SoftwareX 23 (2023) 101448

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

SEBASTiAn: A static and extensible black-box application security testing tool for iOS and Android applications

Francesco Pagano [a,*], Andrea Romdhana [c], Davide Caputo [c], Luca Verderame [a,c], Alessio Merlo [b]

[a] DIBRIS - Università degli Studi di Genova, Genova, Italy
[b] CASD - Centre for Higher Defence Studies, Rome, Italy
[c] Talos srls, Genova, Italy

ARTICLE INFO

Article history:
Received 13 March 2023
Received in revised form 9 June 2023
Accepted 16 June 2023

Keywords:
Static analysis
SAST
Android security
iOS security

ABSTRACT

Despite decades of research, the automatic detection of vulnerabilities in mobile apps remains an open challenge. Among the possible solutions, SAST tools uncover source or compiled code security flaws without needing the app to be executed and tested in a controlled environment. However, SAST tools share several limitations, such as the detection of narrowed vulnerability classes, lack of updates, and limited resiliency to obfuscation techniques. This paper presents SEBASTiAn, a black-box automatic static analysis tool for security vetting Android and iOS apps. It relies on a modular approach to cope with new vulnerabilities.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00167
Permanent link to Reproducible Capsule	
Legal Code License	GNU Affero General Public License
Code versioning system used	git
Software code languages, tools, and services used	Written in Python tested and working on Ubuntu 20.04, Docker image available.
Compilation requirements, operating environments & dependencies	Python packages: androguard, biplist, lief, marshmallow, Pebble, pytest-cov, Yapsy
If available Link to developer documentation/manual	https://github.com/talos-security/SEBASTiAn/blob/master/README.md
Support email for questions	francesco.pagano@dibris.unige.it

1. Motivation and significance

The automatic detection of vulnerabilities in mobile apps is still an open problem despite decades of research efforts in the field. For example, a recent study [1] reported that mobile apps released in Q1 2021 across 18 of the most popular categories have – on average – 39 vulnerabilities per app that can affect the security and privacy of the users. To cope with such issues, the industry and the research community released several security assessment methodologies and tools to detect vulnerabilities by exploiting Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) techniques. Notable examples include MobSF [2], mobsfscan [3], DroidPatrol [4], and Amandroid [5].

SAST tools, in particular, enable the detection of security hazards in the source or compiled code of mobile apps without the burden of executing and testing it in a controlled environment. To this aim, they are easy to integrate into a CI/CD pipeline since the scanning process can be launched as soon as a team member commits code to a source code repository. Unfortunately, SAST tools share several limitations that affect their efficacy. On Android, they are tailored to detect narrowed vulnerability classes. For instance, QARK [6] detects only web and crypto vulnerability categories, while FCDroid [7] is focused on a vulnerability affecting Android WebViews. Trueseeing [8] detects Android Manifest

* Corresponding author.
E-mail addresses: francesco.pagano@dibris.unige.it (Francesco Pagano), andrea.romdhana@talos-sec.com (Andrea Romdhana), davide.caputo@talos-sec.com (Davide Caputo), luca.verderame@dibris.unige.it (Luca Verderame), alessio.merlo@ieee.org (Alessio Merlo).

https://doi.org/10.1016/j.softx.2023.101448

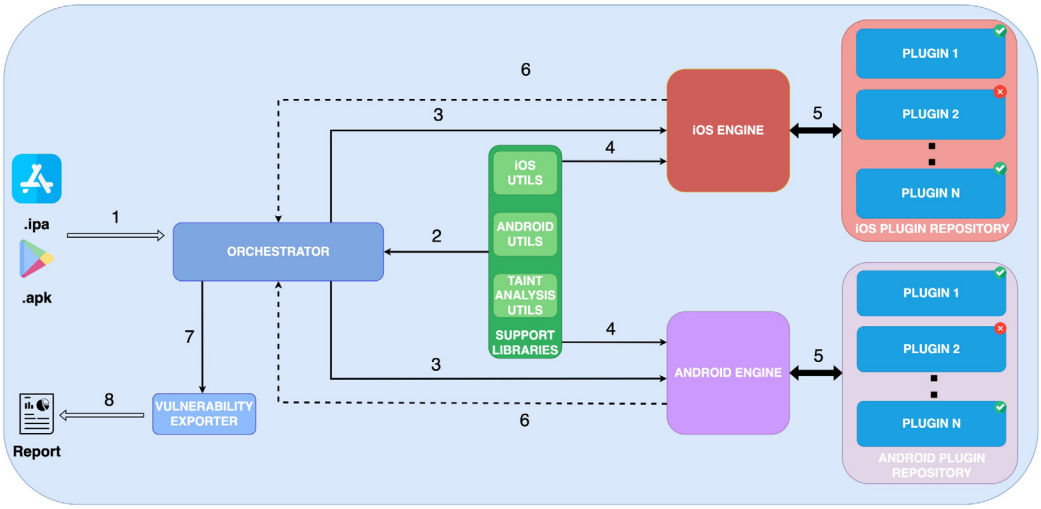

Fig. 1. SEBASTiAn Architecture.

file [9] and insecure connections vulnerabilities. Super [10] relies on regex-based pattern-matching techniques for the vulnerability's detection, which are not accurate in the case of obfuscated code. MobSF [2] analyzes Android and iOS apps and relies on regex-based pattern-matching techniques.

Moreover, SAST tools do not consider maintainability and frequent updates as must-have features, especially in open-source initiatives. Most research tools work under specific constraints (OS and app version) or do not receive updates regularly. For instance, the last version of Super [10] is four years old (2018).

Finally, the advent of security-through-obscurity techniques such as obfuscation [11] increased the complexity of SAST evaluation as they modify the code to counteract automatic code analysis techniques.

To this aim, we advocate that there is a need for novel SAST tools that:

- automatically evaluates a wide range of security vulnerabilities;
- support multiple mobile operating systems (i.e., Android and iOS);
- relies on a modular and extensible approach to cope with new vulnerabilities and analysis techniques;
- support fine-grained updates (e.g., the check of a single vulnerability) without the burden of updating the entire tool;
- could be resilient to obfuscation techniques;

To meet such requirements, we developed SEBASTiAn, a black-box tool for the automated SAST of Android and iOS apps that relies on a modular architecture that supports the definition of new vulnerabilities as plugins and is resilient to obfuscation techniques. We released the first implementation of SEBASTiAn as an open-source tool on GitHub [12], which includes 58 Android security plugins and 19 iOS plugins. We assessed the stability of SEBASTiAn on Android by conducting a test campaign with the Ghera benchmark dataset [13]. To evaluate its performance on obfuscated Android apps, we also tested it with the obfuscated Ghera benchmark dataset. For iOS apps, we verified SEBASTiAn's performance using three benchmark apps [14–16]. Similarly, we evaluated its performance on obfuscated iOS apps by testing it with the obfuscated versions of the same benchmark apps.

2. Software description

2.1. Software architecture

This section introduces SEBASTiAn, a fully automated tool capable of performing SAST on Android and iOS apps by exploiting the architecture depicted in Fig. 1. Thanks to a modular approach, SEBASTiAn supports the definition of per-vulnerability security controls, packed in the form of **Plugins** and stored in dedicated repositories called **Plugin Repositories**. The core of the SAST tool is composed of two engines, namely **Android Engine** and **iOS Engine**, that are responsible for running the set of plugins tailored for the application under test (AUT, hereafter).

The **Orchestrator** module is responsible for receiving the mobile app and triggering the appropriate Engine for analysis. Also, it collects all the results of the plugins and sends them to the **Vulnerability Exporter** that ensemble the results and produces the security report. Finally, SEBASTiAn contains a set of **Support Libraries** that can assist the tool and each plugin in performing general-purpose activities. The libraries consist of additional tools and utility functions to unpack/unzip apps, deobfuscate and decompile the app, simulate `.dex` instructions, and execute taint analysis operations.

SEBASTiAn supports the analysis of Android and iOS application packages. First, the Orchestrator module receives the AUT (step 1 in Fig. 1). It executes a preliminary analysis by (i) detecting the type of app and ii) unpacking and decompiling the package. To do so, it exploits the set of appropriate auxiliary functions from the Support Libraries (step 2).

Then, the Orchestrator sends the AUT to the proper Engine module (step 3).

Each Engine runs the appropriate plugins loaded from the plugin repository (step 5). Both Android and iOS engines support the parallel execution of multiple plugins concurrently, thereby decreasing the overall analysis time.

A Plugin implements a set of security checks to detect a specific vulnerability. It can leverage the API provided by Support Libraries (step 4) to perform common operations, e.g., control flow graph extraction or resource unpacking.

The result of each plugin is reported to the running execution engine using asynchronous callbacks. Once all plugins end their

execution, the Engine sends the raw results to the Orchestrator (step 6). Finally, SEBASTiAn triggers the Vulnerability Exporter module (step 7) to produce the security report (step 8). The report contains the list of vulnerabilities discovered on the AUT along with a detailed description, the classification in the OWASP Mobile TOP 10 [17], and the corresponding CVSS risk score [18].

2.2. Implementation details

SEBASTiAn is entirely implemented in Python 3.7 and can work as a standalone tool or by using a pre-built Docker image. SEBASTiAn relies on the Yapsy library [19] to implement the core mechanisms needed to build the plugin system. In addition, the tool exploits general-purpose libraries for the Support Libraries module, such as Androguard [20] and Lief [21], to compute the app's control-flow graph and to perform integrity checks on the input app. For the complete list of libraries and dependencies, we refer the interested reader to the SEBASTiAn GitHub page.

2.3. Implemented plugins

This section describes the Plugins currently implemented in SEBASTiAn. Each Plugin implements the detection workflow for a specific vulnerability. We implemented 58 Android Plugins and 19 iOS Plugins. The complete Plugin description is available on [22] and on Appendix A section.

2.3.1. Crypto

This Plugin category provides Plugins to discover vulnerabilities in the cryptographic primitives used by the app. Concerning Android, this Plugin category provides 6 Plugins that verify the app uses a constant salt [23] during the encryption processes (CR4). They check if AES in ECB mode is used (CR5) [23] and if the initialization vector (IV) is not randomly generated (CR1). SEBASTiAn also provides Plugins that check if the app stores encryption keys in its code (CR2) or if it stores them inside the device Keystore [24] as unencrypted (CR6). They also check if the app calls the `PBEKeySpec` [25] function, passing it an iteration count smaller than 1000 (CR7). Concerning iOS, this plugin category provides 4 Plugins that check if the `.ipa` file was encrypted (CR8) and is correctly signed (CR9). It also presents Plugins that detect whether the app uses insecure implementations of the random API (CR10). They also check if the app disabled the Perfect Forward Secrecy (PFS) [26] for external hosts (CR11).

2.3.2. ICC

This Plugin category provides Plugins that detect whether the app is vulnerable to Inter Component Communication (ICC) vulnerabilities. In detail, on Android, it provides 6 Plugins that check if the app presents exported components that can be called from other apps (ICC4) if it presents misconfigured intent filters [27] (ICC3), and if the app uses an Implicit Intent [28] to start a service (ICC2). Moreover, it provides Plugins that detect IDOS (Unhandled Exceptions or Denial Of Service), XAS (Cross-application scripting), and FI (Fragment Injection) [29] vulnerabilities (ICC1). Concerning Cordova apps on Android, this Plugin category provides Plugins that check whether the app does not configure the `Intent Whitelist` option [30] or configure it without specifying the only accepted hosts to pass to the Intent (HYB2).

2.3.3. Networking

This Plugin category provides Plugins that check if the app presents networking misconfigurations. In detail, concerning Android, it provides 13 Plugins that verify if the app allows HTTP connections (NET7) and (NET2), and it does not correctly verify server certificates (NET6). They also check if the app filters undesired connections with unknown hosts (NET1) and (NET3). They also detect whether the app opens unsecured sockets, e.g., vulnerable to MitM attacks [31] (NET4) or without a proper encryption level (NET5). This Plugin category provides Plugins to analyze Cordova apps on Android. In detail, it provides Plugins to check if the app properly configures the `Network Whitelist` option [32] to filter the allowed network requests, if it will enable HTTP connections, or if it does not implement a Content Security Policy (CSP) [33] at all (HYB1), and if it configures the `Navigation Whitelist` option [34] to filter the hosts reachable from internal WebView (HYB3). Concerning iOS, this vulnerability category provides 4 Plugins that check whether the app can accept insecure connections (NET8) or uses an old TLS version (NET9). They also check if the app allows HTTP requests (NET10).

2.3.4. Permission

This Plugin category provides Plugins that detect app permissions misconfigurations. In detail, concerning Android, it provides 10 Plugins that check if the app owns permission to install apps or mount filesystems from external storage (PERM4) without needing it. Moreover, it provides Plugins that check if the app defines empty permission groups [35] (PERM5), and it defines dangerous custom permissions [36] other apps can access that (HYB4). It protects at least one class with a custom normal permission, allowing an external app to register and receive messages from itself PERM3. They also check if the app tries to access the internet without the proper permission in the Manifest (PERM1) or if it declares the `Access Mock Location` permission [37] in the Manifest (PERM2). Moreover, they detect whether the app calls the `checkCallingOrSelfPermission` method, which grants permissions to all components that request permission if previously given to another component. This Plugin category provides a Plugin that performs the same check but on the `enforceCallingOrSelfPermi-ssion` method. They also check if the app calls the `checkPermission` method passing it the PID returned from the `Binder.getCallingPID()` because the permission requested could be granted to different unauthorized calling components [38]. This Plugin category provides a Plugin that performs the same check but on the `enforceCallingOrSelfPermission` method.

2.3.5. System

This Plugin category provides Plugins that check if the app misuses OS functionalities to modify system configurations or is misconfigured. In detail, concerning Android, it provides 14 Plugins that detect whether the app declares the `sharedUserId` option inside its Manifest file (SYS8), is compiled in Debug mode (SYS9)), loads code from a .jar file without checking it [39] (SYS10), allows file access in `WORLD_READABLE` or `WORLD_WRITABLE` mode [40] (SYS11), declares the `allowBackup` option inside the Android Manifest file to allow its backup from an external device (SYS5). They also check if the app reads files from the external storage (ST1), uses the device ID [41] or the IMEI [42] to identify the device in which it is installed (SYS4), has a low obfuscation level [11] or it is not obfuscated (SYS7). They check if the app uses APIs that send SMS messages (SYS3), requests root permissions to execute privileged actions (SYS1), or if it executes commands that modify the system configurations (SYS2). Concerning iOS, this Plugin category provides 8 Plugins that check the `Mach-O` binary [43] inside the `.ipa` file. In detail, they check if the Mach-O binary is compiled with the Automatic Reference Counting (ARC) flag [44] (SYS13), presents a restricted segment to avoid arbitrary code injection [45] (SYS12), is compiled with

the NX flag disabled [46] (SYS13), presents symbols (SYS15). They also check if the Mach-O binary misuses the `malloc` system call (SYS17), does not present the stack canary to avoid buffer overflows (SYS18), is not compiled as PIE [47] (SYS19). They also check if the Rpath [48] is properly set SYS16.

2.3.6. Web

This Plugin category provides Plugins that check if the app presents misconfigurations in the WebView [49] it uses. Regarding Android, it provides 10 Plugins that check whatever the app uploads arbitrary code inside the WebView, such as Javascript code, from an external file (WEB2), enables Javascript code loading (WEB5), uses the `addJavascriptInterface` method to load Java code inside the WebView WEB6. They also check if the WebView ignores SSL errors, establishing a connection with an unwanted host (WEB3) or making queries vulnerable to SQL injection to local databases (WEB1). This Plugin category also provides Plugins that check if the app uses a custom `WebView-Client` [50] that overrides the `shouldInterceptRequest` method, improper filtering file loading (WEB4) or overrides the `shouldOver-rideUrlLoading` method allowing every GET requests to be executed (WEB7). They also check whether the app calls the WebView's `loadDataWit-hBaseURL` method to load malicious Javascript code that shares the same privileges as the app and if it encodes URL in Base64 for security aims CR3. Concerning iOS, this vulnerability category presents one Plugin that detects whether the app uses insecure or deprecated APIs functions (WEB8).

2.3.7. Data leakage

This Plugin category provides one Plugin that checks if the app causes an information leakage of its data. In Android and iOS, this vulnerability category provides a Plugin that detects whether the app logs sensitive information inside the OS logging system, such as a user or device information that can be retrieved from a malicious user LEAK1.

3. Assessing SEBASTiAn

3.1. Illustrative example

We used SEBASTiAn to perform security analyses on an Android app. We tested a Ghera [13] app that presents a vulnerability that allows the file URL scheme to be executed inside a web view client provided by the app.[1] We also run all the security analysis Plugins on the app to verify the presence of false positive vulnerabilities. After the security analyses execution, SEBASTiAn produces the final report.[2] The report highlights the presence of a `WebViewAllowFileAccess` vulnerability, which is the target vulnerability in the benchmark app. The report also highlights vulnerabilities related to web view Javascript misconfigurations that allow the target vulnerability to be exploited. Listing 1 shows command line parameters used to run the security analyses on the app.

```
python3 -m cli \
-l en \
-t 1200 \
-gr \
-kf \
-ff \
WebViewAllowFileAccess-UnauthorizedFileAccess-Lean.
apk
```

Listing 1: Example of Command Line Parameters

[1] https://bitbucket.org/secure-it-i/android-app-vulnerability-benchmarks/src/master/Web/WebViewAllowFileAccess-UnauthorizedFileAccess-Lean/Benign/

[2] https://github.com/talos-security/SEBASTiAn/blob/master/docs/WebViewAllowFileAccess-UnauthorizedFileAccess-Lean.apk_report.json

Table 1
The distribution of the Android and iOS vulnerability categories.

Vulnerability Category	Android	iOS
Crypto	5	4
ICC	17	0
Networking	8	1
Permission	2	0
Storage	7	0
System	7	5
Data Leakage	0	1
Web	10	2

Table 2
Results of the benchmark analysis on the Android dataset.

Vuln. Class	SEBASTiAn
Crypto	5/5
ICC	9/17
Networking	6/8
Permission	2/2
Storage	4/7
System	7/7
Web	9/10
True Positives	42
False Positives	0
False Negatives	14

Table 3
Results of the benchmark analysis on the iOS dataset.

Vuln. Class	SEBASTiAn
Crypto	4/4
Networking	1/1
System	6/6
Web	2/2
Data Leakage	1/1
True Positives	14
False Positives	0
False Negatives	0

3.2. Testing the stability of SEBASTiAn

We evaluated the stability of SEBASTiAn through a well-defined benchmark of Android apps. We selected the Ghera [13] benchmark, a dataset that contains 56 vulnerable Android apps. Concerning the iOS case, we evaluate the SEBASTiAn performance using three benchmark apps. To the best of our knowledge, there does not exist a well-defined iOS benchmark at the state-of-the-art, so we could not thoroughly test SEBASTiAn performances against iOS apps. We selected the well-known iOS vulnerable apps DVIA-v2 [14], OversecuredVulnerableiOSApp [16], and Myriam [15] for the iOS evaluation part. Table 1 shows the distribution of vulnerabilities inside the Android and iOS benchmark we used for the SEBASTiAn evaluation. We executed SEBASTiAn on the 56 vulnerable Android apps of the benchmark, and we collected the results from the generated report files. Table 2 shows the vulnerabilities discovered from SEBASTiAn on the Android benchmark compared to the overall vulnerabilities for the category.

The category in which SEBASTiAn reported fewer vulnerabilities compared to its overall number is the ICC [51] one. This lack is due to the difficulty of statically detecting ICC vulnerabilities [52]. Indeed, ICC vulnerabilities can be effectively detected through dynamic analysis techniques unsupported by SEBASTiAn. Table 3

Table 4
Results of the benchmark analysis on the obfuscated Android dataset.

Vuln. Class	SEBASTiAn
Crypto	4/5
ICC	9/17
Networking	5/8
Permission	2/2
Storage	4/7
System	7/7
Web	9/10
True Positives	40
False Positives	0
False Negatives	16

Table 5
Results of the benchmark analysis on the obfuscated iOS dataset.

Vuln. Class	SEBASTiAn
Crypto	2/4
Networking	1/1
System	6/6
Web	0/2
Data Leakage	0/1
True Positives	9
False Positives	0
False Negatives	5

shows the vulnerabilities discovered by SEBASTiAn on the iOS benchmark.

SEBASTiAn also supports the vulnerability analysis of obfuscated apps. To empirically assess the efficacy of SEBASTiAn against Android app obfuscation, we obfuscated the 56 Ghera benchmark apps through the Obfuscapk tool [53]. We applied several obfuscation techniques on the target apps sequentially to increase the obfuscation rate and make the vulnerability detection process more challenging. In detail, we used the following Obfuscapk's plugins to obfuscate the apps:

- **AdvancedReflection**
- **ArithmeticBranch**
- **CallIndirection**
- **Renaming**
- **Nop**
- **Reorder**
- **Rebuild**

Table 4 shows the results of the security analyses execution on the obfuscated benchmarks for Android and iOS apps, respectively. SEBASTiAn detected two fewer vulnerabilities in the obfuscated case compared to the not obfuscated one. This result demonstrates a good level of resilience of SEBASTiAn against app obfuscation.

To demonstrate the SEBASTiAn efficacy on iOS obfuscated apps, we obfuscated the three vulnerable apps we used for the stability evaluation on the not obfuscated case. To accomplish this task, we used the MachObfuscator [54] tool to obfuscate vulnerable iOS apps. To the best of our knowledge, MachObfuscator is the only open-source tool that obfuscates Swift and Objective-C code exploiting symbol renaming techniques.

Table 5 shows the discovered vulnerability number for each category inside the obfuscated iOS apps.

4. Limitations

The stability tests indicate that SEBASTiAn detects 75% vulnerabilities inside the benchmark datasets we used for its evaluation. Also, it can still detect 71% vulnerabilities inside the obfuscated benchmarks, thereby confirming a high resiliency against obfuscation techniques.

Still, it is worth mentioning that SEBASTiAn has several limitations. First, it shares the common limitations of static analysis techniques. For example, one of the libraries used as a foundation of several security plugins, i.e., Androguard [20], has known limitations in terms of accuracy with respect to reflective call [55], thereby limiting its capability to identify parts of the control flow graph used by several SEBASTiAn's plugins. Also, the detection capabilities of SEBASTiAn are directly correlated with the number and type of plugins included. In the current version, SEBASTiAn does not contain plugins for native code, thereby limiting the analysis to the .dex code for Android, the Mach-O code for iOS, and the resources bundled inside the apps. However, the plugin system eases the integration of new detection plugins to extend SEBASTiAn capabilities.

5. Impact and conclusions

In this work, we presented SEBASTiAn, a novel static analysis tool that can be used for the security vetting of Android applications. In particular, SEBASTiAn can find a broad range of security vulnerabilities on Android and iOS. SEBASTiAn can be easily extended to include new security modules. Also, we demonstrated through an extensive experimental campaign that the tool could be effectively applied to analyze real-world Android and iOS apps despite their obfuscation level.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data are available on the tool's GitHub repository.

Acknowledgments

This work was partially supported by the Curiosity Driven grant "Security Assessment of Cross-domain Application Ecosystems" of the University of Genova funded by the European Union - NextGEneration EU program and by project SERICS (PE000 00014) under the NRRP MUR program and funded by the EU - NGEU.

Appendix A

The Appendix lists all the plugins developed for SEBASTiAn, divided into Plugins compatible with Android (Tables 6, 7, 8, 9, and 10) and iOS apps (Table 11). Each table reports the name of each plugin, its detailed description, and the CWE associated with the control. The list is also reported on the GitHub page [12].

Table 6
List of Android Plugins (1/5).

Android Plugins					
Name	CWE	Description	Name	CWE	Description
Crypto Constant IV (CR1)	CWE-798	This Plugin verifies if the initialization vector (IV) is not random. Consequently, encrypting a particular piece of information with a symmetric key will yield the same result every time encryption is applied to that information with the same symmetric key.	Runtime Command Root (SYS1)	CWE-250	This Plugin verifies whether the app requests root permissions through the command `Runtime.getRuntime()-.exec("su")` or the command `Runtime.getRuntime()-.exec("sudo")`.
Runtime Command (SYS2)	CWE-200	This Plugin validates if the app is using the critical function `Runtime.getRuntime()-.exec()`.	Crypto Constant Key (CR2)	CWE-798	This plugin checks if the app stores encryption keys in the source code. The apps that present this vulnerability are vulnerable to forgery attacks and information leaks.
Send SMS (SYS3)	CWE-359	This Plugin verifies if the app can send SMS messages (sendDataMessage, sendMultipartTextMessage, or sendTextMessage) that could be a cost for the user if maliciously exploited.	Access Device ID (SYS4)	CWE-200	This Plugin checks if the app gets the device ID (IMEI) to identify the specific device. This approach has three significant drawbacks: I) it is unusable on non-phone devices, II) it persists across device data wipes, and III) it needs a special privilege to be executed.
External Storage (ST1)	CWE-921	This plugin checks if the app uses the external storage access API. Any app in the system can access external storage; thus, its use for security-critical files is discouraged.	Access Internet Without Permission (PERM1)	CWE-200	This app contains code for Internet access but does not have internet permission in AndroidManifest.xml. This may be caused by an app misconfiguration or a malicious app that tries to access the network interface without proper permission.

Table 7
List of Android Plugins (2/5).

Android Plugins					
Name	CWE	Description	Name	CWE	Description
Intent Vulnerabilities (ICC1)	CWE-925	This Plugin identifies potentially unsafe uses of intents. Specifically, it detects IDOS (Unhandled Exceptions or Denial Of Service), XAS (Cross-application scripting), and FI (Fragment Injection) vulnerabilities.	Access Mock Location (PERM2)	CWE-250	The Plugin checks for the presence of the permission Access Mock Location in the Android Manifest. This permission only works in emulated environments and it is deprecated.
Implicit Intent Service (ICC2)	CWE-927	This Plugin identifies implicit intents within the app that can start services. Using an implicit intent to start a service is a security hazard because of the uncertainty of what service will respond to the intent.	Allow All Hostname (NET1)	CWE-940	This Plugin verifies whether the app validates the Common Name in the SSL certificate. This critical vulnerability allows attackers to implement MitM attacks with their valid certificates. It is a violation of OWASP Mobile TOP 10 Security Risks.
Insecure Connection (NET2)	CWE-319	This Plugin checks whether the app contains URLs that are not under SSL. This security hazard allows the interception of all the information exchanged with those URLs.	Backup Enabled (SYS5)	CWE-250	This Plugin checks whether the ADB Backup is enabled for the app. If this is the case, attackers with physical access to the device can copy all of the sensitive data of the app included in the backup.
Insecure Hostname Verifier (NET3)	CWE-940	The Plugin checks whether the app allows self-defined HOSTNAME VERIFIER to accept all Common Names. This critical vulnerability allows attackers to make MitM attacks with their valid certificates. It is a violation of OWASP Mobile TOP 10 Security Risks.	Base64 Strings (CR3)	CWE-200	This Plugin checks the presence of base64 encoded strings, notifying them to the developer.
Insecure Socket (NET4)	CWE-939	This Plugin checks the usage of `SSLCertificateSocket-Factory.createSocket-()`. The created socket can be vulnerable to Man-in-the-Middle (MitM) attacks.	Insecure Socket Factory (NET5)	CWE-939	This Plugin checks whether the app contains code that relies on instances of socket factory with all SSL security checks disabled, using an optional handshake timeout and SSL session cache. Those sockets are vulnerable to MitM attacks.

Table 8
List of Android Plugins (3/5).

Android Plugins					
Name	CWE	Description	Name	CWE	Description
Cordova Access Origin (HYB1)	CWE-940	This Plugin checks if the app properly configures the Network Request Whitelist options in its Cordova config.xml file. The Plugin also checks if the app configures the Network Request Whitelist options to accept plain HTTP URLs. The Network Request Whitelist options control which network requests are allowed.	Intent Filter Misconfiguration (ICC3)	CWE-926	The app contains misconfiguration in the intent filter of at least one component of the AndroidManifest.xml. The plugin also checks if the app does not include a Content Security Policy (CSP).
Invalid Server Certificate (NET6)	CWE-925	This Plugin inspects whether the app validates the SSL Certificates. It allows self-signed, expired, or mismatched CN certificates for SSL connection. This critical vulnerability allows attackers to make MitM attacks.	Cordova Allow Intent (HYB2)	CWE-941	This Plugin checks if the app configures the Intent Whitelist options to accept HTTPS URLs from any domain. The Intent Whitelist options control which URLs the app is allowed to ask the system to open. This Plugin also checks if the app configures the Navigation Whitelist options to accept HTTPS connections from any domain.
Keystore Without Password (SYS6)	CWE-359	This Plugin checks whether a password protects the Keystore. This security hazard allows malicious users with physical access to the Keystore file to access the keys and certificates.	Obfuscation Rate (SYS7)	CWE-359	This Plugin checks if the app is obfuscated and eventually reports its obfuscation rate.
Cordova Allow Navigation (HYB3)	CWE-940	This Plugin checks whether the Navigation Whitelist accepts HTTPS connections from any domain. Moreover, the Plugin checks if the app does not properly configure the Navigation Whitelist options in its Cordova config.xml file.	Permission Dangerous (HYB4)	CWE-200	This Plugin checks whether the protection level of the app's classes is *dangerous*, allowing any other app to access this permission (AndroidManifest.xml).
Permission Normal (PERM3)	CWE-200	This Plugin checks whether at least one class declared in the Android Manifest of the app is protected with custom permissions, defined with a *normal* or *default* permission level. This allows malicious apps to register and receive messages for this app.	Crypto Constant Salt (CR4)	CWE-312	This Plugin checks if the app uses a constant salt.

Table 9
List of Android Plugins (4/5).

Android Plugins					
Name	CWE	Description	Name	CWE	Description
Shared User ID (SYS8)	CWE-250	The Plugin checks if the app uses the `sharedUserId` attribute. Suppose this attribute is set to the same value for two or more applications. In that case, they will all share the same ID — provided the same certificate also signs them.	Crypto ECB Cipher (CR5)	CWE-312	This Plugin checks if the app uses Block Cipher algorithms in ECB mode for encrypting sensitive information. Block Cipher algorithms in ECB mode are known to be vulnerable.
Sqlite Exec SQL (WEB1)	CWE-200	This Plugin checks for the presence of the `execSQL()` method of SQLite. The `execSQL()` method could create a SQL query based on the user input content, potentially vulnerable to SQL injection attacks.	Crypto Keystore Entry Without Password (CR6)	CWE-359	This Plugin checks if an app stores encryption keys without any protection parameter in a Keystore accessible by other apps. Apps that present this behavior are vulnerable to information exposure. Keystore provides setEntry API to store a key entry. Along with the alias and the key, setEntry API takes an instance of ProtectionParameter as an argument to protect the entry contents.
System Permission Usage (PERM4)	CWE-250	This Plugin checks if the app uses necessary permissions to manage the filesystem and packages. The System Permission should be confined only to device manufacturers or Google system apps. If not, it may be a malicious app.	Crypto Small Iteration Count (CR7)	CWE-312	The Plugin checks, if the app passes an iteration, count smaller than 1000. If so, the PBEParameterSpec and the PBEKeySpec constructors are insecure.
Webview Allow File Access (WEB2)	CWE-359	The Plugin checks for the presence of the `setAllowFileAccess-(true)` method in a WebView.	Debuggable Application (SYS9)	CWE-200	The Plugin checks if the DEBUG mode is on inside the app. Debug mode is discouraged in production since malicious users can debug the app and sniff verbose error information through Logcat.
Webview Ignore SSL Error (WEB3)	CWE-297	The plugin checks if an Android app can display web pages by loading HTML/JavaScript files in a WebView.	Default Scheme HTTP (NET7)	CWE-319	This Plugin checks if the app uses `HttpHost` but its default scheme is `HTTP`.

Table 10
List of Android Plugins (5/5).

Android Plugins					
Name	CWE	Description	Name	CWE	Description
Webview Intercept Request (WEB4)	CWE-939	The Plugin checks if the app loads resources using a WebView, and controls the resources being loaded on a webpage via the `shouldInterceptReque-st` method in WebViewClient.	Dynamic Code Loading (SYS10)	CWE-921	This Plugin checks if the app contains code that dynamically loads classes from .jar files.
Webview Javascript Enabled (WEB5)	CWE-200	This Plugin checks if the app uses `setJavaScriptEnabled-(true)` in WebView. Enabling JavaScript exposes to malicious code injection that would be executed with the same permissions (XSS attacks).	Empty Permission Group (PERM5)	CWE-359	This Plugin searches for a user-defined empty `permissionGroup` in the Android Manifest. Setting the `permissionGroup` attribute to an empty value will invalidate the permission definition, and no other application can use the permission.
Webview Javascript Interface (WEB6)	CWE-939	This Plugin verifies the presence of the `addJavascriptInterfa-ce` method within the app code.	Exported Component (ICC4)	CWE-926	This Plugin detects exported components for receiving external applications' actions. These components can be initialized by other apps and used maliciously. The Plugin also checks exported ContentProvider, allowing any other app on the device to access it (AndroidManifest.xml). Found exported components lacking `android:` prefix in an exported attribute.
Webview Override Url (WEB7)	CWE-939	This Plugin checks whether the Android app shows web content in a WebView, and controls navigation across webpages via the `shouldOverrideUrlLoa-ding` method in WebViewClient.	World Readable Writable (SYS11)	CWE-359	This Plugin checks whether the app allows file access in World Readable/Writable mode. This functionality is deprecated in API Level 17 and removed since API Level 24.

Table 11
List of iOS Plugins.

iOS Plugins					
Name	CWE	Description	Name	CWE	Description
Restricted Segment Missing (SYS12)	CWE-921	The Plugin checks if the Mach-O binary file does not have a restricted segment that prevents dynamic loading of dylib for arbitrary code injection.	Logging Function (LEAK1)	CWE-200	The Plugin checks if the Mach-O binary makes use of logging function(s).
Arc Flag Missing (SYS13)	CWE-772	The Plugin checks if the binary is compiled without the Automatic Reference Counting (ARC) flag. ARC is a compiler feature that provides automatic memory management of Objective-C objects and is an exploit mitigation mechanism against memory corruption vulnerabilities.	Nx Bit Missing (SYS14)	CWE-921	The Plugin checks if the binary does not have the NX bit set. The NX bit offers protection against exploitation of memory-corruption vulnerabilities by marking the memory page as non-executable. However, iOS never allows an app to execute from writeable memory.
Insecure Api (WEB8)	CWE-939	The Plugin checks if the binary makes use of insecure API(s).	Encryption Missing (CR8)	CWE-200	The Plugin checks if the binary is not encrypted.
Insecure Connection Plist (NET8)	CWE-941	The Plugin checks if the app adds exceptions for possible insecure connections in `Info.plist` file.	Insecure Tls Version Plist (NET9)	CWE-940	The application sets the minimum value of TLS version TLSv1.0 or TLSv1.1, which are unsafe.
Symbols Stripped (SYS15)	CWE-200	The Plugin checks if the app does not present symbols.	Code Signature Missing (CR9)	CWE-295	The Plugin checks if the binary does not have a code signature.
Rpath Set (SYS16)	CWE-250	The Plugin checks if the binary has Runpath Search Path (@rpath) set.	Malloc Function (SYS17)	CWE-772	The Plugin checks if the binary may use the malloc function.
Allow Http Plist (NET10)	CWE-319	The Plugin checks if the app allows the use of the HTTP protocol.	Insecure Random (CR10)	CWE-939	The Plugin checks if the binary uses some insecure random API(s).
No Forward Secrecy Plist (CR11)	CWE-940	The Plugin checks if the app disables the server's requirement to support Perfect Forward Secrecy (PFS) through Elliptic Curve Diffie–Hellman Ephemeral (ECDHE).	Weak Hashes (CR12)	CWE-200	The Plugin checks if the binary uses some weak hashing API(s).
Weak Crypto (CR13)	CWE-359	The Plugin checks if the binary uses some weak crypto API(s).	Stack Canary Missing (SYS18)	CWE-672	The Plugin checks if the binary does not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return addresses.
Pie Flag Missing (SYS19)	CWE-200	The Plugin checks if the binary is built without the Position Independent Code flag.			

References

[1] atlasVPN. Over 60% of android apps have security vulnerabilities. 2023, https://atlasvpn.com/blog/over-60-of-android-apps-have-security-vulnerabilities, (Accessed in June 22, 2023).

[2] GitHub. Mobile security framework (mobsf). 2023, https://github.com/MobSF/Mobile-Security-Framework-MobSF, (Accessed in June 22, 2023).

[3] Mobsfscan. 2023, https://github.com/MobSF/mobsfscan, (Accessed in June 22, 2023).

[4] Talukder MAI, Shahriar H, Qian K, Rahman M, Ahamed S, Wu F, Agu E. DroidPatrol: A static analysis plugin for secure mobile software development. In: 2019 IEEE 43rd annual computer software and applications conference, Vol. 1. COMPSAC, IEEE; 2019, p. 565–9.

[5] Wei F, Roy S, Ou X. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. ACM Trans Priv Secur 2018;21(3):1–32.

[6] GitHub. Qark. 2023, https://github.com/linkedin/qark, (Accessed in June 22, 2023).

[7] Caputo D, Verderame L, Aonzo S, Merlo A. Droids in disarray: Detecting frame confusion in hybrid android apps. In: IFIP annual conference on data and applications security and privacy. Springer; 2019, p. 121–39.

[8] GitHub. Trueseeing. 2023, https://github.com/alterakey/trueseeing, (Accessed in June 22, 2023).

[9] Developers A. App manifest overview. 2023, https://developer.android.com/guide/topics/manifest/manifest-intro, (Accessed in June 22, 2023).

[10] Razican. SUPER android analyzer. 2023, https://github.com/SUPERAndroidAnalyzer/super, (Accessed in June 22, 2023).

[11] Developer G. Shrink, obfuscate, and optimize your app. 2023, https://developer.android.com/studio/build/shrink-code, (Accessed in June 22, 2023).

[12] GitHub. SEBASTiAn repository. 2023, https://github.com/talos-security/SEBASTiAn, (Accessed in June 22, 2023).

[13] Mitra J, Ranganath V-P. Ghera: A repository of android app vulnerability benchmarks. In: Proceedings of the 13th international conference on predictive models and data analytics in software engineering. 2017, p. 43–52.

[14] GitHub. DVIA. 2023, https://github.com/prateek147/DVIA-v2, (Accessed in June 22, 2023).

[15] GitHub. Myriam. 2023, https://github.com/GeoSn0w/Myriam, (Accessed in June 22, 2023).

[16] GitHub. OversecuredVulnerableiOSApp. 2023, https://github.com/oversecured/OversecuredVulnerableiOSApp, (Accessed in June 22, 2023).

[17] OWASP. OWASP mobile top 10. 2023, https://owasp.org/www-project-mobile-top-10/, (Accessed in June 22, 2023).

[18] NIST N. Common vulnerability scoring system calculator. 2023, https://nvd.nist.gov/vuln-metrics/cvss/v2-calculator, (Accessed in June 22, 2023).

[19] GitHub. Yapsy. 2023, https://github.com/tibonihoo/yapsy, (Accessed in June 22, 2023).

[20] GitHub. Androguard. 2023, https://github.com/androguard/androguard, (Accessed in June 22, 2023).

[21] GitHub. Lief. 2023, https://github.com/lief-project/LIEF, (Accessed in June 22, 2023).

[22] GitHub. Sebastian readme. 2023, https://github.com/talos-security/SEBASTiAn/blob/master/README.md, (Accessed in June 22, 2023).

[23] Egele M, Brumley D, Fratantonio Y, Kruegel C. An empirical study of cryptographic misuse in android applications. In: Proceedings of the 2013 ACM SIGSAC conference on computer & communications security. Association for Computing Machinery; 2013, p. 73–84. http://dx.doi.org/10.1145/2508859.2516693.

[24] Developer A. Android keystore system. 2023, https://developer.android.com/training/articles/keystore, (Accessed in June 22, 2023).

[25] Developer A. PbeKeySpec. 2023, https://developer.android.com/reference/javax/crypto/spec/PBEKeySpec, (Accessed in June 22, 2023).

[26] VMware. Perfect forward secrecy definition. 2023, https://avinetworks.com/glossary/perfect-forward-secrecy/, (Accessed in June 22, 2023).

[27] Developer A. Intent & intent filter. 2023, https://developer.android.com/guide/components/intents-filters, (Accessed in June 22, 2023).

[28] Developer A. Intents and intent filters. 2023, https://developer.android.com/guide/components/intents-filters#Types, (Accessed in June 22, 2023).

[29] Garcia J, Hammad M, Ghorbani N, Malek S. Automatic generation of inter-component communication exploits for android applications. In: Proceedings of the 2017 11th joint meeting on foundations of software engineering. New York, NY, USA: Association for Computing Machinery; 2017, p. 661–71. http://dx.doi.org/10.1145/3106237.3106286.

[30] Cordova A. Intent whitelist. 2023, https://cordova.apache.org/docs/en/9.x/reference/cordova-plugin-whitelist/#intent-whitelist, (Accessed in June 22, 2023).

[31] Conti M, Dragoni N, Lesyk V. A survey of man in the middle attacks. IEEE Commun Surv Tutor 2016;18(3):2027–51. http://dx.doi.org/10.1109/COMST.2016.2548426.

[32] Cordova A. Network request whitelist. 2023, https://cordova.apache.org/docs/en/8.x/reference/cordova-plugin-whitelist/#network-request-whitelist, (Accessed in June 22, 2023).

[33] Cordova A. Content security policy (CSP). 2023, https://cordova.apache.org/docs/en/10.x/guide/appdev/allowlist/#content-security-policy-csp, (Accessed in June 22, 2023).

[34] Cordova A. Navigation whitelist. 2023, https://cordova.apache.org/docs/en/10.x/guide/appdev/allowlist/#other-notes, (Accessed in June 22, 2023).

[35] Developer A. Permission groups. 2023, https://developer.android.com/guide/topics/manifest/permission-group-element, (Accessed in June 22, 2023).

[36] Developer A. Permissions. 2023, https://developer.android.com/guide/topics/manifest/permission-element, (Accessed in June 22, 2023).

[37] androidpermissionscom. Android permissions. 2023, http://androidpermissions.com/permission/android.permission.ACCESS_MOCK_LOCATION, (Accessed in June 22, 2023).

[38] Ghera. CheckPermission-PrivilegeEscalation-lean. 2023, https://bitbucket.org/secure-it-i/android-app-vulnerability-benchmarks/src/master/System/CheckPermission-PrivilegeEscalation-Lean/, (Accessed in June 22, 2023).

[39] Qu Z, Alam S, Chen Y, Zhou X, Hong W, Riley R. Dydroid: Measuring dynamic code loading and its security implications in android applications. 2017, p. 415–26. http://dx.doi.org/10.1109/DSN.2017.14.

[40] NowSecure. World readable writable. 2023, https://books.nowsecure.com/secure-mobile-development/en/android/implement-file-permissions-carefully.html, (Accessed in June 22, 2023).

[41] Adjust. What is an android id? (AAID). 2023, https://www.adjust.com/glossary/android-id-definition/, (Accessed in June 22, 2023).

[42] androidauthoritycom. What is IMEI and why should you care?. 2023, https://www.androidauthority.com/what-is-imei-923061/, (Accessed in June 22, 2023).

[43] Developer A. Overview of the mach-o executable format. 2023, https://developer.apple.com/library/archive/documentation/Performance/Conceptual/CodeFootprint/Articles/MachOOverview.html, (Accessed in June 22, 2023).

[44] Developer A. Automatic reference counting. 2023, https://opensource.apple.com/source/clang/clang-211.10.1/src/tools/clang/docs/AutomaticReferenceCounting.html, (Accessed in June 22, 2023).

[45] pewpewthespellscom. Blocking code injection on iOS and OS x. 2023, https://pewpewthespells.com/blog/blocking_code_injection_on_ios_and_os_x.html, (Accessed in June 22, 2023).

[46] Unipi. Non executable data. 2023, https://lettieri.iet.unipi.it/hacking/non-exec-data.pdf, (Accessed in June 22, 2023).

[47] Blog C. Nabout ELF – PIE, PIC and else. 2023, https://codywu2010.wordpress.com/2014/11/29/about-elf-pie-pic-and-else/, (Accessed in June 22, 2023).

[48] blogkrzyzanowskimcom. @rpath what?. 2023, https://blog.krzyzanowskim.com/2018/12/05/rpath-what/, (Accessed in June 22, 2023).

[49] Developer A. WebView. 2023, https://developer.android.com/reference/android/webkit/WebView, (Accessed in June 22, 2023).

[50] Developer A. Webviewclient. 2023, https://developer.android.com/reference/android/webkit/WebViewClient, (Accessed in June 22, 2023).

[51] Tian C, Xia C, Duan Z. Android inter-component communication analysis with intent revision. In: Proceedings of the 40th international conference on software engineering: companion proceeedings. 2018, 100403. http://dx.doi.org/10.1145/3183440.3194957.

[52] Demissie BF, Ghio D, Ceccato M, Avancini A. Identifying android inter-app communication vulnerabilities using static and dynamic analysis. In: 2016 IEEE/ACM international conference on mobile software engineering and systems (MOBILESoft). 2016, p. 255–66.

[53] Aonzo S, Georgiu GC, Verderame L, Merlo A. Obfuscapk: An open-source black-box obfuscation tool for android apps. SoftwareX 2020. http://dx.doi.org/10.1016/j.softx.2020.100403.

[54] MachObfuscator. Machobfuscator. 2023, https://github.com/kam800/MachObfuscator, (Accessed in June 22, 2023).

[55] Li L, Bissyandé TF, Octeau D, Klein J. Droidra: Taming reflection to support whole-program analysis of android apps. In: Proceedings of the 25th international symposium on software testing and analysis. 2016, p. 318–29.