e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:01/January-2023 Impact Factor- 6.752 www.irjmets.com

ANDROID VS IOS: A HOLISTIC SECURITY OVERVIEW

Soham Jana*1, Sayak Adhikari*2

*1,2Independent Reasearchers, Kolkata, West Bengal, India.

DOI : https://www.doi.org/10.56726/IRJMETS32838

ABSTRACT

In today's world every single individual has a phone if not a smartphone. Although the smartphone users around the globe approximates to 5 Billion which is forecasted to rise to somewhere around 7 Billion by 2025. Needless to point out the usability and compactness of the equipment, one of the most important factors concerning both manufacturers and the well informed user is data. Since smartphones have become a pivotal device in the computing environment, that one's privacy and scope for sensitive data leakage is a burden to many. Hence, security of data regardless of operating systems has become imperative. The way the security issues are established and resolved resides in the architecture and the OS; which boils down to two main players in the market, android and iOS. In this paper we would like to delve into the attack surfaces on both hardware and software side of the OSes. We would try to explore the differences in the systems based on a data security standpoint and lastly try to provide remediatory suggestions for the issues.

Keywords: Android, Ios, Mobile Security, Privacy.

I. INTRODUCTION

A smartphone has become an un-expendable element of our lives so much so that it has surpassed the functionality and convenience of a traditional personal computer. From calling a person to monitoring your heart rate, an average smartphone has trumped the usage of PCs with respect to visits & bounce rates to websites (without considering the traffic of downloaded applications) [1]. With that said we have primarily two OSes to compare: iOS and android. While iOS is sold by only one manufacturer, for android, we see a lot of different flavours by different manufacturers. Android still leads in the sales of smartphones[2]. If we make a comparison of the version of OSes, we would find that Apple being the sole manufacturer for iOS has more users with the latest OS version than that of android counterparts.

Our approach to make a practical comparison between the security viewpoints of the two types of devices are two-fold: Hardware oriented and software oriented.

II. HARDWARE ORIENTED VIEWPOINT

Although, there's an apparent difference in the entire concept of implementing the software in Android and Apple devices, the hardware architecture remains quite resembling. Majority of the mobile devices use ARM based architecture for its low power consumption, which serves as a primordial aspect for these kinds of devices. We see a plethora of manufacturers(Apple, Qualcomm, Samsung etc.) who tweak the ARM oriented processors to their liking. For example, Qualcomm's Snapdragon 865 is a customized version of ARM Cortex A77 and A55 while Samsung's Exynos 990 tweaks ARM Cortex A76 and A55, for their CPUs[3] while Apple's A13 is based on ARMv8.3-A [4].

In 2017, we see a team of Chinese hackers crack Apple's secure enclave(processors A7-A11 bionic) to successfully retrieve private keys[5]. Few members from Google project zero team found a vulnerability[35] that exploits inherent flaws present in nearly all the processors that can be used to access sensitive data from memory even after security procedures to prevent it. For end-users, stored files, credentials & keystrokes can be compromised if browsers and device manufacturer (OTA) updates are not installed.

Meltdown is a vulnerability which exploits side effects of out-of-order execution on modern processors to read arbitrary kernel-memory locations that may leak personal data. Spectre, on the other hand lets a victim perform operations speculatively instead of the normal trajectory of program's execution, thus leaking victims's private data via a side channel. Effects of speculative processor vulnerability on Intel, Arm, Amd systems for meltdown are briefed as follows[6]:

1. Meltdown is used to read accessible addresses from the address space

2. Meltdown is used to leak the randomization of the direct physical map of KASLR (Kernel Address Space Layout Randomization).

3. Direct physical map is used to dump the content of the physical memory in hexadecimal format.

Although, both meltdown and spectre has affected MacOS and iOS along with other OSs, We could not find a proper source which has implemented these exploits on any vulnerable Apple device, so we are left to believe that on the occasion of exploiting Apple devices properly, we may see data being leaked would be encrypted instead of being in any readable format. And no known exploits were found for Apple devices. It was fixed with Dec 2017 and Jan 2018 security patches from Google and other manufacturers for android devices.

Google Titan M, Samsung knox (Samsung's Real-Time Kernel Protection) arm trust zone & Apple's secure enclave vulnerabilities:

- Vulnerabilities in Google TitanM:

CVE-2021-1043 & CVE-2021-0454/0455/456: contents and size of buffer can be controlled leading to local privilege escalation without user interaction. We may not see apparent exploits of vulnerabilities in the chip but [7] suggests how an attack can go around exploiting the mentioned approaches. Vulnerabilities have been mitigated with newer updates.

- Vulnerabilities in Samsung Knox:

We see the following exploits of vulnerabilities of Samsung's ARM trust zone implementation(which have been mitigated with updates) in paper [8]:

1. Accessing system calls(map physical memory, creating threads).

2. Communication with runtime manager(innit like process).

3. Code execution is EL3(highest ARM privilege level).

4. Public key related key used to sign secure driver is retrieved and can be modified.

- Vulnerabilities related to Apple's secure co-processor and other co-processors:

1. On Apple's co-processor, data of Key generation is stored temporarily in an un-encrypted part of the RAM, thus making it susceptible to attackers.[9]

2. For other co-processors, vulnerabilities posed by side channel attacks[10](remediation of whoch has been discussed under keystore part).

III. SOFTWARE ORIENTED VIEWPOINT OF ANDROID

The android security model stands on these components:

Google Bouncer: Acts as an emulator for the apps to be installed. It basically installs the app, checks for security threats like if the application is using all the allowed APIs, furthermore control required permissions for various apps and users can schedule permissions for specified time[14].

Sandboxing: It is also called Dalvik virtual machine, a VM where each app runs inside its own sandbox, and the app cannot access the data pertaining to other applications(with few exceptions). Directory /data/data is where all the application related data are stored, the content of which is accessible after rooting the device. It is implemented to prevent attacks like remote code execution, buffer overflows and stack smashing [15]. We see CVE-2019-11932 is a Whatsapp Remote Code execution which allows users to access backups and files in whatsapp due to memory corruption[16], which was fixed with newer updates[6]. Secure Inter Process Communication(IPC): It is used when applications communicate with other applications or an application is a multi-process application which has to share its components(like activity, services) to different processes[16]. So there is an ideal isolation between the apps and if one process fails, it wont make others crash. There are three types of IPCs in linux ie, AIDL, messenger, broadcast; of which AIDL is used as it requires two-way communication and support concurrent operations. There are few IPC mechanisms such as:

1. Binders: Remote procedure call mechanisms used in both in-process and cross-process calls.

2. Services: The background processes to start activities.

3. Intents: It is an object which relays an intention to do something. If a component trusts the intent's action with verification, an adversary might be able to perform a functionality through a malicious app. Remediations for flawed intent:

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:01/January-2023 Impact Factor- 6.752 www.irjmets.com

- Signature or Signature Or System permissions limit components data exposure to trusted applications.
- Making distinct components for inter & intra process. Exported components should check for the caller's identity.
- Checking caller's identity before sending return values from exported components.

4. Content Provider: It is an interface that provides access to the data on the device[17]. It is vulnerable to a variety of sql injection attacks and path traversal attacks. There are certain remediations(if content provider is a sql database) against malicious input meant for sql injection by: var selectionClause = "var = $mUserInput" // Constructs a selection clause by concatenating the user's input to the column name.

Application signing: It allows app developers to sign the app before running it on android and export the app signing key to the android Play app signing. It reduces a lot of hassles for developers who want to update the application legitimately. Self signed certificates are allowed, letting anyone create the certificate by signing an application. With vulnerability CVE-2017–13156 (Janus), attacker can append a dex file to apk file without affecting its signature. Janus affects Android devices (Android 5.0 < 8.1) when signed with v1 signature scheme, so it becomes imperative to use version2.

User granted permissions: App developers need to explicitly specify the permissions if they have to handle sensitive data, which is stored in AndroidManifest.xml file[18]. Permission protection levels include:

1. Normal: used when there is no risk leveraging user privacy and operation for other apps.
2. Signature: only used for apps trying to get permission(s) from apps with same signing certificate.
3. Dangerous: When an app uses permission to access user's sensitive data for essential functioning. Users are requested to grant/deny these permissions.

Making sure the user isn't requesting the FINE_LOCATION permission, limiting access to location to apps running in background. Testing permission model for unnecessary permissions since the number of permissions increases the probability for users to deny those.

Keystore: It stores the cryptographic keys that makes it difficult to be extracted from devices without apt priviledges.

We see a lot many attacks(like buffer overflows leading to code execution in the keystore) which requires physical access however in [19] we see an attack which doesn't require any physical access to the device. It is a forgery attack where attacker forge keys by tampering key handlers with a malware, therefore letting the adversary intercept and even tamper exchanged messages.

The following remediations were noted:

- Use of Galois Counter Mode(GCM) to implement AE encryption scheme instead of the h-CBC in keystore, since it is a free alternative to OCB (Offset Codebook Mode).
- Implementing hardware based solutions like Trusted Execution Environment(TEE) to encrypt the keys in AE schemes for key binders, although it would be able to detect forged key handlers but fuzzy attacks making those key handlers valid would lead to a stack overflow attack among others.

Android Kernel:

1. For android 4.3 and above SELinux provides Mandatory Access Control(MAC) i.e, a software has to gain its priviledge to root user account to write to raw block devices. As opposed to discretionary access control (DAC) environments where any user can write to raw block devices[20]. This ensures that a software runs on a minimum privilege level thus reduces the chance of unpriviledged processes from overwriting or transmitting data. Hence reducing the risk of priviledge escalation attacks.
2. Classes of system files like netd, init, and void run as root thus they can access system files, but with SElinux, these system files have R/W permission only to the system server domain and not the entire system files.
3. Since SELinux is a labeling oriented system, every file, directory, ports in the operating system has a label and rules are written for controlling the access of these labels which are referred to as "policy". Assuming if the system is compromised, the attacker cannot access the home directory of an user unless stated otherwise in policy.

There are three kinds of Enforcement levels(of these policies): permissive(mode in which these policies to only be logged), enforced(modes in which these policies are logged and enforced) & disabled(mode is which is neither logged or enforced; not in use).

4. Macros are used to simplify many kinds of complex permissions and to reduce the likelihood of failures related to it. Eg, define(`w_file_perms', `{ open append write lock map }').

5. Security Context(label) and Categories are used to isolate app data from one app to another and from one user to another.

Additinal mitigatory suggestions[21]:

- Avoiding symbolic links(symlinks) as they are often read as files, which can lead to exploits.
- Class of functions running as root should not be able to access app data.
- Commands such as chmod and chown should be restricted to the associated domain of a set of files.

IV. SOFTWARE ORIENTED VIEWPOINT OF IOS

IOS's security perspective is as follows:

Secure Boot Chain & App code signing:

Step1. Boot ROM--> 2. LLB--> 3. iBoot--> 4.iOS kernel--> 5. Apps(code signing)

The boot rom is implicitly trusted in the entire mechanism, it is hard coded during chip fabrication. When the device turns on, it executes the code from this read-only memory, it also contains the Apple root CA public key which in turn is used to verify LLB(Low level bootloader) is signed by Apple. LLB does the same job with iBoot, verifies whether it is signed by Apple, and the same is done to the iOS kernel by iBoot. Thus, this kind of boot chain allows Apple to run only on validated Apple devices.

After the kernel is loaded it runs the required processes and applications which are signed by a trusted Apple issued certificate. For any third party apps, it must be signed by Apple issued certificates only. From step 1 to 5, iOS ensures that no untrusted code is executed, therefore preventing kernel level rootkits from being booted.

App Sandboxing: Similar to android, each app runs its own sandbox which doesn't let other apps share its data. Unlike android, app data in iOS are placed in exclusive directory (which are unique to each app)for each application which are not accessible unless jailbroken.

Developer Account: Any developer who wants to develop an app for iOS has to register with Apple and join their developer program for legitimacy. The developer information is verified by Apple before issuing the cert for signing applications[22]. We find an exploit used by an attacker to plant a backdoor in developer's systems, which in turn taints the xcode executables[23]. An extra layer of security was added by Apple where two factor authentication was required to sign[24].

Software Updates & Downgrade prevention: Software updates for iOS are more frequent with respect to android, across all the iOS devices. Unlike android Apple prevents iOS devices from downgrading its OS by checking which OS user is currently in and which OS needs to be installed, along with that it is checked whether the file is legitimate by verifying whether the update files came from Apple and then signing those files for iTunes.

Plist files, NSUserDefaults & CoreData: Property list or plist files are used by applications(esp. gaming applications) to store codes and/or credentials in .xml or binary format as a key value pair. NSUserDefaults class provides an interface for interacting with default system thus allowing the application to in sync with the user preferences. It also stores its data in plist files. Core Data is a mapping between the interface and the database the most sought after way of storing persistent datas. It provides a visual data model & stores data in SQLite databases with z-prefixed tables. We might see an application storing sensitive data in .plist/.db files[25].

In CVE-2019-8660, we find manipulation of unknown input leading to memory corruption where arbitrary code execution might happen[26] which were remediated in iOS 12.4 update.

Keychain: It is an encrypted password manager used to store credentials, it can only be accessed by apps which are members of the appropriate keychain group. We see an attack where attackers in [27]&[28] compromise the access to the keychain without prompting any passwords from users. Further in [27], the attacker shows that there exists tools used by admins with which they can create, delete and migrate keychains. Although it has been fixed with updates, it can also be compromised if the attacker has access to iTunes backup, which is encrypted[29]. When storing data in the keychain, use the most restrictive protection class (as defined by the kSecAttrAccessible attribute) that still allows your application to function properly, to prevent exposure of keychain from iTunes developers can use This Device Only class and for more sensitive data, rely upon user authentication with application before storing data.

App Types: Using native apps as opposed to hybrid counterparts, gives app an edge in security as code of native app(given it is a non-jailbroken device) is stored in encrypted binary executable which in turn is decrypted by processor while loading it in RAM, as opposed to hybrid apps which has chunks of code in human readable format. While Hybrid apps are much cost efficient, have a single code base and easier to develop; developers must develop apps which deal with extremely sensitive information in Native terms[30][31].

IOS Kernel[32]:

1. Kernel Integrity Protection (A10 onwards): It allows iBoot to load kernel to a protected memory region, which after startup blocks further writing into the protected memory region, preventing modification in kernel data structures and code reuse attacks like Jump-Oriented Programming.

2. Fast Permission Restrictions (A11 onwards): It is a register that can remove execute permissions per thread without the expense of a system call; providing an extra layer for remediation web based attacks, permission re-delegation, vertical privilege escalation and/or even deployment of payloads.

3. System Coprocessor Integrity Protection(A12 onwards): Similar to Kernel Integrity Protection(KIP), iBoot loads firmware of each co-processor to a protected memory region that's separate from KIP. It is meant to prevent:

- Executable mappings outside its part of the protected memory region
- Writable mappings inside its part of the protected memory region

4. Pointer Authentication Codes: It restricts modification of function related pointers and return addresses to prevent exploitation of memory corruption bugs and attacks like kernel buffer overflow & return-oriented programming.

We witnessed threats pertaining to kernel (which have been mitigated by subsequent updates):

Exploitation of unauthenticated kernel memory corruption vulnerability[33].

Allowing adversaries to execute arbitrary code with kernel privilege[34].

V. CONCLUSION

In order to determine which performs better than the other in terms of various areas of security technology, we try to compare the strengths and weaknesses of the two most widely used smartphone operating systems (Android vs. iOS). Android surpassed iOS to take the top spot in sales. Unrelated to security considerations might be the cause of this. Android is the operating system for a range of devices, including the first in sales, unlike iOS, which is only compatible with Apple devices (Samsung smartphones). assuming that the two operating systems utilize different security protection strategies in addition to certain common mechanisms. The iOS security architecture was tight, layered, and complicated. Contrarily, the Android model heavily relies on the permission-based approach in addition to other mechanisms that are derived from Android elements like the Java programming language and its core operating system, Linux. Though Android is working to catch up with upgrades, iOS seems to be more secure. We compare the protection types, their strengths, and their limitations based on both hardware and software considerations. Security plays a powerful role in both due to the use of some protection systems and the contrasting perspectives on others. Comparing the two platforms reveals that Android has more security flaws than iOS. The advantages of iOS outweigh those of Android by a small margin, although the latter is leading in sales. Furthermore, we would like to delve more into the exploits for meltdown and spectre in iOS, for a stricter comparison.

REFERENCES

[1] https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:01/January-2023 Impact Factor- 6.752 www.irjmets.com

[2] https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/
[3] https://www.theverge.com/circuitbreaker/2020/5/26/21267893/arm-cortex-a78-mali-g78-cpu-gpu-designs-smartphones-2021-samsung-qualcomm-apple
[4] https://medium.com/0xmachina/the-apple-a13-bionic-d7f635123eec
[5] https://9to5mac.com/2020/08/01/new-unpatchable-exploit-allegedly-found-on-apples-secure-enclave-chip-heres-what-it-could-mean/
[6] https://github.com/IAIK/meltdown
[7] Melotti D., Rossi-Bellom M., and Continella A. Reversing and Fuzzing the Google Titan M Chip. Proceedings of ROOTS'21: Reversing and Offensive-oriented Trends Symposium November 2021 Pages 1–10.
[8] Peterlin M., Adamski A. and Guilbon J. Breaking Samsung's ARM trustzone, Quarkslab, Black hat USA 2019.
[9] Mandt T., Solnik M. and Wang D. Demystifying the Secure Enclave Processor, Black Hat.
[10] Hwang D., Tiri K. and Hodjat A. AES-Based Security Coprocessor IC in 0.18-um CMOS With Resistance to Differential Power Analysis Side-Channel Attacks, IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 41, NO. 4, APRIL 2006.
[11] https://arxiv.org/abs/1702.08719
[12] https://github.com/osusecLab/SgxPectre
[13] https://patents.google.com/patent/US8832465
[14] https://www.androidauthority.com/bouncer-android-app-905059/
[15] Google_Android_A_Comprehensive_Security.
[16] https://blog.hacktivesecurity.com/index.php/2020/04/05/android-ipc-part-1-introduction/
[17] https://developer.android.com/guide/topics/providers/content-provider-basics
[18] https://medium.com/android-news/permission-protection-levels-969c9d0a7ebc
[19] Sabt M. and Traore J. Breaking Into the KeyStore: A Practical Forgery Attack Against Android KeyStore. 21st European Symposium on Research in Computer Security (ESORICS 2016).
[20] https://source.android.com/security/selinux/concepts
[21] https://source.android.com/security/selinux/implement#use_cases
[22] https://developer.apple.com/programs/
[23] https://thehackernews.com/2021/03/hackers-infecting-apple-app-developers.html
[24] https://www.macrumors.com/2019/02/13/developers-two-factor-authentication-required/
[25] https://resources.infosecinstitute.com/topic/ios-application-security-part-20-local-data-storage-nsuserdefaults-coredata-sqlite-plist-files/
[26] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-8660
[27] https://www.wired.com/story/keysteal-apple-keychain-attack-shenanigans/#:~:text=In%20early%20February%2C%20an%2018,steal%20them%20%3A)%22%20he%20wrote
[28] https://www.forbes.com/sites/thomasbrewster/2017/09/25/apple-mac-os-x-high-sierra-vulnerabilit-hacker-steals-passwords/?sh=12023cea3200
[29] https://books.nowsecure.com/secure-mobile-development/en/ios/use-the-keychain-carefully.html
[30] https://mentormate.medium.com/security-in-ios-protecting-your-business-from-hackers-9053669bc230
[31] https://sagaratechnology.medium.com/what-to-choose-between-native-or-hybrid-development-f214830962cd
[32] https://support.apple.com/en-in/guide/security/sec8b776536b/web
[33] https://www.financialexpress.com/industry/technology/critical-ios-bug-could-have-given-hackers-complete-control-of-your-iphone-over-wi-fi/2142331/
[34] https://threatpost.com/apple-patches-actively-exploited-zero-day-in-ios-macos/168177/
[35] https://www.bgr.in/features/spectre-and-meltdown-security-vulnerabilities-explained-how-to-protect-and-more-551473/