Dynamic Cache Cleaning on Android

Sean Finley, Xiaojiang Du
Department of Computer and Information Sciences
Temple University
Philadelphia, PA 19122, USA
Email: {s.finley, dux}@temple.edu

***Abstract*—Android developers cache data to improve the performance of their applications. Caching is the technique of transparently storing data such that future requests can be accessed more quickly. At times when a mobile device is not under heavy use the cached data, including sensitive data, can remain on the device for an extensive period of time. This poses a security risk, especially when developers do not take the necessary security measures to protect their users' sensitive information. While there does exist ways to clear application caches built within the Android operating system and third-party applications, these approaches require the user to manually perform these tasks. This paper presents a dynamic cache cleaner that more aggressively pushes out unused cache data. We also present other possible solutions to more effectively manage the cache.**

Keywords—Android, cache, performance, security, memory

I. INTRODUCTION

As smart phone and tablet capabilities continue to expand, society's dependence on them is rapidly growing. Businesses are commonly converting their computer applications to work on mobile platforms like iOS, Android and Windows Mobile [1]. Some organizations go as far as relying on mobile technology to conduct day-to-day tasks. However, as the number of users grows and the capabilities of these devices continue to expand, so do the inherent risks attached [9]. Many security researchers expected major breaches in security and privacy through the vulnerabilities of mobile devices [14]. The use of these devices to email, make online purchases with credit cards, and the possibility of a lost or stolen device can leave sensitive data in the wrong hands [7]. This paper uses the Android operating system to demonstrate privacy risks within mobile applications and their caching techniques. While the Android API offers security measures for application developers to implement into their code, it does not guarantee that developers will follow safe programming practices. Furthermore, the question is raised as to whether the application developers should hold responsibility over users' privacy.

Caching is a mechanism for transparently storing data such that future requests can be accessed more efficiently. Any future requests for the data will be satisfied by a quick query within the application's cache storage partition, eliminating the need to open a network connection and access a web server or any other distant entity to fulfill the request. Temporarily storing data into its local cache can greatly improve the performance of an Android application by avoiding redundant downloading of recently requested information.

While the benefits of caching are obvious, hindrances an improperly managed cache can have on a device's performance, memory, and even security do exist. The Android operating system may delete files from application caches when the device is low on internal storage space; however, it is the responsibility of the application developers to maintain their application's cached files. Android's development website suggests to limiting the space consumed by the cache to roughly 1MB [8]. Unfortunately, these practices are not always followed. One can go into their Android mobile device, view the amount of space their applications are hoarding for their cached files, and discover that nearly all of their applications are well above this recommended amount – Note that some heavier applications such as a web browser are expected to exceed the recommended storage capacity for cached files due to the frequency in use and versatility of the application.

The internal storage space of mobile devices, though continually growing, does not compare to their desktop or even laptop counterparts. Therefore, this internal storage is scarce and expensive real-estate. Additionally, when cached files begin to accumulate and clog the limited available space within the device, it negatively affects the device's performance.

II. THREAT MODEL

Mobile devices are beginning to face many of the same security threats that plague conventional desktop and laptop computers. Since the first virus to target mobile devices in 2004 [2], the rate at which is targeting mobile devices, more specifically Android devices, has grown astronomically; especially in the last few years, due to Android's rise in popularity and security flaws. Figure 1 displays the number of new malware targeting Android devices by quarter in 2010 and 2011.

The seemingly endless amount of applications being developed for the Android platform all but guarantees the occurrence of malicious or vulnerable applications from reaching the market. In a study of Android application security, researchers at Pennsylvania State University [12] found that "Many developers fail to securely use Android APIs. These failures generally fall into the classification of insufficient protection of privacy sensitive information."

Android's open source framework leaves it vulnerable to typical mobile device attacks. Attacks through cellular networks, USB, the internet, Bluetooth, and other vectors expose user private information [13]. Their ability to connect

to 3G and now 4G mobile networks and access the internet through WiFi opens an often times unsecure avenue of approach for hackers who plan to steal users' sensitive information and strip them of their privacy. A 2009 survey conducted by AdMob, a mobile advertising researcher, indicates that mobile device users were greatly raised the use of WiFi hotspots [11]. When connected to unencrypted networks, these devices are left vulnerable to man-in-the-middle attacks. A 2011 report by PCWorld stated that "even remote access to your phone to harvest cached data is now becoming possible" [5].

Andrew Hoog, chief information officer of viaForensics, a digital security firm that focuses on computer and mobile forensics, addressed the risk of caching sensitive data on a mobile device in an interview for Eric Huber's "A Fistful of Dongles" blog, "A growing part of our business is performing testing and analysis for corporations who are trying to mitigate the risks introduced by mobile devices. And the risks are considerable. On the obvious side, an enormous amount of corporate data is cached on mobile devices and is outside the control of the IT department. The data can easily end up on personal computers or even eBay/Craigslist." [4]

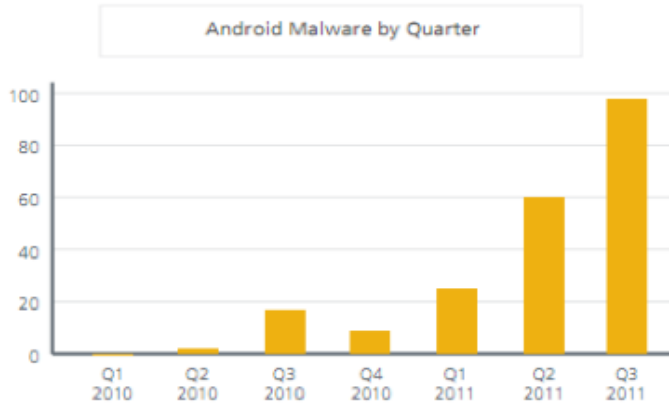

Fig. 1: Android Malware by Quarter [2]

III. Related Work

There exists some cache cleaners in Google's Play Store. These applications simplify the process for a user to clear application caches. Generally, these cache cleaners display a list of applications that are currently storing cached data and how much space these caches are consuming on the mobile device's internal storage. Using these 'one-stop-shops' to clear application caches can be done in one of two ways: clearing a single applications cache, or using a single tap on the screen to remove all cached data from all applications[10].

Albeit this type of model for a cache cleaner does make it easy for a user to more quickly free up some memory than using Android application manager, it does nothing without user interaction. While the Android operating system purges old cached files when additional internal memory is required, stale and potentially private cached files could remain on a device that experiences limited use; despite having a primitive cache cleaner that doesn't manage the device's cache without user interaction.

In an early 2012 article, techShout, a site known for providing critical reviews on the latest in technology, characterizes the seven best cache cleaners meant to speed up a device's performance by clearing redundant data on Android devices [6]. Just by its name alone, the application topping off the list, 1Tap Cache Cleaner, demonstrates the necessity of user interaction.

Asking users to perform such a task through just a single tap of the screen may certainly not be asking for much to keep the user's device to maintaining smooth performance, the average smartphone user is not familiar with how the Android system works, let alone aware of the benefits cache clearing has on device performance.

A few current cache cleaners, like App Cache Cleaner, do possess potential for cache clearing without the need for the user to do so physically [6]. However, these applications have the user define specific intervals to remove all applications cached data, instead of dynamically deciding an appropriate time to remove a specific application's cache.

IV. Dynamic Cache Cleaner Model

The idea behind a dynamic cache cleaner is to remove stale cached files more aggressively without the need of user interaction. This will free up internal memory through transparent background services and in turn result in better device performance. Additionally, it provides more security for the device and its user by removing potentially sensitive files quickly and effectively.

The dynamic cache cleaner measures two things, device idle time and application use, logging key events. This section will discuss these measurements and how they are examined to decide when an applications cache should be cleared.

The cache cleaning process is not a gradual one. Instead, once the idle time of the device and an application reaches a certain point, the application's cache is completely cleared from the device. The equation is simple:

$$\frac{S(D+A)}{I} > T$$

where S is the size of the application's cache in bytes, A being the application's idle time, D as the device idle time in milliseconds, and I the size of the applications cache. I allows applications with less data cached to keep the data for a longer period of time, while the dynamic cache cleaner is more aggressive towards applications with larger caches. T is calculated by the settings made by the user. Once the left surpasses the value of T the application's cache is cleared.

A. Device Idle Time

There are a number of ways to approach tracking whether an Android device is idle. This model tracks CPU load to determine the usage of the device. Other methods, such as checking whether the screen is on or off, hinders additional dynamic components which will be touched upon later. Measuring CPU load tolerates small tasks, such as checking the time and reading an email or text message, allowing the

dynamic cache cleaner to believe the device is still idle when such actions are performed. This sort of behavior leads to more effective cache cleaning. Allowing minor tasks to constantly reset the service's device idle time tracker would hinder the dynamic cache cleaner's ability to clear any application cache.

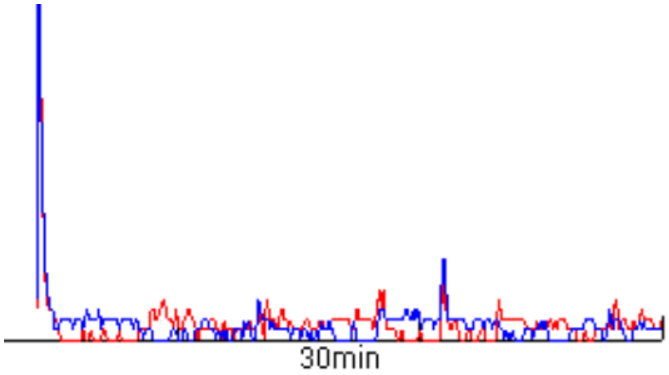

Fig. 2: Samsung Galaxy Tab 2 CPU Spikes Idle Over 30 Minutes

When CPU usage goes above 10% the device is considered under use. However, as shown in Figure 2, most mobile devices can see a spike in CPU load as high as 40% from background services that run while the device is not in use or when simply turning on the screen. These spikes are ignored when surrounded by valleys lower than the minimum threshold and the device is still thought to be idle. Once a steady stream above 15% has been recorded, accepting infrequent drops below this usage percentage is the device considered being used. Additionally, acceptance of peaks when the device is considered idle and valleys when considered in use, prevent the cache cleaner from requiring a high amount of CPU cycles. The process of switching from idle to in-use and vice versa leads to considerable overhead, while the background service required to track the device idle time is comparable to Android's default alarm clock application.

B. *Application Usage*

Figure 3 demonstrates the process the dynamic cache cleaner would follow regarding application usage and when to clear its cache. The idea is simple. Once an application that has cached files is terminated, the dynamic cache cleaner logs the event. At this point one of two things can happen, the application can be reopened, flagging the application and preventing its cache from being cleared, or the applications expiration date can pass, at which point the dynamic cache cleaner will clear the application's cache.

The quantum of time an application can go unused before clearing the cache depends on two things. First, the idle time of the device. If the device is not idle, it will require more time to clear an application's cache, even if the application has not been used in a while. However, once the device goes idle, by its CPU usage dropping below 10%, the quantum begins to drop. A device that has been idle for eight hours or longer will be more aggressive at removing application caches than a device that has only been idle for an hour. Second, the amount of memory an application's cache is using affects the amount of time before its cache is cleared. An application with more cached data will have smaller quantum than one using little storage. This is because the more data stored within an application's cache, the greater chance there is some sensitive information stored within. Additionally, those applications with small caches are not taking up enough memory to negatively affect the device's performance.

V. EVALUATION

For the evaluation, tests are performed on a Samsung Galaxy Tab 2.0 with Android 4.0 Ice Cream Sandwich, testing device memory, performance and security. [Note: Identical test were performed on a Samsung Galaxy Nexus smartphone running Android 4.1 Jelly Bean with nearly identical trending results.] Test cases include the dynamic cache cleaner services

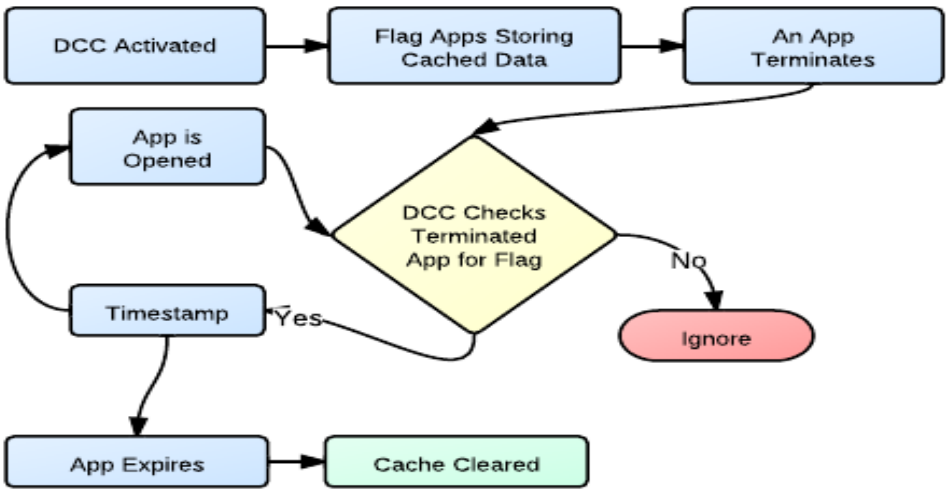

Fig. 3: Dynamic Cache Cleaner Application Usage Flowchart

running and a control experiment where there dynamic cache cleaner is not running on the device. Additionally, tests are done on security by tracking a sensitive file while it is cached by an application and determine the length of time which the file remains on the device, only in the application's cache and no other directory. Each test case is begun with all application caches cleared.

A. Memory Consumption

As mentioned previously, without any user activity to prevent the occurrence, caching from multiple applications on an Android device accumulates and can take up a large portion of the device's internal memory. While the Android operating systems clears out stale application caches when it requires more internal storage than available, this is not aggressive and only clears the minimal amount to allow the requesting process to run.

This specific model tablet comes with 4.59GB internal storage. After initial setup of the device with stock Android 4.0 and installing popular applications from Google's Play Store, the device was down to 3.62GB available storage. [Note: Applications installed include two web browsers, three flash games, two social media, two multimedia, and three news source applications.]

Under the control case, with the dynamic cache cleaner installed on the device but no services running, the applications' caches were able to consume a big portion of the remaining available storage. After just four hours of heavy usage of various applications, available internal storage was down to 2.36GB. This means that application caches were able to collectively take up more than a gigabyte of memory. Considering Android's development website, which gives guidelines for application developers to follow to ensure their application is running as smooth as possible and with no deficiencies to the device, recommends no more than 1MB of cache storage per application.

The dynamic cache cleaner has little effect when the device is under heavy use. Under similar conditions, heavy use rotating between all tested applications, available internal storage after four hours was 2.65GB. This is only 0.29GB less than the control case. This small amount could possibly be a result of the dynamic cache cleaner or just from the random actions performed by the user. However, after setting the control case device down and returning to the device, regardless of length of absence, the cached storage never drops, remaining at 1.26GB. This is not the case when the cleaning service is running. After 24 hours (default setting) of idle time, application caches are completely cleared from the device.

Figure 4 displays memory consumption of a device under heavy use for four hours then left idle for an additional ten hours. As shown, the dynamic cache cleaner is able to keep application caches less than that of the control case even while the device is in use. This demonstrates the need for recording the lack of use of specific applications. Although the device is under heavy use, if a particular application is not used for quite some time, its cache will be automatically cleared after given time.

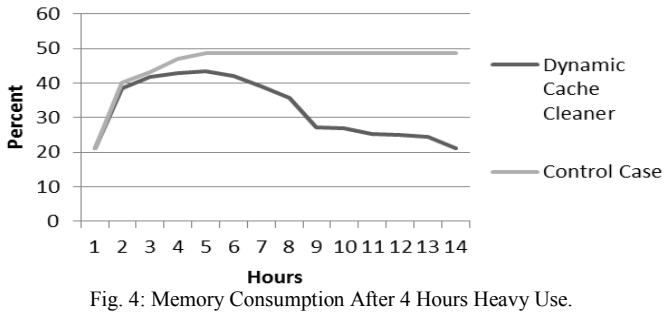

Fig. 4: Memory Consumption After 4 Hours Heavy Use.

Most notable is the effect the dynamic cache cleaner has on the device when idle. As mentioned earlier, the device in the control case, when idle, has no mechanism garbage collecting stale data within the cache. In fact, the only time garbage collecting is performed is when the device is being used and requires additional free internal storage. This is not the case with the dynamic cache cleaner. Even after the device is not used, it continues to clear application caches, more effectively at that.

Lastly, Figure 4 shows how the dynamic cache cleaner's purpose becomes clearer when the device sits idle; to clear unnecessary caches that are consuming internal storage, negatively effecting performance, and potentially harboring sensitive information.

B. Performance

The performance of the device is directly affected by the size of application caches. When there is nothing inside an application's cache, it requires more time to fetch information from the web server. However, when there is too much memory being used by application caches it can negatively affect the device's overall performance.

Performance of running single threaded processes was tested using Linpack for Android. Under both cases the results were similar with no application's storing cached data. After one hundred single thread tests both cases averaged 2.28 seconds. This is evidence that the dynamic cache cleaner's background services have no negative effect on device performance when it is not actively removing application caches.

The differences between the two cases are examined after heavy use followed by idle time. This is when the dynamic cache cleaner begins to garbage collect and clear up internal storage. When the device allowed excessive caching from applications it showed a negative side-effect on performance. At 0.5GB total cache, an application with cached data performed better than with no cache. However, at 1GB total cache, overall device performance suffers. Average single thread start time was up to 2.35 seconds. At 1.25GB it was up to 2.52 seconds. Continuing this trend, at 1.83GB total cache on the tablet, which was the max we could reach due to Android's own cache cleaning mechanism, to run a single thread took 3.09 seconds.

When the device was under heavy use and the dynamic cache cleaner service running, similar results were achieve. However, continuing the trend the benefits were displayed after the device went unused after the period of heavy use.

After cleaning some caches, results began to decrease until all application cache memory was cleared, at which point it was back to 2.28 seconds for starting a single threaded process.

C. Security

Device integrity can be aided with the dynamic cache cleaner's services. When sensitive information is cached by an application, which is never a good programming practice, without the cleaning service it has the potential to remain on the device for a considerable length of time, even days. To test the effects of the dynamic cache cleaner, an application was created whose only function was to cache information input by the user. Under the control experiment this file was able to remain on the device until the operating system cleared it during garbage collection to free requested internal storage. Because Android uses a least-recently-used algorithm, this file would only be removed after all older cached files were removed first. Under ideal settings, where the sensitive file is the first file cached, then the device is run under heavy use, this file was removed in little as 2 hours 43 minutes. Similar results occur while the cleaner is running. Once again, as the device continues to sit idle with the sensitive file on the device, with no dynamic cleaning mechanism running it may and will remain on the device until the user manually clears the application's cache. With the dynamic cache cleaner, this file will remain on the device no longer than 24 hours of idle time as long as the device remains powered.

VI. CONCLUSION

While the benefits from caching with respect to device performance are apparent, excessive caching can lead to unreasonably high memory consumption. This in turn has a negative impact on device performance. In addition, poor programming practices may lead to caching of sensitive information. Under certain circumstances, where the application itself or the Android operating system are not aggressive enough to clear out such a file, it has potential to remain on the device for a great length of time. Mobile device caches are becoming victims of malicious hackers and any sensitive information within an application's cache is vulnerable. The dynamic cache cleaner is set to help resolve these issues by more aggressively, and in turn more effectively, garbage collecting stale cached files.

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation under grants CNS-0963578, CNS-1022552, and CNS-1065444.

REFERENCES

[1] Google. Android – an open handset alliance project. http://code.google.com/android/.2008.

[2] D.-H. Shih, B. Lin, H.-S. Chiang, and M.-H. Shih, "Security aspects of mobile phone virus: a critical survey," Proceeding USENIXATC'10 Proceedings of the 2010 USENIX conference on USENIX annual technical conference, 2008.

[3] Rao, Leena. "McAfee: Nearly All New Mobile Malware in Q3 Targeted at Android Phones". 2011. TechCrunch. http://techcrunch.com/2011/11/20/mcafee-nearly-all-new-mobile-malware-in-q3-targeted-at-android-phones-up-37-percent/

[4] Huber, Eric. "AfoD Blog With Andrew Hoog on Mobile Device Security and Forensics". ViaForensics. 2011. https://viaforensics.com/mobile-security-category/afod-blog-andrew-hoog-mobile-device-security-forensics.html

[5] Geuss, Megan. "App Makers May Be Exposing Your Sensitive Data to Hackers. PCWorld. 2011. http://www.pcworld.com/article/237553/app_makers_may_be_exposing_your_sensitive_data_to_hackers.html

[6] "7 Best Android Cache Cleaner Apps". TechShout. 2012. www.techshout.com/features/2012/08/best-android-cache-cleaner-apps/

[7] Emm, D. 2006. Mobile malware – new avenues. Network Security, 2006(11), 4-6.

[8] "Storage Options". Android Dev Site. 2012. http://developer.android.com/guide/topics/data/data-storage.html#filesIntern

[9] Landman, Max. "Managing Smartphone Security Risks." Information Security Curriculum Conference. 2010.

[10] "Cache Cleaner – Android App Review." AndroidAppdiction. 2011. http://www.androidappdictions.com/archives/415

[11] Aggarwal, Mayank and Troy Vennon. "Study of MITM Attacks Against Smartphone Devies ." SMobile Systems. 2009. http://threatcenter.smobilesystems.com/wp-content/uploads/2009/11/MIMT-Whitepaper031.pdf

[12] Enck W, Octeau D, McDaniel P and Chaudhuri S. "A Study of Adroid Application Security". Proceedings of the 20th USENIX Conference on Security. 2011.

[13] Shabtai A, Fledel Y, Kanonov U, Elovici Y, Dolev S and Glezer C. "Google Android: A Comprehensive Security Assessment". IEEE Security and Privacy. 2010.

[14] Becher, Michael et al. "Mobile Security Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices." IEEE Symposium on Security and Privacy. 2011.