Android OS Information Security Level Assessment

Pavlo Chernenko, Maria Orlova
romankev@scs.kpi.ua
Faculty of Applied Mathematics, Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, Ukraine

Summary
The increase of the efficiency of the collection, processing and exchange of information through modern data transfer technologies, remote monitoring and automated management, is one of the essential directions of the information systems development. Various mobile devices that deal with data transfer and processing issue make the base of this concept. Modern mobile services, consisting of those which exchange and process secret, banking and critical data, show up in light of the consistent increment of the quantity of information security crimes compared to the usage of mobile devices. The extensive use of these devices to access protected data in information systems has given a particular importance to ensuring information security.
Mobile operating systems form the foundation of the Ad hoc mobile networks structure and thus the evaluation of the present status of data security instruments for mobile operating systems is the subject of this paper. The article examines questions of the mobile ecosystem's development and techniques intended to solve these problems, including their principal harmful influences. The article reveals strategies for protection from static and dynamic analysis, and modern security components taking Android OS as an example.

Key words:
Android, information security, malicious impact, mobile OS, protection instruments.

1. Introduction

According to the reports of the rating agency Statcounter (Fig. 1), for the second quarter of 2020, Android is the most common operating system (OS) in the world with a result of almost 38% of the total number of installed OS, which imposes the obligation of this OS to have the highest quality in information security[1]. However, although Google and mobile device manufacturers are constantly improving the security system, the open source code and extensive fragmentation of the platform make this system one of the most vulnerable to malicious exposure.

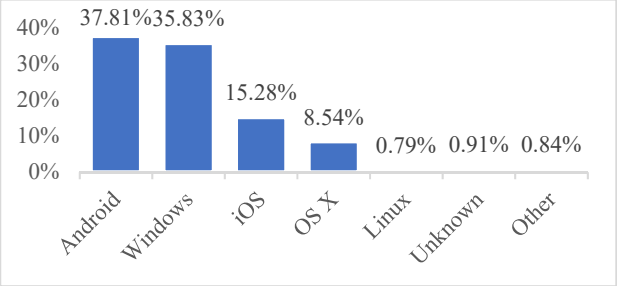

Fig. 1 Operating System Market Share Worldwide

2. Current Android OS security status

The major reason for the fragmentation of the Android ecosystem is the technology used to create mobile devices based on SoC-system (System on a chip). SoC-system consists in the integration of the central processor, graphics accelerator, radio module and various sensor equipment onto one microchip. This concept allows to reduce the physical size of the device, lower the power consumption and increase the productivity with the help of a better integration of components, but the interaction of the entire system requires a development of special drivers. Manufacturers of various SoC-systems develop drivers which are proprietary and unique to each model. As a result, mobile device manufacturers integrate the received drivers for the SoC-system into their own assembly, which makes software update procedures being dependent. Fig. 2 shows the production algorithm for mobile devices based on Android. High fragmentation of the platform doesn't allow to provide relevant updates to mobile devices quickly because of the large number of manufacturers of chipsets and mobile devices (ODM is the manufacturer who create products according to the original project; OEM is the manufacturer who sell parts and equipment to other manufacturers).

Manuscript received August 5, 2020
Manuscript revised August 20, 2020

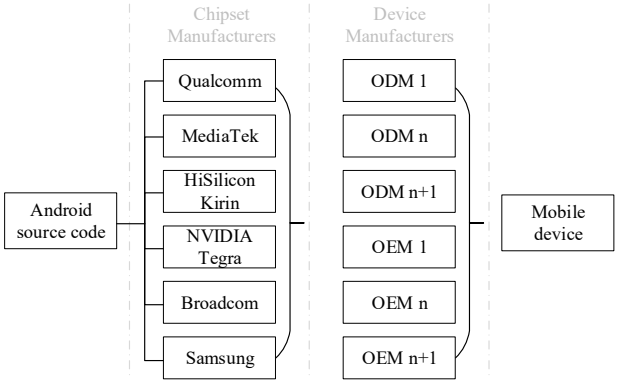

Fig. 2 Algorithm of the production of Android devices

1. Google has taken several important steps to solve the problem of high device fragmentation:
2. In 2014, Google introduces Android One software and hardware standard – an OEM support program aimed to motivate manufacturing devices without OS modifications. The goal of the project is the ability for Google to manage the design, development and support of these devices, while original manufacturers carry out the production.
3. As a result, Google handling security issues and OS updates has increased the level of the security system. However, for now, the largest device manufacturers do not support the concept of a "clean" OS leading to low efficiency in addressing ecosystem fragmentation.
4. In 2017, Google changes the Android architecture by adding the "Project Treble" abstraction layer, which allows separating the layer of hardware code implementation from the operating system code [2]. This step allows to update the operating system code separately from the drivers, but getting recent software versions still depends on the manufacturer. That does not improve the situation.
5. Since 2018, manufacturers have had to release OS updates for at least two years for all Android devices,
6. the number of which exceeds 100,000. If a partner does not fulfill this condition, Google may refuse his license [3].
7. Google's policy aims to provide updates for devices not older than 2-3 years, which leads to serious security issues in the industry. Devices related to this segment should be potentially vulnerable accordingly. Table 1 shows a brief description of the Android versions on the market and their support status.

Table 1: Android versions characteristic.

Version Name	Version Number	Release Year	Support status
Cupcake	1.5	2009	No longer supported by Google
Donut	1.6		
Eclair	2.0 – 2.1	2010	
Froyo	2.2 – 2.2.3		
Gingerbread	2.3 – 2.3.7		
Honeycomb	3.0 – 3.2.6	2011	
Ice Cream Sandwich	4.0 – 4.0.4		
Jelly Bean	4.1 – 4.3.1	2012	
KitKat	4.4 – 4.4.4	2013	
Lollipop	5.0 – 5.1.1	2014	
Marshmallow	6.0 – 6.0.1	2015	
Nougat	7.0 – 7.1.2	2016	
Oreo	8.0 – 8.1	2017	Supported by Google
Pie	9.0	2018	
10 Q	10	2019	Current version
11 R	11	2020	Beta

Based on the analysis of supported versions and statistics of distribution of versions of Android, presented in Table 2, it is possible to calculate the probabilities of a device being in a vulnerable state over the past 7 years under the conditions under which manufacturers perform device updates for two years [4], [5].

Table 2: Android operating system share worldwide by OS version and a probability of a device being in a vulnerable state from 2013 to 2020 (%)

	Froyo	Gingerbread	Honeycomb	Ice Cream Sandwich	Jelly Bean	KitKat	Lollipop	Marshmallow	Nougat	Oreo	Pie	10 Q	**Probability**
2013	2.2	28.5	0.1	20.6	48.6								**30.7**
1H 2014	0.8	14.9		12.3	58.4	13.6							**28.0**

2H 2014	0.7	11.4		9.6	53.8	24.5							**21.7**
1H 2015	0.4	7.4		6.4	44.5	39.7	1.6						**58.7**
2H 2015	0.2	3.8		3.4	30.2	38.9	23.5						**37.6**
1H 2016	0.1	2.2		2.0	20.1	32.5	35.6	7.5					**56.9**
2H 2016	0.1	1.5		1.4	15.6	27.7	35.0	18.7					**46.3**
1H 2017		1.0		0.8	9.1	18.8	32.0	31.2	7.1				**61.7**
2H 2017		0.6		0.6	6.9	15.1	28.8	32.2	15.8				**52.0**
1H 2018		0.3		0.4	4.3	10.3	22.4	25.5	31.1	5.7			**63.2**
2H 2018		0.3		0.3	3.2	7.8	18.3	21.6	29.3	19.2			**51.5**
1H 2019		0.3		0.3	3.2	6.9	14.5	16.9	19.2	28.3	10.4		**61.3**
2H 2019		0.2		0.2	3.0	2.4	6.2	11.3	12.6	22.1	42.0		**35.9**
1H 2020		0.1		0.1	1.5	1.8	4.9	8.7	10.6	18.9	34.7	18.7	**46.6**

Fig. 4 shows a graph displaying a negative trend in the distribution of relevant Android versions.

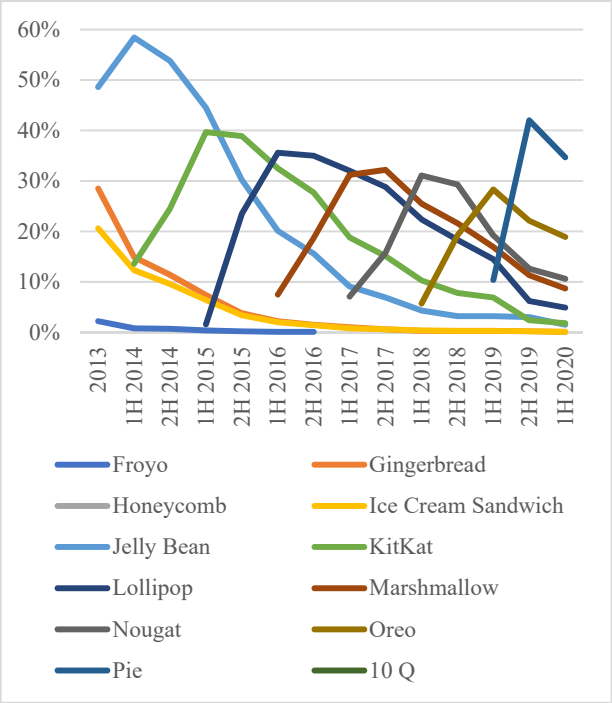

Fig. 4 Negative trend in the distribution of relevant Android versions.

Using the data above, it is possible to forecast a trend of the device being in a vulnerable state for the next few years using a pairwise simple linear regression function (Fig. 5).

Thus, if the Android ecosystem's development trend continues, by 2022 the probability of compromising devices on this platform will tend 65%, which is a serious motivator to create alternative methods for providing security updates for mobile devices.

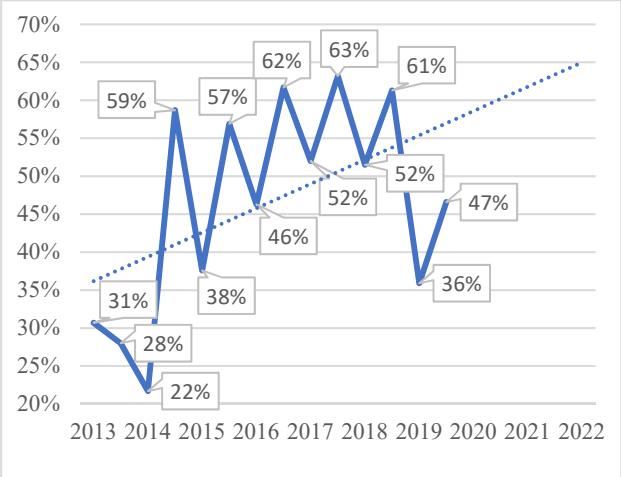

Fig. 5 Chart displaying an increase of vulnerable devices

3. Infection vectors

The study "Android: protecting the kernel", dedicated to the analysis of Android vulnerabilities from 2014 to 2016, showed a notable increase in the detected vulnerabilities in the Linux kernel [6]. Fig. 6 shows statistics characterizing the dynamics of growth of kernel vulnerabilities. In 2014, 4% of vulnerabilities were found in the Android kernel, in 2016 this indicator increased to 36% and for 2018 remains at 33%.

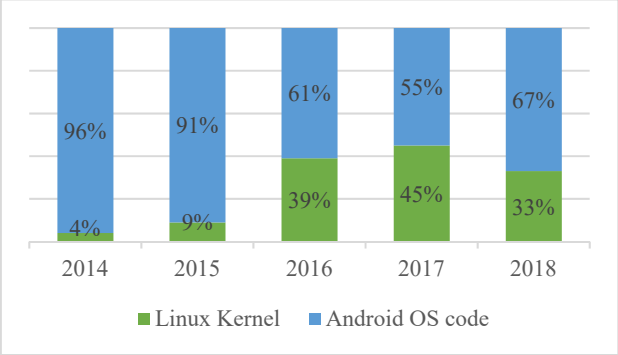

Fig. 6 Android OS vulnerabilities statistics.

According to cvedetails.com, there are 2563 vulnerabilities in Android, moreover 613 and 414 vulnerabilities are discovered in 2018 and 2019 accordingly [7].

The main reason for the high increase of vulnerabilities in the kernel is the introduction in 2013 of Android 4.4 SELinux (Security Enhanced Linux) control system that allows to create policies that define types of interactions allowed and forbidden for each process within the general security context, and also to introduce administrator restrictions

Kernel issues include vulnerabilities not only in the Linux kernel but also in the code of drivers supplied by developers of mobile chipsets. At the international conference on information security (RSA Conference USA 2018), a report on the research of existing vulnerabilities in Android showed that 85% of vulnerabilities detected in the kernel are classified as private drivers[8]. Fig. 7 shows a correlation diagram of various types of kernel errors. As a result, these types of vulnerabilities facilitate attacks on systems regardless of the version of Android.

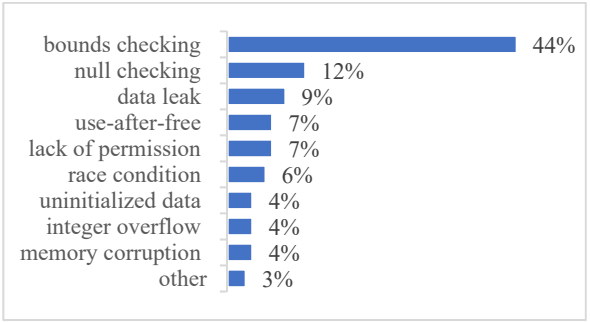

Fig. 7 Kernel error types

Each type of malware designed for desktop computers has an analogue for Android. For example, organizing a botnet, stealing personal data, gaining control of a device or disabling it. Fig. 8 shows a diagram characterizing the ratio of existing types of malware for Android for the 3rd quarter of 2018 [9]. With the development of permission and security systems, the percentage of viruses like Trojan-SMS and Spyware has sharply decreased.

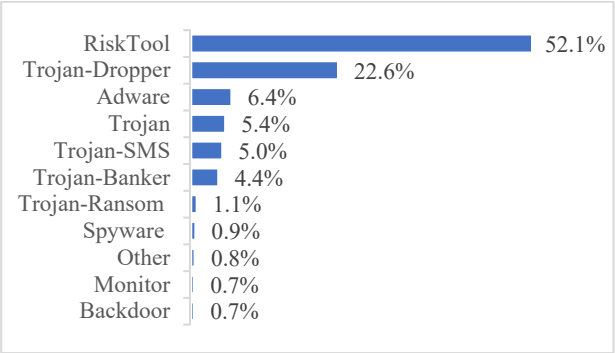

Fig. 8 Malware types

4. Modern malware distribution technologies

Malware distribution technologies are classified as follows [10]:

1. Exploitation of vulnerabilities in the Linux kernel and its modules. Android is a Linux distribution with its own implementation of the inter-process communication function, management of sleep mode, kernel protection (shared memory mechanism, etc.) and memory cleanup. Thus, vulnerabilities found in common kernel components can be applied on mobile devices.
2. Exploitation of vulnerabilities in hardware modules. Mobile devices have many hardware modules designed to interact with other devices [11]. Such vulnerabilities can be exploited in the coverage area of radio modules or with direct access to the device.
3. Exploitation of vulnerabilities in operating system components, programs and drivers. Allows the attacker to bypass Android and SELinux defenses.
4. Exploitation of vulnerabilities in components of mobile device manufacturers. Device manufacturers are modifying Android by placing various applications in the system directory, i.e., processes are launched in privileged mode. These applications may contain vulnerabilities leading to data leak, account hijacking,

and the installation of malware. Also, manufacturers themselves can change the system with undeclared functionality.

5. Exploitation of vulnerabilities in libraries. Several libraries are included in the Android architecture, such as OpenGL, Audio Manager, Media Framework, libc [2]. Exploiting vulnerabilities in these components is one of the main attack vectors at the moment. The most common vulnerabilities of the Media Framework component allow attacks such as remote code execution (RCE) on the affected device (for example, when working with e-mail, browsing the Internet or processing MMS files) [12].
6. Exploitation of vulnerabilities in machine codes. Android contains a toolkit that allows to execute C and C ++ code (Android NDK). This feature generates errors specific to low-level programming languages (memory leaks, buffer overflows, etc.).
7. Exploitation of vulnerabilities in user applications. Applications installed by the user may manage a personal data, but storage and access to this information is not always ensured properly (using HTTP traffic, locating application data files in shared folders, etc.).
8. The use of social engineering methods used for the transfer and subsequent installation of malware from various sources (including the Play Market).

5. Resisting tools of static analysis of applications.

Fig. 9 shows masking methods used to resist the signature analysis. Various ways of code obfuscation bring the source code or executable code of the program to a form that preserves its functionality, but complicates the analysis, understanding of work algorithms and modification during decompilation.

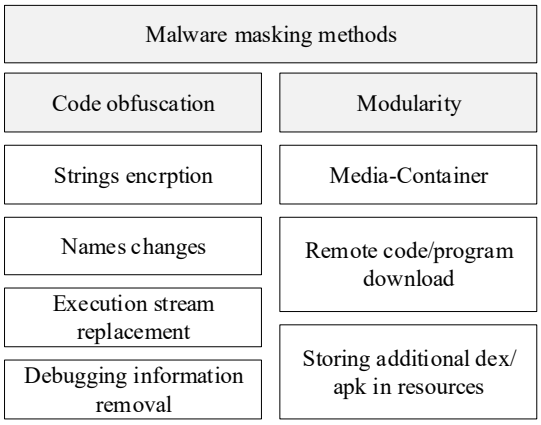

Fig. 9 Malware masking methods

The most common tools for obfuscating application code in Android are DexGuard, Alatori, DashO, and DexProtector. Also, for the initial protection against code analysis, the built-in tools in Android Studio (Proguard or R8) are used, but these tools only rename objects.
Fig. 10 shows the primary stages of converting the source code of programs to the executable code of the Dalvik / ART virtual machine using these tools.

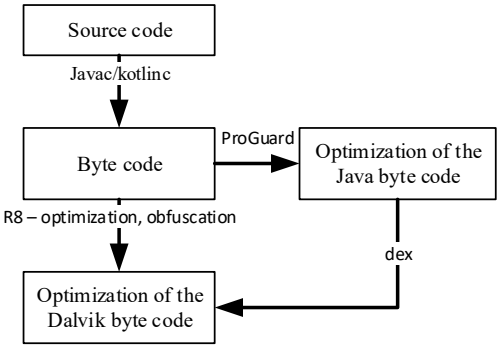

Fig. 10 Stages of an app creation using ProGuard and R8

Modular architecture for building applications allows to apply reflection methods making loading byte-code into a virtual machine during application execution possible.
In Android, to implement the concept of dynamic loading and code execution, the following functionality is used:

1. dalvik.system.DexClassLoader - a class that allows to download ".dex", ".jar" or ".apk" files.
2. Java Reflection API allows, that allows to interact with the downloaded code during program execution.

6. Resisting dynamic application analysis tools

Fig. 11 displays methods used to resist the dynamic analysis of the application. Their goal is to detect attempts to control the behavior during program execution in a sandbox and to get a pattern of the behavior [13]. The static heuristic method checks unique identifiers of a device, such as a serial number or belonging of an external IP address to application verification services. When using a hypervisor, these indicators are equal to the default values, which is an indicator for detecting surveillance systems. A method based on the occurrence of an error in

the readings of sensor equipment (accelerometer, gyroscope, light sensor, GPS, etc.) refers to dynamic heuristics. Modern emulators support simulation of sensors with pseudo-random value update events occurring at random time intervals, which complicates dynamic heuristics.

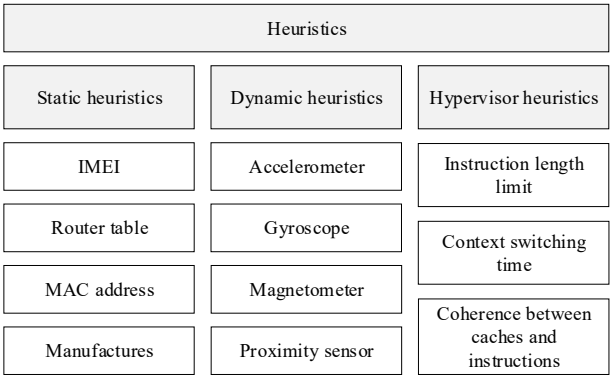

Fig. 11 Methods to resist dynamic analysis tools

Hypervisor heuristics aim is to detect differences in defective emulation of real equipment. The above methods search for differences in the functionality of real and emulated devices. These differences are differences in the values of the MSR registers (model-specific registers), time intervals for switching the context, within the instruction lengths, and specific processor errors, relative performance, etc.

The above methods are often used to bypass the Google Bouncer antivirus scanning system, which operates in the official Google Play repository, as a result allowing malware to get the status of a "trusted application".

7. Modern security approaches and tools in the Android OS

Platform-level security is the most serious change in Android in recent versions. To eliminate discovered vulnerabilities in the Android code efficiently, Google launched monthly security update program. Starting with Android 8.0 they added a component to the platform level of "Play Protect" – a cloud-based security system that provides real-time scanning of applications in the official repository and on users' devices.

Modern tools to ensure information security at the device level and their purposes are the following [14], [15].

1. Encryption. Data protection from unauthorized access using cryptographic information protection tools.
2. Hardware protection tools. Ensuring reliable storage of encryption keys and a secure authentication procedure.
3. Core protection.
   PAN (Privileged Access Never) – restriction to access processes' memory directly and forcing the usage of memory copy functions (Android 8.0);
   CFI (Control Flow Integrity) – creation of a graph of function calls, embedding a verification code with before each function call. This mechanism aims to resist the modification of function pointers and return addresses (Android 9.0);
   IOS (Integer Overflow Sanitization) – integer overflow data protection. Checking the performance of arithmetic operations (Android 7.0);
   KASLR (Kernel Address Space Layout Randomization) – assignment of a random address to the code location area, anonymous memory area and to a kernel data in memory on each boot (Android 8.0);
   PIROM (Post-Init Read-Only Memory) – creation of readable and writable memory areas that could be used only during initialization and are put in read-only mode after initialization (Android 8.0).
4. Isolation of processes Execution of each application in a separate address space with unique values user/group ID. Exchange of data between processes is carried out through OS services.
5. SELinux. Mandatory access control that provides the minimum necessary set of privileges for each process.
6. OS boot verification. Ensuring the operating system's good condition at boot time. Mechanisms to prevent the launch of self-signed code, check the integrity and digital signature of the kernel.

8. Conclusion

Article shows that Android platform security issues are present at all levels of the platform. In modern versions of Android, protection mechanisms have seriously increased the level of device security, but these solutions are not ensuring the security of the ecosystem entirely and cannot guarantee the absolute reliability and security of data in mobile devices.

Despite a significant improvement in the quality of protection mechanisms and big variety of modern security approaches and tools, working separately they cannot secure a device completely against the constantly growing number of vulnerabilities.

The high level of fragmentation of the platform and the predicted increase of the number of vulnerable devices require the use of alternative approaches to ensure the security of mobile operating systems and malware analysis systems. This area is an urgent task with the need for a detailed study of the issues of identifying malicious code.

References

[1] "Operating System Market Share Worldwide". https://gs.statcounter.com/os-market-share (accessed Aug. 13, 2020).

[2] "Android Architecture". https://source.android.com/devices/architecture (accessed Aug. 13, 2020).

[3] "Android Enterprise Recommended". https://www.android.com/enterprise/recommended/ (accessed Aug. 13, 2020).

[4] "Mobile & Tablet Android Version Market Share Worldwide". https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide (accessed Aug. 13, 2020).

[5] J. Stoep Vander, "Android: protecting the kernel", Aug. 26, 2016. https://events.static.linuxfound.org/sites/events/files/slides/Android-%20protecting%20the%20kernel.pdf (accessed Aug. 13, 2020).

[6] "Google Android - CVE Details". https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224 (accessed Aug. 13, 2020).

[7] G. Vigna, "How Automated Vulnerability Analysis Discovered Hundreds of Android 0-days", Apr. 18, 2019. https://www.rsaconference.com/industry-topics/presentation/how-automated-vulnerability-analysis-discovered-hundreds-of-android-0-days (accessed Aug. 13,).

[8] V. Chebyshev, "Mobile malware evolution 2019", Feb. 20, 2020. https://securelist.com/mobile-malware-evolution-2019/96280/ (accessed Aug. 13, 2020).

[9] R. Ferreira, A. Santos, and R. Choren, "Vulnerabilities Classification for Safe Development on Android", Journal of Information Systems Engineering & Management, vol. 1, pp. 187-190, Jun. 2016, doi: 10.20897/lectito.201634.

[10] A. Stern, "Bluetooth Security Vulnerabilities", Apr. 15,2013. https://www.kaspersky.com/blog/bluetooth-security/1637/ (accessed Aug. 13, 2020).

[11] "CVE-2017-0466 - NVD". https://nvd.nist.gov/vuln/detail/CVE-2017-0466 (accessed Aug. 13, 2020).

[12] T. Petsas, Voyatzis Giannis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware", EuroSec '14. 2014.

[13] "Android Security & Privacy 2018 Year In Review". https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf (accessed Aug. 13, 2020).

[14] "Android security Blog". https://security.googleblog.com/search/label/android%20security (accessed Aug. 13, 2020).

[15] "Android: Vulnerability Statistics". https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224 (accessed Aug. 13, 2020).