AUSERA: Automated Security Vulnerability Detection for Android Apps

Sen Chen
College of Intelligence and Computing, Tianjin University
Tianjin, China
senchen@tju.edu.cn

Yuxin Zhang
College of Intelligence and Computing, Tianjin University
Tianjin, China

Lingling Fan
College of Cyber Science, Nankai University
Tianjin, China

Jiaming Li
College of Intelligence and Computing, Tianjin University
Tianjin, China

Yang Liu
Nanyang Technological University
Singapore, Singapore

ABSTRACT

To reduce the attack surface from app source code, massive tools focus on detecting security vulnerabilities in Android apps. However, some obvious weaknesses have been highlighted in the previous studies. For example, (1) most of the available tools such as AndroBugs, MobSF, Qark, and Super use pattern-based methods to detect security vulnerabilities. Although they are effective in detecting some types of vulnerabilities, a large number of false positives would be introduced, which inevitably increases the patching overhead for app developers. (2) Similarly, static taint analysis tools such as FlowDroid and IccTA present hundreds of vulnerability candidates of data leakage instead of confirmed vulnerabilities. (3) Last but not least, a relatively complete vulnerability taxonomy is missing, which would introduce a lot of false negatives. In this paper, based on our prior knowledge in this research domain, we empirically propose a vulnerability taxonomy as the baseline and then extend AUSERA by augmenting the detection capability to 50 security vulnerability types. Meanwhile, a new benchmark dataset including all these 50 vulnerability types is constructed to demonstrate the effectiveness of AUSERA. The tool and datasets are available at https://github.com/tjusenchen/AUSERA and the demonstration video can be found at https://youtu.be/UCiGwVaFPpY.

CCS CONCEPTS

• **Security and privacy → Software and application security**.

KEYWORDS

Android Apps, Security Vulnerability, Vulnerability Detection

ACM Reference Format:
Sen Chen, Yuxin Zhang, Lingling Fan, Jiaming Li, and Yang Liu. 2022. AUSERA: Automated Security Vulnerability Detection for Android Apps. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3551349.3559524

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3559524

1 INTRODUCTION

Nowadays, with the rapid development of smartphones, more and more Android applications (apps) are being developed for different daily tasks, such as shopping, reading, and banking [14, 15]. Actually, Google Play Store contains more than 3 million Android apps to gain mobile users. Consequently, with the growth of Android apps and their users, security and privacy concerns are increasingly becoming the focus of great concern to various stakeholders [16, 17]. For example, more and more users store sensitive data via Android apps, including personal data and financial transfer data. Attackers attempt to exploit app vulnerabilities in order to gain financial gains or sensitive data from Android users, which is one of the most severe security threats in the app ecosystem. Security risk assessment of apps is not only of great significance to Android users but also of significance to guiding app developers during the development process. For example, as shown in Listing 1, in a popular banking app [16, 17] from Google Play, users are asked to register with their personal identity information (e.g., first name, last name, password, and address), which will be sent (in plain text) via SMS to authenticate the user. Unexpectedly, such registration data is also stored in the SMS outbox, attackers with permission to read SMS can easily intercept the sensitive data and impersonate that user to manipulate her legitimate banking account.

Actually, app security vulnerabilities with domain-specific characteristics are different from Android Operating System (OS) vulnerabilities. Vulnerabilities in Android OS are mainly thrown from Linux kernel or architecture design, while the main root cause of app vulnerabilities is due to the improper implementation in Java/Kotlin implementation code, C/C++ native code (.so files), and third-party libraries [19, 20]. Although it is an urgent need to provide an automated security risk assessment system for Android apps, there lacks a comprehensive solution to do this task based on the existing static security analysis tools, such as AndroBugs [1], MobSF [10], Qark [4], Super [5], FlowDroid [13], IccTA [18], JAADAS [3], and CogniCrypt. For example, (1) some existing tools only rely on the pattern-based method to scan the

app code, which introduces a lot of false positives and is imprecise. Besides, they are not able to identify security vulnerabilities related to sensitive data in Android apps such as SMS data leakage and *SharedPreference* data leakage. (2) Although the static taint analysis tools such as FlowDroid [13] and IccTA [18] can alleviate the false positive problem to some extent, they also output hundreds of vulnerability candidates instead of confirmed vulnerabilities such as logging data leakage, which cannot provide insights for app developers to patch the corresponding vulnerabilities. (3) Last but not least, a relatively complete vulnerability taxonomy is missing, which would introduce many false negatives. All in all, existing tools are not precise with low detection accuracy (*high false positive and false negative rate*).

```
// Get sensitive data from EditTexts
public String getregisterSms() {
  StringBuilder m = new StringBuilder("REG");
  m.append(getPin() + "/");
  m.append(getFirstName() + "/");
  m.append(getLastName() + "/");
  m.append(getAddress() + "/");
  return m.toString();
}
// Send the sensitive data via SMS
public void execute() {
  sendSmsMessage(getRegisterSms());
}
private void sendSmsMessage(String message) {
  this.smsManager.setMessage(message);
  this.smsManager.setDestinationAddress("...");
  SmsHandler.builder().activity(this.activity);
  smsManager(this.smsManager).build().send();
}
```

Listing 1: Pseudo code that leaks sensitive data by SMS [17].

To this end, based on our previous work in this research domain [16, 17, 19] and the understanding of the international mobile security standards such as NIST [9], and OWASP [11], we first propose a relatively complete vulnerability taxonomy, which contains 5 main vulnerability categories and 50 vulnerability types within Android apps. Secondly, we further extend AUSERA [16] for Android app vulnerability detection and security risk assessment, by leveraging static program analysis such as data- and control-flow analysis, sensitive data tagging, etc. To demonstrate the effectiveness of AUSERA, we construct a new benchmark dataset involving all these 50 vulnerability types in our proposed vulnerability taxonomy. We finally evaluate AUSERA with 4 academic static analysis tools including AndroBugs [1], MobSF [10], Qark [4], and Super [5].

We conclude our contributions as follows:

- We propose a relatively complete vulnerability taxonomy of app source code in Android apps, which includes 5 main categories and 50 vulnerability types.
- We extend and release an **au**tomated **se**curity **r**isk **a**ssessment tool for vulnerability detection in Android apps, named AUSERA,[1] which effectively reduces the false positives and false negatives compared with the existing tools.

2 AUSERA

As shown in Figure 1, AUSERA takes as input each app, and the keyword set of sensitive data with a formal representation of regular expression, the newly-defined set of sources and sinks guided by the proposed vulnerability taxonomy, and ultimately outputs the set of security vulnerabilities in the app under test, the damage, potential attacks, and patch methods.

2.1 Security Vulnerability Taxonomy

We previously proposed a vulnerability taxonomy specific for banking apps [16]. However, in this paper, the taxonomy of app vulnerabilities is more general for all types of Android apps in the wild. Therefore, one of the most important criteria is to include as many vulnerability types as possible. To do this, we propose the taxonomy according to the following criteria. (1) We keep all of the summarized vulnerability types for the banking apps. (2) We take almost all the vulnerability types implemented in the open-source tools such as AndroBugs and MobSF into the taxonomy. (3) We refer to the international mobile security standards such as NIST [9]. (4) We also integrate the industrial best practice defined in OWASP and security vulnerability database such as CVE [8] and weakness database such as CWE [7]. (5) Last but not least, we take the authors' prior knowledge in this research domain into account to make the taxonomy more complete and accurate. Finally, as shown in Table 1, we conclude 5 vulnerability categories, namely *(sensitive) data storage security* (contains 8 vulnerability types), *data encryption security* (includes 9 vulnerability types), *data access security* involving 19 types, *communication authentication security* (has 9 vulnerability types), and *configuration security* (accounts for 5 types), respectively. There are 50 vulnerability types in total.

To the best of our knowledge, this is the first work to provide a relatively systematic and complete taxonomy of security vulnerabilities for Android apps.

2.2 Used Techniques in AUSERA

For the techniques used in AUSERA, we also integrate the basic design from [16]. The main updates are as follows. (1) We first update the keywords of sensitive data to make it more general for common apps instead of only for one type of those apps. Note that, the set also can be customized by users. For example, if the tool is used to scan healthcare apps, they can add more professional health keywords to represent sensitive data for their scanning tasks. (2) We update the set of sinks to map more vulnerability types introduced in the category of *data storage security* such as android.database.sqlite.SQLiteDatabase: void execSQL for SQLite leakage. The defined sinks can be found in the file of SourcesAndSinks in the configuration folder. (3) For the phase of function identification, we add more APIs to help locate the functional implementations. Note that, the vulnerable functions are not identified as a true vulnerable case until they have been verified for the reachability analysis. For example, if the invalid server verification (do nothing in the verification function body) is not triggered by any other methods or classes, it is considered no threat to mobile users. Reachability analysis can reduce false positives caused by dead code in apps. (4) We also expand the pattern-based analysis by considering the preconditions for vulnerability occurrence. For example, some vulnerable code is only triggered under a specific SDK version range or with exported components. (5) Apart from the vulnerability introduced by the source code, the vulnerable

[1] https://github.com/tjusenchen/AUSERA

Table 1: Taxonomy of security vulnerabilities in Android apps and detection results on the ground-truth dataset.

Category	Taxonomy (50 types)	AUSERA (50 types)	AndroBugs (20 types)	MobSF (20 types)	Qark (11 types)	Super (15 types)
(Sensitive) Data Storage Security (8 types)	Use SharedPreference	●				
	Use SQLite	●				
	Use webview.db	●				
	Use LogCat	●		◐	◐	◐
	Use External Storage	●	◐	◐		◐
	Use Internal Storage	●				
	Use Text files	●				
	Use TempFile	●		◐		
Data Encryption Security (9 types)	Use insecure AES encryption	●			●	●
	Use insecure RSA encryption	●				
	Use insecure SecureRandom	●		●	●	●
	Use insecure MD5 hash function	●		●		
	Use insecure SHA-1 hash function	●		●		
	Use insecure Base64 encryption	●	●			●
	Use insecure Blowfish encryption	●				
	Use insecure DES encryption	●				
	Use hard-coded encryption key	●		◐	◐	
Data Access Security by Vulnerable Function (19 types)	Use dynamically registered Receiver	●				
	Use Implicit Intent	●				
	Use Intent-filter causing component exported	●		●		
	Use empty pending intent	●			●	
	Use exported Activity	●	●	●	●	●
	Use exported Service	●	●	●	●	●
	Use exported Content Provider	●	●	●	●	●
	Use exported Broadcast Receiver	●	●	●	●	●
	Use WebView JavaScript Interface	●	◐		◐	◐
	Use WebView setAllowFileAccess	●	◐		◐	
	Use WebView setPluginState	●				
	Use getRuntime#exec	●	●			
	Use DexClassLoader	●	●			
	Use loadLibary (.so binary files)	●	●	●		
	Use vulnerable binary libraries (supported by Scantist [12])	●				
	Use vulnerable third-party libraries (supported by ATVHunter [19])	●				
	SQL injection attack	●		●		●
	Fragment injection attack	●	◐			
	Content provider file traversal attack (openFile)	●				
Communication Authentication Security (9 types)	Use SMS transmission with sensitive data	●				
	Use HTTP protocol	●	●			
	Use expired certificates	●		●		
	Use certificate with SHA-1 or MD5	●		●		
	Use invalid certificate authentication (Allow all hostname request)	●	●		●	●
	Use invalid hostname verification	●	●	●		
	Use invalid server verification	●	●		●	
	Use invalid certificate authentication in WebView	●	●	●		●
	Use insecure network ports	●				
Configuration Security (5 types)	Allow UI screenshots	●	◐			
	Allow Backup in Manifest	●	●	●		●
	Allow Debug in Manifest	●	●	●		●
	Config MODE_WORLD_READABLE	●				
	Config MODE_WORLD_WRITABLE	●				

●: Can detect the corresponding vulnerability type ◐: Can detect but would introduce false positives

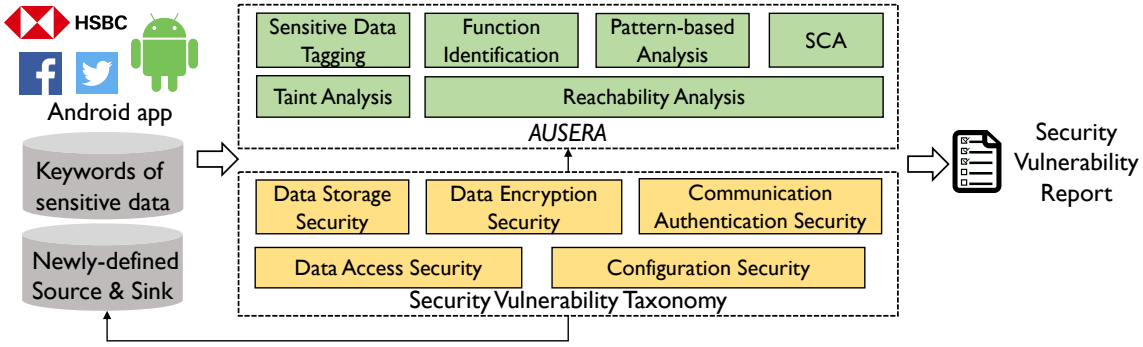

Figure 1: Overview of AUSERA.

```
Public Id: BUG-A003-0001; Type: Security Bug; Risk Level: High; Risk Score: 8;
Sub Type: SMS data leakage; // App vulnerability type
Description: The app sends an SMS attached with the sensitive data (in plaintext) to authenticate that user, but the data is stored in the SMS outbox unexpectedly. If an adversary registers a content observer to the SMS outbox on the mobile device with some permissions, the user's sensitive data can be easily intercepted by the adversary who impersonates that user to manipulate her legitimate banking account.
Location: Found a flow to sink virtualinvoke $r10.<android.telephony.SmsManager: void sendTextMessage(), from the following sources: $r5 = virtualinvoke $r4.<android.widget.EditText: android.text.Editable getText()>() (in <com.globe.gcash.android.activity.transaction.RegistrationTransactionActivity: void doNext()>)
=> RegistrationTransactionActivity;doNext();$r4;$r5 // Activity, Method, Variables logging
==> pin;firstName;lastName;addr // Sensitive data tagging
Patch Method: Avoid sending sensitive data via SMS and store the sensitive data in the SMS outbox accordingly.
```

Listing 2: An example of a security vulnerability report generated by AUSERA.

third-party libraries also cause security threats for users, which are likely to be inadvertently introduced. Therefore, we also integrate ATVHunter [19] to involve the ability of binary SCA for apps. More technique details can be found in [16].

As shown in Listing 2,the final report is a JSON file including basic attributes such as app name, version name, hash value, and vulnerability details. For each detected vulnerability, we have 8 fields (i.e., Public Id, Type, Sub Type, Risk Level, Risk Score, Description, Location, and Patch Method). Currently, we provide English and Chinese versions for users.

3 EVALUATION

To demonstrate the effectiveness of AUSERA, we select 4 open-source and representative tools (i.e., AndroBugs [1], MobSF [10], Qark [4], and Super [5]) to compare the detection results. We use their latest versions on GitHub and conduct the evaluation on Ubuntu 21.04 with 64G memory and Intel@Core i9-100900 CPU@2.80GHz × 20.

3.1 Ground-truth Dataset

Due to the lack of an available benchmark dataset of the vulnerability taxonomy, we first present a ground-truth dataset including all 50 types of vulnerabilities summarized in our taxonomy. We construct the ground-truth dataset based on the following criteria. (1) We first consider the existing benchmark apps as part of the ground-truth dataset, which includes two benchmark apps (i.e., Diva app [2] and MSTG app [6]). DIVA (Damn insecure and vulnerable App) is an app intentionally designed to be insecure with 727 stars. Diva app contains many representative data leakage and data access vulnerabilities. All these vulnerabilities are manually injected into one app. Similarly, the MSTG app [6] is used as an example to demonstrate different vulnerabilities explained in the OWASP Mobile Security Testing Guide. (2) These two benchmark apps cannot cover all vulnerabilities in taxonomy, therefore, we use another 4 apps in our previous studies [16, 17]. These four apps are all banking apps, and the vulnerabilities within these apps have been patched by developers according to our reporting in the latest versions. Therefore, the disclosure will not cause damage in the real world. We also release these 4 apps on the GitHub repository (https://github.com/tjusenchen/AUSERA).

3.2 Evaluation Result

The evaluation result is shown in Table 1. Compared with the existing 4 tools, AUSERA obtains the best performance. Specifically, on the one hand, owing to the sensitive data tagging and reachability analysis, AUSERA significantly reduces the false positives, which has been validated in [16]. On the other hand, owing to the proposed taxonomy, compared with other tools, the result of AUSERA is more complete with low false negatives.

Compared with the vulnerability types, AUSERA achieves a more complete result with all 50 vulnerability types. On the contrary, AndroBugs, MobSF, Qark, and Super support 20 types, 20 types, 11 types, and 15 types, respectively. Meanwhile, even for the support vulnerability types, due to the technical limitations, they will inevitably introduce false positives. For example, sensitive data disclosure through LogCat is always detected by MobSF, Qark, and Super as shown in Table 1, but they just match the following APIs

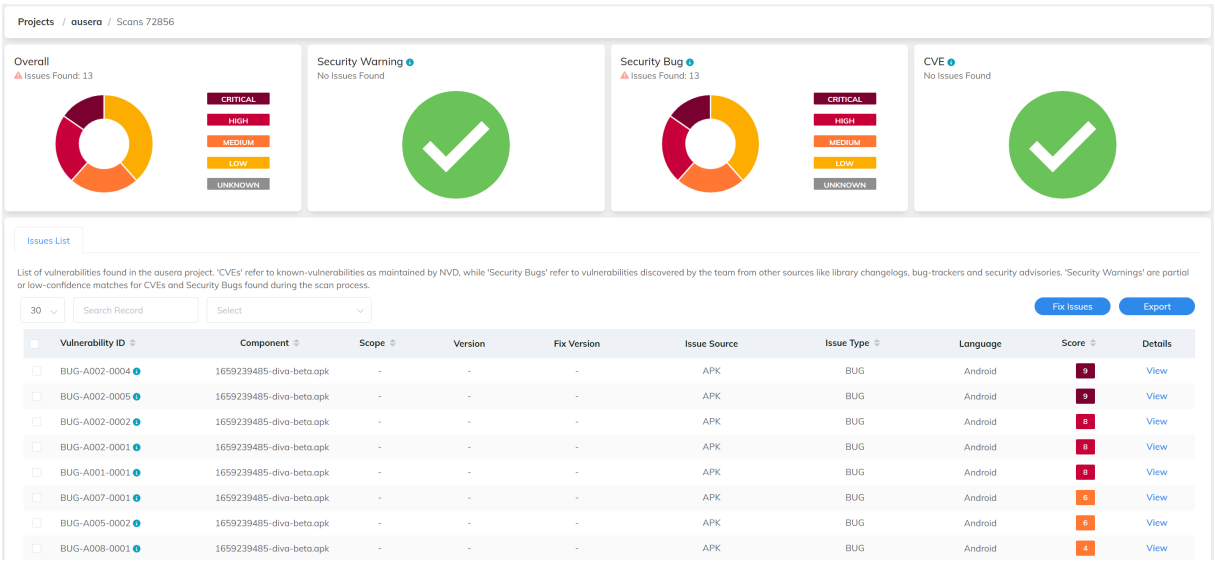

Figure 2: Online scanning service provided by AUSERA.

if used (e.g., Log.e(), Log.d(), and Log.v()). There is no doubt that it has incurred plenty of false positives. If the data are not sensitive, such as "*menu_title*," it is very normal for developers to log or write messages to understand the state of their apps. The risk is that some credentials (e.g., PIN code and password) are also leaked by logging outputs. Another example is the vulnerability of using the WebView JavaScript Interface, which is supported by AndroBugs, Qark, and Super. However, they only check the implementation status of the API *addJavascriptInterface()*, they ignore the vulnerability only occurs when the SDK version is lower than 4.2. Similarly, for the Fragment injection vulnerability, only when the component is exported by default, the vulnerable Fragment code inherited from PreferenceActivity causes a real vulnerability. However, AndroBugs ignores checking the precondition causing false positives.

4 CONCLUSION

In this paper, we proposed a security vulnerability taxonomy for Android apps, which includes 5 categories and 50 vulnerability types, based on which we implemented and released AUSERA to automatically detect vulnerability in Android apps, achieving an accurate result compared with existing tools. Finally, we highlight that the set of sensitive data and vulnerability types can be further expanded according to the detection scenarios for app developers and security analysts.

Through cooperation with a leading security company from Singapore, named Scantist [12], AUSERA is also in the process of product development to provide security scanning service for users, as shown in Figure 2.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (No. 62102284, 62102197).

REFERENCES

[1] 2015. *AndroBugs.* https://github.com/AndroBugs/AndroBugs_Framework
[2] 2016. *DIVA App.* https://github.com/payatu/diva-android
[3] 2017. *JAADAS.* https://github.com/flankerhqd/JAADAS
[4] 2018. *Qark.* https://github.com/linkedin/qark
[5] 2018. *Super.* https://github.com/SUPERAndroidAnalyzer/super
[6] 2020. *MSTG App.* https://github.com/OWASP/MSTG-Hacking-Playground
[7] 2022. *Common Weakness Enumeration: CWE.* https://cwe.mitre.org/
[8] 2022. *CVE.* https://cve.mitre.org/
[9] 2022. *Mobile NIST.* https://www.nist.gov/mobile
[10] 2022. *MobSF.* https://github.com/MobSF/Mobile-Security-Framework-MobSF
[11] 2022. *OWASP.* https://owasp.org/
[12] 2022. *Scantist Pte. Ltd.* https://scantist.io/
[13] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* 49, 6 (2014), 259–269.
[14] Sen Chen, Lingling Fan, Chunyang Chen, and Yang Liu. 2022. Automatically Distilling Storyboard with Rich Features for Android Apps. *IEEE Transactions on Software Engineering* (2022).
[15] Sen Chen, Lingling Fan, Chunyang Chen, Ting Su, Wenhe Li, Yang Liu, and Lihua Xu. 2019. Storydroid: Automated generation of storyboard for Android apps. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 596–607.
[16] Sen Chen, Lingling Fan, Guozhu Meng, Ting Su, Minhui Xue, Yinxing Xue, Yang Liu, and Lihua Xu. 2020. An empirical assessment of security risks of global Android banking apps. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1310–1322.
[17] Sen Chen, Ting Su, Lingling Fan, Guozhu Meng, Minhui Xue, Yang Liu, and Lihua Xu. 2018. Are mobile banking apps secure? what can be improved?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 797–802.
[18] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. 2015. Iccta: Detecting inter-component privacy leaks in Android apps. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 280–291.
[19] Xian Zhan, Lingling Fan, Sen Chen, Feng We, Tianming Liu, Xiapu Luo, and Yang Liu. 2021. ATVhunter: Reliable Version Detection of Third-party Libraries for Vulnerability Identification in Android Applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1695–1707.
[20] Xian Zhan, Lingling Fan, Tianming Liu, Sen Chen, Li Li, Haoyu Wang, Yifei Xu, Xiapu Luo, and Yang Liu. 2020. Automated third-party library detection for Android applications: Are we there yet?. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 919–930.