MUID: Detecting Sensitive User Inputs in Miniapp Ecosystems

Ziqiang Yan
hanshan@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

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

Haoran Wang
wanghaoran@stu.xjtu.edu.cn
Xi'an Jiaotong University

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

ABSTRACT

In recent years, the rise of miniapps, lightweight applications based on WebView, has become a prominent trend in mobile app development. This trend has rapidly expanded on popular social platforms like WeChat, TikTok, Grab, and even Snapchat. In these miniapps, user data is pivotal for providing personalized services and improving user experience. However, there are still shortcomings in identifying the source of sensitive data in miniapps. This paper introduces MUID, an innovative method for detecting user input data in miniapps. MUID integrates an engine that can dynamically test miniapps to overcome the challenges in WebView page extraction, uses a hybrid analysis approach to identify sensitive components, and infers the type of information collected based on contextual hint words. In the evaluation of MUID across 30 popular miniapps randomly selected on WeChat, we demonstrated its high dynamic testing efficiency and its capability to recognize components with a recall rate of 95.74% and a precision rate of 81.32%. The overall precision of MUID is 78.31%, and the recall rate is 92.19%, demonstrating the effectiveness of MUID in conducting security and privacy analyses.

CCS CONCEPTS

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

KEYWORDS

Miniapp security, User input, Mobile security, Privacy

ACM Reference Format:
Ziqiang Yan, Ming Fan, Yin Wang, Jifei Shi, Haoran Wang, and Ting Liu. 2023. MUID: Detecting Sensitive User Inputs in Miniapp Ecosystems. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3605762.3624429

*Corresponding author

SaTS '23, November 26, 2023, Copenhagen, Denmark.

ACM ISBN 979-8-4007-0258-7/23/11…$15.00
https://doi.org/10.1145/3605762.3624429

1 INTRODUCTION

Recent years have marked an intriguing trend within the realm of mobile application development: the rapid emergence of miniapps. These are lightweight applications, grounded in WebView technology, and have become increasingly prevalent across widely used social platforms like WeChat, TikTok, Grab, and Snapchat. These miniapps, which can operate directly within their host applications without separate downloads[26], offer users a range of convenient services[25]. By doing so, They have expanded the functionality of superapps, further evolving them into comprehensive "operating systems" designed to cater to a diverse spectrum of user needs[19]. This miniapp paradigm has garnered substantial global traction. A case in point is WeChat, a pioneer in this space, which reportedly hosts over 4.3 million miniapps[10], spanning services from social to e-commerce, thereby significantly simplifying mobile user experiences.

User data plays an instrumental role in delivering personalized services and enhancing user experiences within these miniapps. The management and safeguarding of such data present significant concerns[6–8]. Prior research has scrutinized the sensitive APIs offered by superapps. For example, Wang et al. [20] identified potential misuse of specific APIs that lacked sufficient security evaluation, and developed APIScope to identify these overlooked APIs. Additionally, Zhang et al. [25] found an insufficient verification process for privileged APIs, often leading to a more extensive range of API permissions than originally anticipated. Nevertheless, these methodologies primarily concentrate on sensitive data derived from API calls, neglecting user-input data in the graphical user interface (GUI), such as addresses and phone numbers. This may lead to a lack of sensitive data leakage detection at the source.

Therefore, effective detection of user-input data is of paramount importance for maintaining user privacy and augmenting the user experience in miniapps. However, this presents a formidable challenge. The lightweight characteristics of miniapps, combined with their unique operational environments, demand specialized data detection techniques divergent from those employed in traditional mobile applications. Previous research [1, 11, 15, 16, 21] fail to handle the dynamic page views predicated on WebView, underlining the necessity for dedicated research focusing on user-input data detection within the realm of miniapps.

In this paper, we present a novel miniapp user-input detection system, MUID, proficient in autonomously analyzing a miniapp's GUI to parse user-input data. MUID utilizes innovative techniques across three primary stages, advancing the state of the art in dynamic execution, component recognition, and semantic parsing. Initially, in the dynamic execution stage, MUID performs testing

on the miniapp, deriving dynamic page layouts. Subsequently, in component recognition stage, MUID employs hybrid analysis to accurately identify sensitive input components. Following this, in semantic parsing stage, MUID infers the type of data collected based on contextual hint words. In summary, the primary contributions of this paper include:

- We develop the first tool specifically engineered for user-input detection in miniapps.
- We propose a novel method for component recognition in WebView pages, which is a critical challenge not tackled in prior research.
- We conduct a empirical assessment of MUID on a comprehensive set of miniapps, with a recall rate of 92.19% and a precision rate of 78.31%, thereby demonstrating its efficacy in performing security and privacy analysis.

2 MOTIVATING EXAMPLE

As shown in Figure 1, we provide an example of user-input on a dynamic page predicated on WebView. In this context, users are required to provide specific details, such as their work email and full address, to complete the enterprise user application. However, the corresponding terms of service do not explicitly enumerate the necessary information: *"When registering a member account and using member services, you should furnish your information accurately and comprehensively in accordance with the prompts on the page."* This vagueness, in conjunction with the 'hot update' feature of webview (allowing developers to alter content within the WebView page without modifying the client source code), equips applications with the ability to modify the data they collect at any point, circumventing platform scrutiny. This practice presents potential threats to user privacy, considering the highly sensitive nature of such data and the grave consequences that could result from improper exposure.

Miniapp UI rendering mechanism: miniapps utilize a rendering mechanism predicated on webview, with the objective of harmonizing the rapid development capacities of web technology and a near-native, immersive user experience. For instance, WeChat miniapps on Android employ the proprietary XWeb engine, derived from the Mobile Chromium kernel, for rendering purposes, while iOS platforms employ WK-WebView for the identical task. WebView is a component deployed for integrating web content within mobile applications, functioning as a display container for web content. As depicted in the component tree on the right side of Figure 1, all miniapp pages are housed within the Webview component. The displayed content, hosted on a remote server, is beyond the complete scope of static analysis, which incorporates reverse parsing of source files, as the WebView components necessitate real-time acquisition.

3 SYSTEM DESIGN

In this section, we delve into the intricacies of MUID's design and the pivotal technologies it employs. The operation of MUID is primarily divided into three essential stages. The first stage employs a dynamic execution engine designed to capture webpage layout information. Following this, during the component recognition stage, the procured page layouts, which comprise both webpage

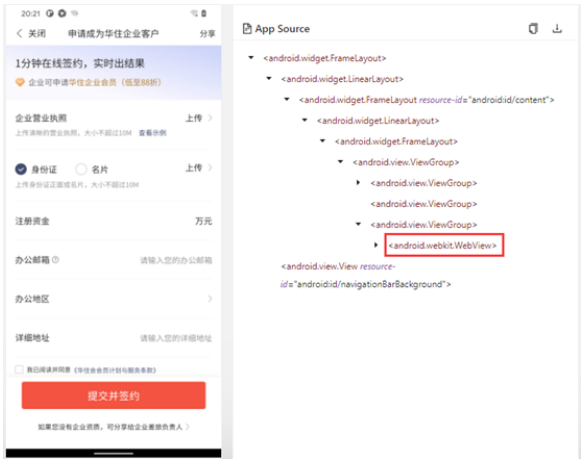

Figure 1: Example of User Input in WebView Page

screenshots and XML layout files, are meticulously analyzed to identify specific sensitive components. Lastly, the semantic parsing stage infers the types of information gathered. It can simultaneously match and bind hint words to the corresponding input components as text is dynamically loaded.

3.1 Dynamic Execution

As the smallest granular component within Android native pages, the WebView component is inseparable in its entirety when the UiAutomator framework [9] is employed for information acquisition and native page modeling, nor can it perceive its internal information. Therefore, to implement a dynamic execution engine, We modified UICrawler[12], an Appium-based testing framework. Through synchronizing the Chromedriver version with the mobile phone's Chrome kernel version, we establish communication with the Chrome kernel via the ChromeDevtoolProtocol, thereby facilitating the acquisition of page structure. Yet, as its original form is a webpage, its GUI hierarchy and component types may not entirely correspond with the native interface. This discrepancy might result in inconsistencies during Appium's component transformation[2]. For example, * * could be converted into *<view>*, which miss the editable attritute. We proposed a solution to this issue in Section 3.2.

Once the page structure is acquired, Appium plays a crucial role in constructing the Document Object Model (DOM) tree. The GUI model is then established with the component as the smallest granularity, and testing is implemented using a Monkey strategy. Concurrently, we employ text similarity to evaluate whether the page has undergone a status change. In a bid to augment testing coverage, we allocate greater weights to components that can potentially trigger page changes, thereby amplifying their likelihood of selection in succeeding tests.

3.2 Component Recognition

Adhering to Google's official definitions of components, we have identified components that can receive user input, such as EditText,

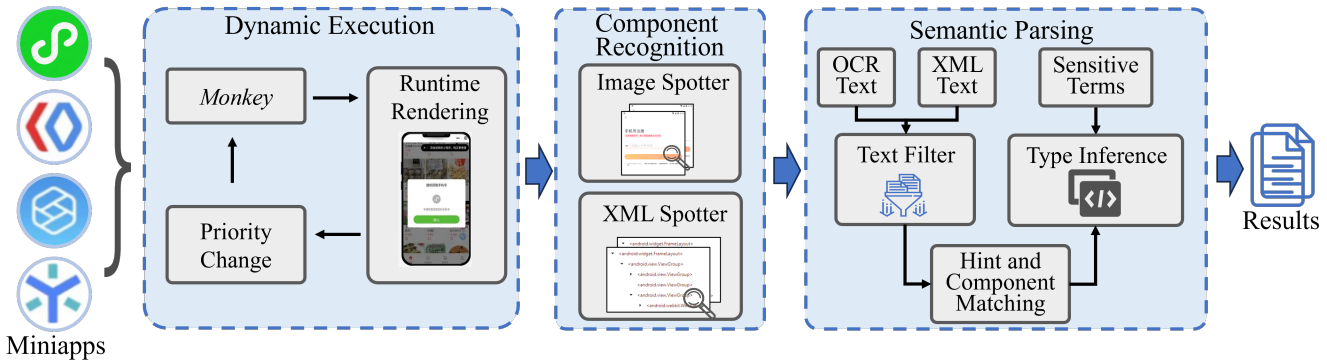

Figure 2: System Overview of MUID

Algorithm 1: Sensitive Components Recognition

```
Input: sen_components = init_list ()
Output: sen_components    //Sensitive Components
components = GenCompFromXml (xml)
supplementary_sen_components =
  GenSenCompFromImg (image)
foreach sup_comp ∈ supplementary_sen_components do
    foreach comp ∈ components do
        if match (sup_comp, comp) and comp.type == View then
            comp.type = sup_comp.type
            break
foreach comp ∈ components do
    if comp.type in sensitive_types then
        sen_components.add (comp)
return sen_components
```

AbsSpinner, CheckedTextView, RadioButton, CheckBox, ToggleButton, Switch, SwitchCompat, and RatingBar, to act as our sensitive components. However, certain inconsistencies may arise when converting components from Web to native format, thereby posing a challenge to the accurate identification of sensitive components. To counter this problem, our component identification engine amalgamates page screenshots with page layout files for recognition. For a detailed examination of the process, refer to Algorithm 1.

Initially, the function *GenCompFromXml* constructs the DOM from the page layout XML file, thus generating a set of components. Subsequently, the function *GenSenCompFromImg* employs a target recognition algorithm to identify specified components from the provided page screenshot, thereby generating a supplementary set of sensitive components. For the purpose of this paper, we utilize the pre-trained model yolov7[23]with training data sourced from rico[3]. Following this, we iterate through the supplementary sensitive component set, cross-referencing it with the original component set. Upon finding a corresponding component that fulfills specific criteria (i.e., position can match and component type is View), we modify the component type to match the corresponding supplementary sensitive component type. Finally, we iterate through the amended component set, incorporating components identified as sensitive components into *sen_components*, which is subsequently returned as the final target value.

3.3 Semantic Parsing

Text Filtering: During the component conversion process facilitated by Appium, some components' text attributes might be lost, potentially leading to oversights if hint words are exclusively sourced from the page layout file. To rectify this, we utilize Optical Character Recognition (OCR) to supplement the text information. However, since OCR recognizes all the text on the page, it significantly escalates the amount of candidate text requiring processing. Hence, we introduce a pre-trained Bert model to filter out irrelevant or noise text. Considering that page text is typically short, we choose to employ ChatGPT to generate relevant training data, which is classified into two categories. The first category includes text featuring sensitive information types and collection intent, such as *"please input your telephone"*. The second category encompasses text devoid of specific sensitive information types, such as *"my service"*, and phrases that, although containing sensitive information types, do not intend to collect them, as in *"under-screen fingerprint mobile phone"* (where the sensitive information type serves as a noun modifying another noun). To enhance the quality of the training data further, we integrate application category restrictions into the ChatGPT prompts, such as *"Please give me 5 hint words for input boxes containing sensitive information types, which may appear in financial category applications"*. Emphasizing categories is because different application categories collect varying ranges of sensitive information. Following training on these two types of data, we derive a binary classifier based on Bert. We then retain only those texts from the OCR-recognized and page layout file that contain sensitive information types and a collection intent as candidate text.

Hint Words and Sensitive Component Matching: Text recognized by OCR includes position coordinates, although the units of measurement might not correspond with the position coordinates of sensitive components in the page layout. Hence, the normalization of these two types of position coordinates becomes our initial task. From the experience, hint words typically appear on the left, above, or encapsulated within sensitive components. Consequently, a mere reliance on the computation of centroid angle relations to ascertain positional relationships proves inadequate. In such instances, guided by the aforementioned experiential knowledge of positional relationships, we initially generate a set of candidate hint words for each sensitive component. Subsequently, we compute the distance between these hint words and the centroid of the sensitive

component, with a preference for matching the hint word that is nearest in terms of distance.

Information Type Inference: In compliance with relevant privacy protection regulations (i.e. GDPR[4] and PIPL[17]), we assemble a two-level sensitive information type dictionary[22]. The first level represents the broad category of the information type, including 13, such as health information, device information, and identity information. The second level clearly indicates the specific information collected, with a total of 80 items. For example, for health information, the second level information will include hospital records, medical records, weight, height, etc. Each specific information has a considerable number of instance forms extracted from real-world code and GUI. For hostalization records, there are forms such as doctor Id, doctor name, admission time, description of the condition, etc. We use Spacy[5] to calculate the semantic similarity between the hint word and all instance forms of the second level information type, then calculate the average value, and select the second level information type with the highest average similarity as the pertinent information type. This two-level information type structure will facilitate subsequent analyses, such as assessing whether the collected information contravenes privacy protection regulations or the application's privacy policy.

4 EVALUATION

In this section, we evaluate the effectiveness of the key modules within MUID. To extract the page layout of miniapps, we ran the dynamic execution of MUID on a Google Pixel4, operating on Android 11. We chose WeChat miniapps as our test samples,given their dominant presence in the field of miniapps, with the WeChat version being 8.0.34. We randomly selected 30 miniapps based on industry and popularity as our samples.

4.1 Dynamic Execution Performance

For the dynamic execution engine, we made a comparative analysis with FastBot[14]. Given that miniapps do not incorporate a distinct concept of Activity, traditional dynamic testing standards, which are guided by Activity coverage, prove inadequate. We propose to evaluate the test depth by quantifying the number of components covered, that is, the sum of components of all pages obtained by MUID running for a period of time. Note that components on similar pages cannot be counted repeatedly. Similarity between pages is assessed by determining whether the text similarity of their layout files exceeds a certain threshold. MUID, on average of executing a 10-minute run, is able to cover 1391.7 components in this time frame, which is 519.6 more than FastBot. This superior performance can be attributed to the fact that FastBot is not optimized for miniapps, leading to frequent exits from the miniapp interface. For the user input pages obtained by the above 30 mini programs, manually mark sensitive component information, hint words, and information types, totaling 56 pages and 141 sensitive components.

4.2 User Input Detection Performance

Sensitive component recognition is the foundation for subsequent analysis of user input detection. We assessed the performance of MUID in recognizing sensitive components within the 56 user input pages acquired as mentioned above. We have categorized these 56 pages into five distinct categories according to the purpose of information collection: travel, login, express, finance, and authentication (auth.). We consider the sensitive components that are consistent with manual labeling as true positives, and those that are inconsistent as false positives. Sensitive components missed in manual labeling are regarded as false negatives. Notably, the concept of true negatives is inapplicable in this context. Table 1 shows the recognition results of MUID's sensitive components, with the highest proportion of login pages (53.57%), an precision rate of 84.84%, a recall rate of 98.24%, and a total precision rate of 81.32%, with a recall rate of 95.74%.

We further evaluated the comprehensive performance of MUID subsequent to component recognition stage and semantic parsing stage. Our standard for measuring overall performance is to detect whether sensitive component information and information type are the same as those manually assigned labeled. If they are consistent, they are considered true positive, otherwise they are considered false positive. The rules for true and false negatives remain consistent with those established for component recognition. Table 1 shows the overall results, with a MUID final precision of 78.31%, a recall rate of 92.19%, and an F1-measure of 84.68%.

The main factor affecting the precision of MUID is the insufficient text filtering during the semantic parsing stage, which leads to some non sensitive prompt words being considered sensitive prompt words. The training data is generated from chatgpt, but this may differ from real-world data.

5 RELATED WORK

Miniapp Security: A multitude of studies have probed into the security facets of miniapps. Yang et al. [24] formulated CmrfScanner to ascertain if there is an absence of appid checks in cross-miniapp requests. Lu et al. [13] studied the resource isolation model of the host App for miniapps and identified some flaws that could potentially leak user privacy. Wang et al. [18] devised TAINTMINI to track tainted data flows in miniapps. Unlike these studies, our primary focus lies in identifying the user-input in miniapps.

User Input Detection: Nan et al. [15] identified user input by training classifiers on static layout resources and code semantic information. Huang et al. [11] introduced SUPOR, a system that procures layout files with coordinates by modifying the ADT static rendering engine, and pairs this with taint analysis technology to construct a privacy leakage discovery system. Andow et al. [1] developed UiRef that obtains page layout by dynamically traversing all Activities in APK. Wang et al. [21] proposed GUILeak, an instrument that discerns whether privacy leakage of user input data breaches the application's privacy policy. However, these tools, which primarily target Android applications, exhibit shortcomings in the sphere of miniapps, as they fall short in analyzing WebView components. Contrarily, the method proposed in this paper is capable of detecting user input in miniapps.

6 CONCLUSION

In this paper, we introduce MUID, an innovative approach for detecting user input data in miniapps. MUID circumvents the prevailing challenges related to WebView analysis by creating a novel

Table 1: User Input Detection Results

	Pages	Components	Prec.(CR)	Rec.(CR)	F1(CR)	Prec.(O)	Rec.(O)	F1(O)
travel	5	19	85%	89.47%	87.17%	85%	89.47%	87.17%
login	30	57	84.84%	98.24%	91.04%	81.81%	94.73%	87.79%
express	12	36	75%	100%	85.71%	70.83%	94.44%	80.94%
finance	6	23	84.61%	95.65%	89.79%	80.76%	91.3%	85.71%
auth.	3	6	66.66%	66.66%	66.66%	66.66%	66.66%	66.66%
sum	56	141	81.32%	95.74%	87.94%	78.31%	92.19%	84.68%

[1] CR refers to MUID's Component Recognition results
[2] O refers to MUID's overall results

GUI model to conduct dynamic testing. It utilizes a hybrid analysis strategy to pinpoint sensitive components and infers the type of information collected based on contextual cues and prompt words. Evaluations conducted on 30 miniapps indicate that MUID can achieve a recall rate of 92.19% and a precision rate of 78.31%.

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] Benjamin Andow, Akhil Acharya, Dengfeng Li, William Enck, Kapil Singh, and Tao Xie. 2017. Uiref: analysis of sensitive user inputs in android applications. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 23–34.
[2] appium. 2023. https://github.com/appium/appium. https://github.com/appium/appium
[3] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*. 845–854.
[4] European Parliament and Council of the European Union. [n. d.]. *Regulation (EU) 2016/679 of the European Parliament and of the Council*. https://data.europa.eu/eli/reg/2016/679/oj
[5] explosion. 2023. spaCy: Industrial-strength NLP. https://github.com/explosion/spaCy
[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] Google. 2023. Write automated tests with UI Automator | Android Developers. https://developer.android.com/training/testing/other-components/ui-automator
[10] THOMAS GRAZIANI. [n. d.]. What are WeChat Mini-Programs? A Simple Introduction - WalktheChat. https://walkthechat.com/wechat-mini-programs-simple-introduction/
[11] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. 2015. {SUPOR}: Precise and scalable sensitive user input detection for android apps. In *24th USENIX Security Symposium (USENIX Security 15)*. 977–992.
[12] lgxqf. 2021. https://github.com/lgxqf/UICrawler. https://github.com/lgxqf/UICrawler
[13] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. 2020. Demystifying resource management risks in emerging mobile app-in-app ecosystems. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security*. 569–585.
[14] Zhengwei Lv, Chao Peng, Zhao Zhang, Ting Su, Kai Liu, and Ping Yang. 2022. Fastbot2: Reusable Automated Model-based GUI Testing for Android Enhanced by Reinforcement Learning. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–5.
[15] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. 2015. {UIPicker}:{User-Input} Privacy Identification in Mobile Applications. In *24th USENIX Security Symposium (USENIX Security 15)*. 993–1008.
[16] Yuhong Nan, Zhemin Yang, Min Yang, Shunfan Zhou, Yuan Zhang, Guofei Gu, Xiaofeng Wang, and Limin Sun. 2016. Identifying user-input privacy in mobile applications at a large scale. *IEEE Transactions on Information Forensics and Security* 12, 3 (2016), 647–661.
[17] Standing Committee of the National People's Congress. 2021. Personal Information Protection Law of the People's Republic of China. http://www.npc.gov.cn/npc/c30834/202108/a8c4e3672c74491a80b53a172bb753fe.shtml
[18] Chao Wang, Ronny Ko, Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Taintmini: Detecting Flow of Sensitive Data in Mini-Programs with Static Taint Analysis. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 932–944.
[19] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. One Size Does Not Fit All: Uncovering and Exploiting Cross Platform Discrepant {APIs} in {WeChat}. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6629–6646.
[20] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. Uncovering and Exploiting Hidden APIs in Mobile Super Apps. *arXiv preprint arXiv:2306.08134* (2023).
[21] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D Breaux, and Jianwei Niu. 2018. Guileak: Tracing privacy policy claims on user input data for android applications. In *Proceedings of the 40th International Conference on Software Engineering*. 37–47.
[22] 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).
[23] WongKinYiu. 2022. https://github.com/WongKinYiu/yolov7. https://github.com/WongKinYiu/yolov7
[24] Yuqing Yang, Yue Zhang, and Zhiqiang Lin. 2022. Cross miniapp request forgery: Root causes, attacks, and vulnerability detection. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3079–3092.
[25] Lei Zhang, Zhibo Zhang, Ancong Liu, Yinzhi Cao, Xiaohan Zhang, Yanjun Chen, Yuan Zhang, Guangliang Yang, and Min Yang. 2022. Identity confusion in {WebView-based} mobile app-in-app ecosystems. In *31st USENIX Security Symposium (USENIX Security 22)*. 1597–1613.
[26] Yue Zhang, Bayan Turkistani, Allen Yuqing Yang, Chaoshun Zuo, and Zhiqiang Lin. 2021. A measurement study of wechat mini-apps. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 2 (2021), 1–25.