Understanding and Identifying Cross-platform UI Framework based Potentially Unwanted Apps

Yichi Zhang, Guoxing Chen, Yan Meng, and Haojin Zhu*
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
Email: {mrzhangyc, guoxingchen, yan_meng, zhu-hj}@sjtu.edu.cn

***Abstract*—Cross-platform UI frameworks may facilitate a new category of Potentially Unwanted Apps, dubbed XPUAs, which uses framework-specific language to implement its UI in the form of cross-platform payload. XPUAs are able to bypass the existing app vetting procedures leveraging their unique technical characteristics and make revenue on addicitive contents that are strictly prohibited by either local laws or app market regulations. In this paper, we first examined the profit chain of XPUAs and then proposed PUAXRAY, a novel detection system that utilized machine learning to identify XPUAs. PUAXRAY used a binary classifier that was trained on features extracted from cross-platform payloads, including semantics information and third-party library usage information. We evaluated PUAXRAY on a dataset that was created for the first time in the community with benign apps from reputable app markets and XPUAs from an industry collaborator. PUAXRAY achieved 95.4% F1-score in the XPUAs identification task, and proved capable to be extended to other cross-platform UI frameworks.**

***Index Terms*—Malware Detection, Potentially Unwanted App, Cross-platform UI Framework, Profit Chain**

I. INTRODUCTION

Potentially Unwanted Applications (PUAs) have become a serious issue to mobile users as over 10% mobile devices are affected by at least one PUA [11]. Instead of causing damage to device or disrupting the system, PUAs target mobile users directly, *e.g.*, causing financial loss or privacy leakage [10]. Various techniques for identifying PUAs have been proposed by analyzing native code [13] or web traffic [14].

On the other hand, due to the use of portable programming languages, the recently emerged mobile cross-platform UI frameworks [20] facilitate a new category of PUAs, dubbed XPUAs (**Cross**-Platform UI based **P**otentially **U**nwanted **A**pplication). One such example was found in Apple's App store (id: 64*****682), which claimed to be appropriate for kids over 4 years old. XPUAs utilized portable languages to create cross-platform payloads that contained UIs specialized in delivering prohibited contents. Furthermore, our preliminary study (Sec. II) revealed its profit chain as depicted in Fig. 1, and that it escaped the vetting process of the app market via the integration with a benign image sharing functionality. Luring victims to pay for the addictive contents that included pornography and gambling games caused harm to the potential audience from app markets (*e.g.* the under-aged kids) and such behavior is considered unwanted according to local regulations and app market guidelines.

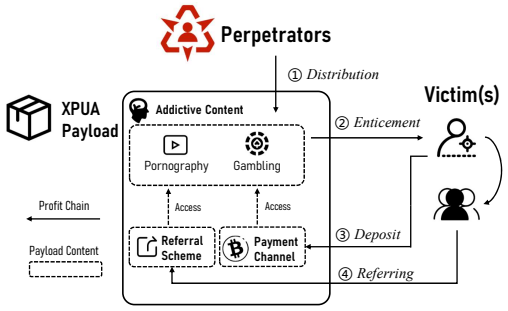

Fig. 1. **The profit chain that characterizes XPUAs in four steps:** 1) Distribution of addictive contents in cross-platform payload. 2) Enticement to access such content. 3) Victims either pay for the access, or 4) refer more victims

Identifying XPUAs poses several technical challenges. Firstly, similar to other PUAs, XPUAs lack definitive fingerprints such as sensitive system API invocations and binary patterns [22]. As stated in previous studies [3], the detection of XPUAs mostly relies on semantic understanding. Therefore, a semantics model is needed to characterize XPUAs. Secondly, XPUAs are implemented using a domain specific portable language with framework-specific format to be shipped as payload across platform. Prior detection system [8], [13] that only considered the platform-native codebase cannot be directly adapted to it. Furthermore, the payload may be in an undocumented binary format that depends on the runtime of the framework and may be compiled or shelled by custom virtual machine instructions. No mature reverse engineering tools can be utilized to decompile such binaries. Thirdly, the wide array of cross-platform frameworks on the market calls for a generic and extensible system.

To this end, we designed and implemented PUAXRAY, a system for identifying XPUAs built with various cross-platform frameworks. To understand the semantics, we first conducted a preliminary study on the motivating example and summarized its profit chain model. We then parsed the framework payload according to its header information to separate the data section which may contain semantics information through text and class names. Next, we conducted a third party library analysis concerning the usage of libraries from the framework community. We utilized the extracted semantics information and third party library usage information to train a classification model to efficiently detect XPUAs. Additionally,

*Haojin Zhu is corresponding author.

979-8-3503-1090-0/23/$31.0 © 2023 IEEE

we created an XPUA dataset for the first time in the community consisting of apps with 1050 android packages and 44 iOS bundles.

In our evaluation, we first measured the effectivity and the efficiency of PUAXRAY. Then we proved its extensibility by adopting the same methodology to identify XPUAs based on another cross-platform UI framework. The result showed that our system can efficiently and effectively identify XPUAs from benign apps with a precision of 94.9% and a recall of 96.0%. The experiment result on extensibility also proved that the proposed system was able to be extended to other cross-platform UI frameworks. The main contributions of our work are:

- We identified a new type of potentially unwanted apps based on cross-platform UI frameworks, coined as XPUAs. XPUAs reaped victims by enticing them to pay for the addictive content contained in the cross-platform UIs. Due to the portability of cross-platform UI frameworks, this kind of PUA may deal wider impact than traditional PUAs.
- We designed and implemented PUAXRAY, an efficient and effective system for identifying XPUAs that is extensible to other cross-platform UI frameworks.
- PUAXRAY achieved great performance in precision and recall. We responsibly disclosed the identified XPUAs to Apple and they were later removed from the App Store.

II. PRELIMINARY STUDY AND PROBLEM STATEMENT

In this section, we first briefly introduce cross-platform UI frameworks, and then describe a preliminary study on an officially downloaded example XPUA that motivates our work.

TABLE I
POPULAR CROSS-PLATFORM UI FRAMEWORKS KEY COMPONENTS

Name	Framework DSL	Runtime Environment	Payload Format	TPL Repo
Flutter	Dart	DartVM	binary	✓
React Native	JS	Hermes	binary	✓
Ionic	JS	Capacitor	source	✓
Cordova	JS	JSEngine	source	✓
Uni-app	JS	JSEngine	source	✓
Weex	JS	JSEngine	source	✓
.NET	C#	Mono	binary	✓
Framework7	JS	JSEngine	source	✓

Cross-platform UI frameworks. Cross-platform UI frameworks may share common key components as shown in the first row in Table I. Among the various frameworks, React Native and Flutter are the most two popular according to a recent survey [19], accounting for over 80% of cross-platform UI framework usage. In general, a mobile cross-platform UI framework consists of a framework Domain Specific Language (DSL), a runtime environment and third-party library (TPL) repositories. Code written with DSL are portable because they can be shipped across platforms as *payload*, which can be either compiled or in minified source code format. The framework runtime is responsible to interpret the DSL and integrate the payload onto native platform. Compared to the Online App Generators (OAGs) which require minimal coding effort [8], [16], the cross-platform UI frameworks require more knowledge on the framework DSL as well as its integration with the native platform.

To implement cross-platform UI, those framework uses different configurations. Mobile web views with JS were common in cross-platform development. Many frameworks in Table I adopted such paradigm, while the recently emerging cross-platform UI frameworks used customized DSL and runtime to improve performance. For example, Flutter uses Dart and DartVM to achieve better performance in UI smoothness and startup time, which essentially results from the pre-compiled machine code and customized runtime. React Native also customized its runtime to enable a natively-feeling integration. Besides coding from scratch, those frameworks allow developers to incorporate third party libraries from developer communities that serve to supply packaged functionality in various purposes including certain UI layouts and widgets, Single Sign-On and analysis tools *etc*.

Preliminary Study on the Motivating Example. The addictive content in the motivating example was implemented with Flutter, a popular cross-platform UI framework. Taking the motivating case from Sec. I as example, we explain its three interesting characteristics that can be used to escape app vetting.

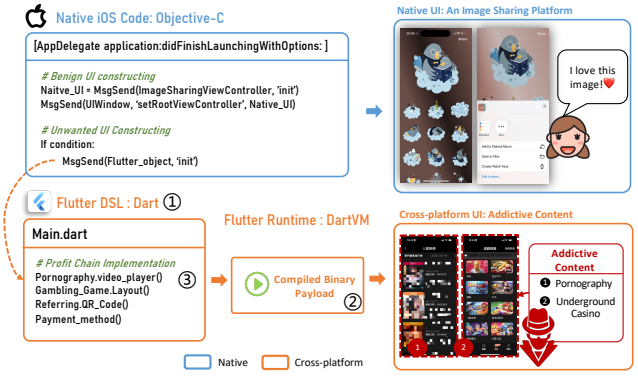

Fig. 2. UI implementation in XPUAs based on the example app, which used Flutter.

As in Fig. 2, we summarize three interesting technical traits of XPUAs. Firstly, the addictive content is delivered in UI that is implemented with a framework-specific portable Domain Specific Language (DSL), which results in a proprietary binary format. Secondly, the platform-native code only serves to launch the unwanted content. XPUAs may reveal no unwanted intention in the platform-native based implementation. Condition judgement is treated as neutral in previous studies [13]. Thirdly, XPUAs are free from system sensitive APIs that are not exposed to the framework runtime. The main target of XPUAs is to generate revenue by luring victims to pay for the addictive content such as pornography and gambling.

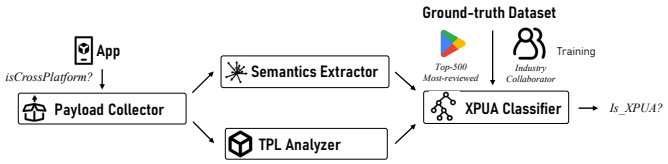

Fig. 3. Overview of the process of our system, including four key components: the Payload Collector, the semantic extractor, the TPL analyzer, and the XPUA Classifier.

To check whether existing PUA detectors could identify such XPUA, we uploaded it to VirusTotal, considering there were no previous studies specialized in XPUAs. To our surprise, none of the engines in VirusTotal flagged it as PUA. Identifying such XPUAs may require a different methodology by addressing the addictive content that is delivered in the cross-platform UI payload directly.

Problem Statement. In this work, we aimed to detect a newly discovered cross-platform UI framework based PUA, dubbed XPUA. Perpetrators escaped the vetting process of app markets by utilizing the distinctive characteristics of cross-platform UI frameworks. Specifically, they used Framework DSL to escape existing app vetting on the platform-native codebases, utilized framework runtime to dynamically load the potentially unwanted UI binary and abused existing functionalities from third-party community to implement their profit chain.

III. PUAXRAY DESIGN

In this section, we present PUAXRAY, a system for identifying XPUAs that directly addressed the cross-platform payload. Particularly, we describe the heuristics of each PUAXRAY component in Sec. III-A, and detail the feature extraction in Sec. III-B.

A. PUAXRAY's Heuristics

To distinguish XPUAs from benign apps, PUAXRAY leverages their unique behavior fingerprints in payload code semantics and third-party library usage. Fig. 3 illustrates the design of PUAXRAY, which consists of a payload collector, a semantics extractor, a TPL analyzer, and an XPUA classifier. The payload collector checks for the existence of cross-platform payload and extracts them from the app package. The semantics extractor and the TPL analyzer extract features about the code semantics and TPL usage from the payload. The XPUA classifier conducts a binary classification based on the extracted feature. We next present more details about PUAXRAY.

Payload Collector. The payload collector extracts the cross-platform payload and feeds it into the Semantic Extractor. To determine if an application adopts the cross-platform development paradigm, we check for framework fingerprints. Specifically for Flutter, we check the existence of the corresponding dynamically linked object located in the `Lib` folder for android and `Frameworks` folder for iOS.

Semantics Extractor. The semantics extractor constructs a feature vector based on the behavior fingerprints of XPUAs. It extracts the embedded texts in the data section according to the header information at the beginning of the payload binary, and uses NLP techniques to convert it into bags of words, which is used in the similarity analysis in semantics based a predefined keyword set characterizing the profit chain.

TABLE II
PRE-DEFINED BUSINESS MODEL KEYWORD LIST

Business Model Step	Example keywords
Addictive Content	['nude', ...], ['lottery', ...]
Financial Enticement	['deposit','membership', ...]
4th Party Payment	['USTD', 'QR Code', ...]
Referral Scheme	['refer code', 'up|down stream', ...]

TPL Analyzer. The Third-Party Library (TPL) analyzer checks for the technical feasibility to implement the profit chain by measuring the usage information of libraries from the developer community that are essential to XPUA's specialized UI for delivering the addictive content. We explain its detail later in Sec. III-B.

XPUA Classifier. The XPUA Classifier conduct the binary classification task to determine whether the input app is XPUA with a set of machine learning models and selects the model with the best performance.

B. Feature Extraction

In PUAXRAY, semantics and TPL usage features are used to quantify the possibility that the payload fits in the business model that we observed in the motivating example. We constructed semantic features and TPL usage features from the text information extracted from the payload collector.

Semantics Features. For semantic features, we summarized a keyword list that reflected the profit chain model of XPUAs. The keyword lists were categorized by the four steps in the profit chain model as shown in Fig. 1. Table III-B lists some example keywords for each step. To calculate how close the payload semantics is to our predefined profit chain model, we updated the feature with each category of keywords. Specifically, for each category in Table III-B, we used Equation (1) to update our feature vector.

$$f_{tpl}[i] = \frac{\sum_{j=1}^{D_i} d_i[j]}{D_j}, \tag{1}$$

where D_i stands for the total number of keywords in category i and

$$d_i[j] = \begin{cases} 1, & \text{if } keyword[i][j] \in matched \\ 0, & \text{if } keyword[i][j] \notin matched \end{cases}$$

We used English as the intermediate language for text embedding. First we removed common stop word. Then we utilized sentence embedding to embed variant lengths of predefined keywords. Next we conducted n-gram iteration on the extracted text according to the number of words in the embedded phrases. Note some of the words such as payment channels were specific words in the profit chain model, and

measuring its cosine similarity may generate false positives. Thus we conducted a strict match for those words.

TPL usage features. The TPL usage features were used to determine whether the payload has the capability to technically realize the profit chain model. To extract the TPL usage features, we quantify the library usage among selected categories listed in Table III. The categories were selected based on the insight that addicitive content relies on specific functionalities, such as the referring scheme prompted victims to share a QR code that was generated using the `qr_flutter` Flutter library.

TABLE III
5 CATEGORIES OF THIRD-PARTY LIBRARIES THAT TECHNICALLY SUPPORT THE BUSINESS MODEL

Third-party Lib Category	Example Lib Name
Addictive Content UI Constructing	video_player
Webpage Rendering	flutter_html
QR Code Generation	qr_flutter
Online Chat	chat_online_customers
Membership Access	modal_barrier.dart

Third-party library usage information can be measured through search the text information in the binary. The naming of library dependencies in cross-platform conforms to specific pattern. Thus we can use regular expression to extract library usage from the text extracted from the payload. Note that code obfuscation would only affect identifier names. Library names are not obfuscated otherwise the payload would not compile. As developers may use multiple libraries in the same category to implement different functionalities, we generate TPL usage feature in a similar way using Equation (2),

$$f_{tpl}[m] = \frac{\sum_{n=1}^{H_m} h_m[n]}{H_m}, \tag{2}$$

where H_m stands for the total number of libraries in category m and

$$h_m[n] = \begin{cases} 1, & \text{if } lib[m][n] \in matched \\ 0, & \text{if } lib[m][n] \notin matched \end{cases}$$

C. XPUA Classification

We concatenated the semantics feature and the TPL usage feature to pass them to the XPUA classifier for detection.

$$F = f_{semantic} \parallel f_{tpl} \tag{3}$$

As the feature space contains both discrete and continuous value, we choose the GBDT as our classifier. We train it using Equation (4) where L stands for the loss function, γ is the observed value and C stands for the ensemble of classifiers.

$$C(x) = \arg\min \sum_{1}^{n} L(y_i, \gamma) \tag{4}$$

IV. IMPLEMENTATION

Our prototype implementation focuses on the detection of XPU-PUAs developed with Flutter framework due to two reasons in the following. On one hand, Flutter has interesting traits that are worth studying, *e.g.*, proprietary object format. Perpetrators may take advantage the gap between the current app vetting and the obscure object format to make effort to infiltrate app markets. By studying Flutter-based XPU-PUAs, we hope to shed light on the detection of XPU-PUAs that are created through a similar methodology. On the other hand, Flutter is currently among the most popular cross-platform UI frameworks for mobile development. The increasing number of Flutter-based app calls for the demand of a semantic checking methodology to regulate the newly-emerging area.

Semantics Extractor. We used different strategies for code semantics and library names. For function and identifier names, we split them using WordNinja (https://github.com/keredson/wordninja) in the same format of raw string. For framework and third party library names, we extracted them using regular expressions based on framework specific patterns.

TPL Analyzer. We checked the documentation for libraries in the repository of Flutter (https://pub.dev/). We used prefix `package:` and suffix `.dart` to filter the third party libraries. For React Native, we conducted a similar procedure by scanning through its repository (https://www.npmjs.com/) and filtering library using suffix `.tsx`.

XPUA Classifier. The concatenated feature vector from Semantics Extractor and TPL Analyzer had 11 dimensions. We used `sklearn`, a python machine learning tool to build our classifier. Our training set contained 70% ground truth data, and the rest 30% was used for validation. We selected the most appropriate learning rate based on the performance on the validation set.

V. EVALUATION

A. Dataset Construction

We constructed the first XPUAs dataset with android packages and iOS bundles. For android packages, we collected apps from 3 sources: I) the top-500 most reviewed apps from Google Play in each of 33 categories, II) apps in reputable app markets from Androzoo [2], and III) PUAs from an industry collaborators. Considering the recent emergence of Flutter, we only crawled apps created after Jan 2020 in source II) and III). We then used the Payload Collector to filter Flutter-based apps. Results are summarized in Table IV. From $10,745$ android packages we collected 94 apps; from $101,226$ apps in Androzoo, we collected $3,772$ apps, and from $45,228$ apps from our industry collaborator we collected $2,996$ apps. For iOS bundles, due to the difficulty in crawling the Apple App Store, we manually downloaded exemplar apps in the Flutter showcase (https://flutter.dev/showcase). Then we collected XPUAs in iOS App Store by visiting XPUA download links with an iOS device.

To create the ground truth dataset, we manually labeled the 94 apps from google play and 347 apps in Androzoo to

construct the benign dataset. We then randomly selected 20% XPUAs from the PUA set to construct the unwanted dataset.

TABLE IV
STATISTICS ON THE APPS COLLECTED IN THIS STUDY

#App	Google Play	Androzoo	PUA	iOS	Total
Downloaded	10,745	101,226	45,228	44	157,243
Flutter-based	94	3,772	2,996	44	6,906
Benign	94	347	-	34	475
Unwanted	-	-	609	10	619

B. PUAXRAY *Performance*

In this section, we evaluate the performance in a cross-platform scenario with android packages and iOS bundles. To demonstrate that PUAXRAY can identify XPUAs in a cross-platform scenario, we trained PUAXRAY with the ground truth using android packages and test it on the iOS bundle dataset.

Model Selection. To choose the best classification model, we trained the XPUA classifier with various classification algorithms. Fig. 4 shows the ROC curves from different machine learning models including logistic regression (LR), support vector machine (SVM) and gradient boosted decision tree (GBDT), where GBDT had the best performance. We therefore adopted the GBDT as our classification model.

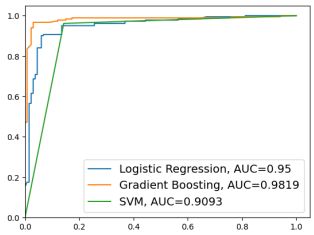

Fig. 4. The classification results on Flutter based XPUAs, where GBDT outperforms SVM and LR.

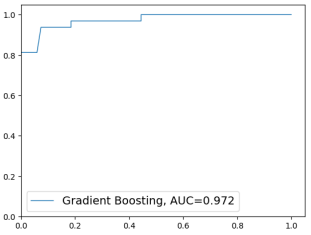

Fig. 5. ROC curve on React Native based XPUAs.

Efficiency. The efficiency of PUAXRAY was evaluated by measuring the time consumption of examining one application on average. The results were measured on a windows PC equipped with i7-8700k CPU and 32 GB DRAM. PUAXRAY took an average of 9.3 seconds to perform a single identification task.

Effectiveness. Table V lists the performance of PUAXRAY in identifying XPUAs. PUAXRAY achieved a 94.9% precision and a 94.9% recall rate. We further tested PUAXRAY's performance on the iOS dataset, and it successfully identified all the XPUAs with 100% precision and recall rate. We also evaluated PUAXRAY using gambling apps from a previous dataset [7]. Among the 100 gambling apps, we found one gambling app was implemented using Flutter. PUAXRAY successfully identified that app.

False Negatives and False Positives. We reviewed the false positive and false negative applications and discovered that (1) for false positives, these apps had similar semantics and implementation to the business model but did not actually contain addictive content; (2) for false negatives, these apps concealed their semantics through code obfuscation and loaded addictive content with online resources. We will discuss potential solutions in Sec. VII.

TABLE V
SYSTEM PERFORMANCE

Precision	Recall	Accuracy	F1-Score	Time(s)
94.9%	96.0%	94.9%	95.4%	9.3

Extensibility. We demonstrated the extendibility of our system by identified XPUAs based on another popular cross-platform UI framework, *i.e.*, React Native. Similarly, we created a dataset containing 315 apps from Google Play Top-reviewed list as the negative ground truth, and 114 PUAs as the positive ground truth. Fig. 5 showed the ROC curve and PUAXRAY achieved a precision of 100% and a recall rate of 88.2%.

In summary, PUAXRAY was capable of identifying XPUAs in a cross-platform scenario with extendibility to other cross-platform UI frameworks with great time efficiency and accuracy.

VI. RELATED WORKS

A. *Potentially Unwanted Apps Identification*

Researchers have proposed traffic-based identification approaches on potentially unwanted content in social media especially when considering children protection [1], [14]. Meanwhile, mobile app markets formulate development guidelines and enforce strict app vetting processes that forbid apps from distributing such content, while recent studies showed that perpetrators kept seeking for new tricks to smuggle the potentially unwanted content into mobile devices for profit. Lee et al. [13] discovered conditionally triggered crowdturfing UIs through static program analysis by vetting the platform native code. More example on PUA Wang et al. [7] discovered that illicit websites could prompt users to side-load their apps on victims' devices, which skips the vetting process of app markets. They revealed such illicit ecosystem by dynamically collected the request backend server URLs. Hong et al. [8] found potentially unwanted contented can also be delivered through the Android WebView UI widget through static taint analysis on the API parameters.

Previous research has identified methods used by perpetrators to create PUAs, either through platform-native code or web hybridization. However, these studies have overlooked the fact that cross-platform UI frameworks can also be exploited by perpetrators who hide potentially unwanted behaviors in framework-specific codebases to evade app vetting processes. Existing defense mechanisms are inadequate because they are targeted to cross-platform payloads. Additionally, automated dynamic traffic analysis is impractical due to encryption and low efficiency. Therefore, a static system is needed to detect such apps before installation.

B. Cross-platform Development Concerns

Among various cross-platform development schemes, the security concern on web view security is the most discussed subject by researchers. Lee et al. [12] detected programming errors from JavaScript source code. Bae et al. [4] extended the former study by formally specifying the interoperability between Java and JavaScript, and achieved a better performance. Further studies conducted measurements on the scale of potential coding bugs in web-view-based development [17], [18]. Another line of works defended vulnerabilities in web view arising from system API exploitation [21], [24].

Apart from web view, the security and safety aspect of other emerging cross-platform UI frameworks is rarely discussed. Existing studies on those frameworks have evaluated the resource consumption [5], [15], automation testing ability [23], motivation of choice [6] and compatibility issues [9].

VII. Discussion

Limitation and future work. PUAXray relied on the semantics information contained in the framework payload. For those app who adopted code obfuscation and load text from the internet, PUAXray may fail to identify the unwanted payload. However, such apps took up a small part from our observation. One of our future work is to combine dynamic traffic analysis with static analysis to identify such apps.

Ethical issue and responsible disclosure. We may only publicize the cross-platform payloads in consensus with our industry partner due to concerns of facilitating the profit chain. Meanwhile, we responsibly disclosed the discovered XPUAs to Apple, who later removed all of them from the App Store.

VIII. Conclusion

In this paper, we discovered XPUAs, a new category of PUA that generates revenue by distributing addictive content such as pornography and gambling through cross-platform payloads. After conducting a preliminary study, we summarized its technical traits and proposed a novel detection system dubbed PUAXray. To evaluate our system, we created an XPUA dataset with android packages and iOS bundles. Experiment results showed that PUAXray was capable of efficiently and effectively detection XPUAs with extendibility to other cross-platform UI frameworks.

Acknowledgment

This research was supported by National Natural Science Foundation of China under Grants No. 62132013.

References

[1] Z. Ahmad and U. Özkaya, "Machine learning and artificial intelligence-based child abusing tracking system for the detection of online sexual predators," in *International Conference on Trends in Advanced Research*, vol. 1, 2023, pp. 131–141.

[2] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th international conference on mining software repositories*, 2016, pp. 468–471.

[3] B. Andow, A. Nadkarni, B. Bassett, W. Enck, and T. Xie, "A study of grayware on google play," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 224–233.

[4] S. Bae, S. Lee, and S. Ryu, "Towards understanding and reasoning about android interoperations," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 223–233.

[5] H. Brito, A. Gomes, Á. Santos, and J. Bernardino, "Javascript in mobile applications: React native vs ionic vs nativescript vs native development," in *2018 13th Iberian conference on information systems and technologies (CISTI)*. IEEE, 2018, pp. 1–6.

[6] L. Delia, P. Thomas, L. Corbalan, J. F. Sosa, A. Cuitiño, G. Cáseres, and P. Pesado, "Development approaches for mobile applications: Comparative analysis of features," in *Intelligent Computing: Proceedings of the 2018 Computing Conference, Volume 2*. Springer, 2019, pp. 470–484.

[7] Y. Gao, H. Wang, L. Li, X. Luo, G. Xu, and X. Liu, "Demystifying illegal mobile gambling apps," in *Proceedings of the Web Conference 2021*, 2021, pp. 1447–1458.

[8] G. Hong, Z. Yang, S. Yang, X. Liaoy, X. Du, M. Yang, and H. Duan, "Analyzing ground-truth data of mobile gambling scams," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2176–2193.

[9] M. Q. Huynh and P. Ghimire, "Browser app approach: Can it be an answer to the challenges in cross-platform app development?" *Journal of Information Technology Education. Innovations in Practice*, vol. 16, p. 47, 2017.

[10] M. Kan. (2022) Google pulls 6 fake antivirus apps from play store. https://www.pcmag.com/news/ Accessed April 2, 2023.

[11] P. Kotzias, J. Caballero, and L. Bilge, "How did that get in my phone? unwanted app distribution on android devices," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 53–69.

[12] S. Lee, J. Dolby, and S. Ryu, "Hybridroid: static analysis framework for android hybrid applications," in *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, 2016, pp. 250–261.

[13] Y. Lee, X. Wang, K. Lee, X. Liao, X. Wang, T. Li, and X. Mi, "Understanding ios-based crowdturfing through hidden ui analysis." in *USENIX Security Symposium*, 2019, pp. 765–781.

[14] Q. Luo, J. Liu, J. Wang, Y. Tan, Y. Cao, and N. Kato, "Automatic content inspection and forensics for children android apps," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7123–7134, 2020.

[15] P. Nawrocki, K. Wrona, M. Marczak, and B. Sniezynski, "A comparison of native and cross-platform frameworks for mobile applications," *Computer*, vol. 54, no. 3, pp. 18–27, 2021.

[16] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes, "The rise of the citizen developer: Assessing the security impact of online app generators," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 634–647.

[17] B. Shen, W. Zhang, A. Yu, Z. Wei, G. Liang, H. Zhao, and Z. Jin, "Cross-language code coupling detection: A preliminary study on android applications," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 378–388.

[18] A. Tiwari, J. Prakash, S. Groß, and C. Hammer, "Ludroid: A large scale analysis of android–web hybridization," in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2019, pp. 256–267.

[19] L. S. Vailshery, "Cross-platform mobile frameworks used by developers worldwide 2019-2021," 2022, https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ Accessed April 5, 2023.

[20] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proceedings of the 6th Balkan Conference in Informatics*, 2013, pp. 213–220.

[21] G. Yang, J. Huang, and G. Gu, "Iframes/popups are dangerous in mobile webview: Studying and mitigating differential context vulnerabilities." in *USENIX Security Symposium*, 2019, pp. 977–994.

[22] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–40, 2017.

[23] H. A. Zahra and S. Zein, "A systematic comparison between flutter and react native from automation testing perspective," in *2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. IEEE, 2022, pp. 6–12.

[24] L. Zhang, Z. Zhang, A. Liu, Y. Cao, X. Zhang, Y. Chen, Y. Zhang, G. Yang, and M. Yang, "Identity confusion in {WebView-based} mobile app-in-app ecosystems," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1597–1613.