International Journal of Contemporary Research in Computer Science and Technology (IJCRCST) ISSN: 2395-5325
Volume 3, Special Issue 3 (September '2017)
Proceeding of - International Conference on Recent Trends in Computational Life Science and Information Technology,
Conference held at Tiruppur Kumaran College for Women,Tirupur,Tamilnadu, India.

DESTINATION DISCONTINUITY IN ANDROID APPS

R.Soniya,
M.Phil., Research Scholar.,
Department of Computer Science,
K G College and Arts Science,
Coimbatore, India.

Boopalan S,
Assistant Professor,
Department of Computer Science,
K G College and Arts Science,
Coimbatore, India.

Abstract: Target version is declared for the android apps. At the point when keep running on gadgets with later Android forms, applications are executed in a similarity mode that endeavors to copy the conduct of the more established target rendition. This outline has genuine security results. Applications that goal out of date Android adjustments cripple basic security changes to the Android organize. We call the issue of uses concentrating on out of date Android frames the target irregularity issue. We separate a dataset of 1,232,696 free Android applications accumulated between May, 2012 and December, 2015 and exhibit that the target intermittence issue is an authentic stress over the entire application organic group and has not changed amazingly in a significant extended period of time. By and large, 93% of current applications center out of date arrange frames and have a mean oldness of 686 days; 79% of uses are starting at now obsolete on the day they are exchanged to the application store. Finally, we investigate seven security related changes to the Android arrange that are disabled in applications that goal out of date organize frames and exhibit that target brokenness hamstrings attempts to upgrade the security of Android applications.

Keywords: Android Apps, Application Program Interface, SDK,Security

I. INTRODUCTION

Android has transformed into the most standard mobile phone organize the world over, with more than one billion dynamic contraptions [1]. Android goes up against two critical security challenges in passing on secure code to customers, break inside devices and discontinuity inside applications. Contraption irregularity is a remarkable concern [2]. Google does not control the dispersal of Android devices or Android programming revives. Google rather relies upon an arrangement of various associations to pass on cutting edge Android programming to customers, as new devices or programming revives. Because of the scattered thought of this strategy, a far reaching number of Android devices are missing the mark on date types of the Android organize. Essential security patches don't accomplish countless customers, growing the lifetime of security vulnerabilities. Interestingly with device crack, brokenness inside applications has become little thought paying little heed to passing on practically identical security results.

Android uncovered various fundamental library elements to applications through the Android API. New forms of the Android plat-frame can acquaint changes with the conduct of existing library highlights. For instance, Android 4.4 changed the conduct of AlarmManager set to cluster cautions set for comparable circumstances. This change enhances battery life to the detriment of estimated alerts. Behavioral changes, for example, this present an issue for applications, which may all of a sudden break when a gadget updates to another Android form. Google allocates every Android stage form a whole number called an API level. To keep up a level of advances compatibility and keep applications from significantly changing their conduct all of a sudden, each application has an objective API level. Applications that are keep running on gadgets with a higher API level than their objective API level are executed in a similarity mode that endeavors to coordinate the conduct of gadgets with the objective API level as nearly as could be expected under the circumstances. For instance, applications that set their objective API level to 18 (Android 4.3) will utilize the unbatched alert conduct notwithstanding when keep running on a la mode Android gadgets.

Android stage changes can incorporate vital new security highlights that either settle known issues with the Android APIs or give additional security against assault. Any of these elements handicapped by the similarity mode will be inaccessible to applications that objective obsolete Android levels. This outline implies that, notwithstanding when running a progressive gadget, an application that objectives an obsolete API level won't approach the most current security highlights. Google does not distribute an application's objective API level on the Google Play store so there is no basic route for clients to know whether an application focuses on an obsolete API level. Rather, clients are completely helpless before application engineers. We call this issue of applications focusing on obsolete API levels the objective fracture issue.

The most surely understood outcome of the objective fragmentation issue identifies with a remote code execution weakness in Android WebView [3]. Programming interface level 17 added new conducts to determine this weakness, however this change is just connected to applications that objective API level 17 or higher. A long time after the helplessness was unveiled and settled in the Android stage, applications can at present be powerless, notwithstanding when keep running on the new stage, by focusing on obsolete API levels. A few examinations inspecting this helplessness have revealed the level of applications that objective levels 16 or beneath [4, 5]. Be that as it may, this

defenselessness is only one case of how the objective fracture issue makes Android applications less secure and no exploration has studied target discontinuity in its own particular right.

To the best of our insight, this paper reports the primary examination to recognize and measure the objective fracture issue and its security suggestions at an expansive scale. In this paper we think about target fracture in five datasets of free Android applications gathered from the Google Play store over right around four years. Altogether, these datasets incorporate 1,232,696 applications. We measure inclines in the objective fracture issue utilizing metadata acquired from the Google Play store. At long last, we consider the security ramifications of the objective fracture issue regarding a few solid vulnerabilities. The major research questions and results of our study are as follows:

What is the condition of the objective discontinuity issue in the Android application biological community? We look at a dataset of 60,086 free applications gathered from the Google Play store in December, 2015 and locate that 93% of applications focus outdated API levels. We characterize a measure of "obsoleteness" and find that applications, all things considered, target API levels that are 686 days outdated.

Do designers target obsolete Android versions or is discontinuity caused by engineers deserting their applications? To represent unmaintained applications, we characterize a measure of "careless obsoleteness" that measures obsoleteness from the date an application was transferred to the application store and demonstrate that objective fracture is not recently caused by dormant applications. We find that applications gathered in December, 2015 have a mean careless obsoleteness of 536 days.

Is target discontinuity an issue in the most mainstream applications? We inspect the objective fracture issue as for application prevalence and presume that objective fragmentation is a major issue even among the most famous applications. We locate that 88% of applications gathered in December, 2015 and introduced more than one million times focus outdated API levels. These applications have a mean obsoleteness of 607 days and a mean careless obsoleteness of 493 days, just somewhat lower than the overall public.

Is the objective fracture issue ending up less extreme after some time? We analyze the objective fracture comes about because of the December, 2015 dataset with four different datasets gathered from the Google Play store between May, 2012 and July, 2014. These datasets consolidated contain 1,232,696 applications. We locate that, other than a developing tail of to great degree obsolete applications, obsoleteness dispersions among the four most as of late gathered datasets are fundamentally the same as, proposing that the seriousness of the objective discontinuity issue has not changed extensively in quite a while.

What are the particular security ramifications of the objective discontinuity issue today? We extend the discourse of the objective fracture issue by looking at seven security applicable changes in the Android stage and give the main quantitative investigation of the expansive ramifications of target discontinuity on the security of the Android biological community

II.BACKGROUND

Bundled with every Android application is a show record. The show record is a XML report that contains data around an application, for example, the rundown of utilization parts, asked for consents, and framework occasions to which the application reacts [6]. The show contains two credits significant to this investigation: a base API level (minSdkVersion) and an objective API level (targetSdkVersion)1. The importance of the base API level is extremely clear. An application can't be introduced on a gadget with an Android level beneath the base API level. This plan guarantees that applications are not introduced on gadgets that need fundamental usefulness.

[1]Manifests can also include a maximum API level, but since Android 2.0.1 the maximum API level is no longer used for anything other than filtering searches on the Google Play store.

Code name	Version number	Initial release date	API level
N/A	1.0	September 23, 2008	1
	1.1	February 9, 2009	2
Cupcake	1.5	April 27, 2009	3
Donut	1.6	September 15, 2009	4
Eclair	2.0–2.1	October 26, 2009	5–7
Froyo	2.2–2.2.3	May 20, 2010	8
Gingerbread	2.3–2.3.7	December 6, 2010	9–10
Honeycomb[a]	3.0–3.2.6	February 22, 2011	11–13
Ice Cream Sandwich	4.0–4.0.4	October 18, 2011	14–15
Jelly Bean	4.1–4.3.1	July 9, 2012	16–18
KitKat	4.4–4.4.4, 4.4W–4.4W.2	October 31, 2013	19–20
Lollipop	5.0–5.1.1	November 12, 2014	21–22
Marshmallow	6.0–6.0.1	October 5, 2015	23

TABLE I: An abbreviated Android version history.

An application's objective API level is utilized to keep up advances similarity with new Android stages. In theevent that the API level of a gadget is higher than an application's objective API level, the gadget will empower similarity components to coordinate the conduct of the objective API level as nearly as could be expected under the circumstances. The arrangement of similarity highlights empowered in every API level can be found in the Android documentation [7]. Note that applications can be securely introduced on gadgets running lower API levels than the objective API level. Designers can focus on the most current API level without making their application inconsistent with more established Android gadgets.

On the off chance that an application does not pronounce an objective API level or if the objective API level is lower than the base API level, the objective API level is set to the base API level. For the rest of this paper we recognize the crude target API level, which is the objective level recorded in the show document, and the objective API level, which is the objective level processed utilizing this lead and really utilized by the Android working framework. Roughly 8% of

applications today don't announce a legitimate crude target API level.

The targetSdkVersion and minSdkVersion at-tributes take whole number esteems called "Programming interface levels" that compare to the diverse Android variant codes. For the rest of this paper we utilize the API level esteems (e.g., 17, 18, 19) as opposed to the variant codes (e.g., 4.2, 4.3, 4.4) while talking about various API adaptations. The correspondence between late API levels and Android rendition codes is displayed in Table I.

A. Security Concerns

It is not promptly evident from the Android documentation that focusing on obsolete API levels can have security suggestions. On the documentation page for the targetSdkVersion property [8], Google recommends that de-velopers should "increment the estimation of [the targetSdkVersion] ascribe to coordinate the most recent API level," yet there is no specify of the security outcomes of focusing on obsolete API levels. On the "Security Tips" page [9] there is no specify of target API level. One may accept that the security outcomes of focusing on obsolete API levels are negligible or nonexistent. In any case, there are critical security changes in late Android forms that are not connected to applications that objective obsolete API levels. For instance, API levels 17 and 19 both contain changes that counteract code infusion vulnerabilities in broadly utilized components. A few other API levels change famous components to have more secure default practices, giving an additional layer of assurance. Table II records the real security changes to the Android stage that can be handicapped by focusing on

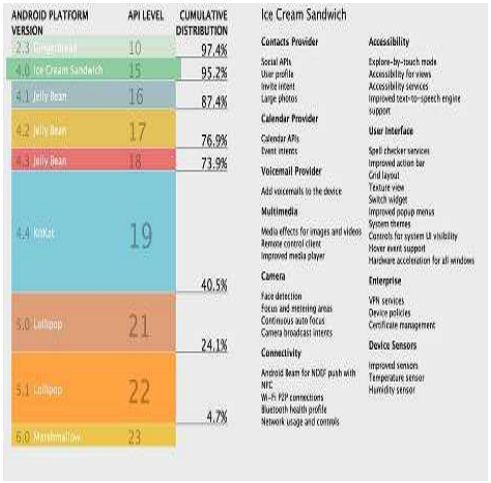

TABLE II: Selected security-relevant changes to the Android plat-form.

Apps that target API levels below the listed levels do not have the benefit of any security protection provided by these changes. outdated Android levels. The details of these changes and the vulnerabilities they close are discussed in Section V. If a large number of apps target out-of-date API levels then these changes, no matter how well intentioned, are made ineffective and apps are put at unnecessary risk.

III.METHODOLOGY

Our investigation breaks down a dataset of 1,232,696 free applications collected from the Google Play application store between May, 2012 and January, 2016. To gather these applications we built up a framework to creep the Google Play store to distinguish new applications, rub metadata from the Google Play store, and download real application documents. This framework was operational amid five brief time windows, normally isolating our dataset into five littler (Datasets A, B, C, D, and E, in turn around sequential request) that relate to these time windows. Table III portrays the subtle elements of these datasets and records the most current API level at the time each dataset was gathered.

A. Collecting Apps

Our framework initially slithers the Google Play store for applications to download. We consider an application interesting on the off chance that it has a remarkable application id2. To slither the Google Play store, we utilize the accompanying four systems:

(1) creep the Google Play assigned classes for famous applications and accumulations,

(2) creep arbitrary known engineer pages to search for new applications,

(3) seek on the Google Play store utilizing words from known application portrayals, and (4) separate all application ids from URLs on crept pages.

Google distributes some metadata about applications on the store. We rub and gather metadata about each slithered application including the date the latest adaptation of that application was transferred to the application store and the quantity of gadgets on which the application has been installed3.

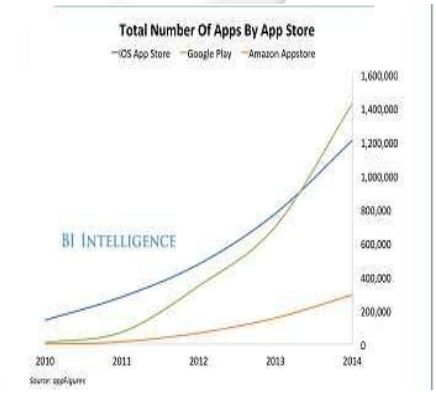

TABLE III: An overview of the five datasets used in this study and the most current API level at the time each dataset was collected.

To download applications we utilize a strategy like the one portrayed by Viennot, Garcia, and Nieh [10]. While trying to productively gather as assorted an arrangement of utilizations as could reasonably be expected, we just download applications with at no other time seen application ids. For each dataset aside from Dataset A, we endeavored to download each application with an at no other time seen application id recognized amid slithering. Because of time imperatives (and specialized difficulties), Dataset An is just

a subset of the accessible applications. We talk about the impact of this gathering strategy in Section VI-B.
The value of the id query parameter of the app's Google Play URL, for example com.instagram.android in the URL play.google.com/store/
apps/details?id=com.instagram.android. Note that this is not necessarily the same as the app's package name
Precise install counts are not available. Google Play reports a range of possible install counts with two buckets per order of agnitude. For example, an app might have a reported install count between 10,000 and 50,000.

B. Analysis

The Google Play store distributes each application's base API level however does not distribute target API levels. We utilize apktool [11], a static investigation apparatus that believers bundled applications into comprehensible documents, to extricate shows and record their objective API levels. Since our database of applications is to a great degree extensive, it is unrealistic to perform complex static investigation. The greater part of the static examination utilized as a part of this investigation is absolutely syntactic, which we perform by handling the small portrayal of application bytecode separated by apktool.

IV.EVALUATION

In this segment we measure the degree of the objective fragmentation issue. We start by showing that the dominant part of inspected applications target obsolete API levels. We characterize an obsoleteness metric that measures the seriousness of the objective fracture issue for individual applications and in addition over a populace of applications. We demonstrate that the objective discontinuity issue is basically caused by designer carelessness as opposed to applications that lie decrepit on the application store. We look at our obsoleteness metric amongst well known and disliked applications and demonstrate that objective discontinuity is an issue even among the most famous applications. At long last, we demonstrate that obsoleteness bends are comparative in the four most as of late gathered datasets. This outcome recommends that, unless the objective fracture issue is reevaluated, it might proceed with a similar scale later on. Because of the vast sizes of our datasets, we trust that these outcomes apply comprehensively to the whole Google Play biological community.

A. Destination Discontinuity Today

Figure 1 demonstrates the dissemination of target API levels for applications in Dataset A, the dataset containing the most current applications. It is quickly evident that the tremendous dominant part of applications doesn't target API level 23, the most current API level at the time Dataset A was gathered. All the more definitely, we locate that 93% of applications in Dataset An objective API levels 22 or lower.

Applications that objective more obsolete API levels will probably miss essential security changes. We are not quite recently keen on if an application is obsolete yet additionally in how obsolete an application is. We characterize a quantitative measure called "obsoleteness" as the distinction (in days) between the discharge date of an application's objective API level and the discharge date of the most current API level at the season of the application was gathered. We locate a middle obsoleteness of 704 days and a mean obsoleteness of 686 days

V.CONCLUSION

Android applications indicate an objective API level and keep running in a similarity mode on gadgets with higher API levels. The similarity mode can incapacitate vital security changes in the Android stage. We call the issue of applications focusing on obsolete API levels the objective fracture issue. In this investigation we examine a dataset of more than one million Android applications gathered more than four years and demonstrate that the vast dominant part of gathered applications target obsolete API levels. We inspect the handy ramifications of target discontinuity on seven security changes to the Android stage and demonstrate that objective fracture hamstrings new security highlight
We trust that applying security changes in this discretionary way is a defective approach that penances security at the sacrificial stone of similarity. Designers turn into another snag to securing applications and clients have no methods for guaranteeing that their applications focus on the most current API levels. The objective fracture issue is additionally intensified by the coupling of security changes and non-security changes. We trust that by revealing insight into this issue, engineers can turn out to be more mindful of the outcomes of focusing on obsolete API levels and this defective plan can be rethought and changed so that there is less open door for Android applications to work without access to imperative security highlights.

VI.REFERENCES

[1]. D. R. Thomas, A. R. Beresford, and A. Rice, "Security metrics for the android ecosystem," in *Proceedings of the 5th ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015.
[2]. Webview addjavascriptinterface remote code execution. [Online]. Available: http://labs.mwrinfosecurity.com/blog/2013/09/24/webview-addjavascriptinterface-remote-code-execution
[3]. P. Mutchler, A. Doupe,´ J. Mitchell, C. Kruegel, and G. Vigna, "A large-scale study of mobile web app security," in *Mobile Security Techologies*, 2015.
[4]. D. R. Thomas, A. R. Beresford, T. Coudray, T. Sutcliffe, and A. Taylor, "The lifetime of android api vulnerabilities: Case study on the javascript-to-java interface," *Security Protocols XXII*, 2015.
[5]. App manifest. [Online]. Available: http://developer.android.com/guide/
topics/manifest/manifest-intro.htmlBuild.version codes. [Online]. Available: http://developer.android.com/
reference/android/os/Build.VERSION CODES.html
[6]. N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *Proceedings of the 2014 ACM Conference on Measurement and Modeling of Computer Systems*, 2014.

[7]. Apktool. [Online]. Available: http://code.google.com/p/android-apktool
[8]. Webview. [Online]. Available: http://developer.android.com/refe rence/ android/webkit/WebView.html
[9]. E. Chin and D. Wagner, "Bifocals: Analyzing webview vulnerabilities in android applications," in *Information Security Applications*, 2014.
[10]. D. Wu and R. K. Chang, "Analyzing android browser apps for file://vulnerabilities," in *Information Security*, 2014.
[11]. M. Georgiev, S. Jana, and V. Shmatikov, "Breaking and fixing origin-based access control in hybrid web/mobile application frameworks," in *Proceedings of the 2014 Network and Distributed System Security Symposium*, 2014.
[12]. M. Backes, S. Gerling, and P. von Styp-Rekowsky, "A local cross-site scripting attack against android phones," 2011.[Online]. Available: http://infsec.cs.uni-saarland.de/projects/android-vuln/android xss.pdf
[13]. What is mixed content?[Online].Available http://developers.google.com/web/fundamentals/sec urity/prevent-mixed-content/what-is-mixed-content
[14]. Mixed content blocking in firefox. [Online]. Available: http://support. mozilla.org/en-US/kb/mixed-content-blocking-firefox
[15]. Binding javascript code to android code. [Online]. Available: http:// developer.android.com/guide/webapps/webview.html #BindingJa vaScript
[16]. N. Bergman. Abusing webview javascript bridges. [Online]. Available: http://50.56.33.56/blog/?p=314<provider>. [Online]. Available: http://developer.android.com/guide/ topics/manifest/provider-element.html
[17]. Y. Z. X. Jiang, "Detecting passive content leaks and pollution in android applications," in *Proceedings of the 20th Network and Distributed System Security Symposium*, 2013.
[18]. R. Hay. A new vulnerability in the android framework: Fragmentinjection. [Online]. Available: http://securityintelligence.com/new-vulnerability-android-framework-fragment-injection
[19]. Intents and intent filters. [Online]. Available: http://developer.android. com/guide/components/intents-filters.html
[20]. E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, 2011.
[21]. Api parser. [Online]. Available: http://github.com/yeganehs/API-Parser Dashboards. [Online]. Available: http://developer.android.com/about/ dashboards/index.html Google Play Stats. [Online]. Available: http://appbrain.com/stats
[22]. T. McDonnell, B. Ray, and M. Kim, "An empirical study of api stability and adoption in the android ecosystem," in *Proceedings of the 29th IEEE Conference on Software Maintenance*, 2013.
[23]. L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in
[24]. *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.
[25]. X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri, "Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.
[26]. X. Zhou, Y. Lee, N. Zhang, M. Naveed, and X. Wang, "The peril of frag-mentation: Security hazards in android device driver customizations," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, 2014.
[27]. L. Xing, X. Pan, R. Wang, K. Yuan, and X. Wang, "Upgrading your android, elevating my malware: Privilege escalation through mobile os updating," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, 2014.
[28]. C. Mulliner, J. Oberheide, W. Robertson, and E. Kirda, "Patchdroid: Scalable third-party security patches for android devices," in *Proceed-ings of the 29th Annual Computer Security Applications Conference*, 2013.
[29]. A. Moller, F. Michahelles, S. Diewald, L. Roalter, and M. Kranz,
[30]. "Update behavior in app markets and security implications: A case study in google play," in *Proceedings of the 3rd Workshop on Research in the Large.*, 2012.
[31]. S. McIlroy, N. Ali, and A. E. Hassan, "Fresh apps: An empirical study of frequently-updated mobile apps in the google play store," in *Empirical Software Engineering*, 2015.