WJCMS

Wasit Journal of Computer and Mathematics Science

Journal Homepage: https://wjcm.uowasit.edu.iq/index.php/WJCM
e-ISSN: 2788-5879 p-ISSN: 2788-5887

Leveraging Social Engineering Techniques for Ethical Purposes: An Approach to Develop Fake Android App for Collecting Valuable Data Discreetly

Hussein Abdulkhaleq Saleh[1]*

[1]Directorate General of Education in Dhi Qar, Al-Haboubi Street, Nasiriyah, 64001, Dhi Qar, IRAQ.

Corresponding Author: Hussein Abdulkhaleq Saleh

DOI: https://doi.org/10.31185/wjcms.268
Received 24 July 2024; Accepted 27 Augest 2024; Available online 30 September 2024

ABSTRACT: Social engineering techniques are often viewed negatively due to their association with deceptive practices. However, these techniques can also be utilized ethically, as many cybersecurity professionals do, particularly when evaluating vulnerabilities and testing security defenses. This paper presents EDC (Ethical Data Collector), an Android application that utilizes social engineering techniques to discreetly collect valuable data from an Android device for ethical purposes. EDC employs deception through a simulated UI (fake) to engage the target for a period, while secretly collecting data such as device information, active phone number, and images in the background, then sending them to a designated server via the internet. The researcher argues that EDC could help identify inexperienced cybercriminals or extortionists without complex efforts or significant cost, provided that its capabilities are judiciously utilized and subject to proper controls and oversight. EDC's development methodology emphasizes understanding the target's personality, predilections, and preferences to tailor the app experience as required for attracting the target to install and run the application. The paper describes the core functions and workflows for collecting and sending data. Additionally, permissions handling has been addressed as being critical for enabling EDC to collect the required information. Testing on Android emulators demonstrated that the EDC's APK file size is 4 MB, and data collection and transmission processes functioned as intended across various Android versions. The minimum SDK version required to run EDC is level 16. The total estimated time to complete the fake UI process (cumulative user engagement time) is 57-60 seconds, where each activity takes 8 seconds based on the adopted development approach.

Keywords: Android development, Data collection, Ethical Social engineering, Mobile application, Security

1. INTRODUCTION

Social engineering is considered one of the prevalent cybersecurity terms in the last years. It describes how an attacker can exploit some vulnerabilities of a human to breach security or privacy through social interaction [1]. A social engineering attack usually targets the weakness of humans using various techniques and scenarios to elicit sensitive data [2]. Based on various perspectives, social engineering attacks can be classified into several categories, such as fake software attacks, phishing attacks, etc. [3]. What this paper cares about in the previous classification is fake software attacks. This kind refers to a cyber-attack in which an attacker develops or distributes software that appears legitimate, but in reality, is designed to deceive users and perform harmful activities, such as stealing data, or controlling the system without user knowledge [3].

After reading the previous paragraphs, any person simply concludes that the social engineering reputation is negative and causes harm, due to its association with deceptive practices and malicious intent. However, it is important to recognize that not all social engineering utilizes are unethical. Many professionals in cybersecurity fields employ those techniques ethically for vulnerability assessment and security measures tests [4]. From this perspective, utilizing social engineering ethically could be beneficial in various security situations, such as gathering information about malicious

*Corresponding author: hussein.abd.alkhaliq@gmail.com

actors. One of the possible ways to gather information is through designing and implementing special software or applications, which is what we focus on in this work.

Over the past decade, the use of social engineering techniques in cybersecurity has garnered significant attention, with numerous studies focusing on the potential risks and vulnerabilities these methods expose. A common theme in this body of research is the unethical application of social engineering to deceive users and compromise security systems.

For instance, Hadikusuma et al. [5] explore stealing personal data on Android using a Remote Administration Tool (RAT) with social engineering techniques, highlighting how social engineering can facilitate data theft on Android devices. Whereas Wang et al. [1] present a comprehensive domain ontology for social engineering in cybersecurity, highlighting how attackers exploit human vulnerabilities to gain unauthorized access to sensitive data. Similarly, Mouton et al. [2] developed a social engineering attack framework that categorizes various attack methods based on their modus operandi, further emphasizing the destructive potential of these techniques. Also, Salahdine and Kaabouch [3] conducted a survey on social engineering attacks, underscoring the increasing sophistication of these techniques in the digital age.

From another perspective, Blancaflor et al. [6] discuss social engineering attacks on Android devices using StormBreaker, highlighting the risks of malicious links and recommending firewall implementation for enhanced security against such threats. Moreover, Raymond et al. [7] explore social engineering attacks on Android apps by modifying code through reverse engineering, aiming to highlight vulnerabilities and countermeasures using tools like Kali Linux and Metasploit.

However, while extensive research has explored the malicious use of social engineering, there is a noticeable gap in studies examining the ethical application of these techniques for security purposes in the context of Android devices. This gap is particularly significant considering the popularity of Android devices [8], plus the potential for ethically designed software to enhance security protocols by simulating attacks and identifying vulnerabilities before malicious actors can exploit them. Although the existing literature extensively covers social engineering's unethical applications, it pays limited attention to leveraging social engineering to proactively identify and neutralize security threats, particularly in Android applications. The present study aims to address this gap by integrating social engineering methods into the design and functionality of a special Android application, which sheds new light on the dual-use nature of these techniques, thereby advancing the current understanding of how these techniques, and offering a framework for their ethical application in cybersecurity.

The primary contribution of this research is to present an approach for developing an Android application with a customizable UI, called Ethical Data Collector (EDC). This application employs social engineering techniques in an ethical form, to collect some kinds of valuable data stored in an Android device without target knowledge. Targeting an individual using EDC only requires customizing the UI layouts, without modifying core code. Our research findings offer both academic contributions and practical insights, suggesting that applications leveraging this approach could be valuable in cybersecurity contexts. By analyzing the collected data, security professionals may uncover the true identities of inexpert cyber criminals or blackmailers, thereby enhancing security measures through innovative yet ethical means.

2. IMPLEMENTATION METHODOLOGY

The development process of EDC does not require any complex arithmetic or logical operations. It simply requires:

1- Designing the visible contents of the app based on an attractive subject, in a way that deceives and entices the target to download and run EDC. This can be achieved effectively by displaying a fake UI, designed by utilizing social engineering tactics, remaining visible for a period, and effectively engaging and preoccupying the target.
2- Implementing the background processes that properly perform the collection and sending of data silently to a designated server through an internet connection.

In accordance with the above, the core function of EDC is to: collect some personal data stored on an Android-based device; and transmit it silently to a designated server via the Internet, while the target user interacts with the UI. The development process of EDC begins with identifying the core functions and tasks, followed by design and implementation stages. The generated app then undergoes testing in real-world scenarios across various Android versions, to evaluate its effectiveness and discuss the results. Regarding the collected data analysis processes, it is not within the scope of this work, which focuses primarily on constructing EDC.

Before starting the design and programming process, setting up an appropriate IDE is important. In this work, Android Studio IDE (Integrated Development Environment) was chosen as it is the official IDE for creating Android applications [9]. For programming, there are several languages for developing Android apps such as Java, Kotlin, and C++. We intend to build this application with Java, to produce a small APK (Android Application Package) file size [10],

which is one of the critical points that will be seen when testing EDC. To support a wider range of Android devices to run this app, level 16 is the minimum SDK (Software Development Kit) version for developing this application, while the targeted SDK will be level 29. This choice is due to its compatibility with 100% of supported devices, as reported by Android Studio.

3. EDC'S CORE FUNCTIONS

As described later in Figure 1, core functions will be divided as follows:

- Application-side:
  1- Graphical part:
    - Display a fake UI layout on the screen for a period.
    - Request the required permissions from the user to perform the application's tasks.
  2- Background-logic part:
    - Handle the result of the requested permissions.
    - Collect the data, including:
      - Device info (model, manufacturer, Active cell phone number).
      - A certain number of Images.
    - Send the collected data via HTTP (Hypertext Transfer Protocol) or HTTPS (Hypertext Transfer Protocol Secure) connections over the internet to be stored on a designated server.
- Server-side: Receive the transmitted data; store it on the server in the appropriate folders, then send an acknowledgment, or notify if an error occurs.

In EDC, the data collecting task is a scalable process, which can encompass collecting additional data kinds as the case requires, and developer intentions, without impacting UI design. However, in this paper, the exclusive focus is on the mentioned data kinds, for test purposes.

The implementation of server-side processes can vary, customized based on several factors like needs and developer preferences, such as using custom, paid, or free services as a server for hosting the backend code to receive the data. However, this study will not delve into detailed explanations of these technical details. Rather than constructing a dedicated server, a free hosting service such as 000webhost will be utilized effectively for testing purposes in this study.

It is crucial to note that all the mentioned processes should only be executed during EDC's first launch, not subsequently. This is because sending the same data multiple times from the same device is not helpful. To ensure this, the application should check if it is the first launch. A simple approach to achieve that is executing all collecting and sending processes during handling permissions results when all have been granted as illustrated in Figure 1. Or, by creating a Boolean reference, called "launch-state" that stores appropriately. For the second option, the SharedPreferences interface is a commonly used method for storing local data in Android applications [11]. The default value of the "launch-state" reference should be set to false, indicating that the app has not been launched before. Later, when the application is launched, this value will be updated to true.

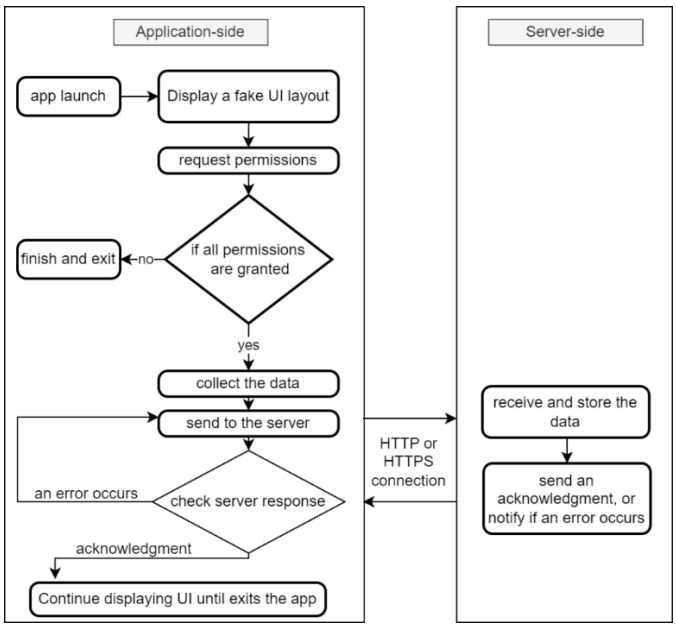


FIGURE 1. A flowchart illustrates EDC's Core-functions workflow.

4. UI DESIGN

4.1 NAME AND THEME

Application name is often the first interaction users have, and usually signifies the primary purpose of the app [12]. A well-chosen name can:

1. Create a positive impression.
2. Generate curiosity or interest.
3. Enticing users to explore further.

Based on the previous points, EDC should have an attractive name, matched with an interesting theme. Understanding the personality of the target (his predilections and preferences) plays a significant role in determining the adequate theme of EDC, therefore assigning the appropriate name for EDC. This is where the social engineering art can help us, by selecting the appropriate subject and name that convinces the target and encourages him to install and launch the application. For the app's theme and name, the concept of "free money" will be adopted as a theme for attracting the target [13]. Therefore, the application name will be "Watch and Earn", which means you can watch videos and earn money.

4.2 UI LAYOUT DESIGNING

Similar to the approach for app name assignment, the UI designing methodology in EDC is likewise connected to the use-case scenario and customization requirements, employing social engineering tactics. It is important to emphasize that the UI design should visually and functionally align with the overall concept of the application theme and name, enhancing trustworthiness. Therefore, constructing a tailored UI layout for EDC must meet the following requirements:

- Being attractive and legitimate to gain trust.
- Being lightweight as much as possible.
- Engaging the target with some activities to provide sufficient time for background processes to be done.
- Being congruent with the application theme and name.

This study will be content with constructing a simple UI design that aligns with the application's objectives and allows sufficient time for data collection and transfer. Implementing a simulated (fictitious) registration process (create an account), complete with various screens and intentional delays, will adequately enable EDC to perform its tasks seamlessly in the background while the user is engaged with the UI. When designing a UI for EDC, it is preferred to be as simple as possible. UI layouts with less memory will decrease the APK file size, where APK file size is considered a significant factor in motivating the target to install the app rapidly. The fake registration process will include five activities, as its layouts design is shown in Figure 2.

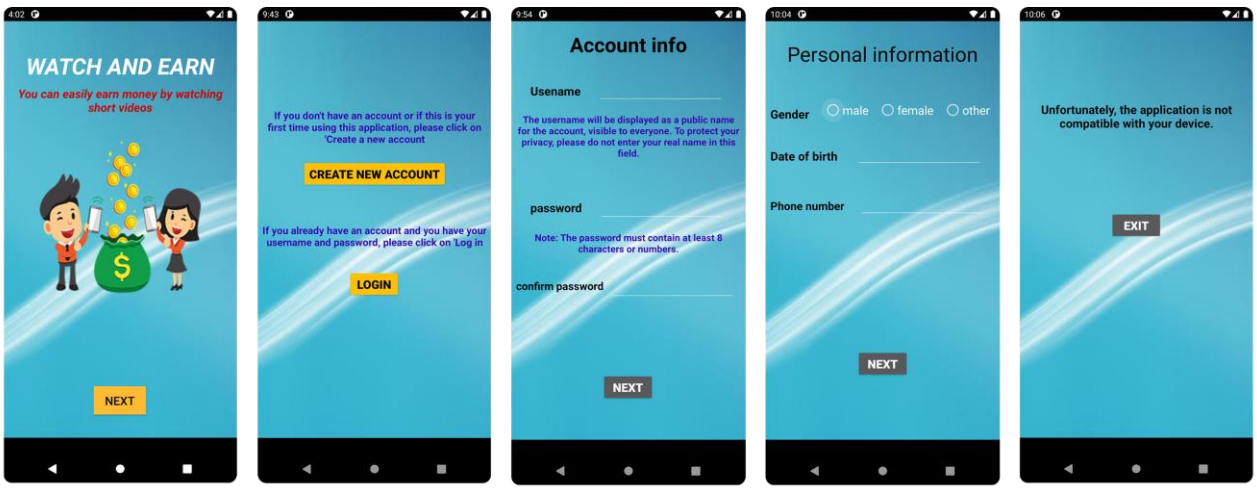


FIGURE 2. UI layouts of the fake registration process (This UI consists of five activities as shown. The first activity is the welcome screen. The other activities require the user to interact and engage to input his data, while background processes collect and send the data. The last activity for notifying the user of the incompatibility problem.)

During the transition between activities, as the user progresses from one activity to another, EDC will implement a consistent delay period (8000 milliseconds in this study for testing purposes) indicated by a progress bar, along with a special message (Toast) informing the user to wait until a connection with the server is established. The cumulative estimated time for all activities is expected to be 57 seconds, considering the utmost effort to complete the entire process from the initial activity to the final one, where this will be sufficient to collect and send some data. Extending the delay period will undoubtedly result in a greater amount of fetched data, particularly for images, but it will also inconvenience the user. As illustrated in Figure 2, The last activity provides an option to exit from the application, where the journey ends by claiming that the app is incompatible with the target device.

5. PERMISSIONS HANDLING

Android permissions are authorization settings that control an app's access to device features and data [14]. Permissions play a crucial role in the Android operating system, where Android employs a model that requires apps to request and obtain permission from the user before accessing sensitive resources or performing certain actions [15]. Based on that, obtaining the required permissions stands as a critical procedure in EDC and serves as the initial step toward initiating the actual functionality of the application. Without successfully obtaining these permissions, EDC is unable to collect or send any data.

As we mentioned before, EDC's theme, name, and UI design should be persuasive for the target to gain the necessary permissions from the target, where the higher the persuasion rate, the higher the chances of obtaining the required permissions. In addition to that, user awareness is another key factor. The awareness regarding permissions among users is still low, and Android users often agree to app permissions without fully understanding [16][17]. According to a previous study, overall, 84% of the permission requests were accepted by Android users [18].

Android implements a sophisticated permissions model to enhance user privacy and security, categorizing permissions into several types, including install-time permissions and runtime permissions. Install-time permissions, granted during app installation, provide access to less sensitive data and functionality. In contrast, runtime permissions, also known as dangerous permissions, are particularly significant as they grant additional access to restricted data or allow for the execution of restricted actions [19].

According to Google's Android developer documentation, the runtime permissions model is designed to provide users with greater control and transparency. This system requires explicit user consent for access to sensitive information and device features, such as the camera, contacts, or location data. When an app requests such permissions, it must present a prompt in context, providing a clear explanation of why the permission is needed. For example, when an app requests access to the device's location, users receive a notification detailing the necessity of this access. The Android system provides options for users to grant or deny these permissions, and importantly, users can modify their decisions at any time through the device's settings. This flexibility allows users to maintain control over their data and device features throughout their interaction with various apps.

In Android 5.1 (Lollipop) and lower, dangerous permissions are granted during app installation, while in Android 6.0 - 9, the permissions model for Android applications was redesigned, where users grant dangerous permissions while the app is running. Starting from Android 10, users see increased transparency and control over activity recognition (AR) runtime permissions [20].

To enable EDC to collect and send the mentioned data types in Sec. 3, the required permissions must be included in the Manifest file, which are:

1- For collecting data:
   - READ PHONE STATE (dangerous permission)
   - READ PHONE NUMBERS (dangerous permission)
   - READ EXTERNAL STORAGE (dangerous permission) (d) WRITE EXTERNAL STORAGE (dangerous permission)

2- For sending:
   - ACCESS NETWORK STATE (normal)
   - INTERNET (normal)

It is clear to see that all four permissions required for collecting the data are classified as dangerous permissions, which necessitate runtime permission requests, while the others are categorized as normal permissions according to the Android Developers website.

As stated in Sec. 2, EDC's minimum SDK is level 16, which corresponds to Android 4.1 "Jellybean", and level 29 is the target one. Consequently, the application supports installation and running on either "Jellybean" or a higher version, depending on the user's device, which necessitates handling different permission models. In this scenario, EDC needs to initially check the SDK version (whether it is lower, equal to, or higher than SDK level 23 "Marshmallow") of the user's device to determine the appropriate approach for handling permissions. So, EDC's required permissions handling process will follow two different paths based on the device's SDK version, as illustrated in Figure 3.

It is worth noting that sometimes users choose to decline the grant of runtime permissions to see if they can avoid granting them. In such situations, EDC should finish its activities and exit without any results, as shown in Figure 1. This 'Exit process' serves as a necessary motivation to prompt the user to reopen the app again, and grant the required permissions, especially when the user is excited about exploring the app.

In the latest Android versions (SDK level 23 or higher), the permissions dialog will be hidden after the user declines it once or twice according to Android developers' documentation. To prevent the permissions dialog from being hidden by the Android OS during the next app launch, EDC will perform a complete reset, clearing all its data when exiting, to restore the app to its initial launch state, as shown in Figure 3. This process requires the inclusion of the 'CLEAR APP USER DATA' permission in the Manifest file to enable the app to clear its data according to the Android Developers website.

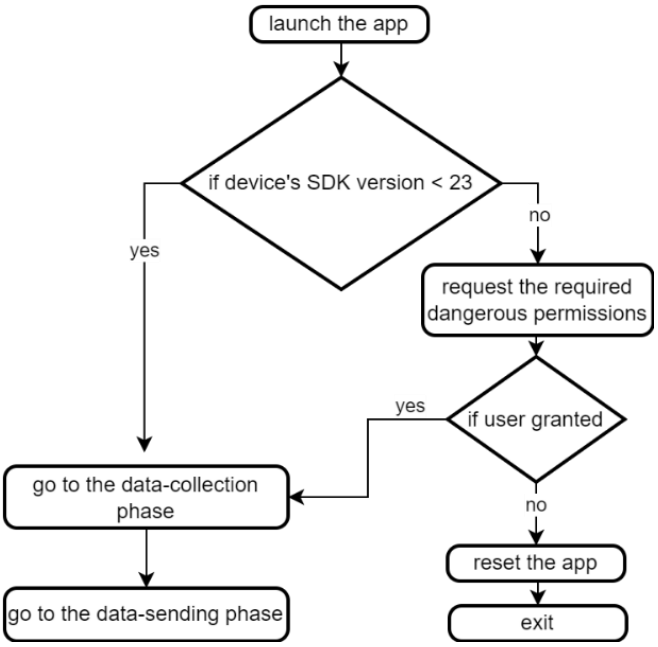


FIGURE 3. A flowchart of the Permissions-handling process in EDC

6. COLLECT AND SEND THE DATA

6.1 COLLECTING PROCESS

After the target grants the required permissions, the way becomes clear for the data collection process. To obtain the device's model and manufacturer as strings, a common class provided by Android called 'Build' can be utilized. This class can extract information about the current build from system properties according to the Android Developers website.

When it comes to retrieving the phone number, the capabilities of the TelephonyManager class can be leveraged. This class provides access to essential information and states related to telephony services. The process entails utilizing the 'getLine1Number()' method from the TelephonyManager class to retrieve the phone number associated with the device as a string according to Android developers' documentation, as shown in Figure 4.

To collect a particular number of images, EDC will utilize the MediaStore API, which provides an indexed collection of common media types such as Images from any attached storage devices according to the Android Developers website. This process involves systematically extracting file paths of available externally stored images by querying using a 'Cursor'. The extracted file paths are then employed for the transmitting process. This method ensures an organized approach to image collection, facilitating their subsequent upload to a remote server, as described in Figure 4.

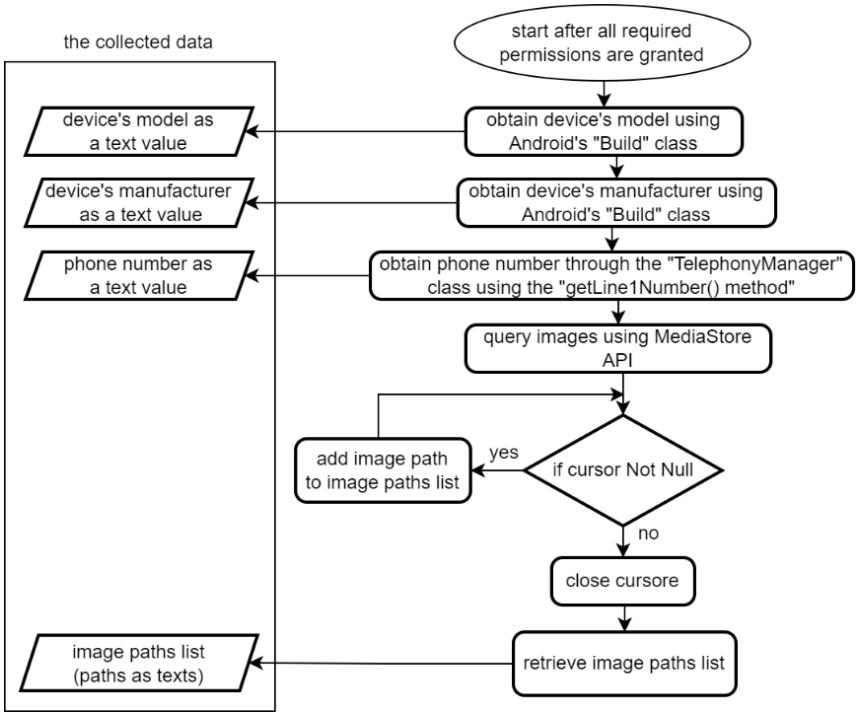


FIGURE 4. A flowchart illustrates the data-collecting phase in EDC.

6.2 SENDING PROCESS

After completing the data collection phase, the subsequent stage involves data transmission. The required data, including the device's model, manufacturer, phone number, and images, is transmitted in this phase.

To achieve this efficiently, a multi-threading approach (Runnable interface structure) is adopted. This implementation strategy involves the creation of dedicated threads for sending each kind of data, even for each image. This approach ensures efficient execution, particularly in some scenarios where one of these pieces of information is unavailable. By employing separate threads, the application can continue sending the available data, without being impacted by the absence of any specific piece of information. The process of sending the collected data involves the following key steps:

1. Check network and internet connection availability using the "ping" method. If unavailable, notify the user through a message "Toast" about the connection's necessity to complete the registration process in the app.
2. A temporary directory is created, or an existing folder is determined, to house all files that will be created later.
3. Several files are created to hold the required data. For textual types (device manufacture, model, and phone number). The file name should indicate the nature of the contained information. For the image, its original name will be adopted. In addition, a prefix number (acting as a user ID) will be appended to the transmitted file name for user identification on the server side, where this prefix can be generated during the application launch or sending phase.
4. For textual types (device manufacture, model, phone number), the obtained data is written directly to the file as bytes. The collected images will undergo a transformation process, including decoding and compression to the 'JPEG' format, and then written to new files (with their original names) as bytes, as illustrated in Figure 5. This transformation employs Android's 'Bitmap' and 'BitmapFactory' classes to instantiate a lightweight image for fast-speed transmission over the internet.

5. To avoid the device hanging and UI freezing during the sending phase, the sending procedure should not be implemented in the UI thread. Instead, the app will create an AsyncTask () for sending every file separately in the background.
6. Establish a connection with the designated server through HTTP or HTTPS, based on the device's Android SDK version, as shown in Figure 5. HTTP connection transmits data in plaintext, while HTTPS connection encrypts data for secure transmission [21]. This means every AsyncTask () has its own connection to send its related data. On the server side, it prefers to use two separate PHP pages for handling textual types and images, therefore the URLs used for connection will be different, one for sending textual data, and the other for images.
7. Read the file and send its contents to the server as part of a multipart form-data POST request. The request includes boundary and file information.
8. Receive a response message from the server to ensure receipt (an acknowledgment). In case of a negative response, such as an error, the app should resend data, as illustrated in Figure 5.
9. If an acknowledgment has been received from the server, delete the created directory and files, to free up its memory and remove evidence of its existence. Then, close the connection, and change the value of "launch-state" in SharedPreferences to true to ensure that the collected data is not sent multiple times for the same device as explained in the third section.

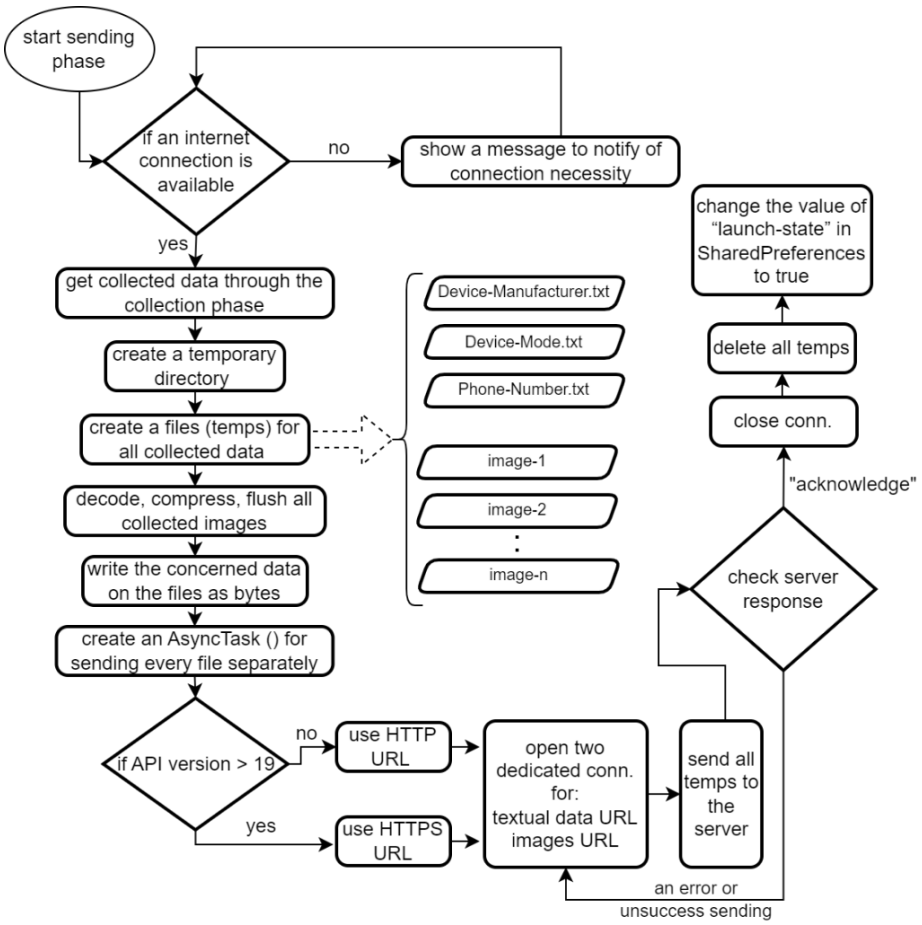


FIGURE 5. A flowchart illustrates the data-sending process in EDC. (after granting all required permissions, this process will begin directly)

6.3 RECEIVING PROCESS

As mentioned before in Sec. 3, server-side implementation can vary and be customized. In the EDC situation, and for testing purposes, a free service like 000webhost can be utilized as a PHP server without any costs.

The receiving process on the server side involves several key steps to handle incoming data efficiently, where two PHP scripts are utilized to manage uploaded files: one for receiving textual data (textual_data_receiving.php) and another for image data (image_data_receiving.php). Both scripts are identical and perform the same job, as illustrated in Figure 7. The use of separate PHP scripts for textual and image data, despite their current functional similarity, is primarily for performance optimization. This separation allows for more efficient handling of unsuccessful data transmissions and reduces the need for re-sending processes.

Dedicated endpoints for each data type allow the server to process and respond to uploads more quickly, minimizing latency and improving overall system responsiveness, especially when dealing with large volumes of data or potential network issues. Textual files such as device manufacturer, model, and phone number are often small in size. Using a separate URL allows for quick delivery and receipt isolatedly from image files, which take more time to send. These scripts are accessed via specific URLs, which are used during the initiation of HTTP or HTTPS requests on the application side, as shown in Figure 7.

Upon receiving data, the server first checks if a file with a specific name was uploaded via an HTML form. If no file is detected, the script echoes "No file uploaded" to the application. For successful uploads, the process continues as follows:

1- The script specifies a target directory (e.g., "uploads/") where the files will be stored.
2- It then attempts to move the uploaded file from its temporary location to the target directory.
3- If the file move is successful, the script echoes "file was saved successfully" to the app.
4- In case an error occurs during the file move, an error message along with the error code is echoed back to the app as illustrated in Figure 6.

This process ensures that all received files, whether text or image-based, are properly stored and accounted for. The server's response to each upload attempt is crucial for the application to confirm successful receipt or to handle any errors. This two-way communication ensures reliability in the data transfer process.

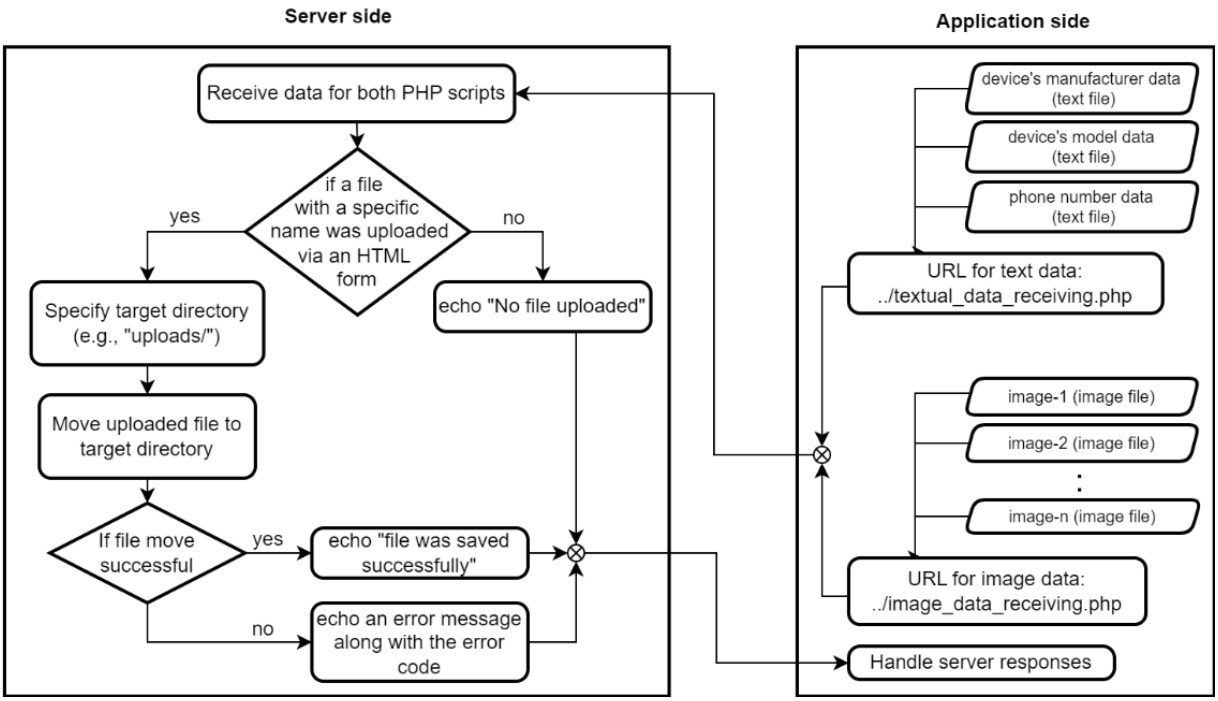


FIGURE 6. A flowchart illustrates the data-receiving process on the server side.

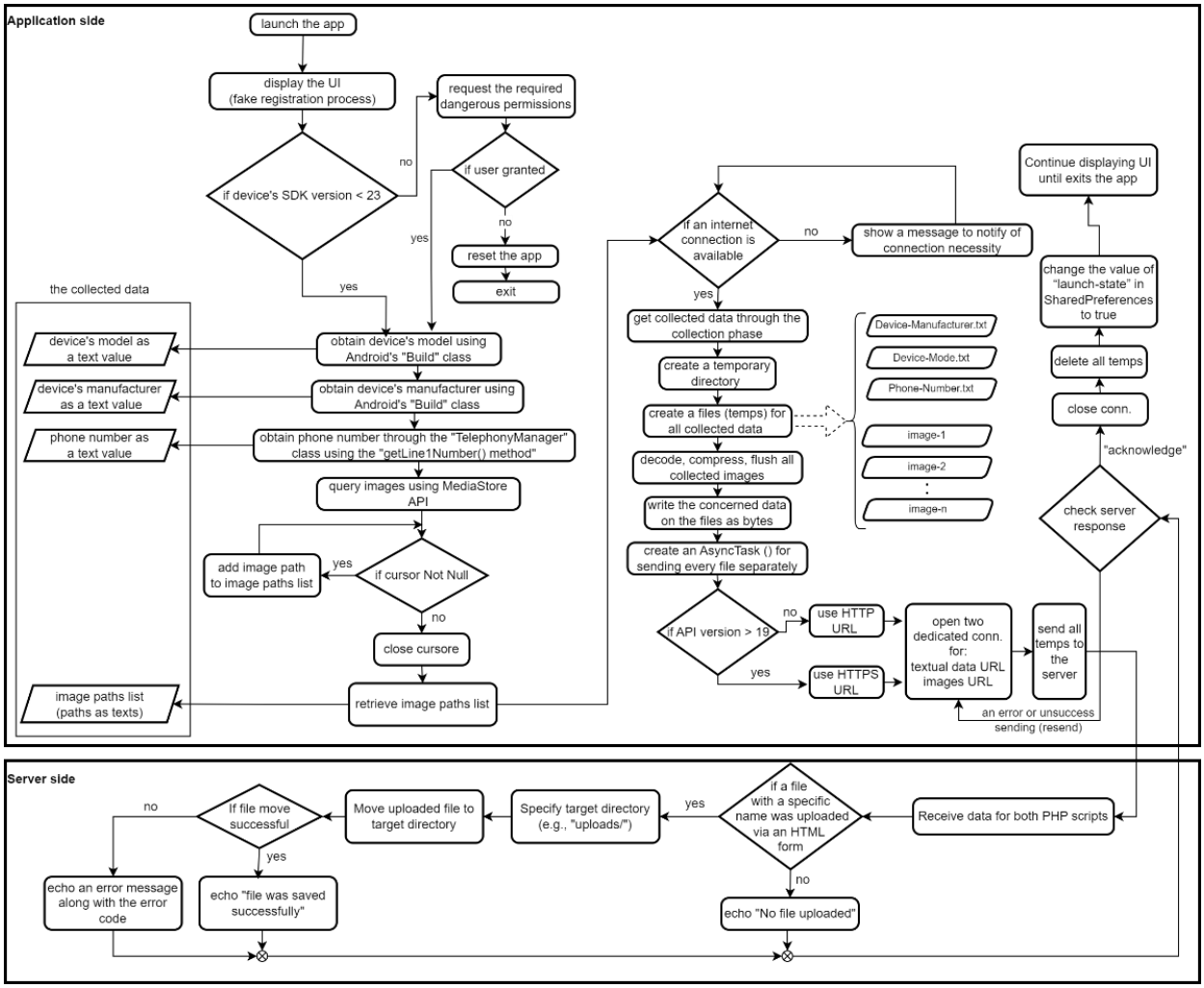


FIGURE 7. A comprehensive flowchart illustrates all EDC's processes, including server-side operations.

7. EDC TESTING

In this section, EDC will undergo comprehensive testing across various Android versions, ranging from SDK level 16 to level 29. This approach ensures that the testing accurately reflects user experiences across different Android versions, regardless of how the app is sent to the user. But before conducting the test, an APK file for EDC should be built by Android Studio. After building, the APK file (which was 4MB in size) was ready for testing.

The testing environment includes:

- A laptop computer (Lenovo Yoga 910-13IKB) will be used for installing and running the Android emulator, with the following specifications:
  - processor: Intel Core i7-7500U
  - installed RAM: 8.00 GB.
  - powered by: Windows 10 (64-bit operating system).

- Android Studio IDE will be used for utilizing its official emulator as an Android device, with the following specifications:
  - version: 2022.3.1 (Giraffe).
  - used build Gradle version: 8.1.0.
  - used virtual device (android device emulator): 13 Android device emulators have been created with different versions (from SDK Level 16 "Jellybean" to SDK Level 29 "Quince Tart"), to test EDC across multiple Android versions.

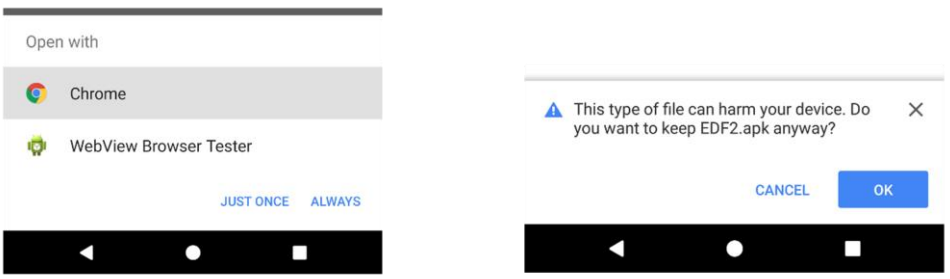
7.1 DOWNLOADING TEST:

In our testing approach, the APK file of EDC was uploaded to a free server (such as 000webhost) to obtain a URL for downloading the APK file inside the targeted device. Then, the URL of the file was sent to all 13 emulators via an SMS message, where this step simulates an authentic scenario for targeting a real user. This could also be achieved in real-world scenarios through WhatsApp, Facebook Messenger, or any other social network with chat capabilities.

After all 13 emulators received the SMS message and clicked on the URL, the downloading procedure was as follows:

- When downloading EDC's APK file on an Android emulator with Android version 6.0 or lower (SDK Level 23 down to Level 16), there are no security alerts presented to the user, and the download process begins directly.
- When downloading EDC's APK file on an Android emulator with Android version 7.0 and higher (SDK Level 24 up to Level 29), the Android operating system prompts the user to open the downloading URL using the available browser installed on the device (such as Google Chrome), as shown in Figure 8-a. After choosing a web browser, a security alert appears as shown in Figure 8-b. This differs from the download process for Android version 6.0 or lower.

Ultimately, the downloading test revealed important differences in the downloading process across Android versions. For devices running Android 6.0 or lower, the lack of security alerts during the download process potentially facilitates easier app distribution. In contrast, Android versions 7.0 and higher introduce additional steps, including browser selection and security alerts, which may require more understanding of the target user's preferences to choose an appealing subject and generate interest in downloading EDC.



A- choose from the available browsers to open the download link *B- security alert notifying before starting the download*

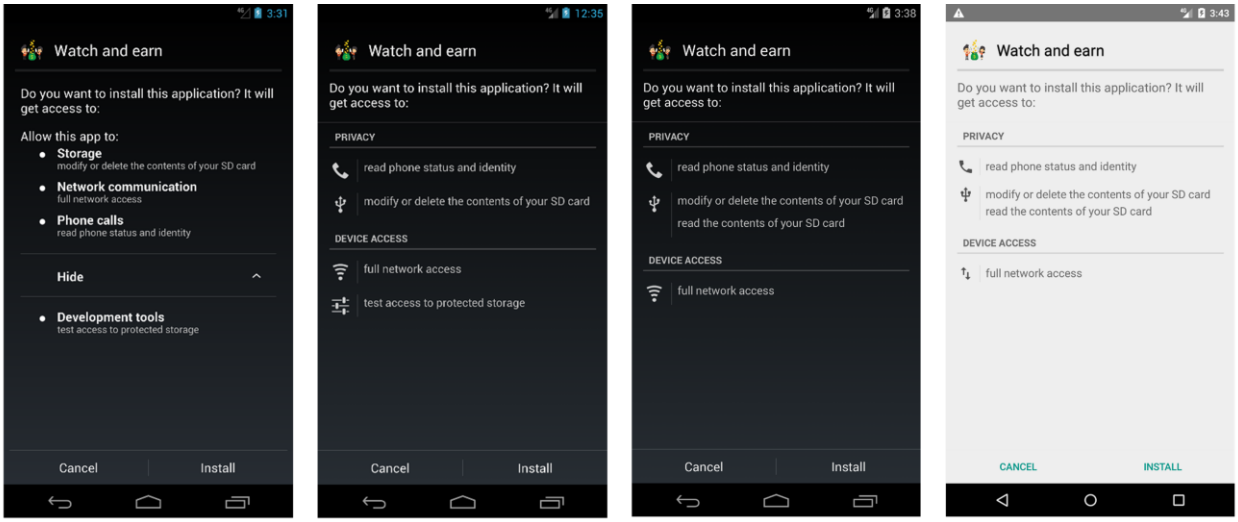
FIGURE 8. EDC's downloading process on Android version 7.0 and higher.

7.2 INSTALLING AND LAUNCHING TEST:

When running the downloaded APK file to install EDC on Android version 5.1 or lower, the permissions window appears directly as shown in Figure 9, because the normal and dangerous permissions are granted simultaneously during app installation as explained in Sec. 5.

When the permissions are granted during installation, and then launch the app, it successfully fetches and sends the data (model, manufacturer, phone number, and images) to the server. The number of uploaded images primarily depends on the compression factor applied, plus device speed, and internet connection speed. A higher compression factor value allows to transmission of more images, though at a reduced quality. Also, engaging time with the fake UI (registration process) is a crucial factor for maximizing the number of uploaded images. When measuring the cumulative time for that process, the estimated time was 57 - 60 seconds considering the utmost effort to complete the fake registration process.

Compared to older Android versions, installing EDC on 6.0 and higher involves directly installing the app without a permissions dialog showing. But, when launching the app for the first time, the dangerous permissions dialog will appear directly as illustrated in Figure 10, because on newer Android versions, dangerous permissions must be granted during runtime as explained previously in Sec. 5. After permissions are granted by the user, EDC successfully collects and sends the data (model, manufacturer, phone number, and images) to the server.



A : EDF's permissions dialog on Android 4.1 | B : EDF's permissions dialog on Android 4.2 and 4.3 | C : EDF's permissions dialog on Android 4.4 | D : EDF's permissions dialog on Android 5.0 and 5.1

FIGURE 9. EDC's permissions dialogs that appear on Android version 5.1 or lower.

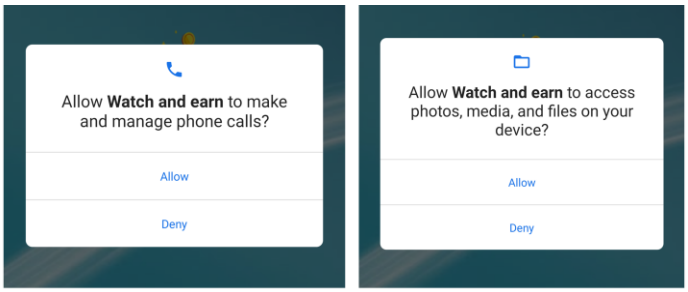


FIGURE 10. EDC's permissions dialogs that appear on Android version 6.0 or higher.

The installing and launching test provides insights into EDC's behavior across different Android versions, highlighting the variations in the downloading procedure and permission handling.

For Android version 5.1 and lower, the test reveals that both normal and dangerous permissions are requested simultaneously during installation, presenting users with a comprehensive permissions window. This allows EDC to immediately access and transmit sensitive data upon launch, including device information and images. The test also explores factors affecting data collection efficiency, such as image compression, device speed, and internet connection.

In contrast, for Android 6.0 and higher, the installation process occurs without an initial permissions dialog. Instead, the app requests dangerous permissions at runtime when first launched. This change in Android's permission model affects how EDC operates, potentially influencing user perception and the app's ability to collect data immediately.

Despite these differences, the test demonstrates that once permissions are granted, EDC successfully collects and transmits the targeted data (model, manufacturer, phone number, and images) to the designated server across all tested Android versions. This consistency in data collection capabilities, regardless of the Android version, underscores the effectiveness of the application's design in achieving its data-gathering objectives across a range of Android environments.

7.3 LIMITATIONS AND POTENTIAL DRAWBACKS

While EDC demonstrates the potential for ethically using social engineering techniques to collect data discreetly, several limitations and potential drawbacks should be considered, as follows:

1- EDC's success depends on the target user granting the necessary permissions. If permissions are denied, the application cannot function as intended. If permissions are denied, the application cannot function as intended.
2- EDC performance may vary across different devices and Android versions, potentially affecting data collection efficiency.

3- Ethical considerations regarding the use of deception, even for positive purposes, must be carefully managed to avoid misuse.
4- EDC's APK file size is another significant factor. Increasing file size can decrease download speed and discourage quick installation.
5- Understanding the target user's personality (including their predilection and preferences) plays a significant role in encouraging downloading and running EDC. Choosing subject matters, an appropriate app name, and an attractive user interface appearance are crucial factors to appeal to the user's interests and elicit enthusiasm. Without considering these factors, the app may not perform its job effectively.
6- The target's attention should be held for as long as possible to allow sufficient time for background processes in EDC, where engaging time with the UI is a significant factor in maximizing the uploaded data.
7- The UI's visual and functional design should align with the overall concept of the application name to enhance trustworthiness.
8- To target a specific user, the authorized developer is required to customize UI layouts for the application based on the use case scenario.
9- EDC downloading on Android version 6.0 or lower initiates directly without any security alerts presented to the user, which provides a higher chance of installing and running the app. In contrast, on Android version 7.0 and higher, a security alert appears before the download starts, which reduces its chances of being operated.
10- Granting the required permissions is crucial for EDC to access needed data, where this relies mainly on the target's awareness. Without granting the required permissions, EDC cannot collect or send any data.
11- Data-sending processes in EDC depend on the availability and speed of the internet connection on the target device for transmitting collected data via an HTTP or HTTPS connection.
12- The choice of using HTTP or HTTPS depends on compatibility with the target device, which means no chance of sending data over HTTPS for older devices.
13- The number of collected images primarily depends on the compression factor applied and device speed. Using a higher compression factor allows transmitting more images, though at a reduced quality.

8. CONCLUSIONS

Through implementing and testing EDC, it became clear that social engineering techniques can be ethically employed to develop an Android application that can collect some personal information such as cell phone numbers and images. The idea behind EDC is to deceive bad actors by engaging them with a fake yet attractive scenario. While occupying the target user, the application collects and silently sends types of stored data like the active phone number and images to a designated server via an internet connection in the background.

Analyzing the collected data could help identify the true identities of cyber criminals or blackmailers. Understanding the target user's personality (including their predilection and preferences) plays a significant role in customizing EDC based on the use-case scenario requirements. Choosing subject matters that appeal to the user's interests and elicit enthusiasm for downloading and running the app is important. Relevant factors include the app name and user interface appearance. EDC should have an attractive name, and the UI should be lightweight, appealing, and hold the target's attention for as long as possible to allow sufficient time for background processes. Importantly, the UI's visual and functional design should align with the overall concept of the application name to enhance trustworthiness. APK file size of EDC is another significant factor, where minimizing the size can boost download speed and motivate the target to install rapidly.

To ethically target an Android user, the authorized developer is required to customize user interface (UI) layouts for EDC without modifying its core code logic. EDC downloading on Android version 6.0 or lower initiates directly without any security alerts presented to the user, which provides a higher chance of installing and running the app. In contrast, on Android version 7.0 and higher, a security alert appears before the download starts, as illustrated in Figure 8. Granting the required permissions is crucial for EDC to access needed data, where this relies mainly on the target's awareness. Without granting the required permissions, EDC cannot collect or send any data, as described in Figure 7.

Data-sending processes in EDC depend on the availability and speed of the internet connection on the target device for transmitting collected data via an HTTP connection. The number of collected images primarily depends on the compression factor applied and device speed. Using a higher compression factor allows transmitting more images, though at a reduced quality. Engaging time with the fake UI is a significant factor in maximizing the number of uploaded images. For receiving data sent by EDC, lower-cost alternatives to custom server implementation include utilizing free web hosting. Services like 000webhost or InfinityFree can be used for server-side scripting, at no monetary cost.

9. FUTURE WORKS

While this study has demonstrated the feasibility and ethical potential of the Ethical Data Collector (EDC) application in collecting sensitive data through social engineering techniques, several avenues for future research and development could further enhance its effectiveness and applicability.

- Expansion of Data Collection Scope: Future iterations of the EDC application could explore the inclusion of additional data types, such as geolocation, contact lists, or behavioral analytics. This expansion would allow the app to gather a more comprehensive set of data for profiling potential cybercriminals, thereby enhancing the identification process.
- Adaptive User Interface (UI): Enhancing the adaptability of the EDC's UI to better align with individual user behaviors and preferences could increase the success rate of data collection. This could involve integrating machine learning algorithms to dynamically modify the UI based on real-time user interactions, optimizing the engagement time for more efficient data capture.
- Integration with Advanced Security Frameworks: The application could be integrated with advanced security frameworks or AI-driven threat detection systems to provide real-time analysis of the collected data. This would enable quicker identification of threats and potentially automate the process of neutralizing them.
- Cross-Platform Development: Expanding the EDC application to other mobile platforms such as IOS would increase its applicability and effectiveness. This cross-platform development could also provide insights into differences in user behavior and security measures across different operating systems.
- Ethical Considerations and Policy Development: As the application of social engineering techniques for ethical purposes continues to evolve, it will be crucial to develop clear guidelines and policies to govern their use. Future research should focus on establishing a framework that balances the benefits of these techniques with the ethical implications, ensuring that they are used responsibly and with proper oversight.
- Real-World Testing and Validation: Conducting real-world tests of the EDC in various environments and against diverse user profiles would provide valuable data on its effectiveness and limitations. This could involve collaboration with cybersecurity professionals in controlled settings to validate the app's performance and refine its functionalities.
- User Awareness and Education: Finally, future work should consider the development of educational programs or tools to raise awareness about social engineering tactics and how individuals can protect themselves. These programs could be integrated into the EDC framework to provide dual functionality: collecting data for security purposes and educating users on cybersecurity best practices.

Funding

None

ACKNOWLEDGEMENT

None

CONFLICTS OF INTEREST

The author declares no conflict of interest.

REFERENCES

[1] Z. Wang, H. Zhu, P. Li, and L. Sun, "Social Engineering in Cybersecurity: A Domain Ontology and Knowledge Graph Application Examples," *Cybersecurity*, vol. 4, no. 1, 2021, pp. 1-21. DOI: 10.1186/s42400-021-00094-6.

[2] F. Mouton, M. M. Malan, L. Leenen and H. S. Venter, "Social Engineering Attack Framework," *2014 Information Security for South Africa, Johannesburg, South Africa,* 2014, pp. 1-9. DOI: 10.1109/ISSA.2014.6950510.

[3] F. Salahdine and N. Kaabouch, "Social engineering attacks: a survey," *Future Internet*, vol. 11, no. 4, pp. 1-10, 2019. DOI: 10.3390/fi11040089.

[4] Ş. A. Duman, R. Hayran, and İ. Sogukpinar, "Impact Analysis and Performance Model of Social Engineering Techniques," *2023 11th International Symposium on Digital Forensics and Security (ISDFS)*, Chattanooga, TN, USA, 2023, pp. 1-6. DOI: 10.1109/ISDFS58141.2023.10131771.

[5] R. Satrio Hadikusuma, L. Lukas, and E. Rizaludin, "Methods of Stealing Personal Data on Android Using a Remote Administration Tool with Social Engineering Techniques," *Ultimatics: Jurnal Teknik Informatika*, vol. 15, no. 1, pp. 44-49, June 2023. DOI: 10.31937/ti.v15i1.3122

[6] E. Blancaflor, H. K. S. Billo, B. Y. P. Saunar, J. M. P. Dignadice, and P. T. Domondon, "Penetration Assessment and Ways to Combat Attack on Android Devices Through StormBreaker - A Social Engineering Tool," *2023 6th*

International Conference on Information and Computer Technologies (ICICT), Raleigh, NC, USA, 2023, pp. 220-225. DOI: 10.1109/ICICT58900.2023.00043.

[7] J. Raymond and P. Selvaraj, "An Effective Approach to Explore Vulnerabilities in Android Application and Perform Social Engineering Attack," in *Proceedings of International Conference on Deep Learning, Computing and Intelligence*, G. Manogaran, A. Shanthini, and G. Vadivu, Eds., vol. 1396, *Advances in Intelligent Systems and Computing*, Singapore: Springer, 2022, pp. 349–360. DOI: 10.1007/978-981-16-5652-1_32.

[8] Statcounter Global Stats, "Mobile Operating System Market Share Worldwide." Accessed: May 30, 2023. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[9] E. Wihidayat, "Pengembangan aplikasi android menggunakan integrated development environment (ide) app inventor-2," *Edutic - Scientific Journal of Informatics Education*, vol. 4, no. 1, 2017. DOI: 10.21107/edutic.v4i1.3229.

[10] B. P. D. Putranto, R. Saptoto, O. C. Jakaria and W. Andriyani, "A Comparative Study of Java and Kotlin for Android Mobile Application Development," *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI),* Yogyakarta, Indonesia, 2020, pp. 383-388, DOI: 10.1109/ISRITI51436.2020.9315483.

[11] S. S. Kumbhar, Y. Lee and J. Yang, "Hybrid Encryption for Securing SharedPreferences of Android Applications," 2018 1st *International Conference on Data Intelligence and Security (ICDIS),* South Padre Island, TX, USA, 2018, pp. 246-249, DOI: 10.1109/ICDIS.2018.00047.

[12] N. Bu, S. Niu, L. Yu, W. Ma, and G. Long, "Bridging semantic gap between app names: collective matrix factorization for similar mobile app recommendation," *Web Information Systems Engineering – WISE* 2016, pp. 324-339, 2016, DOI: 10.1007/978-3-319-48743-4_26

[13] Y. Hu, H. Wang, L. Li, Y. Guo, G. Xu and R. He, "Want to Earn a Few Extra Bucks? A First Look at Money-Making Apps," 2019 IEEE 26th *International Conference on Software Analysis, Evolution and Reengineering (SANER),* Hangzhou, China, 2019, pp. 332-343, DOI: 10.1109/SANER.2019.8668035.

[14] O. Hussein, "Detection of integrity attacks on permissions of Android-based mobile apps: Security evaluation on PayPal," *International Journal of Computers and Information (IJCI),* vol. 11, no. 2, pp. 25-43, 2024, DOI: 10.21608/ijci.2024.277929.1156

[15] S. Li, W. Jiang, Y. Yao, and C. Xu, "Permission analysis based on android app store," Proc. SPIE, vol. 12718, *International Conference on Cyber Security, Artificial Intelligence, and Digital Economy (CSAIDE 2023)*, 127180Z, 2023, DOI: 10.1117/12.2681726.

[16] N. S. Awang Abu Bakar, "Users comprehension and behavior study on Android permissions," *IJPCC*, vol. 2, no. 2, pp. 1-15, Oct. 2016, DOI: 10.31436/IJPCC.V2I2.29.

[17] N. S. A. A. Bakar and I. Mahmud, "Empirical Analysis of Android Apps Permissions," 2013 *International Conference on Advanced Computer Science Applications and Technologies*, Kuching, Malaysia, 2013, pp. 406-411, DOI: 10.1109/ACSAT.2013.86.

[18] B. Bonné, S. T. Peddinti, I. Bilogrevic, and N. Taft, "Exploring decision making with Android's runtime permission dialogs using in-context surveys," in *Proc. 13th Symp. Usable Privacy and Security (SOUPS 2017),* Santa Clara, CA, USA, Jul. 2017, pp. 195-210.

[19] R. Baalous and R. Poet, "Factors Affecting Users' Disclosure Decisions in Android Runtime Permissions Model," 2020 IEEE *19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom),* Guangzhou, China, 2020, pp. 1113-1118, DOI: 10.1109/TrustCom50675.2020.00147.

[20] S. Wang, Y. Wang, X. Zhan, Y. Wang, Y. Liu, X. Luo, and S.-C. Cheung, "Aper: Evolution-aware runtime permission misuse detection for Android apps," *Proc. 44th International Conference Software Engineering (ICSE '22),* New York, NY, USA, 2022, pp. 125-137, DOI: 10.1145/3510003.3510074.

[21] S. Bistarelli, M. Ceccarelli, C. Luchini, I. Mercanti, and F. Santini, "Design and Implementation of a Covert Channel based on HTTP Headers," *IJCI. International Journal of Computers and Information*, vol. 11, no. 2, pp. 25-43, Apr. 2024. DOI:10.2139/ssrn.4822138.