An Enhanced Framework to Mitigate Post-Installation Cyber Attacks on Android Apps

Vijay Koka
Department of CSE, GITAM School of Technology, Hyderabad, India
kokavijay58@gmail.com

Kireet Muppavaram
Department of CSE, GITAM School of Technology, Hyderabad, India
kireet04@gmail.com (corresponding author)

Received: 12 April 2024 | Revised: 29 April 2024 | Accepted: 7 May 2024

 DOI: https://doi.org/10.48084/etasr.7467

ABSTRACT

The widespread use of smartphones worldwide has led to a corresponding rise in the number of mobile applications available for Android devices. These apps offer users convenient ways to perform various daily tasks, but their proliferation has also created an environment in which attackers can steal sensitive information. Insecure options employed by many app developers create vulnerabilities that can be exploited by attackers to gain access to most smartphones. While existing methods can detect malware during app installation, they do not sufficiently address post-installation attacks, such as those resulting from fake apps or Man-in-the-Disk (MitD) attacks. To address this issue, the current study conducted research on post-installation attacks, including data leakage, malware injection, repackaging, reverse engineering, privilege escalation, and UI spoofing. MitD attacks are particularly challenging to counter, so, to mitigate this risk, the Post-Installation App Detection Method is proposed to monitor and regulate sensitive information flow and prevent MitD attacks.

Keywords-cyber attacks; malware; cyber attacks in android apps; post installation attacks; fake apps; MITD attacks

I. INTRODUCTION

Smartphones have become an indispensable part of daily life, with 6.648 worldwide users. Nearly 45% of these users own Android smartphones, which are popular for their openness and ease of use. Unfortunately, this accessibility has led to the proliferation of malicious apps on the Android platform, and recent reports indicate that over 9.5 million malwares have been blocked to secure smartphones. Although Android offers many security provisions to detect malware, its existing mechanism only checks app permissions and faults without verifying the logical connectivity of the app. This leaves open attack vectors [1] that hackers can exploit to introduce malicious activities and access sensitive information.

This paper focuses on investigating post-installation attacks in mobile applications, particularly one of the most significant post-installation attacks, Man-in-the-Disk (MitD) attacks [2]. MitD attacks occur when an attacker intercepts data between an app and external storage, potentially compromising the external storage and extracting sensitive information from the user, such as login credentials or financial data. The proposed Post-Installation App Detection Method offers a solution to MitD attacks by detecting them and regulating the flow of sensitive information to the attacker. Detecting post-installation attacks [3] in mobile applications is critical for enhancing the security of sensitive user information. By providing a solution to mitigate MitD attacks, this paper improves the security of mobile applications and reduces the risk of sensitive information exposure to attackers.

The attacks which occur after the app installation and functioning are termed as post installation attacks. Authors in [3] identified how the attackers intercept the sensitive data sent from the app to the server, such as login credentials and personal information, using methods like Man-in-the-Middle (MitM). Authors in [5] presented how attackers inject malware into the app's code, which can allow them to control the device or steal sensitive data. Authors in [6] studied how the attackers download the original app from the app store, modify its code, and repackage it as a new app, allowing them to access the user's sensitive information. Authors in [7] discussed how the attackers decompile the app's code, identify vulnerabilities, and exploit them. Authors in [8] reported how the attackers gain escalated privileges within the mobile app to access sensitive information or control the device. Authors in [9] analyzed how the attackers modify the app's user interface to deceive users into providing sensitive information, such as login credentials.

Automatic Android Malware Detection [18] is a robust method which detects the malware using static analysis by passing API to the deeplearning models. Antibypasssing four

stage dynamic behavior [19] shows the four stage dynamic behavior which addresses the dynamic behavior of an attack and when this method is applied to recent attacks it bypasses a few post installation attacks like MitD, but does not bypass the few cases where the apk is not extracted. EnDroid [20] has worked on post installation attacks. It extracts the critical behavior of the attacks.

It is recommended that users only download apps from reliable sources, keep their devices updated with the latest security patches, and use strong passwords and multi-factor authentication. Additionally, mobile app developers can implement security measures, such as encryption, secure data storage, and code obfuscation to make vulnerability exploitation more challenging. However, despite the security measures provided by Android, recent attackers still target smartphones through post-installation attacks. Extensive research on post-installation attacks has identified solutions for common attacks, entailing MitM [8], malware injection, privilege escalation, and UI spoofing. However, many reports indicate that MitD attacks are becoming increasingly common and that the existing studies do not have adequate detection methods to identify these types of attacks. The present study's investigation into MitD attacks reveals that attackers depploy update app package insertion as a major attack vector. This MITD attack is similar to the MitM attack in which the attacker uses the app data stored employing external storage permission.

MitD attacks are similar to MitMattacks [9] and involve replacing legitimate update files with malicious program code. Once the app is installed, the attacker can obtain necessary access and extract user information. MitD attacks can also disable the app's functionality, rendering its services inaccessible to the user. The external storage on an Android device is usually located on the SD card or within the storage partition, and it enables components and files to be shared among various mobile apps. This sharing of data creates a vulnerability that can be exploited by collusion attacks [10], inter-leaking attacks [11], and MITD attacks [11].

II. THE PROPOSED METHOD FOR DETECTING POST INSTALLATION ATTACKS IN MOBILE APPLICATIONS

In this paper, the Post Installation App Attack Method is proposed to identify recent exploits in Android mobile applications [13, 14], such as MitD attacks. The recommended approach is designed to cover attacks that occur after the installation of Android applications. Most current techniques [9, 15] only offer malware detection [15] prior to app installation, and do not provide detection after installation. Developers regularly update their apps by publishing new packages, which are often used in mobile application attacks [16]. While the Play Store has some methods to validate the packages, MitD attacks still occur while utilizing these updated packages. To address this issue and identify any potential MitD attack, the Post Installation App Detection Method has been developed. The method involves four steps:

1) Feature Extraction from Apps

The initial step of the proposed method involves the extraction and pre-processing of app features. The Virustotal tool was employed to extract the complete package details of an app, including permissions, system calls, certificate details, version, developer data, and intent filters used in the app. These details were pre-processed although some outdated apps had compatibility issues, which were addressed by engaging an APK decompiler.

2) Pre-Processing of Key Features

In the second step, the proposed solution pre-processes the features of permissions and certificate attributes. Putting into service the Virustotal tool, the APK file of the app is extracted, including permissions, services, activities, intent filters, receivers, and providers. Focus is given on pre-processing the requested permissions and certificate attributes since they are crucial for app detection.

3) Detection of Fake Apps

The third step of the proposed technique involves utilizing an algorithm for detecting fake apps.

4) Detection and Identification of MITD Attacks

This step encompasses detecting and identifying MitD attacks.

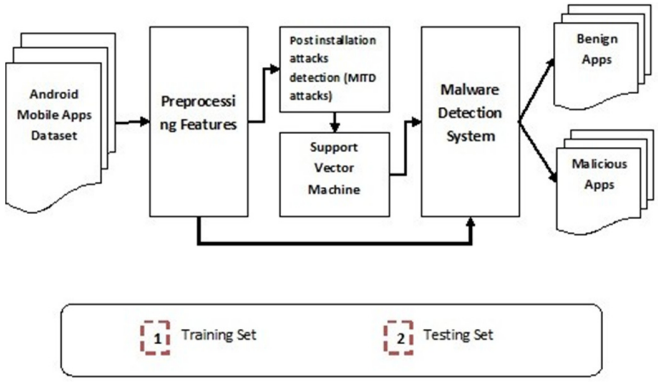

Fig. 1. Detection of MitD post installation attacks.

An investigation of apps was conducted using various sources, such as DREBIN, Virusshare, Google Play store, etc. to gather information on app packages. The performed investigation revealed that permission features and certificate attributes are crucial aspects for detecting fraudulent apps. The current approaches adopted to detect recent malware attacks require more than five attributes. The accuracy of the detection was tested at different levels of feature reduction. During the pre-processing stage, it was observed that the outcomes of the three app features and the two app features were identical.

The proposed method relies on two key features: permissions and certificate attributes. To obtain these features, the Virustotal tool was implemented to extract them from the app apk file. When the app apk file is provided to the Virustotal tool, it extracts various components, namely basic properties, history, names, Android information as summary, certificate attributes, certificate subject, certificate issuer, permissions, activities, services, receivers, and intent filters [17, 18]. The proposed method determines whether the app is a fake or not.

The detection of a fake app is the first stage in detecting an attack. If the app does not contain any data, it is a dummy or fake app installed by the attacker. The certificate attributes are very important in determining whether an app is a fake app or a legitimate app.

Algorithm 1: Fake App Detection

```
Input: P as Permissions, C as Certificate
CA as Certificate Attributes, CS as
Certificate Subject, CI as Certificate
Issuer, CN as Common Name, OG as
Organization, OGU as Organization Unit, St
as State, and LC as Locality
Output: App legitimacy status (genuine or
fake)
Extract app permissions and certificate
attributes using the VirusTotal tool.
if app permissions listed in P and
certificates in C are present then
Verify the attributes of CN, OG, OGU, ST,
and LC
if CN equals OG, OG equals OGU, and OGU is not
equal to ST or LC then
Set app legitimacy status to "genuine."
else
Set app legitimacy status to "fake."
end if
else
Set app legitimacy status to "fake," and
Print "App is fake due to mismatch in
permissions and certificate details."
end if
```

The algorithm first extracts app permissions and certificate attributes using the VirusTotal tool. It then verifies if the app permissions listed in P and the certificates in C are present in the app. If both are present, the algorithm proceeds to verify the attributes of CN, OG, OGU, ST, and LC. If CN equals OG, OG equals OGU, and OGU is not equal to ST or LC, the app is considered genuine, and the algorithm sets the app legitimacy status to "genuine." If these conditions are not met, the algorithm sets the app legitimacy status to "fake." If the app permissions and certificates do not match, the algorithm sets the app legitimacy status to "fake" and prints "App is fake due to mismatch in permissions and certificate details." Overall, this algorithm also aims to identify potential attacks by distinguishing between genuine and fake mobile applications.

Algorithm 2: MitD Attack Detection

```
Input: App data including update
information, original application
information, and additional permissions.
Output: Indication of whether the updated
app may lead to a MitD attack.
Retrieve information on the updated app.
Retrieve information on the original app.
Determine if any additional permissions
are present in the updated app. Verify if
the updated app is genuine.
if the updated app is genuine and the
additional permissions are the same as the
original app, then
Verify if the package name of the updated
app matches the package name of the
original app.
if the package names match then
Print "App is safe."
else
Print "Updated app may lead to a MitD
attack due to mismatched package details."
end if
else if the updated app is genuine and the
additional permissions are different from
the original app, then
Print "Updated app may lead to a MitD
attack due to additional permissions."
else
Print "Updated app may lead to a MitD
attack due to being a fake app."
end if
```

This algorithm takes as input app data and outputs whether the app may lead to an MitD attack. It utilizes the variables upd app to represent the mobile application update information, org app to represent the original application information, and P1 to represent additional permissions. The algorithm begins by checking if the app is genuine by calling the CheckAppGenuine() function and comparing the result to the input app. If the app is genuine, it proceeds to check if the app has the same permissions as listed in P and if the package names match between the updated app and original app. If the permissions and package names match, the algorithm prints, "App is safe". If the package names do not match, the algorithm prints, "Updated app may lead to a MitD attack due to mismatched package details." If the permissions do not match, the algorithm prints, "Updated app may lead to a MITD attack due to additional permissions." If the app is determined to be fake (i.e. not genuine), the algorithm prints, "Updated app may lead to a MITD attack due to being a fake app."

III. IMPLEMENTATION

The proposed approach aims to identify MitD attacks following two methods. Firstly, it detects fake apps by utilizing the first algorithm, while the second algorithm verifies the update file. To test this approach, 300 unsafe and 300 safe apps were employed, and the datasets were balanced accordingly. The pdfebookconverter app was deployed as a case study to demonstrate the post-installation app detection approach, and its features were extracted using the Virustotal tool. The app's features are presented in Figure 2, where they were combined for easy observation. The first algorithm was applied to check if the app data are empty, while the second algorithm verified the certificate attributes, such as the common name, organization, and organizational units. By utilizing the suggested approach, the certificate attributes were verified, and it was observed that the CN, OG, OGU, ST, and LC were identical, whereas the updated package app details led to an

MitD attack. The algorithm printed the app and certificate details, revealing that it was a fake.

Fig. 2. Sample of a fake app.

Figure 3 portrays the extracted features of the Adobe app deploying the proposed method. The app's features were extracted using the Virustotal tool. By analyzing Figure 3, it is evident that the certificate attributes of CN, OG, OGU, ST, and LC are different and not identical. Furthermore, the updated package app details do not lead to any type of MitD attacks, and the original package file details are the same as those of the updated files.

Figures 2 and 3 indicate that this study's algorithms can easily detect whether an app is malicious or benign. Updated app package files stored in different versions of the app were collected to verify if the proposed method is effective in detecting apps. The investigation carried out clearly revealed that attackers have significant opportunities to attack an app by using storage access permissions.

Fig. 3. Sample of a genuine app.

IV. METRICS AND RESULT ANALYSIS

The effectiveness of the proposed model in detecting MitD attacks was evaluated using industry-standard metrics, such as True Positive Rate (TPR) and False Positive Rate (FPR). TPR measures the number of apps accurately identified as exposed to MitD or fake app attacks, while FPR measures the number of apps wrongly identified as MitD attacks or fake apps. To analyze the performance of the suggested approach, experiments were conducted on two categories of apps - whitelisted and blacklisted - as well as on standard malicious apps from the DREBIN datasets [11]. Harmful apps that were downloaded from various sources were categorized as blacklisted, whereas apps downloaded from the Google Play store were considered safe and placed on the whitelist. The obtained results can be observed in Table I.

TABLE I. RESULTS OF THE POST INSTALLATION APP DETECTION METHOD

Apps	TP	FP
Blacklisted	278	21
Whitelisted	297	23
Standard	292	24

TABLE II. RESULTS OF THE POST INSTALLATION APP DETECTION METHOD USING DIFFERENT DATASETS

APP_Datasets	TP	TN	FP	FN
Health_apps_dataset	179	174	21	26
Sports_apps_dataset	178	184	21	16
Fake_apps_dataset	176	179	24	21
Karon_malware_apps_dataset	173	188	27	12
Deep_zone_apps_dataset	196	182	4	17

The results presented in Table I illustrate the True Positives (TP) and False Positives (FP) identified by the proposed Post Installation App Detection method when applied to datasets from both internal and external sources. Regardless of whether the method was applied to malicious applications, safe apps, or ordinary apps, consistent TP were obtained, with fewer FP. Specifically, this work managed to achieve a 97% TPR when detecting standard datasets from DREBIN. Figure 4 displays the accuracy obtained when the proposed method was applied to different datasets. The results indicate that the average attained accuracy is 93%. The proposed method is accurate in detecting the most common post-installation attack, i.e. MitD.

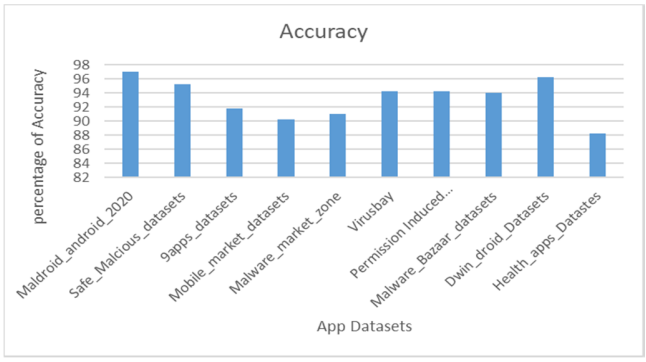

Fig. 4. Accuracy achieved when applying different datasets.

The Post Installation Attacks Detection Method is highly effective in detecting threats in an Android environment due to its advanced behavioral analysis, dynamic monitoring of app behavior after installation, anomaly detection capabilities, integration of machine learning and AI for improved accuracy, real-time alerting and response mechanisms, continuous

monitoring and updates to stay ahead of the evolving threats, and its low FPR, ensuring accurate identification of malicious activities while minimizing disruptions for users when compared with existing and standard recent malware detection models.

TABLE III. PERFORMANCE COMPARISON OF THE PROPOSED METHOD AND STANDARD AND RECENT MALWARE DETECTION MODELS

Model Name	Precision	Recall	Accuracy
Post-Installation-attacks detection method	95.79%	98.94%	97.32%
Automatic Android Malware Detection	86.11%	77.22%	82.66%
Endroid	91.33%	95.56%	93.35%
DREBIN	93.75%	97.57%	95.61%

Recent developments in post-installation attack detection [18-2] for Android apps focus on utilizing machine learning algorithms to better detect sophisticated threats, such as zero-day exploits and polymorphic malware. Additionally, there has been a shift towards leveraging behavioral analytics and anomaly detection techniques to identify subtle deviations in app behavior that may indicate malicious activity. Real-time monitoring and response capabilities have also seen improvements, allowing for quicker mitigation of the emerging threats. Overall, these advancements aim to strengthen the resilience of the Android app security against any evolving post-installation attacks.

V. CONCLUSION

This study focused on post-installation attacks on Android smartphones, with a specific emphasis on MitD attacks, which are a recent and prevalent attack vector that lacks adequate detection solutions. A post-installation app detection method is proposed that uses two algorithms to determine whether an app is susceptible to MitD attacks. The recommended approach was tested on various datasets and it was found that it effectively detects MitD attacks. The former was compared with several standard models and it was discovered that it outperforms them in post-installation attack detection. The investigation conducted also revealed that attackers use several attack paths to gain access to devices and steal personally identifiable information. Therefore, it is crucial for Android developers to strictly adhere to Google's recommendations to prevent cybercriminals from launching attacks. Finally, the suggested approach can handle MitD attacks, which is essential for safeguarding sensitive user data. Overall, the proposed method provides an effective solution to mitigate the risks of post-installation attacks on Android smartphones.

REFERENCES

[1] J. Kumar and G. Ranganathan, "Malware Attack Detection in Large Scale Networks using the Ensemble Deep Restricted Boltzmann Machine," *Engineering, Technology & Applied Science Research*, vol. 13, no. 5, pp. 11773–11778, Oct. 2023, https://doi.org/10.48084/etasr.6204.

[2] M. Kireet, P. Rachala, M. S. Rao, and R. Sreerangam, "Investigation Of Contemporary Attacks In Android Apps," *International Journal of Scientific & Technology Research*, vol. 8, no. 12, pp. 1789–1794, 2019.

[3] S. Nasiri, M. T. Sharabian, and M. Aajami, "Using Combined One-Time Password for Prevention of Phishing Attacks," *Engineering, Technology & Applied Science Research*, vol. 7, no. 6, pp. 2328–2333, Dec. 2017, https://doi.org/10.48084/etasr.1510.

[4] Y. Sun *et al.*, "Detecting Malware Injection with Program-DNS Behavior," in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, Genoa, Italy, Sep. 2020, pp. 552–568, https://doi.org/10.1109/EuroSP48549.2020.00042.

[5] M. Conti, N. Dragoni, and V. Lesyk, "A Survey of Man In The Middle Attacks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2027–2051, 2016, https://doi.org/10.1109/COMST.2016.2548426.

[6] M. Yaseen *et al.*, "MARC: A Novel Framework for Detecting MITM Attacks in eHealthcare BLE Systems," *Journal of Medical Systems*, vol. 43, no. 11, Oct. 2019, Art. no. 324, https://doi.org/10.1007/s10916-019-1440-0.

[7] S. Anand and V. Perumal, "EECDH to prevent MITM attack in cloud computing," *Digital Communications and Networks*, vol. 5, no. 4, pp. 276–287, Nov. 2019, https://doi.org/10.1016/j.dcan.2019.10.007.

[8] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, "Intelligent Vision-Based Malware Detection and Classification Using Deep Random Forest Paradigm," *IEEE Access*, vol. 8, pp. 206303–206324, 2020, https://doi.org/10.1109/ACCESS.2020.3036491.

[9] K. Aldriwish, "A Deep Learning Approach for Malware and Software Piracy Threat Detection," *Engineering, Technology & Applied Science Research*, vol. 11, no. 6, pp. 7757–7762, Dec. 2021, https://doi.org/10.48084/etasr.4412.

[10] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, Jan. 2018, Art. no. 3, https://doi.org/10.1186/s13673-018-0125-x.

[11] W. Enck *et al.*, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *ACM Transactions on Computer Systems*, vol. 32, no. 2, pp. 5:1-5:29, Mar. 2014, https://doi.org/10.1145/2619091.

[12] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proceedings of the 19th Network and Distributed System Security Symposium NDSS 2012*, San Diego, CA, USA, Jan. 2012.

[13] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, Feb. 2012, pp. 95–109, https://doi.org/10.1109/SP.2012.16.

[14] A. A. Alhashmi, A. M. Alashjaee, A. A. Darem, A. F. Alanazi, and R. Effghi, "An Ensemble-based Fraud Detection Model for Financial Transaction Cyber Threat Classification and Countermeasures," *Engineering, Technology & Applied Science Research*, vol. 13, no. 6, pp. 12433–12439, Dec. 2023, https://doi.org/10.48084/etasr.6401.

[15] A. Al-Marghilani, "Comprehensive Analysis of IoT Malware Evasion Techniques," *Engineering, Technology & Applied Science Research*, vol. 11, no. 4, pp. 7495–7500, Aug. 2021, https://doi.org/10.48084/etasr.4296.

[16] "VirusTotal - Home," *Virus Total*. https://www.virustotal.com/gui/home/upload.

[17] M. İbrahim, B. Issa, and M. B. Jasser, "A Method for Automatic Android Malware Detection Based on Static Analysis and Deep Learning," *IEEE Access*, vol. 10, pp. 117334–117352, 2022, https://doi.org/10.1109/ACCESS.2022.3219047.

[18] Y. Zhang, S. Luo, H. Wu, and L. Pan, "Antibypassing Four-Stage Dynamic Behavior Modeling for Time-Efficient Evasive Malware Detection," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 3, pp. 4627–4639, Mar. 2024, https://doi.org/10.1109/TII.2023.3327522.

[19] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A Novel Dynamic Android Malware Detection System With Ensemble Learning," *IEEE Access*, vol. 6, pp. 30996–31011, 2018, https://doi.org/10.1109/ACCESS.2018.2844349.

[20] H. Lu *et al.*, "EAODroid: Android Malware Detection Based on Enhanced API Order," *Chinese Journal of Electronics*, vol. 32, no. 5, pp. 1169–1178, Sep. 2023, https://doi.org/10.23919/cje.2021.00.451.