 2018, we see the most pronounced peak of high severity level patched vulnerabilities; to a lesser degree there is also a peak in July 2017. In both these security bulletins, there is a separate section that mentions cumulative updates of Qualcomm closed-source components. For example, in April 2018, there are 286 patched vulnerabilities that belong to the cumulative update [34]. 240 out of 286 are high severity patched vulnerabilities.Similarly, there are 94 patched vulnerabilities that belong to the cumulative update of Qualcomm closed-source components in July 2017 [29]. Table 2 and Table 3 show the mean and standard deviation values, respectively, of the monthly number of patched vulnerabilities for each severity level annually.

Severity Level	2015	2016	2017	2018
Critical	3.5	11.5	17.83	8.41
High	3	30.5	42.25	52.41
Moderate	0.91	13	17,58	1,41
Low	0.41	0.25	0.58	0

Table 2: Mean of monthly number of patched vulnerabilities for each severity level annually

Severity Levels	2015	2016	2017	2018
Critical	6.23	6.05	10.20	4.05
High	4.11	21.61	22.55	74.22
Moderate	1.37	8.89	12.13	2.67
Low	0.90	0.45	0.99	0

Table 3: Standard deviation of monthly number of patched vulnerabilities for each severity level annually

To check whether each of moderate, high, and critical severity levels has statistically significant differences in their

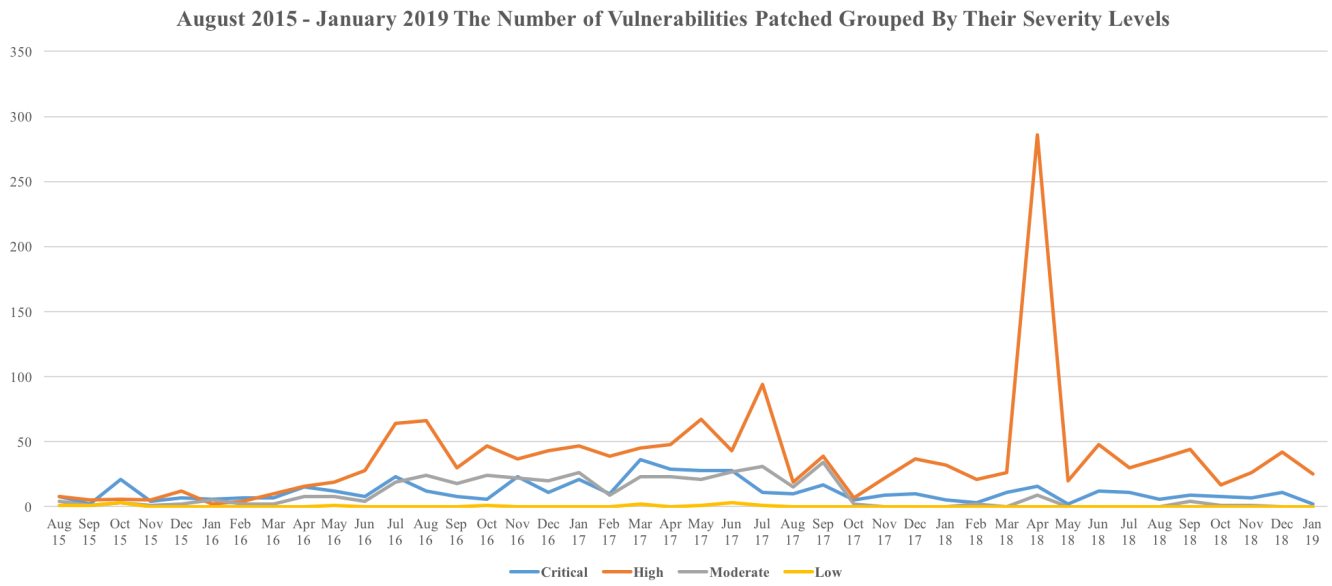

Figure 4: Number of monthly patched vulnerabilities from August 2015 to January 2019 grouped by their severity levels

Severity Level	F-value	p-value
Critical	8.85566	0.00001048
High	3.35922	0.0270576
Moderate	14.2759	0.0000012

Table 4: F and p-values of ANOVA test performed on absolute values for different severity levels of all patched vulnerabilities

Severity Level	F-value	p-value
Critical	1.36501	0.99999999
High	4.65718	0.99999999
Moderate	2.02166	0.99999999

Table 5: F and p-values of ANOVA test performed on percentage values for different severity levels of all patched vulnerabilities

monthly means between years, we perform several different Analysis of Variance (ANOVA) tests. Note that we exclude low severity patched vulnerabilities as we do not have enough samples to justify a meaningful result. Our null hypothesis for all of our tests is the same: For each of these three severity levels, the population means of their monthly patched vulnerabilities are the same in every year, given confidence level $\alpha = 0.05$. For the first ANOVA test, we consider the absolute number of patched vulnerabilities in each month for a severity level and compare between different years. Table 4 depicts the results of our ANOVA analysis. F-values are relatively high and p-values are lower than our confidence level, so we can reject the null hypothesis. This means that the mean number of patched vulnerabilities in each year is different from other years for moderate, high, and critical severity levels.

We also perform an ANOVA test on the monthly percentage of each severity level (as a fraction of all vulnerabilities) whose results are shown in Table 5. Here, we cannot reject our null hypothesis. This means that if we consider the percentages of patched vulnerabilities of a severity level (as a fraction of all vulnerabilities) instead of their absolute values, the means of each severity level are uniform over the years.

To examine whether Qualcomm-related vulnerabilities have a significant effect on the mean numbers, we also perform ANOVA tests by excluding these vulnerabilities. Based on Figure 4, there are peak points that Qualcomm patched vulnerabilities cause. Table 6 and Table 7 show the ANOVA test results for absolute and percentage values, respectively. Here, we observe results that are similar to those including the Qualcomm vulnerabilities. For absolute values, the means are not equal, while for percentage values, the population means are equal in different years. In sum, our analysis shows that even though the average number of patched vulnerabilities for each severity level changes annually, their average percentage (as a fraction of all vulnerabilities) is the same over years. Therefore, we can expect distributions with the same average percentages in the future.

5.1.2 Common Weakness Enumerations

CWE is a formal list which describes a common language to classify software security weaknesses. `Buffer Overflows`, `Structure and Validity Problems`, and `Authentication Errors` are some examples of enumera-

Severity Level	F-value	*p*-value
Critical	6.60945	0.00087576
High	11.28015	0.00087576
Moderate	14.14615	0.00087576

Table 6: F and *p*-values of ANOVA test performed on absolute values for different severity levels by excluding patched Qualcomm vulnerabilities from the security bulletin dataset

Severity Level	F-value	*p*-value
Critical	4.51910	0.99999999
High	4.51910	0.99999999
Moderate	3.99611	0.99999999

Table 7: F and *p*-values of ANOVA test performed on percentage values for different severity levels by excluding patched Qualcomm vulnerabilities from the security bulletin dataset

tion categories [52]. We analyze these enumerations because they specify what causes vulnerabilities. Therefore, we can determine what are the common software weaknesses among the patched vulnerabilities. Note that CWE information is not indicated on security bulletins. Thus, we use the field *CWE ID* and its details, which we scrape from the web page CVEDetails.

Among the 2,470 patched vulnerabilities, 109 do not have any details listed on the CVEDetails web page. In addition, 58 vulnerabilities have details but do not have any CWE. After excluding these, we continue our analysis with the remaining 2,303 patched vulnerabilities.

Among the common weakness enumeration categories, there is a parent-child hierarchy [53]. For instance, `CWE-306: Missing Authentication for Critical Function` is a child of `CWE-285: Improper Authorization`. Furthermore, this relation can be one-to-many. In other words, there are enumerations that are a child of more than one other enumeration. For instance, `CWE-358: Improperly Implemented Security Check for Standard` is a child of both `CWE-693: Protection Mechanism Failure` and `CWE-254: Security Features`. Moreover, `CWE-693` is a child of `CWE-254: Security Features`. Hence, `CWE-254` should also be considered when a vulnerability belongs to the CWE category of `CWE-358`.

For a concrete example, consider `CVE-2017-0757`, which was published in September 2017 [30]. According to the CVEDetails web page, this vulnerability has the enumeration of `CWE-284: Access Control (Authorization) Issues`. Since `CWE-284` has three different parent enumerations, which are `CWE-693: Protection Mechanism Failure`, `CWE-264: Permissions, Privileges and Access Control`, and `CWE-664: Improper Control of a Resource Through its Lifetime`, we also consider these three parent enumerations.

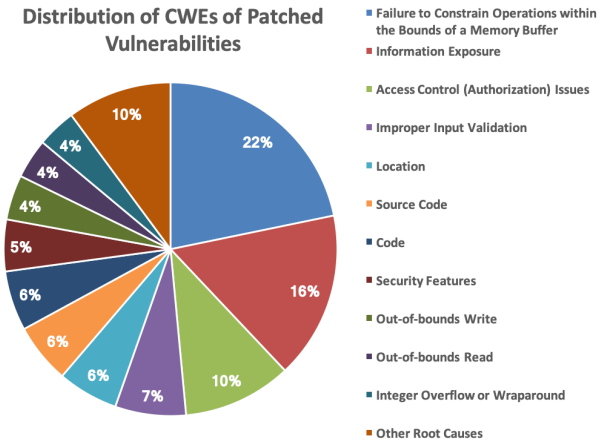

Figure 5: Distribution of CWEs of Patched Vulnerabilities

The overall distribution of CWE categories among the Android patched vulnerabilities and the annual percentage of the four most frequent CWE categories are plotted in Figure 5 and Figure 6, respectively.

Based on Figure 5, the four most frequent CWE categories make up more than 50% of all the vulnerabilities patched. The reason for patching mostly these CWE categories might be that the developers—especially those who lack coding experience or familiarity with language features—tend to introduce such weaknesses quite often.

Figure 6 shows the annual percentages of the four most frequently patched CWE categories. According to Figure 6, in contrast with the stability of the severity level ratios, there is more variation in the occurrence of the four most frequent CWE categories. For instance, although `CWE 119: Failure to Constrain Operations within the Bounds of a Memory Buffer` is the most frequently patched category in 2015 and 2018, it is only the second most frequent one in 2016 and 2017.

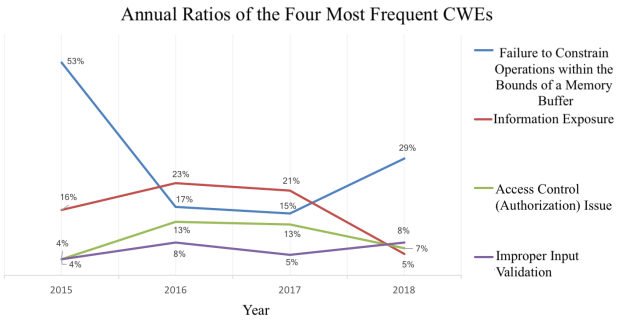

Figure 6: Annual ratios of the four most frequent CWEs

5.2 RQ2: Security Support Duration

The Android Open Source Project (AOSP) is a ready to be released version of Android. The source code of AOSP can be customized and adapted by original equipment manufacturers (OEMs) to run on their devices [1]. In this subsection, we investigate the number of patched vulnerabilities that affect each AOSP versions. Android has 28 different API levels and 63 different AOSP versions. In our security bulletins dataset, there exist 15 different AOSP versions. Note that Android was released in 2008 and the first security bulletin was published in August 2015. Thus, some AOSP versions have reached their end of life before the first bulletin, so we only see a limited number of AOSP versions in the Android security bulletins. Further, we exclude AOSP version 6.1 from our analysis because we only see two patched vulnerabilities for this version: `CVE-2016-2496` and `CVE-2016-3843`, which were published in June 2016 and August 2016, respectively [21, 23]. Therefore, in our analysis, we consider only 14 different AOSP versions.

Early Android security bulletins do not specify AOSP versions but instead use the terms *5.1 and below*, *6.0 and below*, *5.0 and above*, and *6.0 and above*. As a result, we first need to map the terms *below* and *above* to AOSP versions. We use a conservative approach which is as follows: For the term *X and below*, we consider version $\mathcal{X}$ as well as versions that were introduced earlier than version $\mathcal{X}$. For the term $\mathcal{X}$ *and above*, on the other hand, we consider version $\mathcal{X}$ as well as versions that were introduced later than version $\mathcal{X}$ but earlier than the security bulletin that we are studying. For example, we replace the expressions *5.1 and below* and *5.0 and above* with {*4.4, 4.4.4, 5.0, 5.0.2, 5.1*} and {*5.0, 5.0.2, 5.1*}, respectively.

As we mentioned earlier, there are 2,470 patched vulnerabilities. Among these, 889 have an entry in the field for updated AOSP versions in the Android security bulletins. Figure 7 shows the number of vulnerabilities for each AOSP version from August 2015 to January 2019.

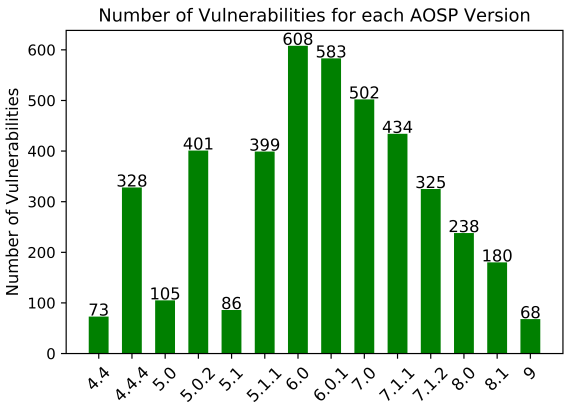

Figure 7: The number of patched vulnerabilities for each AOSP version

5.2.1 Release Date and Update Duration

In this subsection, we conduct pairwise comparisons for consecutive AOSP versions focusing on the gap in release time between the two versions vs. the gap in the time when the final patch for a particular version was released. If all versions are treated equally in terms of support, we would expect that these two gaps are identical. Also, we would expect that the data is similar over time for different pairs of versions.

In our analysis, we exclude the versions 7.0, 7.1.1, 7.1.2, 8.0, 8.1, and 9 because these versions are still receiving updates at the time of our data collection (January 2019). The result of our analysis is presented in Figure 8. In this figure, the horizontal axis shows the time difference of end-of-life (EOL) between two versions in months. Note that we calculate the EOL based on the last time that we see an update for an AOSP version on Android security bulletins. The vertical axis represents the time difference in the release time of two versions in months. Each point indicates these two differences for two versions. The blue line represents the point where the time difference of EOL and release time for the two versions are equal.

According to Figure 8, the pair consisting of version 5.0.2 and version 5.1.1 is the only one located on the blue line. Version 5.1.1 was introduced 4 months after version 5.0.2. Also, version 5.0.2 stopped receiving updates 4 months sooner than version 5.1.1.

Points that are under the blue line are those pairs of versions where the newer version continues receiving updates comparatively longer than the difference in their release dates would indicate.

There are three points in that region. We describe each of them in detail in the following.

- The difference in introduction date of version 4.4 and 4.4.4 is about 8 months. However, version 4.4 stopped receiving updates 22 months sooner.
- Version 5.0.2 was introduced one month after 5.0. Version 5.0.2 continued receiving updates 21 months after the last update date of version 5.0.
- Version 5.1 was introduced one month before version 5.1.1. The former stopped receiving updates 27 months sooner than version 5.1.1.

The points above the blue line are those pairs of versions where the gap between release times is larger than the gap between EOL times. We have 4 pairs of such situations that are also plotted in Figure 8.

- Version 6.0 and version 6.0.1 stopped receiving updates at the same time while 6.0.1 was introduced 2 months after version 6.0.

- Version 5.0.2 was introduced 6 months after version 4.4.4. However, version 4.4.4 stopped receiving updates just one month sooner than version 5.0.2.
- Version 4.4.4 was introduced 18 months sooner than version 6.0.1. But, it stopped receiving updates just 10 months before version 6.0.1.
- Version 6.0.1 was introduced one year after version 5.0.2. But, version 6.0.1 continues receiving updates only 9 months after version 5.0.2.

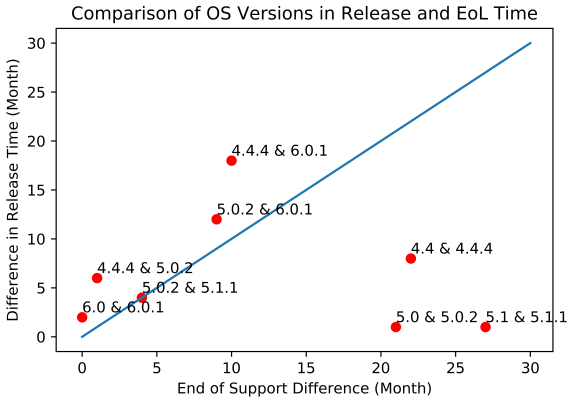

Figure 8: Pairwise comparison of time differences in release time and end-of-life in Android security bulletins. Here, the blue line shows that the difference in release time and end-of-life is equal for two versions

For our sample, we can deduce that newer pairs of versions tend to have shorter gaps between EOL times in comparison to their release time gaps. In contrast, service for some older OS versions was abandoned comparatively early (i.e., for 4.4, 5.0, and 5.1), when compared to the gaps in release times with their successors.

Please note that our observations are based on a reduced sample size. On the one hand, AOSP versions 7.0 and higher are still receiving security updates and we cannot make any claim about them. On the other hand, Google started Android security bulletins in August 2015 and does not provide information before that.

5.2.2 Updated AOSP Versions vs. Affected Versions

In our security bulletin dataset, there are 782 CVEs that have AOSP versions on both the Android Security bulletin and CVEDetails. Among these 782 CVEs, different OS versions are listed on the bulletin and on CVEDetails for 483 of them. In this part, we do not focus on the quantitative analysis of these differences between Android security bulletins and CVEDetails. Rather, we focus on two examples and the implications of these differences.

The first example that we study is `CVE-2016-5348`, which was mentioned in the October 2016 Android security bulletin [25]. According to the Android security bulletin, the updated AOSP versions are as follows: 4.4.4, 5.0.2, 5.1.1, 6.0, 6.0.1, 7.0. However, according to CVEDetails, the affected products for this CVE are as follows: 4.0, 4.0.1, 4.0.2, 4.0.3, 4.0.4, 4.1, 4.1.2, 4.2, 4.2.1, 4.2.2, 4.3, 4.3.1, 4.4, 4.4.1, 4.4.2, 4.4.3, 4.4.4, 5.0, 5.0.1, 5.1, 6.0, 6.0.1, 7.0. As we can see, the affected products in CVEDetails are more comprehensive and contain some older versions that are not in the list of updated AOSP versions in Android security bulletins, such as 4.0. Hence, this shows that Google does not provide security updates for all affected versions.[4] The underlying reason is that a version has reached its end-of-life and does not receive security patch updates anymore. Therefore, the older versions of Android are at risk.

The second example is `CVE-2017-0807`, published in December 2017 [33]. On Android security bulletin, the updated AOSP versions are listed as follows: 5.1.1, 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2. However, in CVEDetails, the affected products are mentioned as follows: 4.0, 4.0.1, 4.0.2, 4.0.3, 4.0.4, 4.1, 4.1.2, 4.2, 4.2.1, 4.2.2, 4.3, 4.3.1, 4.4, 4.4.1, 4.4.2, 4.4.3, 4.4.4, 5.0, 5.0.1, 5.0.2, 5.1, 5.1.1, 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2. Similar to the first example, Google does not provide security updates for all affected versions that are mentioned on CVEDetails, which puts many older devices at risk.

In the following subsection, we investigate the common CVEs among different versions and the possibility that some of them are applicable to older versions.

5.2.3 Common Vulnerabilities Among AOSP Versions

Before starting the analysis of common vulnerabilities among different versions, we introduce the term ***latent CVEs*** for an AOSP version. The latent CVEs for an AOSP version are all CVEs that affect that version but do not receive security updates like newer versions since the AOSP version has reached its end-of-life.

We analyze the number of common vulnerabilities among different AOSP versions based on the information in Android security bulletins (see Figure 9), however, paying attention to the fact that different versions receive security updates for a different amount of time. The Android security bulletins provide related information about patches, but note that the number of vulnerabilities for different versions varies a lot, see Figure 7. As a result, it is highly likely that some Android versions that already receive common patches have even more common vulnerabilities than is reflected in the bulletins. This

[4] We assume that information on the Android security bulletins correctly reflects the released patches. It is possible that some information about patched vulnerabilities is not communicated correctly on the Android security bulletins. In particular, that information about released patches for versions that have reached EOL is incorrect. However, we have found no evidence of such practices so far.

suggests that many CVEs for a newer version are also applicable for older versions. In the following, we consider the information related to version 4.4 for further analysis.

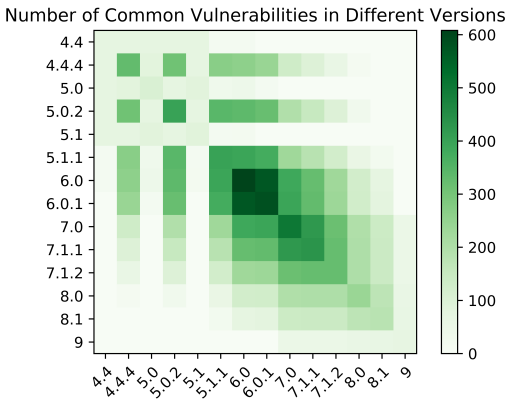

Figure 9: Heatmap of common vulnerabilities among the 14 AOSP versions

To understand latent vulnerabilities in version 4.4, we limit ourselves to versions 4.4.4, 5.0, 5.0.2, 5.1, 5.1.1, and 6.0 for comparison purposes. The reason why we do not consider version 6.0.1 and later versions is that we only see updates for version 4.4 until December 2015. Version 6.0.1, on the other hand, was introduced on December 7, 2015. Hence, it is obvious that there should be no common updates between 6.0.1 and versions after that with version 4.4 in our dataset. Note that version 4.4 has 73 patched vulnerabilities.

Version 4.4.4 has 328 patched vulnerabilities in total and 70 of them are in 2015. All these 70 vulnerabilities are common with version 4.4, while version 4.4 has 3 vulnerabilities which are not reported as common with version 4.4.4. On the other hand, version 4.4.4 continues receiving updates 22 months longer than version 4.4. Therefore, based on our previous discussion and definition of latent CVEs, it is likely that most of the remaining vulnerabilities in version 4.4.4 are also applicable to version 4.4.

The above observation might suggest that it is only applicable to sub-versions. To extend our analysis to more significant version changes, we take AOSP 4.4 and 5.x versions as examples. All patched vulnerabilities of version 4.4 are common with version 5.1. With respect to versions 5.0 and 5.0.2, all CVEs are also common (except 1 for 5.0, and 3 for 5.0.2). On the other hand, these three 5.x sub-versions have more vulnerabilities in the observed time period. Therefore, these three sub-versions of version 5 Android have almost all version 4.4 and 4.4.4 vulnerabilities, but also many other CVEs which are not reported as common with the older Android versions (which had an earlier EOL). In other words, version 4.4 CVEs are a subset of the vulnerabilities of versions 5.0, 5.0.2, and 5.1 (while all these versions received updates). This also suggests that many vulnerabilities that were patched later are likely common among versions 4.4 and 5.x.

Based on the exploratory analysis above, many CVEs are likely common among different versions. Once Google stops providing patches for older versions, publishing Android security bulletins provides information to attackers about possible vulnerabilities in these older versions. If the market share of older versions is still non-trivial, this may place many users at risk.

5.2.4 Unknown CVE ID

There are 107 CVEs in the Android security bulletins that do not have corresponding entries in CVEDetails. For these CVEs, the CVEDetails website reports "Unknown CVE ID," which means that the corresponding CVE has been reserved, but no information has been provided for it. Other public data sources also return similar results. For instance, `cve.mitre.org` describes these unknown CVE IDs as follows: "This candidate has been reserved by an organization or individual that will use it when announcing a new security problem. When the candidate has been publicized, the details for this candidate will be provided." We believe that this discrepancy is due to the delay in updating these publicly available datasets. To evaluate the role of such delay, we study CVEs in Android bulletins that have unknown CVE IDs. Table 8 shows the number of CVEs with unknown CVE IDs for each year from 2015 to 2019. Most of these CVEs are in bulletins from 2018 and 2019. Note that for 2019, we have only the January bulletin. In this month, there are 27 CVEs in total and only 7 of them have corresponding information on the CVEDetails website. Moreover, all of these 7 CVEs are related to the kernel component, which means that these CVEs are not specific to Android but to Linux.

Year	2015	2016	2017	2018	2019
No. Of CVEs	0	5	6	76	20

Table 8: Number of CVEs with unknown CVE ID in CVEDetails website for each year

5.3 RQ3: Vulnerabilities Originating from Qualcomm and Linux

To study how long it takes for Google to patch vulnerabilities in components from other vendors and organizations, we focus on kernel and Qualcomm-related vulnerability patches that affect Android. In total, there are 1,092 Qualcomm-related and 210 kernel layer patches in the Android security bulletins. However, only 414 Qualcomm and 144 kernel layer patched vulnerabilities have references. For these vulnerabilities, we calculate the time difference of the published date on the respective Android security bulletin and the fix-commit date, which is shown in Figure 10. According to this figure, there is only one patched vulnerability, `CVE-2017-8281`,

whose publication date (September 2017 [30]) precedes its fix-commit date (December 2017 [46]). For all other vulnerability patches, the time difference is positive (see Figure 10). In other words, both Qualcomm and Linux patch vulnerabilities before Google publishes them on its security bulletins. The time differences vary mostly between roughly 120 and 450 days (i.e., between 4 months and 15 months). Table 9 shows the mean and standard deviation of these time differences for both Qualcomm and kernel layer patched vulnerabilities. The mean values are 307 and 324 days for Qualcomm and kernel layer patches, respectively, and the standard deviations are relatively high for both. These results indicate that there exists a considerable delay from Google to provide patches for vulnerabilities originated from Qualcomm and Linux. This issue is even more daunting considering that these fix-commit dates are publicly available. Hence, this delay from Google puts many devices at risk. In other words, some of these vulnerabilities originating from Qualcomm and Linux may be interpreted as zero-day vulnerabilities for Android devices.

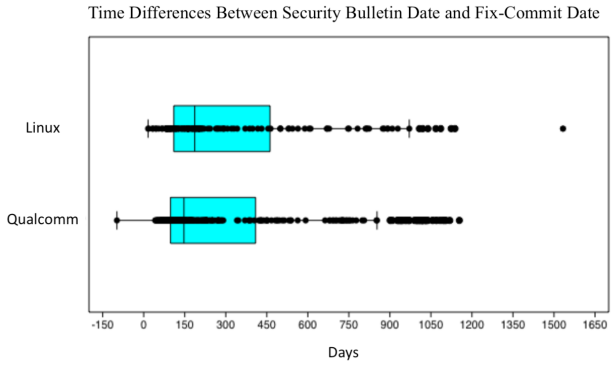

Figure 10: Time differences for patched vulnerabilities between the published dates in the Android security bulletins and fix-commit dates of both Qualcomm and Linux patches.

Vendor	Mean	Standard Deviation
Qualcomm	307.80	323.53
Linux	324.86	310,54

Table 9: Mean and standard deviation of the time difference between the Android security bulletin time and the fix-commit date for Qualcomm and Linux patched vulnerabilities

We perform a *Mann-Whitney U* test to see whether there exists a statistical difference between Qualcomm and Linux when Google handles the vulnerabilities originating from them. Since the p-value of the test result is 0.033 and is lower than our confidence level which is $p = 0.05$, these two distributions are statistically different from each other. Therefore, this shows that Google handles vulnerabilities originated from open source platforms differently from the closed source platforms, i.e., Qualcomm. In this case, the mean time for Linux is higher than for Qualcomm. The underlying reason might be the more open, but diverse, software community in the Linux ecosystem, which corresponds to a longer time for Google to identify and address relevant vulnerabilities. As a result, it is crucial for the Android security team to closely monitor the Linux-related CVEs.

5.4 RQ4: Public Disclosure vs Patch Release

To investigate this question, we compare three different times. The first one is the **patch release date**, which indicates when a vulnerability patch is published on an Android security bulletin. The second one is the **public repository disclosure date** representing when a vulnerability detail is available in public repositories like CVEDetails. The third one is the **last commit date** indicating when the last fix-commit is made. Having the first commit and the bug creation date would also help us to understand the entire timeline of a vulnerability. However, since these dates are not publicly available, we focus on the aforementioned three dates. The relation between these three dates in an ideal setting should be as follows. The first date should be the last commit date, which should then be followed by the patch release date. The patch release date is the point in time where Android releases its security patches. If a public repository disclosure date is sooner than the patch release date, this may place many Android devices at risk. As a result, the public repository disclosure date should be after the patch release date.

We limit our analysis to those patched vulnerabilities that have (only) AOSP Git repository references. In other words, we do not consider other external references/repositories like Qualcomm and Linux, since those vulnerabilities are patched in other repositories in addition to AOSP Git repositories. Note that a patch release date does not exactly correspond to the time of disclosing a patched vulnerability. Only 632 patched vulnerabilities fall into this criterion, i.e., having AOSP Git repository references. Also note that a patched vulnerability can have multiple references. Therefore, the time sequence can change since each reference has its own last fix-commit date.

Given that, we separately analyze the following two different groups: Patched vulnerabilities that have only one reference and the others that have more than one reference. For the public repository disclosure date, we use the date that is indicated as *publish date* on the CVEDetails web page. Our level of granularity to capture these three dates are year and month. Therefore, we do not consider the exact calendar day of release/publication in our analysis.

In our analysis, we use the symbols $\mathbb{B}$, $\mathbb{C}$ and $\mathbb{D}$ to denote the *patch release date*, *fix-commit date*, and *public repository disclosure date*, respectively. To express the sequence of these three dates, we separate them with "-" if they happened at different times. The date on the left side of "-" is earlier than

the date on the right side. For example, $\mathbb{C}-\mathbb{B}-\mathbb{D}$ means that the commit occurred first; then, the patch was released, which was later followed by public repository disclosure. If two dates are not separated by "-" (e.g., $\mathbb{BC}$), then they are in the same month. For instance, $\mathbb{BC}-\mathbb{D}$ means that patch release and last commit are in the same month, and the public repository disclosure is after them. Table 10 shows the frequency of all the time sequences that we observe in the Android ecosystem.

Time Sequence	Frequency
$\mathbb{BC}-\mathbb{D}$	1
$\mathbb{C}-\mathbb{B}-\mathbb{D}$	66
$\mathbb{C}-\mathbb{D}-\mathbb{B}$	3
$\mathbb{C}-\mathbb{BD}$	530
$\mathbb{CBD}$	2
$\mathbb{D}-\mathbb{C}-\mathbb{B}$	19
$\mathbb{DC}-\mathbb{B}$	11

Table 10: The frequencies of the time sequences of patched vulnerabilities that have only one AOSP reference

According to Table 10, of 632 patched vulnerabilities, only 66 followed the ideal time sequence $\mathbb{C}-\mathbb{B}-\mathbb{D}$. However, 530 of them follow a near ideal sequence, the only difference being that the patches are released *and* publicly disclosed in the same month. Therefore, 94% of the vulnerabilities are patched no later than their disclosure date in public repositories. The remainder, on the other hand, have different timelines. The most problematic one is the vulnerability being disclosed in public repositories and then receiving a patch, which is represented by $\mathbb{D}-\mathbb{C}-\mathbb{B}$ and $\mathbb{DC}-\mathbb{B}$ in our dataset (4%). For instance, `CVE-2017-6983` was published in September 2017 [30] and its last commit date is on August 2018. However, it was disclosed in public repositories on May 2017. Similarly, `CVE-2017-13078` was published in November 2017 [32], but its disclosure date in public repositories and its last commit data are in October 2017.

Similarly, Table 11 shows the frequencies of time sequences for patched vulnerabilities that have more than one AOSP reference. In other words, in this table, each patched vulnerability has more than one last commit date. Based on Table 11, the majority of the patched vulnerabilities still follow the same timeline. Sequences $\mathbb{C}-\mathbb{BD}$ and $\mathbb{C}-\mathbb{B}-\mathbb{D}$ contain 304 out of 321, i.e., 94% of total commits that belong to 125 different patched vulnerabilities. This shows that for the majority of patched vulnerabilities we do not see potentially dangerous practices. Particularly, the disclosure date is not sooner than the patch time. `CVE-2014-6060`, published in April 2016, is the one vulnerability corresponding to $\mathbb{C}-\mathbb{D}-\mathbb{B}$ [20]. This vulnerability was disclosed in public repositories in September 2014, and one of its two references has June 2014 as the last commit date. The other reference is in September 2014, which is the same as the public repository disclosure date. In addition, vulnerability `CVE-2014-6060` is the only one that fits more than one time sequence. Since the first commit is in June 2014, it fits $\mathbb{C}-\mathbb{D}-\mathbb{B}$. It also fits $\mathbb{CD}-\mathbb{B}$ because the second commit is in September 2014, which is the same as its public repository disclosure date. The other vulnerabilities fit only one time sequence each.

Time Sequence	Frequency
$\mathbb{C}-\mathbb{B}-\mathbb{D}$	34
$\mathbb{C}-\mathbb{D}-\mathbb{B}$	1
$\mathbb{C}-\mathbb{BD}$	270
$\mathbb{D}-\mathbb{C}-\mathbb{B}$	15
$\mathbb{DC}-\mathbb{B}$	1

Table 11: The frequencies of the time sequences among the patched vulnerabilities that have more than one AOSP reference

Considering these two tables, the majority falls into $\mathbb{C}-\mathbb{B}-\mathbb{D}$ and $\mathbb{C}-\mathbb{BD}$ which is a secure practice. However, there are some exceptions. For example, `CVE-2014-6060` follows the following two time sequences: $\mathbb{C}-\mathbb{D}-\mathbb{B}$ and $\mathbb{CD}-\mathbb{B}$. For this patched vulnerability, the difference between disclosure date in public repositories and patch release date is around 1.5 years. However, these two aforementioned tables do not indicate the actual time gaps. Therefore, we further analyze the last commit, public repository disclosure, and patch release dates. The main goal is to determine the distribution of time gaps between these three points in time. Figure 11 shows these dates for all patched vulnerabilities that have public repository disclosure dates and have at least one AOSP reference. Note that, if there is more than one reference, we calculate the time differences for each of them. For example, if a patched vulnerability has two references, there are two different time results that specify the comparison of both patch release date and disclosure date in the public repository. Table 12 shows the means and standard deviations of the results represented in Figure 11. For the analysis of **disclosure date of the public repository and patch release date**, we analyze 758 patched vulnerabilities. For both analysis of **patch release date - last commit** date and **disclosure date of public repository - last commit date**, we analyze 954 patched vulnerability references. The reason for having more patched vulnerabilities for the last two analyses is due to multiple references for some patched vulnerabilities.

Time Gap Analyze	Mean	Standard Deviation
Disclosure Date - Patch Release date	-2.43	68.2071
Disclosure Date - Last Commit Date	54.26	81.5345
Patch Release date - Last Commit Date	61.84	43.7562

Table 12: Means and standard deviations of time differences among patch release date, last commit date, and disclosure date in public repository

According to the Figure 11, there are no negative values

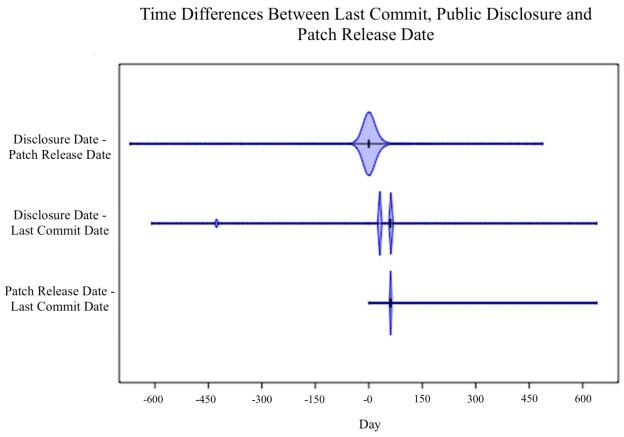

Figure 11: Time gaps between the last commit, publicly disclosure and patch release dates

for the time differences between the patch release date and the commit date. In other words, patch release dates occur after the commit date for all patched vulnerabilities. This is expected since a patch release date indicates when a patch is available for a particular vulnerability; it should also occur after its last fix-commit date. The mean of this time gap is 61.84 days. In other words, for publishing a vulnerability patch after making the last fix-commit, Google spends an average of around 62 days (2 months). However, there are some extreme cases. The longest time gap is 640 days which belongs to `CVE-2014-6060`, which we already mentioned above. The second longest time gap is 639 days belonging to `CVE-2016-1621`. It is published in March 2016 [19] and disclosed in a public repository in the same month. However, it has 3 different references. Although two of them have the same last commit dates (January 2016), the third one has a date of June 2014.

For the time differences between the disclosure date in a public repository and the patch release date, even though we see large variation, from lower than -600 days to around 488 days, the majority of them, i.e., for 639 vulnerabilities, occurs at the same time. In other words, 84% of them are disclosed at the same time in a security bulletin as in a public repository. 10% of them have positive values which means that they are disclosed in public repositories after the patch release date. In total, around 95% of them are disclosed in a public repository after or at the same date as the patch release date. The mean of this time difference is equal to -2.43, i.e., very close to zero. Around 5% of them have a negative value. In other words, they are publicly disclosed before Android published them on its security bulletins, which is the same date when Google provides its patches. Even though this occurs for a limited number of vulnerabilities, it is a potentially dangerous case which puts many devices at risk considering that a vulnerability has been published publicly but the patch has not been provided.

Lastly, we investigate the time difference of disclosure time in a public repository and the last commit date. 94% of them have a positive value meaning that they are publicly disclosed in a repository after the last commit is made. For 66% and 12% of our total samples, the time differences are 60 and 30 days, respectively, which we can also see in Figure 11. The mean value for this time difference is equal to 54.26. Hence, it takes around 2 months for public repositories to disclose a vulnerability after its last commit date. Note that for 3%, i.e., 34 of them, the disclosure time in a public repository is even sooner than the last commit date. This means that a vulnerability has been disclosed in a public repository while the fix has not been yet finalized. This gives an attacker an advantage to compromise Android devices.

Based on our above analysis, for the majority of vulnerabilities, these three time sequences follow the pattern which can be considered secure. However, there are some instances that put the security of Android devices at risk. All of these instances show that a vulnerability has been introduced in public repositories when the patch is not available or even before the last fix-commit in Google Git. Even though these instances are not common, i.e., about 5%, the potential consequences might be significant. It follows that we need better coordination between Google and public repositories with respect to their time management in announcing vulnerabilities.

5.5 RQ5: Vulnerability Lifetime

Vulnerability lifetime means the time difference between the introduction of vulnerability in the code and when the vulnerability is eventually patched [87]. It is difficult to find an actual patching time for Android vulnerabilities, i.e., when Google releases the security patches for a vulnerability. For this reason, the published time of a security bulletin gives the upper bound for when a patch is released; and we consider this as the patching date. In order to calculate the introduction time of a vulnerability in the code, we use an algorithm called SZZ [95]. The general idea behind the SZZ algorithm is to identify the changed lines for fixing a bug and then identify when these lines were added for the first time. By doing so, first, we manually clone all the AOSP Git repository branches being used for patching the vulnerabilities.

After cloning the branches, for each reference, we take the commit ID. With this commit ID, we issue the command *diff* [55] under the corresponding branch in order to detect which lines are changed from the previous revision to last revision. Since the algorithm relies on only the line deletions, we have to find when these deleted lines were added in the first place. Hence, we use the command *annotate* on the previous revision for each changed file where the changes were made [54]. Following this process, we can find the first time when the lines were added to the code. Note that there can be

more than one deleted line. Furthermore, it is possible that each deleted line is added at different times. As a result, we use the terms ***maximum lifetime*** and ***minimum lifetime*** for better understanding of this time interval. These terms are in agreement with the original SZZ paper [87]. The maximum lifetime shows the time interval between the line addition time of the first deleted line and the time of publishing the vulnerability patch on the security bulletin. On the other hand, the minimum lifetime is the time interval between the line addition time of the last deleted line and the time of publishing the vulnerability patch on the security bulletin.

As mentioned before, this algorithm only checks the deleted lines. As a result, it excludes the cases where only additions are made. To identify that, a static code analysis must be done which is not the focus of this work. Besides, we also exclude the patched vulnerability that do not have any references. After excluding all of them, we have 549 patched vulnerabilities for which we are able to identify the maximum and minimum lifetimes. Among them, we notice 133, 300, 112, and 4 patched vulnerabilities with critical, high, moderate, and low severity levels, respectively. The result of our analysis for both maximum lifetime and minimum lifetime is represented in Figure 12.

As we see in Figure 12, both minimum and maximum patched vulnerability lifetimes are very high. In particular, there are patched vulnerabilities with high and low severity levels that have a maximum vulnerability lifetime of more than 6000 days, which is longer than the lifetime of Android. The reason is that the algorithm also checks files and lines that might be irrelevant for the vulnerability like a log, build files and even comments. For instance, if the fix-commit includes a comment deletion, then the algorithm takes this into consideration. Therefore, the above outcomes can occur, for example, when a build file or log file has not been changed for 7-8 years and then has been changed in the last fix-commit. Further, since this issue might also happen on external branches that have older commit histories than Android AOSP, we can observe these outliers. In general, the minimum patched vulnerability lifetimes fluctuate between 300 and 800 days, which is still high. On the contrary, the maximum vulnerability lifetimes vary from 700 to 2200 days (including the aforementioned outliers).

Table 13 and Table 14 report the mean, and standard deviation results for both maximum and minimum vulnerability lifetimes.

In order to see whether these data are statistically different, we perform a *Mann Whitney U* test for all minimum and maximum vulnerability lifetime data with each other. In other words, each minimum/maximum vulnerability lifetime set is compared to another dataset of a minimum/maximum vulnerability lifetime with different severity levels. Our null hypothesis, H_O, is that all minimum/maximum vulnerability lifetime datasets are equally distributed. Table 15 and Table 16 depict the p-values of the Mann-Whitney U tests.

Severity Rankings	Mean	Standard Deviation
All Vulnerabilities	1350.2	981.69
Critical	1254.49	860.21
High	1366.49	1042.53
Moderate	1386.66	949.10
Low	2295.25	391.20

Table 13: Mean and standard deviation values of maximum vulnerability lifetimes

Severity Rankings	Mean	Standard Deviation
All Vulnerabilities	882.78	789.59
Critical	868.53	697.47
High	890.68	796.92
Moderate	876	871.68
Low	954	1018.65

Table 14: Mean and standard deviation values of minimum vulnerability lifetimes

According to the results, the p-values of all pairs are larger than the significant level, i.e., $p = 0.05$. This means that the null hypothesis cannot be rejected. In other words, there are no statistical differences between the datasets.

Maximum Lifetime	U-value	p-value
Critical-High	18994.5	0.497
Critical-Moderate	6760.0	0.204
Critical-Low	99.5	0.396
High-Moderate	16267.5	0.168
High-Low	203.5	0.399
Moderate-Low	90.0	0.375

Table 15: Comparison results of Mann Whitney U test performed on maximum lifetime values for each severity levels

6 Discussion

We believe that our work represents important steps in understanding the security practices in the Android ecosystem, as well as its likely impact on users. We now present the emerging themes and practical security policy recommendations based on our study.

6.1 Comprehensive Security Bulletins

The first commercialized version of Android was released on September 23, 2008. Since then, Google has released 63 different AOSP versions with 28 API levels [1]. Due to the openness of the platform, Android has been adopted by different vendors, like Samsung, LG, etc., which results in the

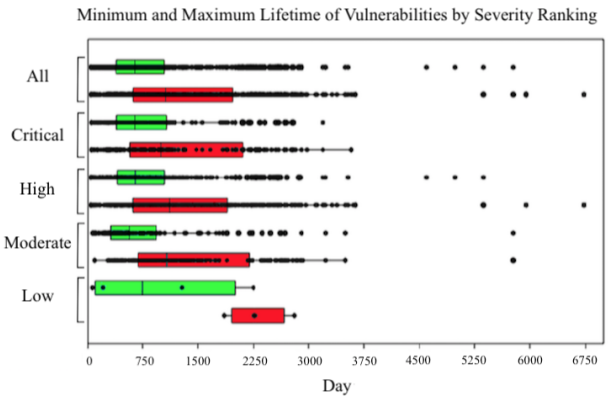

Figure 12: Green and red bar represents the minimum and maximum vulnerability lifetime, respectively

Minimum Lifetime	U-value	*p*-value
Critical-High	19943.0	0.497
Critical-Moderate	6992.0	0.204
Critical-Low	245.0	0.346
High-Moderate	15765.0	0.168
High-Low	555.0	0.399
Moderate-Low	202.5	0.375

Table 16: Comparison results of Mann Whitney U test performed on minimum lifetime values for each severity levels

highest market share among mobile phone devices. Considering the large and widespread use of Android devices and their impact on humans lives, it is vital to keep Android devices secure over time. Because of that, Google provides monthly security patch updates to fix security vulnerabilities. Based on data from the CVEDetails website [12], the Android-related CVEs existed back in 2009. However, Google started its Android security bulletins only in August 2015. Subsequently, also some vendors began releasing their own security bulletins. For instance, Samsung started in October 2015 and LG started in May 2016.

According to the CVEDetails website, there are 2,146 Android-related vulnerabilities by the end of 2018. However, the recorded number of Android-related vulnerabilities from 2009 to the end of 2014 is only equal to 43 [12]. That means, from 2015 (start of Android security bulletins), the number of Android CVEs has increased drastically. Therefore, it seems that releasing Android security bulletins has certainly provided better knowledge about Android security. In contrast, we do not have too much information about the early days of Android (2008) until 2015. From a research perspective, it would be useful if Google starts adding also security bulletins that belong to the time before August 2015 to enable a comprehensive overview of security patches from the introduction of Android to the present day.

6.2 Coherent Security Platforms

There are many different platforms providing detailed information of CVEs, like CVEDetails [50], MITRE [51] and the National Vulnerability Database (NVD) [65]. These three are general purpose databases and are not limited to one vendor or platform. Therefore, we can potentially find any vendor-related as well as Android-related vulnerabilities on them. In this paper, we use CVEDetails and Android security bulletins to investigate Android patched vulnerabilities. The expectation is that all of these publicly available platforms should be consistent with each other; and that there are no contradictions. In other words, only checking one of them should be sufficient for the majority of problems.

However, based on our results in Section 5.2.2, we notice inconsistencies between the Android security bulletins and CVEDetails. These inconsistencies included but are not limited to different information regarding updated and affected AOSP versions, and unknown CVE IDs on the CVEDetails website. Due to these inconsistencies, for example, if a person only relies on CVEDetails, s/he might miss some of the Android-related vulnerabilities on Android security bulletins considering that they are unknown in CVEDetails. Furthermore, Google provides updated versions while CVEDetails provide affected versions. In some cases, these two seem to systematically differ from each other. As a result, an observer cannot rely on only one of them. We believe it is essential that different publicly available websites that explain the vulnerabilities in detail should strive to be consistent with each other. Consistency would help security practitioners to find reliable information easily and they would not have to check different resources.

6.3 User-Centric Security Policy and Security after End-Of-Life (EOL)

Each vendor stops providing software updates after some period of time. Microsoft clearly states the end of support for its Windows operating systems [73]. For example, for Windows 7, the end of mainstream support was January 13, 2015, and the end of extended support is January 14, 2020. Microsoft provides the exact date of support clearly on their website. Unlike Microsoft, Google does not mention these dates exactly [66]; however, clearly announcing such a schedule would be a desirable practice for developers that work within the Android ecosystem as well as consumers.

Further, there are still a considerable number of consumers who use Android devices that do not receive security updates anymore. By taking our analysis and results in Section 5.2 into account, many CVEs are common among different versions including versions that have reached EOL. As a result, out-of-date versions are vulnerable as they do not receive security patch updates. Hence, we believe that in order to have a more secure Android ecosystem, the EOL policy should

take the consumers more directly into account. Our proposition is that the EOL date should be related to the market share of an Android version. For example, if the market share of an Android version (e.g., 4 or 4.x) drops below a certain threshold, then an EOL date can be announced and notifications should be sent to the users. The other advantage of our proposed policy is that when a market share of an Android version is lower than a threshold, it might not be beneficial (in terms of cost-benefit analysis) for rational attackers to exploit a vulnerability.

7 Limitations

In the following, we discuss limitations of our work. The first security bulletin was published in August 2015 by Google while Android was commercialized in 2008. As a result, our analysis and investigation are limited to the last four years (version 4.4 and above) and cannot be extended to 2008-2015. However, we believe that our study and analysis are representative of the current practices of Android and can lead to better policy design in this ecosystem.

Another limitation of our work is that we have asymmetric information for different CVEs. For example, for some CVEs in Android security bulletins, we have a reference link which gives some information about the commit date, etc. But, for other CVEs, there are no such references. Another example is the *reported date*. Only a limited number of CVEs have this attribute. As a result, our analysis for these attributes works with a reduced sample size of those CVEs that have that information.

The SZZ algorithm has its own limitations as well. First, the algorithm only considers code deletions when it tries to find the first commit that might cause a particular vulnerability. However, there are other commits that only have code additions and analyzing code additions needs static code analysis to find the cause of vulnerabilities. Furthermore, the SZZ algorithm also checks the files that might not cause a vulnerability. Log files and build files are examples of them. Since there might be lots of different types of files to exclude, we did not do anything further in this case and leave a more nuanced analysis to future work.

The algorithm also proposes to draw information from the issue tracker [95] such as bug creation date. However, while there is an Android issue tracker [6], we were not able to identify relevant information about patched vulnerabilities. Therefore, it is very likely that these bugs are internally tracked.

8 Conclusion

Our paper provides, what we believe to be the most detailed and comprehensive study on patched Android vulnerabilities. We have collected 2,470 vulnerabilities with their detailed information from both Android security bulletins and the CVEDetails website for August 2015 to January 2019.

First, we investigated the overall trend of the patched vulnerabilities by analyzing the number of vulnerabilities per year for each severity level. Second, we analyzed the distribution of the root causes of patched vulnerabilities by studying their common weakness enumerations (CWE). Third, we studied the duration of support for different AOSP versions. Fourth, we calculated the time gap between the fix-commit and the published date of vulnerabilities that originate from Linux and Qualcomm. Fifth, we examined the sequences of the public disclosure, the patch release, and the last commit dates of the patched vulnerabilities. Last, but not least, we analyzed the maximum and minimum vulnerability lifetimes.

For example, our findings demonstrate that the most common root cause of patched vulnerabilities is `CWE 119: Failure to Constrain Operations within the Bounds of a Memory Buffer`. Further, we showed that the length of security support varies for different AOSP versions. In addition, there are versions that are affected by Android-related vulnerabilities but not updated due to Google's patch policy, which leaves users unprotected after an Android version has reached EOL status.

We hope that our research contributes to a better understanding of the security practices in the Android ecosystem, and helps to develop better policies for security management in the future.

Acknowledgments: We thank the anonymous reviewers for their constructive comments and feedback. We further want to thank Ikra Gizem Yildiz and Ece Kubilay for their feedback. This work was supported by the German Institute for Trust and Safety on the Internet (DIVSI).

References

[1] All AOSP versions. https://en.wikipedia.org/wiki/Android_version_history. Accessed: 01/19/2019.

[2] Android architecture. https://source.android.com/devices/architecture. Accessed: 01/29/2019.

[3] Android audio. https://source.android.com/devices/audio. Accessed: 01/30/2019.

[4] Android bluetooth. https://source.android.com/devices/bluetooth. Accessed: 01/30/2019.

[5] Android Git repo. https://android.googlesource.com. Accessed: 02/15/2019.

[6] Android issue tracker. https://issuetracker.google.com/issues. Accessed: 03/02/2019.

[7] Android kernel stack layer. https://source.android.com/devices/architecture/kernel/android-common. Accessed: 01/19/2019.

[8] Android market share. https://www.idc.com/promo/smartphone-market-share/os. Accessed: 02/17/2019.

[9] Android media. https://source.android.com/devices/media. Accessed: 01/30/2019.

[10] Android namespace libraries. https://source.android.com/devices/tech/config/namespaces_libraries. Accessed: 01/29/2019.

[11] Android NDK native APIs. https://developer.android.com/ndk/guides/stable_apis. Accessed: 01/29/2019.

[12] Android related vulnerabilities. https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224. Accessed: 02/17/2019.

[13] Android runtime. https://en.wikipedia.org/wiki/Android_Runtime. Accessed: 01/30/2019.

[14] Android security bulletin 2015-08. https://source.android.com/security/bulletin/2015-08-01. Accessed: 01/30/2019.

[15] Android security bulletin 2015-10. https://source.android.com/security/bulletin/2015-10-01. Accessed: 01/26/2019.

[16] Android security bulletin 2015-12. https://source.android.com/security/bulletin/2015-12-01. Accessed: 01/26/2019.

[17] Android security bulletin 2016-01. https://source.android.com/security/bulletin/2016-01-01. Accessed: 01/26/2019.

[18] Android security bulletin 2016-02. https://source.android.com/security/bulletin/2016-02-01. Accessed: 02/17/2019.

[19] Android security bulletin 2016-03. https://source.android.com/security/bulletin/2016-03-01. Accessed: 02/17/2019.

[20] Android security bulletin 2016-04. https://source.android.com/security/bulletin/2016-04-02. Accessed: 02/17/2019.

[21] Android security bulletin 2016-06. https://source.android.com/security/bulletin/2016-06-01. Accessed: 02/02/2019.

[22] Android security bulletin 2016-07. https://source.android.com/security/bulletin/2016-07-01. Accessed: 01/29/2019.

[23] Android security bulletin 2016-08. https://source.android.com/security/bulletin/2016-08-01. Accessed: 02/17/2019.

[24] Android security bulletin 2016-09. https://source.android.com/security/bulletin/2016-09-01. Accessed: 01/26/2019.

[25] Android security bulletin 2016-10. https://source.android.com/security/bulletin/2016-10-01. Accessed: 01/26/2019.

[26] Android security bulletin 2017-01. https://source.android.com/security/bulletin/2017-01-01. Accessed: 01/26/2019.

[27] Android security bulletin 2017-02. https://source.android.com/security/bulletin/2017-02-01. Accessed: 01/30/2019.

[28] Android security bulletin 2017-06. https://source.android.com/security/bulletin/2017-06-01. Accessed: 01/27/2019.

[29] Android security bulletin 2017-07. https://source.android.com/security/bulletin/2017-07-01. Accessed: 01/30/2019.

[30] Android security bulletin 2017-09. https://source.android.com/security/bulletin/2017-09-01. Accessed: 01/19/2019.

[31] Android security bulletin 2017-10. https://source.android.com/security/bulletin/2017-10-01. Accessed: 01/29/2019.

[32] Android security bulletin 2017-11. https://source.android.com/security/bulletin/2017-11-01. Accessed: 01/19/2019.

[33] Android security bulletin 2017-12. https://source.android.com/security/bulletin/2017-12-01. Accessed: 02/09/2019.

[34] Android security bulletin 2018-04. https://source.android.com/security/bulletin/2018-04-01. Accessed: 02/03/2019.

[35] Android security bulletin 2018-06. https://source.android.com/security/bulletin/2018-06-01. Accessed: 01/30/2019.

[36] Android security bulletin 2018-08. https://source.android.com/security/bulletin/2018-08-01. Accessed: 01/29/2019.

[37] Android security bulletin 2018-09. https://source.android.com/security/bulletin/2018-09-01. Accessed: 01/27/2019.

[38] Android security bulletin 2018-10. https://source.android.com/security/bulletin/2018-10-01. Accessed: 01/23/2019.

[39] Android security bulletin 2019-01. https://source.android.com/security/bulletin/2019-01-01. Accessed: 02/18/2019.

[40] Android security bulletins. https://source.android.com/security/bulletin. Accessed: 01/23/2019.

[41] Android severity rankings. https://source.android.com/security/overview/updates-resources.html#severity. Accessed: 02/02/2019.

[42] Android stack layers. https://source.android.com/setup. Accessed: 01/19/2019.

[43] Android stagefright media player. https://quandarypeak.com/2013/08/androids-stagefright-media-player-architecture/a. Accessed: 01/30/2019.

[44] Beautifulsoup. https://www.pythonforbeginners.com/beautifulsoup/beautifulsoup-4-python. Accessed: 01/23/2019.

[45] Code Aurora. https://www.codeaurora.org. Accessed: 02/17/2019.

[46] Code Aurora. https://source.codeaurora.org/quic/la/kernel/msm-3.18/commit/?id%3D9be5b16de622c2426408425e3df29e945cd21d37&sa=D&usg=AFQjCNHuM63XOo5Y0C7bMJQIIedBHSDKjw. Accessed: 02/17/2019.

[47] CVE-2016-5131 git repository. https://android.googlesource.com/platform/external/libxml2/+/0eff71008becb7f2c2b4509708da4b79985948bb. Accessed: 01/27/2019.

[48] CVE-2018-9440 first git repository. https://android.googlesource.com/platform/frameworks/av/+/2870acaa4c58cf59758a74b6390615a421f14268. Accessed: 01/27/2019.

[49] CVE-2018-9440 second git repository. https://android.googlesource.com/platform/frameworks/av/+/8033f4a227e03f97a0f1d9975dc24bcb4ca61f74. Accessed: 01/27/2019.

[50] CVE details. https://www.cvedetails.com. Accessed: 01/23/2019.

[51] CVE Mitre. https://cve.mitre.org/. Accessed: 02/03/2019.

[52] CWE. https://cwe.mitre.org/about/. Accessed: 01/28/2019.

[53] CWE hierachy. 'https://nvd.nist.gov/vuln/categories/cwe-layout#'. Accessed: 01/27/2019.

[54] Git annotate. https://git-scm.com/docs/git-annotate. Accessed: 03/02/2019.

[55] Git diff. https://git-scm.com/docs/git-diff. Accessed: 03/02/2019.

[56] Google kernel git. https://android.googlesource.com/kernel/. Accessed: 01/30/2019.

[57] HAL. https://source.android.com/devices/architecture/hal. Accessed: 01/29/2019.

[58] HAL & HIDL package and interfaces. https://source.android.com/devices/architecture/hidl/interfacesl. Accessed: 01/29/2019.

[59] HIDL. https://source.android.com/devices/architecture/hidl. Accessed: 01/29/2019.

[60] Json. https://en.wikipedia.org/wiki/JSON. Accessed: 01/23/2019.

[61] LG security bulletin. https://lgsecurity.lge.com/security_updates.html. Accessed: 02/17/2019.

[62] Linux patchwork. https://patchwork.kernel.org. Accessed: 02/17/2019.

[63] Mobile Android Version Market Share Worldwide. http://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide. Accessed: 02/10/2019.

[64] Mongodb. https://www.mongodb.com/what-is-mongodb. Accessed: 01/23/2019.

[65] National Vulnerability Database. https://nvd.nist.gov/. Accessed: 02/03/2019.

[66] Nexus Device Update. https://support.google.com/nexus/answer/4457705?hl=en. Accessed: 02/09/2019.

[67] NVD severity rankings. https://nvd.nist.gov/vuln-metrics/cvss. Accessed: 02/02/2019.

[68] Python 3.6.0. https://www.python.org/downloads/release/python-360/. Accessed: 02/02/2019.

[69] Samsung security bulletin. https://security.samsungmobile.com/securityUpdate.smsb. Accessed: 02/17/2019.

[70] Selenium. https://www.seleniumhq.org. Accessed: 01/23/2019.

[71] Studio3t. https://www.mongodb.com/partners/studio-3t. Accessed: 01/23/2019.

[72] Total number of vulnerabilities. https://www.cvedetails.com/browse-by-date.php. Accessed: 02/17/2019.

[73] Windows lifecycle fact sheet. https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet. Accessed: 02/07/2019.

[74] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Proceedings of the USENIX Security Symposium*, 2013.

[75] Omar Alhazmi and Yashwant Malaiya. Modeling the vulnerability discovery process. In *Proceedings of the 16th International Symposium on Software Reliability Engineering*, pages 1–10. IEEE, 2005.

[76] Ashish Arora, Ramayya Krishnan, Rahul Telang, and Yubao Yang. An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *Information Systems Research*, 21(1):115–132, 2010.

[77] Thomas Duebendorfer and Stefan Frei. Why silent updates boost security. Technical report, 2009.

[78] Keith Edwards, Erika Shehan Poole, and Jennifer Stoll. Security automation considered harmful? In *Proceedings of the 2007 Workshop on New Security Paradigms (NSPW)*, pages 33–42. ACM, 2008.

[79] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding Android security. *IEEE Security & Privacy*, (1):50–57, 2009.

[80] Sadegh Farhang, Aron Laszka, and Jens Grossklags. An economic study of the effect of android platform fragmentation on security updates. *arXiv preprint arXiv:1712.08222*, 2017.

[81] Sadegh Farhang, Jake Weidman, Mohammad Mahdi Kamani, Jens Grossklags, and Peng Liu. Take it or leave it: A survey study on operating system upgrade practices. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*, pages 490–504. ACM, 2018.

[82] Stefan Frei, Bernhard Tellenbach, and Bernhard Plattner. 0-day patch exposing vendors (in) security performance. *BlackHat Europe*, 2008.

[83] Christos Gkantsidis, Thomas Karagiannis, and Milan Vojnovic. Planet scale software updates. *ACM Computer Communication Review*, 36(4):423–434, 2006.

[84] Arrah-Marie Jo. The effect of competition intensity on software security-an empirical analysis of security patch release on the web browser market. In *16th Annual Workshop on the Economics of Information Security (WEIS)*, 2017.

[85] Aron Laszka, Mingyi Zhao, and Jens Grossklags. Banishing misaligned incentives for validating reports in bug-bounty platforms. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 161–178, 2016.

[86] Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2201–2215. ACM, 2017.

[87] Mario Linares-Vásquez, Gabriele Bavota, and Camilo Escobar-Velásquez. An empirical study on Android-related vulnerabilities. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 2–13. IEEE, 2017.

[88] Thomas Maillart, Mingyi Zhao, Jens Grossklags, and John Chuang. Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs. *Journal of Cybersecurity*, 3(2):81–90, 2017.

[89] Arunesh Mathur and Marshini Chetty. Impact of user characteristics on attitudes towards automatic mobile application updates. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, 2017.

[90] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 692–708. IEEE, 2015.

[91] Chuangang Ren, Peng Liu, and Sencun Zhu. Windowguard: Systematic protection of GUI security in Android. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2017.

[92] Chuangang Ren, Yulong Zhang, Hui Xue, Tao Wei, and Peng Liu. Towards discovering and understanding task hijacking in Android. In *Proceedings of the USENIX Security Symposium*, pages 945–959, 2015.

[93] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Securing Android-powered mobile devices using SELinux. *IEEE Security & Privacy*, 8(3):36–44, 2010.

[94] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X Liu. A large scale exploratory analysis of software vulnerability life cycles. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 771–781. IEEE, 2012.

[95] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *ACM Sigsoft Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[96] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An empirical study of SSL warning effectiveness. In *Proceedings of the USENIX Security Symposium*, pages 399–416, 2009.

[97] Kami Vaniea and Yasmeen Rashidi. Tales of software updates: The process of updating software. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pages 3215–3226. ACM, 2016.

[98] Timothy Vidas, Daniel Votipka, and Nicolas Christin. All your droid are belong to us: A survey of current Android attacks. In *WOOT*, pages 81–90, 2011.

[99] Luyi Xing, Xiaorui Pan, Rui Wang, Kan Yuan, and XiaoFeng Wang. Upgrading your Android, elevating my malware: Privilege escalation through mobile OS updating. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 393–408. IEEE, 2014.

[100] Karim Yaghmour. *Embedded Android: Porting, Extending, and Customizing*. O'Reilly Media, Inc., 2013.

[101] Guangliang Yang, Jeff Huang, Guofei Gu, and Abner Mendoza. Study and mitigation of origin stripping vulnerabilities in hybrid-postmessage enabled mobile applications. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 742–755. IEEE, 2018.

[102] Mingyi Zhao, Jens Grossklags, and Peng Liu. An empirical study of web vulnerability discovery ecosystems. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1105–1117. ACM, 2015.

A Android Stack Layers

Android stack layers are software components for a wide array of devices with different form factors. It helps understanding where a software component is located. Figure 13 represents all of its five main stack layers and one additional component [42]. In this section, we investigate the role of Android stack layers in vulnerabilities.

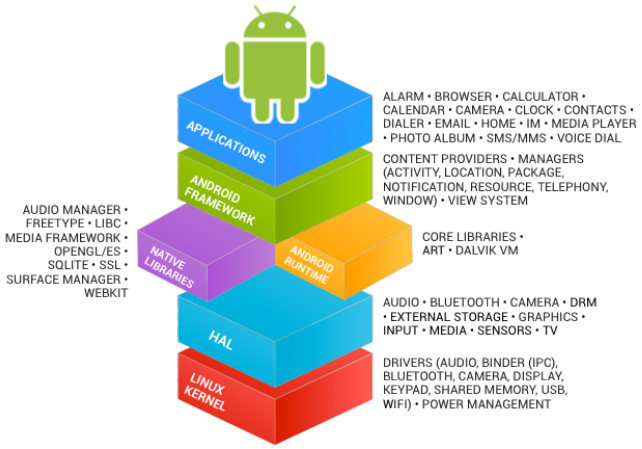

Figure 13: Android stack layers [42]

It is crucial to examine that which Android stack layer is affected by which vulnerability. Thus, one can see which layers are more vulnerable. However, to the best of our knowledge, none of the publicly available platforms provide explicit information about affected layers for Android-related vulnerabilities. Thus, a further analysis is essential to determine them. Note that patched vulnerabilities have references in our dataset. They contain links that navigate to the code repositories such as Qualcomm Code Aurora, AOSP Git repository and Linux Patchwork. In this classification, we only investigate AOSP Git repositories. For instance, `CVE-2016-5131`, published in June 2017, has one reference link that navigates to the AOSP Git repository [28]. In AOSP Git repository, the followings are published: the branch, the commit id, the changed files, the author, the commit message and commit logs. [47].

The references can be more than one as well. For instance, `CVE-2018-9440` is published in September 2018 and has two different references [37]. Although they have the same branch which is *frameworks/av*, they have different commit IDs. Besides, one of the references has one additional changed file than the other one which is *media/libstagefright/httplive/M3UParser.h* [48, 49]. Moreover, the branches can also be different among the references that belong to the same patched vulnerability. For instance, `CVE-2017-0831` is published in November 2017 and has two different references [32]. The first reference has the branch *frameworks/base* while the other one has the branch *packages/apps/Settings*. Given this, the changed files can differ from each other as

well.

Another issue we notice is that there might be patched vulnerabilities that do not have any references. According to the information that is published on each Android security bulletin, these patched vulnerabilities are marked with the sign of asterisk * since their references are not publicly available.

For our classification method, only a branch name itself is not enough to find which layer is affected by a vulnerability. The reason is that there might be directories on the branch that might point to the different stack layers. For this reason, we also need to check the changed files' directories to achieve more accurate classification. Since the changed files have their own directories, we combine the branch path with the changed file patch to get *full path*.

Aside from the full paths, we also use *component name* and **category name** to identify the stack layers of patched vulnerabilities. Component name is presented as a column of each patched vulnerabilities. Note that there might be such cases where it does not have the component name. Category name, on the other hand, is the title of each category on the Android security bulletins. For instance, in September 2017 [30], the first category name is *Framework* which consists of only one patched vulnerability. The second category name is *Libraries* that contains three different patched vulnerabilities. Although the component and category names do not provide too much detailed information, we benefit from them to identify external components and kernel stack layer vulnerabilities.

The following subsections describe the classification methods for each stack layers.

A.1 Kernel Layer

All the kernel layer changes are implemented under the branch name *kernel/* [7, 56]. Thus, a patched vulnerability can be classified as a kernel layer by looking at this branch name. In case of where a patched vulnerability does not have any references, then we look at its category name and the component name. If either of these names has the name *kernel*, we consider it as a Kernel Layer patched vulnerability. For example, `CVE-2014-9322`, published in April 2016, has the branch name *kernel/* [20] which makes it a kernel layer patched vulnerability. On the other hand, `CVE-2017-0648`, published in June 2017, does not have any references [28]. However, it has the category name *kernel*. Thus, this patched vulnerability is classified as a kernel layer. As a result, we check the branch name, category name, and component name of the patched vulnerabilities to classify them as kernel layer.

A.2 Hardware Abstraction Layer

"Hardware Abstraction Layer (HAL) defines a standard interface for hardware vendors to implement, which enables Android to be agnostic about lower-level driver implementations" [57]. An interface description language (IDL) that determines the interface between users and HAL is called HAL interface definition language (HIDL). This also specifies types and method calls, collected into interfaces and packages [59].

Based on the Android source [58], if a patched vulnerability has the following branch names, then it can be classified as HAL: *hardware/*, *hidl/* and *hwservicemanager/*. The branches *vendor/* and *device/* can also be counted as HAL-related branches, according to [58] and [100] Page 88. Moreover, the branch *system/bt/* also indicates a Bluetooth stack located in HAL [4].

For instance, `CVE-2017-0767`, published in September 2017, has two references [30]. One of them has the branch name *hardware/*. Therefore, we classify this patched vulnerability as HAL. The second example, `CVE-2017-0812`, published in October 2017, has the branch name *device/* [31]. Hence, we also classify this as HAL.

A.3 External Layer

Android uses external libraries that are developed by third-parties. These externals might affect different Android stack layers. However, they need further analyses in order to classify them as one of the Android Stack Layers. Thus, we consider them separately. External-related patched vulnerabilities have their own unique branch name *external/*. Thus, we use this branch name to classify the patched vulnerabilities as an external-related one. It consists of all external libraries such as *chromium-libpac* and *v8*. For example, `CVE-2018-9490`, published in October 2018, has two references [38]. Both of them have the branch name *external/*. Therefore, we classify this patched vulnerability as an external-related patched vulnerability.

A.4 Applications Layer

Applications Layer is the top layer on all Android stack layers. It consists of different stock applications such as Calculator and Email that the end user interacts with. Based on [100] (page 88, Figure 3.2), we consider all the changes implemented in the branch *packages/apps* as application layer patched vulnerabilities. For instance, `CVE-2018-9501`, published in October 2018, has one reference under the branch name *packages/apps* [38].

A.5 Application Framework Layer

Application framework layer consists of abstractions and APIs in order to provide a communication between the applications and the native libraries [2]. According to [100] (page 65, Figures 2-4), there are Java-built services under the system servers. Java providers and managers are also under the application frameworks layer [100] (page 73). More-

over, compatibility test suites (CTS)[5] are also considered as one of the part of this layer. We use the following branches and directories to classify the patched vulnerabilities as application framework layer:*platform/packages/providers*, *platform/frameworks/base/*, *platform/packages/services/*, *platform/frameworks/opt/*, *platform/cts/*, *platform/libcore/*.

The branch *platform/frameworks/base/* shares files which also belong to the native libraries layer. To address this issue, the files that have *.java* file extension are counted as the application framework layer while the others that have *.c* or *.cpp* file extensions are considered as native libraries layer.

For example, CVE-2018-9438, published in August 2018, has *platform/packages/providers* branch [36]. Another example is CVE-2018-9493 which is published in October 2018 [38]. It has branch name *platform/frameworks/base/*. Therefore, we count these two patched vulnerabilities as application framework layer due to their branch names.

A.6 Native Libraries Layer

Native libraries consist set of native headers and shared library files in order to provide a solid connection between the upper layers and lower layers [10, 11]. Audio and Media are two examples that part of native libraries [3, 9] The branch */platform/frameworks/av/* indicates the media related libraries [43]. We also count *platform/frameworks/base/* since it is the old version of *platform/frameworks/av/*. Further, the branch *platform/frameworks/native/libs/* consists of several native libraries as well [100] (page 89).

We also consider some parts of the Android architecture, such as *Native Daemons* and *Init/Toolbox* as parts of the native libraries layer. Since we analyze the patched vulnerabilities by attaching to the Android stack layers and Android stack layer schema does not explicitly indicate these parts, we consider these parts as native libraries as well. For instance, CVE-2018-9491, published in October 2018, has the branch *platform/frameworks/av* [38] and CVE-2017-0426, published in February 2017, has the branch */platform/system/core/* [27]. Therefore, we consider both of them as native libraries patched vulnerabilities.

A.7 Android Runtime

Android Runtime (ART) is an application runtime environment. Replacing Dalvik, the process virtual machine originally used by Android, ART performs the translation of the application's bytecode into native instructions that are later executed by the device's runtime environment [13]. According to [100] Page 88 Figure 3.2, */dalvik/* branch is under ART. Besides, with the investigation of the directories, we also see files/directory names *androidruntime* or *android_runtime* in some references. Therefore, we look at these branches and directories/file names to consider a patched vulnerability as ART. For instance, CVE-2015-3865, published in August 2015, has the directory *android_runtime* in its reference [14]. Similarly, CVE-2016-3758, published in July 2016, has the branch *platform/dalvik* [22].

In Appendix A.8, we list all branches and directories that we use for the classification of the vulnerabilities published in Android security bulletins.

[5] CTS tests the overall functionality including the application layer framework

A.8 All Branches and Directories for the Classification of the Patched Vulnerabilities

- **Kernel Layer Branches and Directories**
  "platform/kernel/"

- **HAL Branches and Directories**
  "platform/hardware/",
  "platform/device/",
  "platform/vendor/"
  "platform/system/bt/",
  "platform/system/nfc/",

- External-Related Branches and Directories:
  "platform/external/"

- **Applications Layer Branches and Directories**
  "platform/packages/apps/"

- **Application Framework Layer Branches and Directories**
  "platform/packages/providers/",
  "platform/frameworks/base/",
  "platform/packages/services/",
  "platform/frameworks/opt/",
  "platform/cts/",
  "platform/libcore/",
  "platform/frameworks/base/services/",

- **Native Libraries Layer Branches and Directories**
  "platform/frameworks/av/",
  "platform/frameworks/base/lib/",
  "platform/frameworks/native/",
  "platform/frameworks/base/core/" ,
  "platform/frameworks/base/media" ,
  "platform/frameworks/minikin/libs/",
  "platform/system/",
  "platform/frameworks/ex/",
  "platform/bionic/" ,
  "platform/bootable/" ,

- **Android Runtime Layer Branches and Directories**
  "platform/dalvik/",
"platform/frameworks/base/include/android_runtime",