JSLibD: Reliable and Heuristic Detection of Third-party Libraries in Miniapps

Junjie Tao
taojunjie@stu.xjtu.edu.cn
Xi'an Jiaotong University

Jifei Shi
sjf528@stu.xjtu.edu.cn
Xi'an Jiaotong University

Ming Fan*
mingfan@mail.xjtu.edu.cn
Xi'an Jiaotong University

Yin Wang
wy0724@stu.xjtu.edu.cn
Xi'an Jiaotong University

Junfeng Liu
liujunfeng@stu.xjtu.edu.cn
Xi'an Jiaotong University

Ting Liu
tingliu@mail.xjtu.edu.cn
Xi'an Jiaotong University

ABSTRACT

Miniapps have become an indispensable part of people's lives. Meanwhile, the utilization of third-party libraries greatly streamlines, expedites, and enhances the development of miniapps. However, ensuring the security of these third-party libraries presents a challenge, as they may harbor security vulnerabilities, such as plaintext transmission.

In this paper, we propose **JSLibD**, an automated extraction method for third-party libraries in miniapps. Unlike conventional extraction methods that heavily rely on prior knowledge, **JSLibD** introduces a heuristic prediction approach, comprising two integral components: a whitelist matching method to match the known libraries and a heuristic prediction method to extract the unknown libraries using function call relationships. The results demonstrate that **JSLibD** can efficiently match known libraries, and accurately predict unknown libraries, achieving an impressive precision rate of 85.9% and a high recall rate of 97.2%.

CCS CONCEPTS

• **Security and privacy → Web application security**; **Software security engineering**; • **Software and its engineering**;

KEYWORDS

Mobile Security, Miniapp, Third-party Library

ACM Reference Format:
Junjie Tao, Jifei Shi, Ming Fan, Yin Wang, Junfeng Liu, and Ting Liu. 2023. JSLibD: Reliable and Heuristic Detection of Third-party Libraries in Miniapps. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3605762.3624428

**Corresponding authors

SaTS 2023, November 26, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0258-7/23/11…$15.00
https://doi.org/10.1145/3605762.3624428

1 INTRODUCTION

With the exponential growth of super apps' users, such as WeChat, Alipay, and other app-carrying manufacturers, miniapps have garnered a substantial user base, leading to the accumulation of users' privacy information. According to relevant data, by the end of 2022, miniapps have exceeded 7.8 million, and DAU exceeded 800 million[11]. However, this rapid expansion has also exposed security risks in data collection, deletion, and transmission processes, primarily concerning data security issues[23].

The key parties responsible for miniapp information collection and utilization include the first and third parties. The first party, referring to the miniapp developers, collects and utilizes information with user consent to fulfill its functional requirements. And the third-party encompasses the third-party SDKs used in miniapp, such as AutoNavi SDK and advertising libraries. These third-party SDKs can share user's sensitive information with the first party while providing their services. Presently, numerous third-party resource databases, including CDNJS, Baidu Cloud, Alibaba Cloud, etc., cover diverse types of third-party databases, such as advertising, statistical analysis, maps, and payment services. However, ensuring the security of these third-party libraries presents a challenge, as they may harbor security vulnerabilities, such as plaintext transmission.

Meanwhile, there is no work that investigates the third-party libraries of miniapps yet. Therefore, in this paper, we propose a **JSLibD** method for automated detection and matching of third-party libraries for miniapps. Specifically, our method consists of three steps.

This paper conducts a manual analysis of 635 miniapps with privacy policies across seven platforms, including Baidu, Tiktok, JD, Taobao, Toutiao, WeChat, and Alipay. Among these, 310 miniapps employ third-party libraries, accounting for 8.8% of the total. In total, 972 third-party libraries are utilized, with an average of 1.53 third-party libraries being used per miniapp, while the Suning.com miniapp use up to 14 third-party libraries. Additionally, the study collects 24 commonly used third-party libraries for miniapps as shown is Table 1, yet ensuring the security of these third-party libraries proves challenging. A significant majority of these libraries lack security auditing, and many code segments contain security vulnerabilities, such as plaintext transmission, posing substantial challenges to the security of miniapps. Furthermore, in the vast majority of miniapps, locating privacy policy protection agreements for third-party libraries is difficult or impossible. Moreover, some third-party platforms may collect user application information,

account details, and device data without obtaining user consent, leading to uncontrollable privacy threats for both developers and application users.

Table 1: List of commonly used third-party libraries for miniapps

Number	Third-party Library Name	Third-party Library Description
1	Taro-ui	Taro UI Component
2	MD5.js	Encryption
3	Aes.js	Encryption
4	Qrcode.js	Generate QR Code
5	Base64.js	Base64 Encoding and Decoding
6	City.js	Provincial and Municipal Plugin
7	Mtj-wx-sdk.js	Accessing Baidu Statistics
8	Ald-stat.js	Aladdin Data Statistics
9	We-cropper.js	WeChat Image Cropping Tool
10	Wxcharts.js	Tencent Chart Service
11	Qqmap-wx-jssdk	Tencent Location Services
12	Amap-wx.js	Gaode Location Service
13	QiniuUploader	Qiniuyun Upload SDK
14	Common.js	Modular Canonical Function Set
15	Sha1.js	SHA-1 Encryption Algorithm
16	Moment.min.js	Date Processing Class Library
17	Weui	WeUI Component Library
18	Uni-ui	Cross-end UI Library of DCloud
19	Weapp	Lightweight UI Component
20	Colorui	Highly Attractive Miniapp
21	WxParse	WeChat Rich Text Parsing
22	Bmap-wx.min.js	Baidu Maps Service Interface
23	Verify_mpsdk	Tencent Cloud Facial SDK
24	Wepy	Wepy Development Framework

Firstly, the wxapkg file is decompiled into source codes. After that, the source codes will be parsed into AST structure. The second step is to extract features from these ASTs, and the function call graph (FCG) and file dependency graph (FDG) will be generated. Finally, the whitelist matching method will use the FCG to match the known libraries and the heuristic prediction method, based on FDG and the invoking rules, will be executed for the unmatched files to find the unknown libraries.

In summary, we make the following contributions:

- We collect a third-party library whitelist for miniapp, encompassing 852 commonly used third-party libraries, totaling 18,516 files.
- We are the first to introduce a heuristic third-party library extraction method, based on extensive measurement and observation of function call relationships in numerous miniapp samples, yielding three types of invoking rules for miniapps.
- We implement a prototype tool for third-party library extraction named **JSLibD**, integrating both whitelist matching and heuristic prediction. In a manually labeled dataset, this method achieves an precision of 85.9% and a recall rate of 97.2%.

2 BACKGROUND

2.1 Miniapp Reverse

Miniapp code consists of four types of files: JS logic files, WXML page files, WXSS style files, and JSON configuration files. Upon uploading, miniapps are packaged into a wxapkg file, akin to a compressed archive. Leveraging an offline package and web approach, users download the packaged wxapkg file (usually under 2MB) when accessing a miniapp, ensuring rapid and efficient usage. For extracting third-party libraries from a miniapp, the starting point is the source code comprising the aforementioned four file types. In this paper, we use the **wxappUnpacker**, an open source reverse tool, to extract Wechat miniapp source code.

2.2 Third-Party Libraries

Third-party libraries for miniapps are written in JavaScript and are hosted on platforms such as CDNJS, Baidu Cloud, Qiniu Cloud, and GitHub. These libraries enhance development efficiency when integrated by program developers. Due to the characteristics of miniapps, third-party libraries can be categorized as either single-file or multi-file libraries. Single-file libraries, like Baidu Map Services (bmap-wx.min.js) and Aladdin Data Statistics (ald-stat.js), consist of a single JS file. Multi-file libraries, on the other hand, comprise multiple JS files that are often organized within a specific directory structure. Notable examples include wxParse for rich text parsing and Tencent Cloud Face Recognition SDK verify_mpsdk.

3 METHOD

The workflow of **JSLibD** is shown in Figure 1. The **JSLibD** system takes wxapkg files as the input. Initially, the wxapkg file is decompiled into source codes using the wxappUnpacker tool. After that, the source codes will be parsed into AST structure using the esprima tool. After the feature extracting steps, the function call graph (FCG) and file dependency graph (FDG) will be generated. Then, the whitelist matching method will use the FCG to match the known libraries and the heuristic prediction method, based on FDG and the invoking rules, will be executed for the unmatched files to find the unknown libraries.

3.1 Whitelist Matching Method

The whitelist matching method operates on files that have undergone the feature extraction. For each file, Potential Known Libraries (PKL) will be selected through comparing the size of FCGs. Subsequently, employing graph similarity measurement between files and its PKLs, each file will be ascertained whether it is a known library.

To get the whitelist, this paper utilizes two distinct methods. Firstly, we crawls suitable JavaScript libraries for miniapps from various third-party static resource hosting platforms. Secondly, we employs a large-scale clustering approach [14, 16, 25] to extract frequently used third-party libraries from a dataset comprising over 100,000 miniapps.

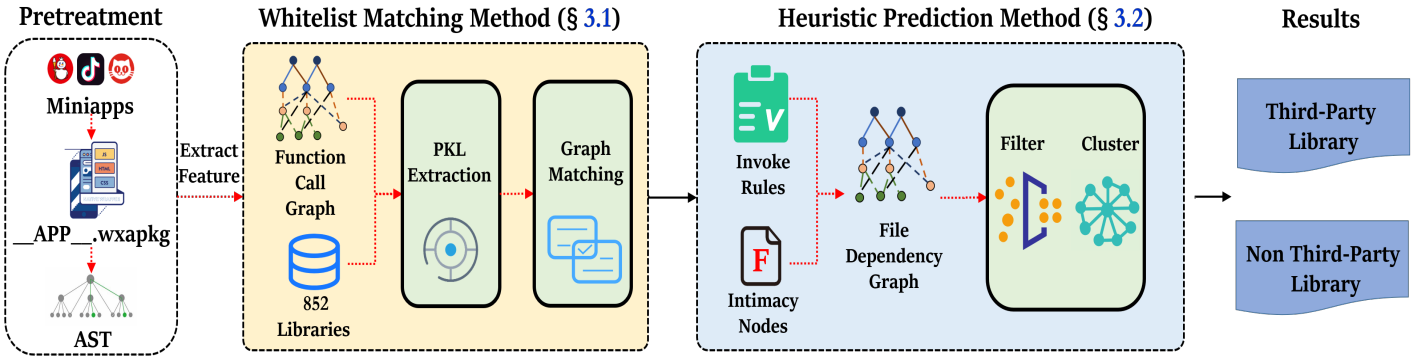


Figure 1: JSLibD Detection Workflow

3.2 Heuristic Prediction Method

Based on extensive measurements and observations of function call relationships in numerous miniapp samples, this paper identifies three types of invoking rules:

- Non-library files calling other non-library files.
- Non-library files calling library files.
- Library files calling other library files, where both the caller and callee belong to the same library type.

An example shown in Figure 4 is given to visualize these rules. There are three colorful calling relationships in the example complying with the invoking rules above. Specifically, the red relationship which represents pages.js calling utils.js follows the first rule. And the green relationship concurs with the second rule, while the yellow relationship aligns seamlessly with the third rule.

Based on these invoking rules, this paper proposed an innovative heuristic prediction method for third-party library extraction. The method introduces the concept of "intimacy" to measure the level of coupling between a JS file and the entire miniapp.

Definition 1. `intimacy`: *It is used to measure the degree of coupling between a JS file and the entire miniapp. JS files with high intimacy are code files written by miniapp developers themselves.*

Theoretically, by traversing the FDG, once nodes with excessively high intimacy are identified, all parent nodes can be considered as non-library file nodes, given that the invoking rules indicate that library files cannot invoke non-library files. Subsequently, after filtering out the non-library file nodes, the method clusters the left files belonging to the same library type using the third rule. Ultimately, the method outputs the unknown libraries within the miniapp.

To filter JS files authored by miniapp developers, this paper employs several methods to measure intimacy. For example, we can extract the function call graph at the miniapp level, where nodes with high coupling (e.g., the blue nodes in Figure 2 representing utility nodes within utils authored by miniapp developers, which have the same feature with libraries in the code level) become the target nodes for the first extraction method.

Furthermore, we can extract the APIs called by the JS files, then calculate the total weight based on preassigned weights for each API, and consider a node to have excessively high intimacy if its total weight exceeds the threshold. The extracted APIs mainly include page navigation, routing, and page interaction APIs, which are typically not used by third-party libraries. Detailed information on the extracted APIs is provided in Table 2.

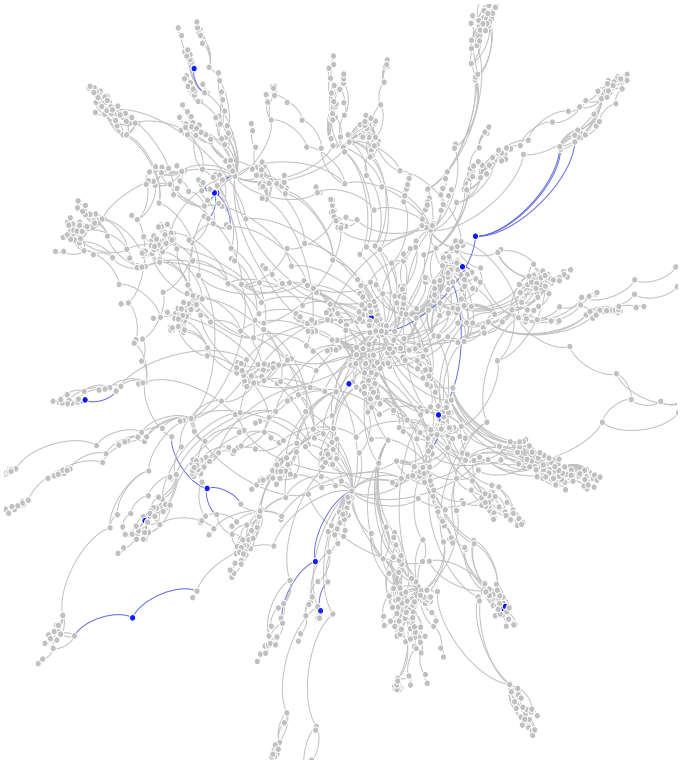


Figure 2: Miniapp Global Function Call Graph

4 EXPERIMENTAL RESULTS

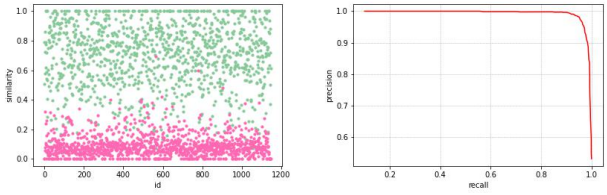
We collected 839 third-party libraries totaling 18,277 files to test the performance of the whitelist matching method. And the overall detection performance of **JSLibD** was tested by 35 WeChat miniapps with manually calibrated data. Finally, the case of FinLove miniapp is used to illustrate the anti confusion ability of the detection tool.

Table 2: API Information

Number	API	Weight
1	wx.switchTab	2
2	wx.reLaunch	2
3	wx.redirectTo	2
4	wx.navigateTo	2
5	wx.navigateBack	2
6	wx.showToast	1
7	wx.switchModal	1
8	wx.showLoading	1
9	wx.showActionSheet	1

4.1 WhiteList Fast Matching Method Results

To determine the optimal threshold for the proposed method, we selected 1,147 third-party library files and measured their similarity with files from different versions. Furthermore, random files from other libraries were selected and calculated their similarity with the above 1147 files.



(a) Scatter plot **(b) P-R curve**

Figure 3: Scatter plot and P-R curves of 1147 third-party library similarity measurement results

The results presented in Figure 3a illustrate the average similarity scores between different versions of the same library (green nodes) and between different libraries (red nodes). As observed from the plot, the similarity between files from the same library predominantly exceeds 0.4, while the similarity between files from different libraries mostly falls below 0.4. In addition, for matching methods, higher precision and recall are desirable, as depicted in the precision-recall (P-R) curve shown in Figure 3b, where the upper left corner represents the optimal performance. Therefore, we selected the threshold corresponding to the convex point of the P-R curve [21].

Furthermore, a total of 360 libraries with 5,967 files were selected for evaluating the performance. Subsequently, 100 JS file samples were chosen, with 50 samples present in the 5,967 files and the remaining 50 samples absent. The experimental results demonstrate that each sample file requires an average search of 22.69 third-party libraries and 68.07 files to obtain matching results, effectively achieving fast matching performance. In addition, the results indicate that out of the 100 samples, only one sample was falsely identified as a known library, resulting in an impressive accuracy of 99%.

4.2 Performance of Detection

The overall detection performance of the tool was assessed by combining the tool's results with manual analysis of 35 WeChat miniapps, comprising a total of 5,556 files. The results are summarized in Table 3. We used the confusion matrix method to evaluate the results and the performance yielded 588 true positive files, 96 false positive files, and 17 false negative files, which indicates **JSLibD** have an precision of 85.9% and a recall rate of 97.2%.

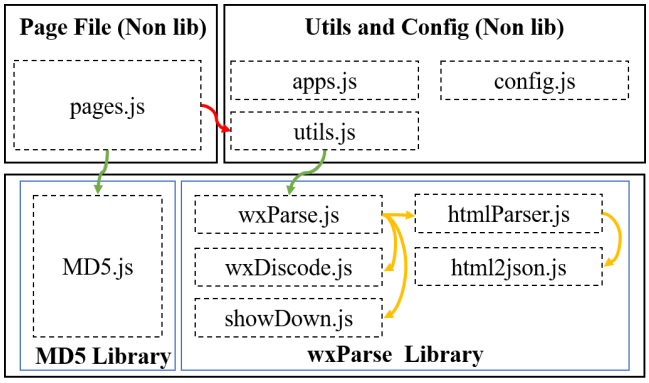


Figure 4: Code structure of FinLove miniapp

4.3 Cases of Anti Confusion Ability

To verify the actual effectiveness of **JSLibD**, we selected the FinLove miniapp for analysis and testing. Its code structure is shown in Figure 4. It can be found that the miniapp consists of page files and some configuration files, which are non third-party libraries, as well as MD5 files and wxParse files, which are third-party libraries. After manual analysis, the corresponding situation of the third-party library source files and their confusion version in the miniapp is shown in the Table 4, where '\' indicates that the file is a non third-party library and '✓' indicates that the tool is able to detect obfuscated third-party library files.

From Table 4, it can be seen that all third-party library files within the FinLove miniapp were successfully obtained, which indicate that **JSLibD** can effectively extract third-party libraries for miniapps and has certain anti confusion ability.

5 DISCUSSION

This paper acknowledges certain limitations of **JSLibD** while proposing potential solutions:

Limited Detection of Front-end UI Libraries. JSLibD focuses solely on detecting back-end JavaScript logic code libraries and does not account for front-end UI libraries. As third-party libraries may introduce security risks due to the lack of security auditing, **JSLibD's** main goal is to produce outputs that can be utilized by other privacy and security analysis tools to assess library security risks comprehensively. However, front-end UI libraries typically do not involve sensitive data, leading to their exclusion during the initial design of **JSLibD**. Future research could involve a detailed analysis of front-end UI libraries to incorporate a separate module within **JSLibD**, specifically designed to detect and analyze them.

Table 3: Overall Tool Detection Performance

Miniapp name	TP	TN	FP	FN
CMC Cinema	6	123	1	0
Instabolt flash charge	0	25	1	0
OIAM plays big names	0	6	0	0
Mom and Dad camp good things	0	478	0	0
Class Gang	24	284	1	0
Play with you	2	139	0	0
Check out the QR code	126	384	5	2
Wheel driver's license pass	3	126	0	0
The car has material	4	145	0	0
Do not disturb	9	6	1	0
Hi repair mobile phone platform	10	90	1	1
Dada exclusively	23	179	7	0
State Department client	36	615	0	0
Wanting to fall in love	6	73	1	0
Mango TV video	0	51	0	0
Qiaqia Food Lite	0	442	0	0
QQ Music	1	62	2	0
Networking business cards	5	34	0	0
Oil price map	2	25	0	0
Suning.com	41	525	4	0
Play card OneWeek	20	98	9	0
Little Red Book APP	49	116	6	3
Chengdu court service platform	1	18	0	0
Automatically generate sheets	15	129	1	0
GitHub applets	2	13	3	0
pModify the map	21	22	0	2
Aika Cars	6	31	0	0
U guest cloud	11	69	2	0
Lucky little weather	0	3	4	0
Baby Bus Kids Bedtime Story	42	56	23	0
Open volume audio subscription	6	61	1	0
Yuanzhou decoration store	81	317	11	8
Love Xianyang	1	45	2	0
Xi'an Policy Pass	19	9	5	0
High-tech office	16	56	5	1

Dependency Graph Reliance on Path Information. The process of generating the file dependency graph heavily relies on path information, which may become erroneous when decompiling project source code from wxapkg files. A complete and accurate file dependency graph is essential for the Heuristic Prediction Method, as any missing dependencies can lead to numerous false positives. To address this issue, two potential solutions are proposed: Firstly, the development of a custom decompilation tool that ensures the accuracy of path information during the decompilation process, ensuring reliable file dependency graphs. Secondly, a modification of the current method employed by **JSLibD** to generate file dependency graphs, making it less sensitive to path information and capable of producing complete and accurate dependency graphs. The feasibility and effectiveness of these approaches require further investigation.

6 RELATED WORK

Static Analysis of Super Apps. One method that cannot be separated from the analysis of miniapps is the static analysis technology for super apps, which involves analyzing the source code without executing it. Static analysis was widely used to find security and privacy issues[13]. For example, some approaches were developed based on static analysis techniques to detect privacy leaks[1, 8, 12, 18, 19, 24] and detect malicious code[4, 6, 7, 9, 20].

Third-party Library Detection of Miniapps. The existing research on third-party library detection and extraction methods has predominantly focused on Android apps, leaving a notable gap in the context of miniapps. In the domain of Android apps, two primary streams of approaches for detecting third-party libraries can be identified.

The first approach entails constructing a whitelist of known libraries and subsequently performing similarity matching. Chen et al.[5] compiled a collection of 73 commonly used third-party libraries to detect cloned apps. Similarly, Grace et al.[10] and Book et al.[3] curated lists of commonly used advertising libraries and employed package name matching to detect third-party libraries in app programs. To improve resistance to obfuscation, many detection tools generate robust feature values for the known libraries and then utilize feature matching or similarity comparison methods to identify third-party libraries. For instance, LibRadar[16] and Wukong [22] extracted system-level APIs from each library to serve as feature values. Despite exhibiting a certain level of obfuscation resistance and enhancing detection robustness, these methods still suffer from a critical drawback of being unable to identify third-party libraries not included in the whitelist.

The second approach is based on machine learning techniques and can be divided into two categories. The first category involves feature-based classification methods. Tools like PeDal [15] and AdDetect [17] extract permission and API features to train classifiers capable of distinguishing advertising libraries from non-advertising libraries. Additionally, there are clustering-based third-party library detection tools, including LibRadar[16], LibD [14], and LibExtractor [25]. These tools utilize large-scale app datasets, often comprising millions of apps, to group third-party libraries with similar features together and then apply a defined threshold to determine their status as third-party libraries. However, some of these clustering methods will use large-scale Apps as input to generate enough feature signatures, while others will discover incomplete third-party libraries. Moreover, the package names or structures that rely on clustering methods are easily confused by existing obfuscators[2].

7 CONCLUSION

This paper successfully designed and implemented **JSLibD**, a robust third-party library extraction method for miniapps. Comprising the WhiteList Fast Matching Method and the Heuristic Prediction Method, **JSLibD** provides an effective solution for identifying third-party libraries within miniapps. The evaluation of **JSLibD** was carried out on 35 WeChat miniapps, encompassing a total of 5,556 files, and the results showed **JSLibD** have an precision of 85.9% and a recall rate of 97.2%.

Table 4: The file connection and tool detection of third-party library source files and code confusion

Miniapp Category	Miniapp Code File	Library Source File	Detection Result
Pages File	\	\	\
Utils and Configs	\	\	\
MD5 Library	OAFDED1185C4F1DE6C9B85166B2B3782.js	MD5.js	✓
wxParse Library	wxParse.js	wxParse.is	✓
	E9B9EF8285C4F1DF8FDF87859F0B3782.js	wxDiscode.is	✓
	D261394285C4F1DEB4075145CC1B3782.js	showDown.js	✓
	C91EB0B585C4F1DEAF79D8B2610B3782.js	htmlParser.js	✓
	4278396785C4E1DE241E516010EA3782.js	html2json.js	✓

ACKNOWLEDGEMENTS

This work was supported by National Key R&D Program of China (2022YFB2703500), National Natural Science Foundation of China (62232014, 62272377, 62293501, 62293502, 72241433, 61721002, 62032010, 62002280), CCF-AFSG Research Fund , China Postdoctoral Science Foundation (2020M683507, 2019TQ0251, 2020M673439), and Young Talent Fund of Association for Science and Technology in Shaanxi, China.

REFERENCES

[1] 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.

[2] Arini Balakrishnan and Chloe Schulze. 2005. Code obfuscation literature survey. *CS701 Construction of compilers* 19 (2005), 31.

[3] Theodore Book, Adam Pridgen, and Dan S Wallach. 2013. Longitudinal analysis of android ad library permissions. *arXiv preprint arXiv:1303.0857* (2013).

[4] David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. 2008. Automatically identifying trigger-based behavior in malware. *Botnet Detection: Countering the Largest Security Threat* (2008), 65–88.

[5] Kai Chen, Peng Liu, and Yingjun Zhang. 2014. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In *Proceedings of the 36th International Conference on Software Engineering*. 175–186.

[6] Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, and Ting Liu. 2018. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Transactions on Information Forensics and Security* 13, 8 (2018), 1890–1905.

[7] Ming Fan, Xiapu Luo, Jun Liu, Meng Wang, Chunyin Nong, Qinghua Zheng, and Ting Liu. 2019. Graph embedding based familial analysis of android malware using unsupervised learning. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 771–782.

[8] Ming Fan, Le Yu, Sen Chen, Hao Zhou, Xiapu Luo, Shuyue Li, Yang Liu, Jun Liu, and Ting Liu. 2020. An empirical evaluation of GDPR compliance violations in Android mHealth apps. In *2020 IEEE 31st international symposium on software reliability engineering (ISSRE)*. IEEE, 253–264.

[9] Yanick Fratantonio, Antonio Bianchi, William Robertson, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2016. Triggerscope: Towards detecting logic bombs in android applications. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 377–396.

[10] Michael C Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. 2012. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. 101–112.

[11] Aladdin Institute. 2023. 2022 White Paper on the Internet Development of Miniapps. [Online]. https://www.aldzs.com/viewpointarticle?id=16573/ Last accessed on 2023-08-09.

[12] 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.

[13] Li Li, Tegawendé F Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Le Traon. 2017. Static analysis of android apps: A systematic literature review. *Information and Software Technology* 88 (2017), 67–95.

[14] Menghao Li, Wei Wang, Pei Wang, Shuai Wang, Dinghao Wu, Jian Liu, Rui Xue, and Wei Huo. 2017. Libd: Scalable and precise third-party library detection in android markets. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 335–346.

[15] Bin Liu, Bin Liu, Hongxia Jin, and Ramesh Govindan. 2015. Efficient privilege de-escalation for ad libraries in mobile apps. In *Proceedings of the 13th annual international conference on mobile systems, applications, and services*. 89–103.

[16] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. 2016. Libradar: Fast and accurate detection of third-party libraries in android apps. In *Proceedings of the 38th international conference on software engineering companion*. 653–656.

[17] Annamalai Narayanan, Lihui Chen, and Chee Keong Chan. 2014. Addetect: Automated detection of android ad libraries using semantic analysis. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 1–6.

[18] Jordan Samhi, Alexandre Bartel, Tegawendé F Bissyandé, and Jacques Klein. 2021. Raicc: Revealing atypical inter-component communication in android apps. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1398–1409.

[19] Jordan Samhi, Jun Gao, Nadia Daoudi, Pierre Graux, Henri Hoyez, Xiaoyu Sun, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. 2022. Jucify: A step towards android code unification for enhanced static analysis. In *Proceedings of the 44th International Conference on Software Engineering*. 1232–1244.

[20] Jordan Samhi, Li Li, Tegawendé F Bissyandé, and Jacques Klein. 2022. Difuzer: Uncovering suspicious hidden sensitive operations in android apps. In *Proceedings of the 44th International Conference on Software Engineering*. 723–735.

[21] Helen R Sofaer, Jennifer A Hoeting, and Catherine S Jarnevich. 2019. The area under the precision-recall curve as a performance metric for rare binary events. *Methods in Ecology and Evolution* 10, 4 (2019), 565–577.

[22] Haoyu Wang, Yao Guo, Ziang Ma, and Xiangqun Chen. 2015. Wukong: A scalable and accurate two-phase approach to android app clone detection. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 71–82.

[23] Yin Wang, Ming Fan, Junfeng Liu, Junjie Tao, Wuxia Jin, Qi Xiong, Yuhao Liu, Qinghua Zheng, and Ting Liu. 2023. Do as You Say: Consistency Detection of Data Practice in Program Code and Privacy Policy in Mini-App. *arXiv preprint arXiv:2302.13860* (2023).

[24] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. 2018. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. *ACM Transactions on Privacy and Security (TOPS)* 21, 3 (2018), 1–32.

[25] Zicheng Zhang, Wenrui Diao, Chengyu Hu, Shanqing Guo, Chaoshun Zuo, and Li Li. 2020. An empirical study of potentially malicious third-party libraries in android apps. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 144–154.