© 2015 IJSRST | Volume 1 | Issue 4 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Engineering and Technology

Mitigation of Security Attack in Android Application using Pin Tool A Review

Sisara Lalji C., Prof. B. V. Buddhadev
Department of Information Technology, Shantilal Shah Government Engineering College, Bhavnagar, Gujarat, India

ABSTRACT

The popularity and adoption of smartphones has greatly stimulated the spread of mobile malware, especially on the popular platforms such as Android. In light of their rapid growth, there is a pressing need to develop effective solutions. In the past few years, mobile devices (smartphones, PDAs) have seen both their computational power and their data connectivity rise to a level nearly equivalent to that available on small desktop computers, while becoming ubiquitous. On the downside, these mobile devices are now an extremely attractive target for large-scale security attacks. Mobile device middleware is thus experiencing an increased focus on attempts to mitigate potential security compromises. In particular, Android incorporates by design many well-known security features such as privilege separation. In this thesis the Android security model and some potential weaknesses of the model is described. Thesis provides taxonomy of attacks to the platform demonstrated by real attacks that in the end guarantee privileged access to the device and mitigation technique for the same attack would be proposed. The result analysis and testing would be done on mitigation technique.

Keywords: Dynamic Analysis, Runtime, Binary Instrumentation, Pin, Pin tool, Intel, Just-in-time compiler, security attack, android Attack.

I. INTRODUCTION

(1) Pin and Pin Tool

Instrumentation

Instrumentation is a simple technique for inserting any extra line of code in to an application to observe its behavior. It can be performed at various stages – inside the source code, at compile time, post link time, or even at run time. Source Code Instrumentation is a way to instrument source programs and Binary Instrumentation is to instrument binary executable directly Static binary instrumentation (SBI) occurs before the program is run phase, a phase in which we can rewrite executable code or object code. Dynamic binary instrumentation (DBI) is done at run time.

Program Analysis
Type of program analysis

	Static Category	Dynamic Category
Source Type	Static source analysis	Dynamic source analysis
Binary Type	Static binary analysis	Dynamic binary analysis

Table (1). Types of Program Analysis

Static Analysis and Dynamic Analysis:

Static analysis is the process of analyzing the source code or machine code of the program without need of running it **Dynamic analysis** is the process of analyzing program as it executes or at the runtime**.**

Source Analysis and Binary Analysis:

Source analysis is the process of analyzing programs at the level of source code. Source analysis are generally done for the points of programming language constructs such as expressions, statements, functions, and variables.

Binary analysis is the process of analyzing programs at the level of machine code, that stored either as object code (pre-linking) or executable code (post-linking). In this category, we have analysis that are performed at the level of executable intermediate representations, such as byte-codes, that runs on a particular virtual machine. Binary analysis are generally done for the points machine entities, such as registers, memory locations, procedures, and instructions.

A Pin

Pin has been the framework of choice for researchers working on program analysis and related tools. It can be used for several purposes, but mostly for program analysis (memory allocation analysis, error detection, performance profiling, etc...) and for architectural study (processor and cache simulation, trace collection, etc…). PIN is a dynamic binary instrumentation engine or framework. Pin is used for the instrumentation of software programs. It supports many platforms like Windows, Linux, Mac OS and Android executable for IA-32, and Intel(R) 64[4]. The Pin allows a programmer to insert any arbitrary code (written in C or C++) at arbitrary places in the executable (run time of any program). The code is added dynamically while the executable (program) is in the running phase. The input to this compiler is not byte code, but a regular executable. Pin dynamically re-compiles the application during execution. The Pin kit includes many tools (they can be found at: pin-w-x-y-android/source/tools). The tools are provided as source files .Pin provides the framework and API.

Pin Architecture:

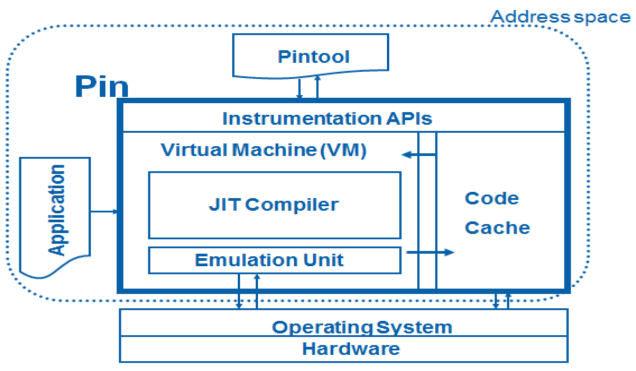


Figure 1. Pin Architecture [1]

Pin consists of a virtual machine (VM), a code cache, and an instrumentation API invoked by Pin tools. The VM consists of a just-in-time compiler (JIT), an emulator, and a dispatcher. After Pin control of the application, the VM coordinates its components to execute the application. The JIT compiles and instruments application code, which is then launched by the dispatcher. The compiled code is stored in the code cache. The emulator interprets instructions that cannot be executed directly. It is used for system calls which require special handling from the VM. (E.g. system calls)

Pin Tool

Pin tool is the instrumentation program. Pin tools run on Pin to perform meaningful tasks. The inscount pin tool is used to find out the number of instructions in the running program.

Instrumentation consists of two components:

1. A mechanism that decides where and what code is inserted

2. The code to execute at insertion points

These two components are instrumentation and analysis code.

(2) Android

Android is a powerful Operating System supporting a large number of applications in Smart Phones. These applications make life more comfortable and advanced for the users. Hardware's that support Android is mainly based on ARM architecture platform. Android comes with an Android market which is an online software store. It was developed by Google. It allows Android users to select, and download applications developed by third party developers and use them. There are around 2.0 lack+ games, application and widgets available on the market for users. Android applications are written in java programming language. Android is available as open source for developers to develop applications which can be further used for selling in android market. There are around 200000 applications developed for android with over 3 billion+ downloads.
Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. For software development, Android provides **Android SDK**

(Software development kit). Read more about open source software.

Android uses Following Tools:

Eclipse, ADT Plugins, SDK toolkit, AVD toolkit.

Android Architecture

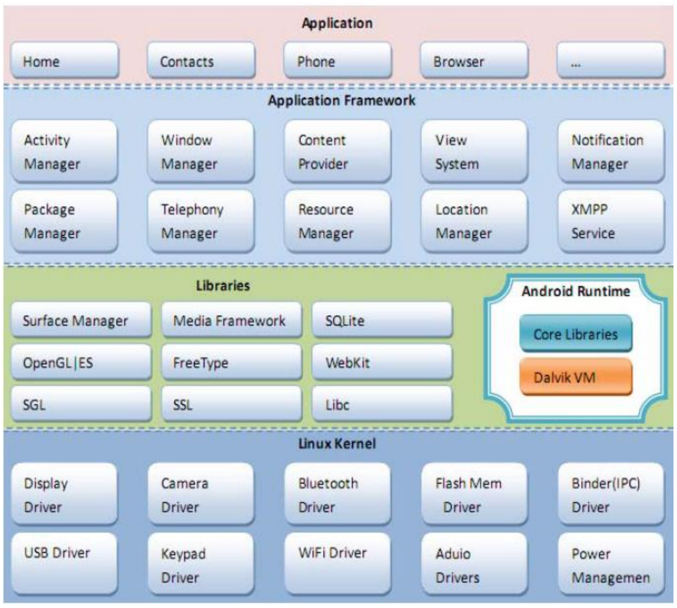
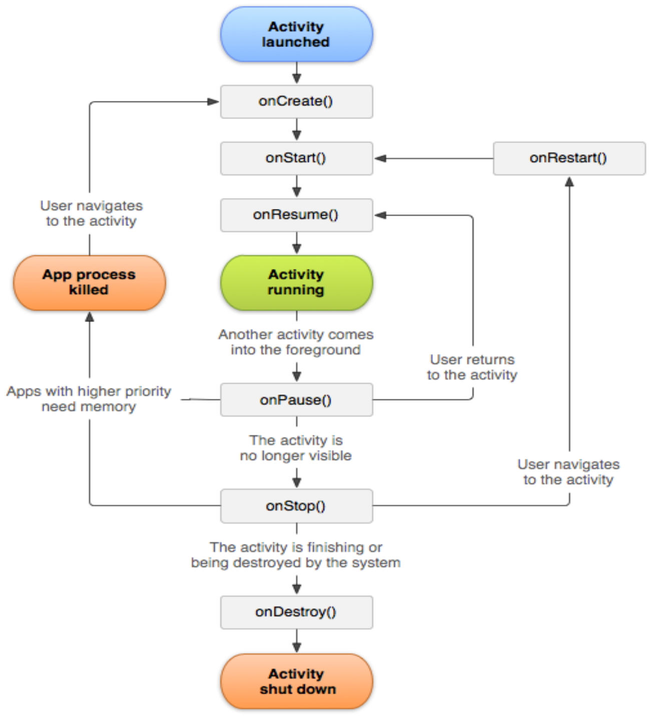


Figure 2. Android Architecture [3]

Android Activity Lifecycle

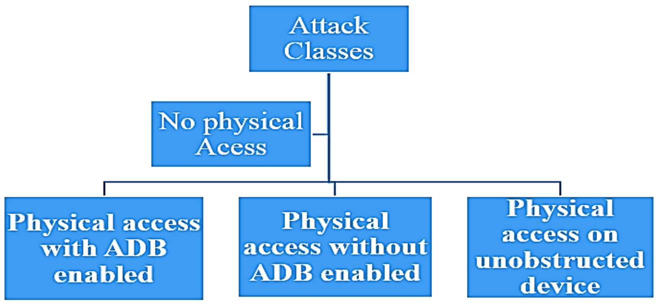


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 3. Android Activity Lifecycle [2]

(3) Mitigation

Mitigation is the effort to reduce loss life and property by Lessing the impact of disasters.



Attack Classes
No physical Acess
Physical access with ADB enabled
Physical access without ADB enabled
Physical access on unobstructed device

Figure 4. Attack classes

No Physical Access

Attack circumstances where it is impossible to gain physical access to a user's device. Then the attacker must get the user to perform actions on the attacker's behalf. Such remote attacks commonly rely heavily on social engineering [5]. To achieve the appropriate initial access to the user's device an attacker must get some malicious software running on the device. To run code remotely on a user's device, the attacker typically must convince the user to either download a malicious application or access malicious content via one of the applications already installed on the device. If the attacker can exploit vulnerability on the user's device, then this access may be used further to gain privileged access.

Physical Access with ADB Enabled

If the attacker finds a device left unattended, yet obstructed via a password or screen lock, the attacker may be able to exploit the device through the Android Developer Bridge.

Physical Access without ADB Enabled

If the attacker finds an obstructed Android device left unattended, but is unable to use the ADB service, the attacker may still gain privileged access via recovery boot.

Physical Access on Unobstructed Device

In some cases the attacker may actually have access to a device without a password protected screen lock. Such a situation allows the attacker to actually leverage any other attack method since the attacker can choose to install applications, visit malicious websites, enable ADB on the device, etc.

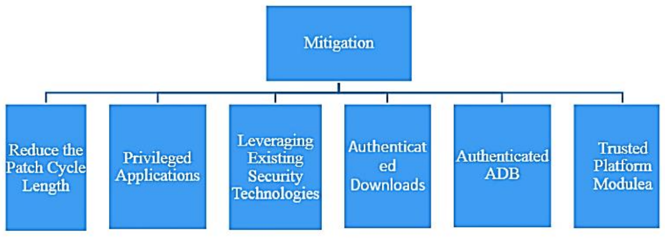


Figure 5. Type of Mitigation

Reduce the Patch Cycle Length

Attackers exploit some flaw in the operating system to gain root privileges. Reducing the patch cycle length would mitigate these threats with greater effectiveness. Zero-day exploits would still be possible, however the common lingering threat will be reduced. While Google has already demonstrated willingness to act quickly with out of band patch releases in reaction to certain attacks (e.g., [6]), reducing complete patch cycles is a more difficult problem. Indeed, manufacturers make changes to the Android source to create a competitive advantage. A fundamental separation between the core of Android and manufacturer modifications should be established.

Privileged Applications

To mitigate application attacks that take advantage of Android's permission model many solutions have been proposed. Propose lightweight application certification comparing the requested permissions of an application to a set of security rules. If the application does not pass any of the security rules, then possible malicious activity is brought to the attention of the user.

For example, Google could validate that certain software vendors create security software and grant applications created by these vendors additional API functionality. Applications signed by such a vendor could, for example, have read access to the file system in order to facilitate anti-virus scanning beyond limited scope typically granted to applications. Such a configuration would allow users to install security related applications without having to first root their device. Because privileged applications will have unrestricted access to the device, these applications should be certified by some governing entity before they can be downloaded. This certification process could also help mitigate some weaknesses of an unmoderated market. With access to trusted security tools, users would be able to monitor untrusted applications and provide appropriate feedback.

Leveraging Existing Security Technologies:

There are several existing operating system security enhancements that could be ported to Android. Instrumenting Android to monitor applications and understand how they interact with the user's sensitive information. A realized implementation of Taint Droid could give users real-time information about how an application uses the permissions it is granted. Generally, operating system level software modifications such as adding a firewall to Android involve porting existing technology to the Android kernel and creating an application to facilitation administration.

Authenticated Downloads:

Once an attacker has physical access to a device, adding malicious applications becomes simple and quick by posing as the legitimate user and downloading them from the Android Market. To ensure downloads are made only by the user, the market should require authentication before every transaction, similar to the model currently used by the iPhone.

Authenticated ADB:

Because of the power given through the ADB, it should not be accessible to unauthorized users. Android should require the device to be unlocked before ADB can be used. Any legitimate user should be able to unlock the device and once the connection is made, the session could be maintained by preventing the screen from locking while it is connected via USB. With ADB authentication, the attacker no longer has a backdoor to bypass the lock mechanism's authentication process, mitigating the ADB attack against obstructed devices.

Trusted Platform Module:

To secure a device in a managed model scenario a root of trust must be established. Using a Trusted Platform Module (TPM) provides a ground truth on which device security could be built, providing authentication of device state. Using a TPM would mitigate the recovery image attack, which relies on the ability to change the boot image. Assuming signed byte code and authentication of the boot image, updates running unauthorized code would become extremely difficult.

II. LITERATURE REVIEW

(1)Paper title: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation

In this Paper, They have described that robust and powerful software instrumentation tools are essential for program analysis tasks such as profiling, performance evaluation, and bug detection tool writer to analyze an application at the instruction level without the need for detailed knowledge of the underlying instruction set. The API is designed to be architecture independent whenever possible, making Pin tools source compatible across different architectures. Pin uses dynamic compilation to instrument executable while they are running. For efficiency, Pin uses several techniques, including in lining, register reallocation, liveness analysis, and instruction scheduling to optimize instrumentation. This fully automated approach delivers significantly better instrumentation performance than similar tools.

(2)Paper title: Behavioral Analysis of Android Applications Using Automated Instrumentation

They describe that Google's Android operating system has become one the most popular operating system for hand-held devices. Due to its ubiquitous use, open source nature and wide-spread popularity, it has become the target of recent mobile malware. In this paper, they present efforts on effective security inspection mechanisms for identification of malicious applications for Android mobile applications. To achieve that, they developed a comprehensive software inspection framework. Moreover, to identify potential software reliability flaws and to trigger malware, they develop a transparent instrumentation system for automating user interactions with an Android application that does not require source code. Additionally, for run-time behavior analysis of an application, they monitor the I/O system calls generated the by application under monitoring to the underlying Linux kernel.

(3)Paper title: All Your Droid Are Belong to Us: A Survey of Current Android Attacks

Mobile devices (smartphones, PDAs) have seen both their computational power and their data connectivity rise to a level nearly equivalent to that available on small desktop computers, while becoming ubiquitous. Mobile device middleware is thus experiencing an increased focus on attempts to mitigate potential security compromises. The Android security model also creates several new security sensitive concepts such as Android’s application permission system and the unmoderated Android market. In this paper we look to Android as a specific instance of mobile computing. We first discuss the Android security model and some potential weaknesses of the model. We then provide a taxonomy of attacks to the platform demonstrated By real attacks that in the end guarantee privileged access to the device. Where possible, we also propose mitigations for the identified vulnerabilities

(4)Paper title: Analysis and Research of System Security Based on Android

In this paper, it has analysis Android system's security mechanisms with widely used in mobile platforms. It has separately introduced its system architecture, security mechanism and safety problems. Through it has analysis Android security mechanisms and its components, it has set to the Android security, safety mechanism side, system security and data security. It has promoted system security to system permission. At the same time it analysis the Android security risks, it has deeply researched the attack based on Linux kernel. It has proposed security mechanisms based on SELinux policy theory to ensure system security on application program framework layer. Not only from the Linux kernel layer, it uses Android's security framework to ensure system security from the application layer intrusion, so it is essential to research and develop the method to protect the Android framework. This work will be the reference base to the Android further security analysis.

(5)Paper title: Patch droid: scalable third party security patches for android devices.

In this paper they have presented patch droid, a System to patch security vulnerabilities on legacy android Android devices, Patch droid uses dynamic instrumentation techniques to patch vulnerabilities in memory, and uses a path distribution service so that patches only have to be created once and can be deployed on every devices. Because patches are injected directly into the processes, Patch droid does not need to flash or modify system partitions or binaries, making it universally deployable even on tightly controlled devices.

(6)Paper title: Dissecting Android Malware: Characterization and Evolution

In this paper, we focus on the Android platform and aim to systematize or characterize existing Android malware. Particularly, with more than one year effort, we have managed to collect more than 1,200 malware samples that cover the majority of existing Android malware families, ranging from their debut in August 2010 to recent ones in October 2011. In addition, we systematically characterize them from various aspects, including their installation methods, activation mechanisms as well as the nature of carried malicious payloads. The characterization and a subsequent evolution-based study of representative families reveal that they are evolving rapidly to circumvent the detection from existing mobile anti-virus software. Based on the evaluation with four representative mobile security software, our experiments show that the best case detects 79.6% of them while the worst case detects *only* 20.2% in our dataset. These results clearly call for the need to better develop next-generation anti-mobile-malware solutions.

III. METHODS AND MATERIAL

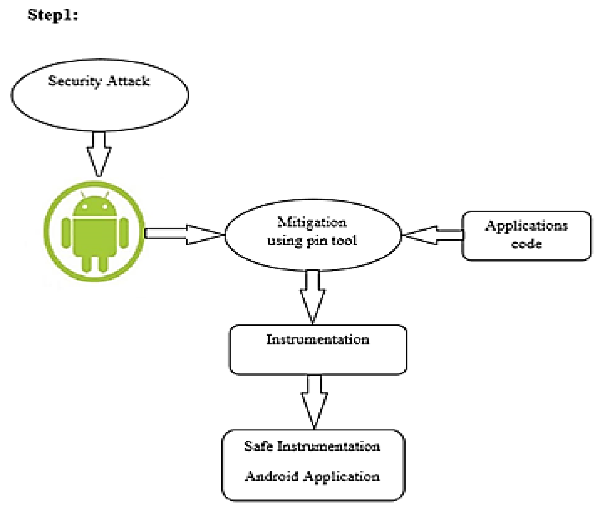

Step1:
Security Attack
Mitigation using pin tool
Applications code
Instrumentation
Safe Instrumentation Android Application

Figure 6. Mitigation of security attack in android application using pin tool

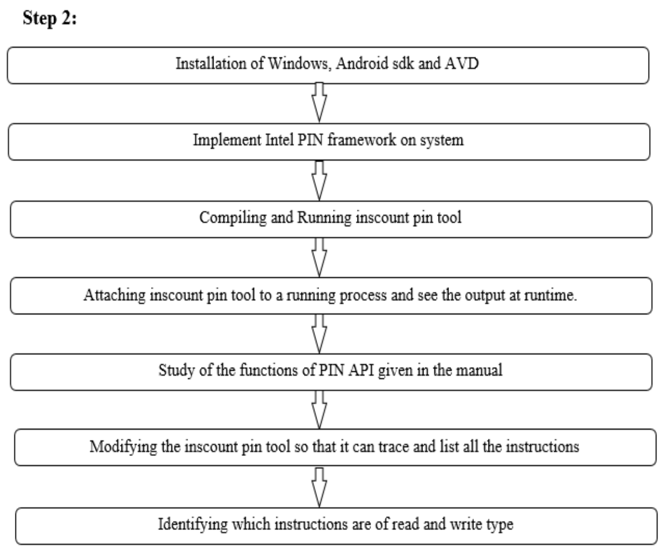

Step 2:
Installation of Windows, Android sdk and AVD
Implement Intel PIN framework on system
Compiling and Running inscount pin tool
Attaching inscount pin tool to a running process and see the output at runtime.
Study of the functions of PIN API given in the manual
Modifying the inscount pin tool so that it can trace and list all the instructions
Identifying which instructions are of read and write type

Figure 7. Instruction count in this android application

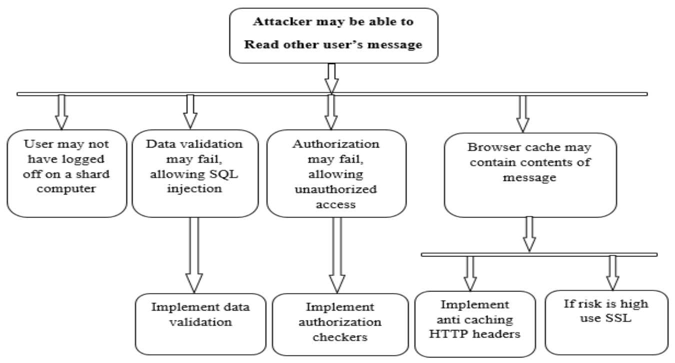

Attacker may be able to Read other user's message
User may not have logged off on a shard computer
Data validation may fail, allowing SQL injection
Authorization may fail, allowing unauthorized access
Browser cache may contain contents of message
Implement data validation
Implement authorization checkers
Implement anti caching HTTP headers
If risk is high use SSL

Figure 8. For Example: attack

IV. CONCLUSION

We have presented a method of mitigation of security attack in Android Applications using Pin tool which allows the user to instrument an Android Application. Instrumented code alters the behaviour of the original application thus the attacker can't find the right way to inject his own code into the running Application. Android applications obtained from untrusted sources can be instrumented to enforce some sort of policies to prevent application from doing data leaks of confidential information. Moreover instrumentation can also be used as a protecting weapon. So here we have used pin tool to mitigation of security attack in android application.

We have described about the Pin, an Intel framework that provides portable, transparent, easy-to-use, robust and efficient dynamic binary instrumentation. It supports various architectures like IA32, EM64T, Itanium R, and ARM.

After describing the PIN tools and the pin tool kit, we have described the pin tool used for counting the instructions known as inscount. The code as well as the screenshot when it was attached to a running application is also shown. After that we have modified and extended the functionality of this pin tool to change the arguments to get trace the instructions.

Pin tool works as a dynamic binary instrumentation engine. It always detects vulnerabilities or malicious activities during the execution process of application. It provides run time monitoring and profiling for vulnerability analysis.

V. FUTURE WORK

In future instrumentation the android application to perform various attacks dynamically can be carried out. We need to develop various drives classes that can perform various kinds of activities. Thus with the help of instrumentation we can develop a whole new era of android application with fully featured profilers, debuggers and tools for controlling the applications at runtime.

In future on an android application, the attack will be performed. Using pin tool, its mitigation will also be provided. Mainly privileged types of attack will be done and mitigation technique will be developed to protect application from any attack.

VI. REFERENCES

[1] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney , Steven Wallace, Vijay Janapa Reddi, Kim Hazelwood,"Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation" PLDI '05 Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation Pages 190-200. and www.pintool.org

[2] http://developer.android.com

[3] Android Kernel Issues.http://www.kandroid.org.

[4] Intel. IA-32 Intel Architecture Software Developer’s Manual Vols 1-3, 2003.

[5] Just because it’s signed doesn’t mean it isn’t spying on you. http://www.f-secure.com/weblog/ archives/00001190.html, May 2007.

[6] J. Oberheide. Remote kill and install on google android. http://jon.oberheide.org/blog/2010/06/25/remote-kill-and-installon- google-android/