A Technique to Steal OAuth Tokens in Android-Based Devices Using a Malicious Application

MSc Research Project
MSc. Cybersecurity

Rajendra Yashwant Topare
Student ID: 21222061

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rajendra Yashwant Topare		
Student ID:	21222061		
Programme:	MSc in Cybersecurity	**Year:**	2022-23
Module:	MSc Research Project		
Supervisor:	Mr. Vikas Sahni		
Submission Due Date:	18/09/2023		
Project Title:	A Technique to Steal OAuth Tokens in Android-Based Devices Using a Malicious Application		
Word Count:	5343	**Page Count**:	19

Signature: Rajendra Yashwant Topare

Date: 16/09/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A Technique to Steal OAuth Tokens in Android-Based Devices Using a Malicious Application

Rajendra Yashwant Topare
21222061

Abstract

The OAuth protocol is used to authorize and authenticate access to third-party services. But its implementation in Android-based mobile devices is vulnerable to attacks, which allows attackers to get unauthorized access to private information. This research demonstrates a new technique to capture OAuth tokens using a malicious Android application while accessing third-party applications using a Facebook login. It presents useful insights into the different strategies used by cybercriminals to obtain OAuth tokens and highlights flaws in existing security mechanisms. It highlights the need for the development of new countermeasures to mitigate these threats. These insights can be utilized by mobile app developers to improve app security while protecting user data.

Keywords: *Mobile Threats, OAuth tokens, Android Security.*

1 Introduction

By using Open Authorization (OAuth), a user can authorize a third party (such as a mobile app) to access the user's data on a resource provider (RP) by using the user's credentials. Due to OAuth, users don't have to trust third-party applications with their credentials in order to provide them access. Google, Facebook, and Twitter are just a few of the major SPs that give software development kits (SDKs) that can be integrated into mobile applications to make using their services effortless. Third-party applications often use OAuth to connect to the HTTP services of well-known Service Providers (SPs). OAuth implementations in Android applications are not always consistent with the recommended standard, primarily due to differences between providers and platforms and the various types of implementations that often leave apps vulnerable to attacks (Rahat, Feng and Tian, 2019).

The stealing of OAuth tokens on Android devices is a big threat to OAuth security. Android has become by far one of the most popular mobile operating systems for attackers to target, with over 3.3 billion active users[1] (Abdullah and Zeebaree, 2021). To gain access to sensitive information such as emails, social networking accounts, and bank account information, attackers may exploit gaps in Android applications or the operating system itself. Unfortunately, OAuth token thefts can happen invisibly, and users may be unaware of security vulnerabilities until it is too late. The compromised credentials or OAuth tokens can be used to gain access to user accounts and get involved in a variety of illegal activities such as data mining, spamming, and posting offensive information on social networking sites. To secure

[1] https://www.bankmycell.com/blog/how-many-android-users-are-there

user information and privacy, it is critical to research mobile issues such as the stealing of OAuth tokens on Android devices.

The purpose of this research study is to highlight the seriousness of the OAuth token hijacking issue on Android smartphones while accessing third-party apps. The OAuth protocol is often used to offer authorization and authentication for third-party services (Fett, Küsters and Schmitz, 2016). However, the way it's implemented in Android smartphones leaves it vulnerable to attack, enabling attackers to get unauthorized access to personal data. Despite widespread acceptance of the necessity of OAuth token security, there is still a significant gap in understanding of how to successfully secure tokens on Android devices. This study aims to identify the strategies that attackers can use to steal OAuth tokens from Android devices and to evaluate the effectiveness of current security measures in mitigating this threat. This study contributes to the world of mobile security by putting the spotlight on the present status of OAuth token security on Android devices and pointing out the need for more research to enhance it.

The research on OAuth token stealing on Android devices is critical because it increases awareness of the severity of this security issue and the possible risks associated with utilizing mobile apps that utilize the OAuth protocol. The research gives useful insights into the various strategies used by cybercriminals to steal OAuth tokens and reveals flaws in existing security mechanisms, which may be utilized when developing new countermeasures to reduce this risk. Mobile application developers can use the insights to enhance application security and protect user data. Furthermore, this study emphasizes the necessity for future research to address the security flaws in the OAuth protocol in order to enhance its implementation on Android devices. This study also adds to the area of mobile security by emphasizing the present status of OAuth token security on Android devices and highlighting the need for ongoing efforts to enhance mobile application security.

Research Question: How OAuth tokens can be stolen on Android devices?

2 Related Work

Many researchers have discovered a number of security issues associated with poor OAuth implementation. The protocol is multi-party and is intended for helping third-party authorizations and authentications. It combines password authentication with third-party delegated authentication. Significant security problems can be introduced by either the service provider (SP) because of an unsafe OAuth implementation or by developers incorrectly interpreting or abusing the protocol. Because of variations in operating systems, mobile app developers frequently do mistakes in their code. Due to a lack of complete security analysis, poor OAuth implementation is not going to have its security problems recognized and corrected. Without a comprehensive and thorough examination, even competent analysts may overlook various intricacies and minor flaws in an Android-based OAuth process.

One of the major security risks related to OAuth session handling highlighted by (Abdullah and Zeebaree, 2021) in their paper was improper session management. For session management, developers commonly use OAuth tokens, SSO, and cookies. However, if someone with malicious intent obtains these tokens, they can carry out activities as authorized users and get access to higher privileges. Mobile application vulnerabilities are caused by developers' lack of awareness of security and bad coding practices. Furthermore, Android's permission system has problems, and inappropriate permission system usage leads to vulnerabilities and data thefts. To prevent this issue, implement proper session management by verifying the user's identity through the backend, creating difficult-to-guess session tokens, and defining session timeouts. If they are correctly implemented, attackers might need to deploy new token-stealing strategies.

(Rahat, Feng and Tian, 2019) investigated for flaws in the design as well as implementation of dependent parties that use OAuth protocols, assuming that service providers use OAuth security techniques effectively. The research looked at authentication attacks, where an attacker acquired access to mobile apps on behalf of victims, and authorization attacks, in which malicious dependant parties attempt to get unauthorized access to user data. The article described TikTok, a popular application that connects user accounts from service providers (SP) such as Facebook, Instagram, and Google by applying the standard authentication mechanism using the OAuth 2.0 protocol. However, after running OAUTHLINT, the authors uncovered many flaws in the app's OAuth protocol implementation.

The activity-in-the-middle attack which was highlighted by (Xiao *et al.*, 2017) can occur during the OAuth 2.0-based Single Sign-On (SSO) process. In this attack, the attacker executes an undetectable malicious activity that prevents communication from the user's device system that should be forwarded to the Service Provider (SP) and transmits its own data. The attacker can subsequently utilize the access token to get access to the user's private information hosted on the SP's servers. If strong SSL setups are used, a secure connection between the user's device along with the service provider is ensured, making it harder for attackers to intercept or alter communications.

(Fett, Küsters and Schmitz, 2016) discussed several attack strategies such as State Leak attacks and Cross Social-Network Request Forgery. In OAuth 2.0, a state leak attack can allow session altering or login CSRF. The attack can cause a browser to be logged in at an RP or encourage a Relying Party (RP) to utilize an attacker-controlled website instead of a user-controlled resource by using the Referrer header in the HTTP request. The analysis also mentions that compromises of sensitive data by using the Referrer header are not uncommon, based on previous research.

When paired with IdPs that simply fail to validate the redirect URI, cross-social-network request forgery occurs. This applies to RPs with basic user intention monitoring. This research emphasized that even if an IdP carefully reviews redirect URIs, the IdP mix-up attack can be successful and that this class of attacks is not restricted to certain implementations, but rather a systematic vulnerability in the OAuth standard. It also disclosed the Malicious Endpoints Attack, which is limited to OpenID Connect and uses the OpenID Connect Discovery technique. This vulnerability indicates the RP is vulnerable to CSRF.

(Shehab and Mohsen, 2014, 2016) explained a JavaScript injection technique for OAuth token theft that can be used when the source code of a mobile app has the WebView component. In this attack, an attacker may inject JavaScript into the WebView, jeopardizing the secrecy and integrity of the OAuth transaction. An attacker can obtain a user's credentials during the authentication step by injecting JavaScript to capture the user's email address and password when clicking the submit button. The injected JavaScript can then send the login information to the mobile application, which can subsequently transfer them to the malicious developer's remote server. An attacker can also alter the authorisation page presented to the user during the OAuth procedure by manipulating the list of requested permissions. The mitigation solution proposed in the research paper is a whitelist access control framework called "SecureOAuth" to fight against JavaScript injection attacks on the OAuth-WebView implementation.

(Wang et al., 2015) highlighted the various attack surfaces which are linked with OAuth implementation on Android, putting user data at risk. User-agent hijacking is an example of such an attack surface, in which a malicious RP application takes control of the WebView to launch attacks against user authentication and app permission. This may lead to the theft of critical user information as well as the manipulation of WebView content. Another major difficulty with the Android OAuth implementation is the network attack surface. Because the protocol is completely dependent on SSL for server authentication and secrecy, sensitive data such as login details and OAuth credentials can be captured and altered by a network attacker if transport security measures are not correctly implemented. This can result to data theft and a violation of user privacy.

(Li and Mitchell, 2014) explained Client secret theft in their lecture notes which threaten OAuth protocol security. This vulnerability is caused by an attacker gaining the client secret, which authenticates the client with the authorization server. The attacker can obtain tokens for the vulnerable client or repeat refresh tokens and authorization codes. An attacker could obtain the client's secret by gaining access to the source code, binaries, or deployment-specific secret. This risk allows an attacker to bypass client authentication and get refresh tokens or authorization codes. Theft of refresh tokens also jeopardizes OAuth security. An attacker acquires the refresh token, which allows a client to receive a new access token when the previous token expires. To get access to the protected resources, the attacker could pretend as the user. The refresh token can be stolen, duplicated, or obtained through online or native applications. This flaw allows an attacker to get unauthorized access to protected resources, which is dangerous.

3 Research Methodology

In recent years, a lot of research has been conducted on the various methods by which attackers could compromise OAuth frameworks. Others have undertaken security research, investigating how SP and RP implementations are utilized in the real world, and discovered several major flaws that an attacker can exploit to get control the victim user's resources. Facebook's OAuth 2.0 implementation has security weaknesses that might allow OAuth credentials to be stolen (Wang, Chen and Wang, 2012). As most previous research has focused on security issues of OAuth implementations on Web platforms, this study seeks to analyze the potential security difficulties that could occur when implementing OAuth on the Android operating system. Several researchers investigated OAuth implementation difficulties in mobile settings. Researchers discovered major discrepancies between mobile platforms and the Web, as well as the challenging areas of the OAuth protocol flows for mobile app developers (Wang *et al.*, 2015). Even with OAuth, native mobile apps make stealing login credentials easier. They advised avoiding implementing the end-user authorization procedure inside an embedded browser due to risks connected with the OAuth 2.0 threat model and security issues. In contrast to the most of previous work, which focuses on the authorization phase, this study focuses only on the process of authentication.

Each method for obtaining OAuth tokens includes its own set of limitations, though some of them may not be appropriate in specific situations. An approach for creating an Android application for acquiring OAuth tokens has been presented to bring attention to an additional potential vulnerability in the OAuth protocol that needs to be fixed.

The technique used for the perpetration of OAuth Token theft includes a methodical strategy aimed at comprehending, evaluating, and enhancing its performance inside the Android application. Once the development environment and projects have been set up, attention is directed toward understanding the manner in which the code will manage deep linkages using the Intent object. Upon careful examination of the layout XML file and the UI components, it becomes evident that a clear linkage exists between the code and the TextView element, which serves the purpose of presenting data. The focus is directed on the logical process that verifies intents and data, guaranteeing the extraction and display of the data as a string in the TextView. The application behaviour is tested by conducting a series of test cases, which include starting the application with and without intent data.

This work builds an Android app for stealing OAuth tokens using a novel technique. This technique has the potential to succeed in certain conditions, while earlier approaches are likely to not work in the same circumstances. To ensure the security of user information and prevent unauthorized access, it is critical to regularly discover and resolve any vulnerabilities associated with the OAuth protocol.

4 Design Specification

Fig.1 shows the OAuth flow for the client apps that uses an embedded Web browser (Web view) (Shehab and Mohsen, 2014). Facebook login for authentication complies with a well-defined process to ensure secure verification of user identities and the permission of application access. The authentication process is initiated by the user by the action of clicking the "Login with Facebook" option, which then redirects the user to Facebook's server. After that, the application transmits an authorization request including its unique client identifier and the permissions it intends to get. After being provided with a consent screen, users give approval for the required rights, therefore enabling Facebook's server to generate an authorization grant, which is then returned to the application. The application initiates a token request to facilitate the exchange of permission for an access token. This access token serves as the necessary credentials to get user data from Facebook's servers. This functionality allows the application to get authorized data, such as the user's name and profile image, and verify the user's identity within its own system. The access token, which is followed by a specified expiry time frame, enables authorized interactions to occur without the need for repetitive login processes. The continuity of user sessions is ensured by means of this token until the user decides to log out or the token is revoked. The OAuth flow not only guarantees secure access to user data but also enhances the process of user authentication and improves user experience.

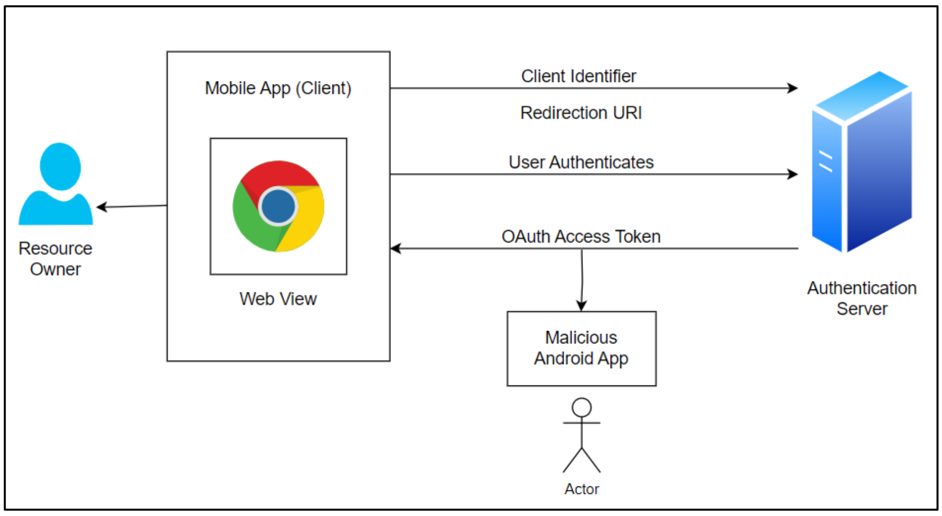


Figure 1. OAuth Web Client Flow

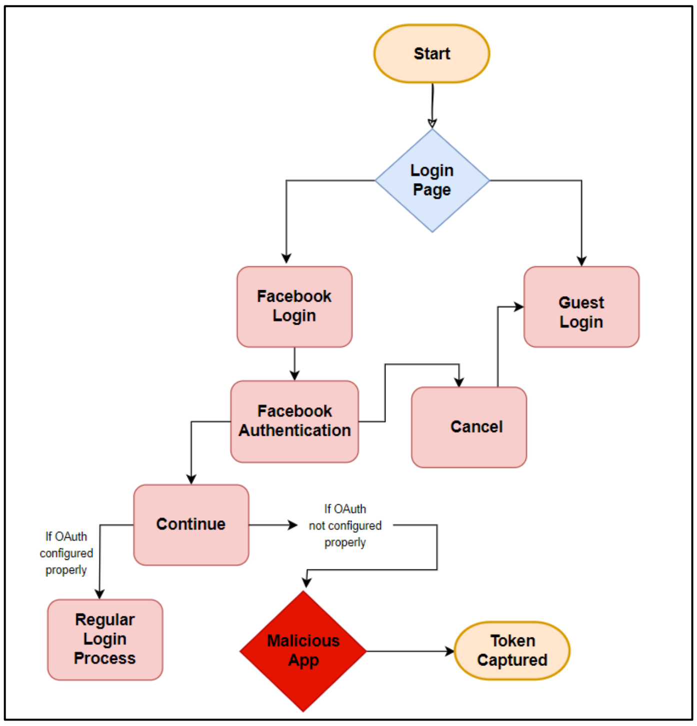


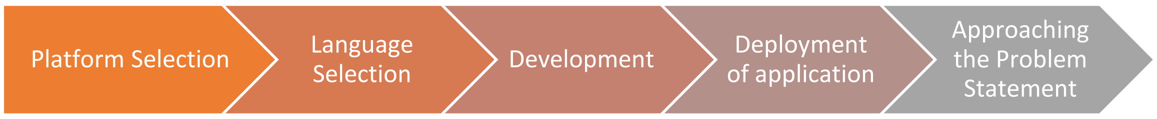
Figure 2. Flowchart of token theft during the Authentication Process

The flowchart demonstrates each step through which a user interacts with an application using Facebook Login. It includes the smooth authentication procedure as well as the possible challenges that may arise due to improper OAuth configurations. The process begins when the user interacts with the "Login with Facebook" button, prompting the application to send an authorization request to Facebook. This request includes the necessary credentials and permissions that the application is requesting. The obtaining of user permission is made easier via a consent screen, which ultimately results in the issuing of an authorization grant after acceptance. But at this point, a key turning point appears: if OAuth is not carefully implemented, vulnerabilities appear. In the given situation, it is possible for a malicious application to take advantage of this vulnerable setup, secretly acquiring the OAuth token. whereas when appropriately configured, the flowchart progresses to the stage when the authorization grant is exchanged for an access token. This exchange allows for the recovery of user data and enables smooth authentication inside the application's domain. The access token serves as a means for enhancing user interactions, ensuring a seamless experience by reducing the need for repeated logins. This feature contributes to the enhancement of the overall user experience. Users maintain control over their accounts and have the ability to log out or adjust application permissions via their Facebook accounts.

For instance, the diagram shows how different the two situations are: if OAuth is not set up properly, a malicious app can steal tokens, but if it is set up correctly, this threat is eliminated, and regular login processes can be used. The substantial discrepancy shown in this scenario highlights the critical need for implementing strong security mechanisms in OAuth, which serve to enhance the safety of user data and maintain the authenticity of the authentication process.

5 Implementation

During the implementation of the project following steps have been followed:



5.1 Platform Selection:

The main objective of the project implementation is to create an Android application that will capture the OAuth token while accessing the third-party application. Android Studio has been used to create this application. Android Studio is an official IDE to create and deploy Android applications. It is an open-source application and can be installed and used by any user. In 2023, 49.11% of all smartphone users around the world are likely to be using an Android phone[2]. API 23: Android 6.0 (Marshmallow) was selected for SDK development so that it can be installed on approximately 97.7 percent of devices[3].

5.2 Language Selection:

The choice of a suitable programming language within the Android Studio framework has significant implications. The application can be developed in Java or Kotlin languages[4]

[2] https://www.bankmycell.com/blog/how-many-android-users-are-there
[3] https://developer.android.com/studio
[4] https://developer.android.com/training/basics/supporting-devices/languages

however Java has been chosen over Kotlin for the development due to its maturity, large developer base, compatibility, and established effectiveness. This decision is based on Java's extensive adoption and established reliability within the Android development community. Considering the broad community support and exhaustive documentation available for Java in the context of Android. This deliberate decision enabled the development of a streamlined and effective application that successfully met the project's objectives.

5.3 Development:

The code and configuration aim to create an Android app that can manage and display deep links, especially those associated with Facebook authentication, without any issues. When a user clicks on a link with the specified scheme (for example, "fb_login_protocol_scheme"), the app's MainActivity will launch and display the deep link's content in a TextView. This integration improves the user experience by enabling the app to respond to relevant actions activated by external sources, such as web browsers or other apps, and display the relevant information within the app's interface.

5.4 Deployment of Application:

After the application has been developed and its functionality has been tested in Android Studio, an APK is generated and deployed on Genymotion and an Android device. Genymotion is a versatile fast Android emulator used for testing and developing applications[5]. It provides a variety of virtual devices with different Android versions and configurations, allowing developers to simulate real-world scenarios and test the applications efficiently in a variety of environments.

5.5 Approaching the Problem Statement:

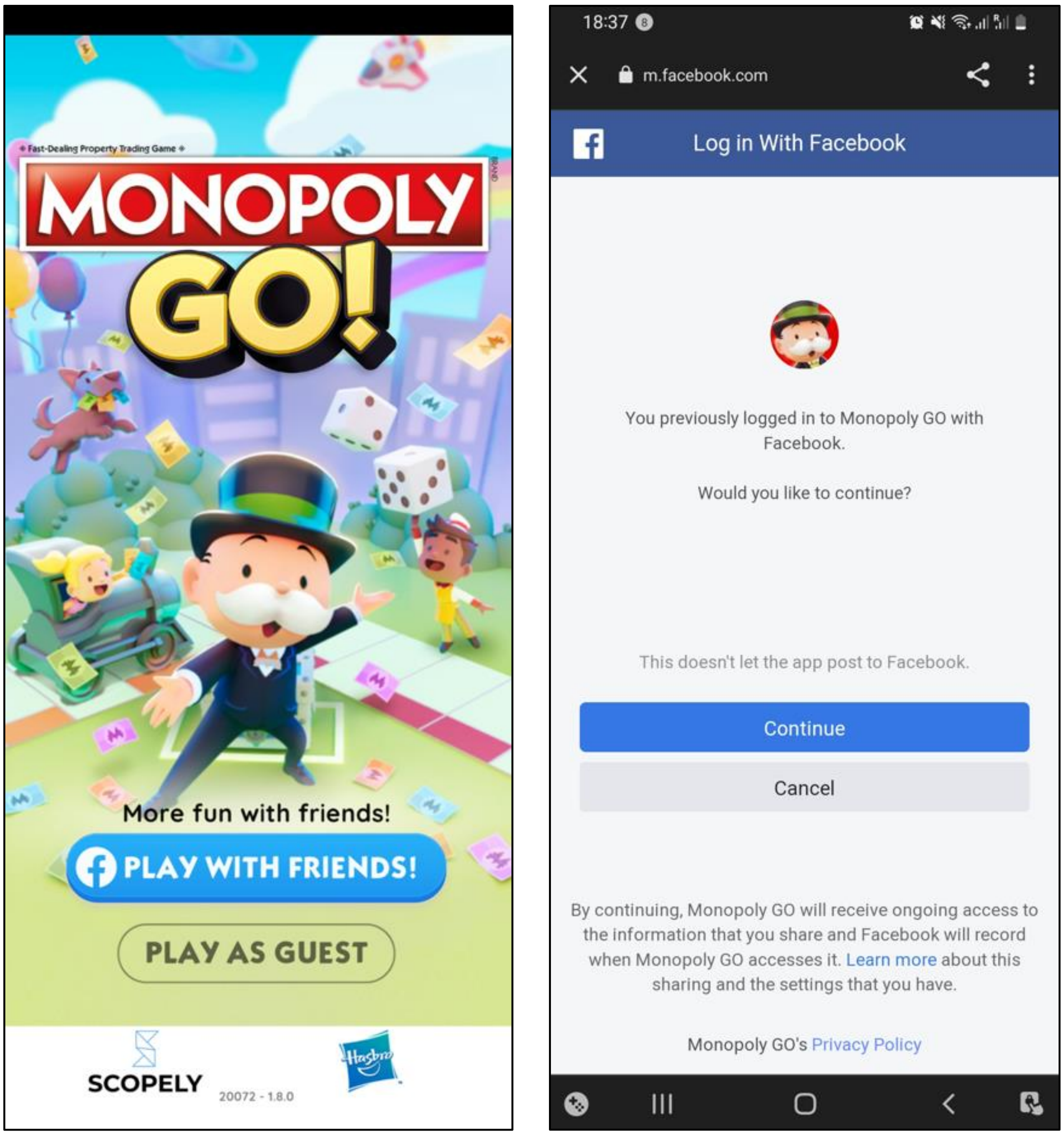
A website and mobile application were evaluated via testing on both an Android device and the Genymotion Android emulator. The primary objective of this study was to examine and analyze the challenges associated with Oauth configurations in the Facebook login procedure. During the examination of this testing, a comprehensive analysis was conducted on the Android application that was created for the project. Additionally, throughout the testing process, it was discovered that the OAuth token-gathering method in the Android application worked efficiently. The application successfully acquired Oauth tokens, hence highlighting its possibilities in addressing the Oauth configuration issues pointed out during the evaluation of the website and mobile application.

[5] https://www.geeksforgeeks.org/how-to-install-genymotion-emulator-and-add-its-plugin-to-android-studio/

6 Evaluation

6.1 Experiment 1 – Stealing token of an Android application.

The following experiment demonstrates the token theft inside a Monopoly game performed using a malicious application. Users are presented with the choice of playing the game through a Guest login or by playing alongside their Facebook friends. When the user tries to authenticate using valid Facebook credentials, the gaming application leads to a malicious app that has been installed on the Android mobile device. This malicious application then collects the login token, which may then be misused by an attacker with malicious intentions.



You previously logged in to Monopoly GO with Facebook.

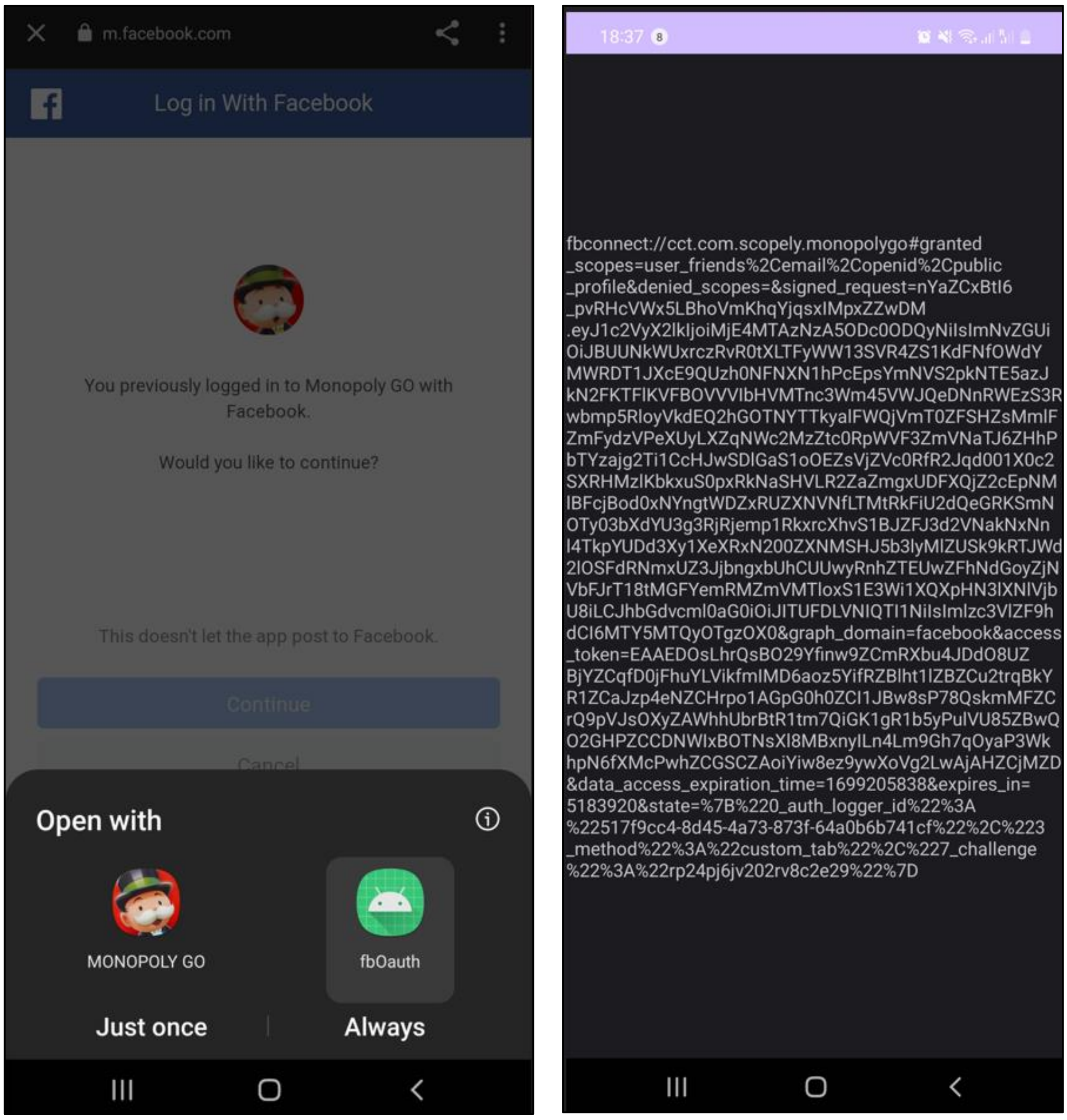
Would you like to continue?

This doesn't let the app post to Facebook.

By continuing, Monopoly GO will receive ongoing access to the information that you share and Facebook will record when Monopoly GO accesses it. Learn more about this sharing and the settings that you have.

Monopoly GO's Privacy Policy

Figure 3. Monopoly Game Application Launch Using Facebook Login



You previously logged in to Monopoly GO with Facebook.

Would you like to continue?

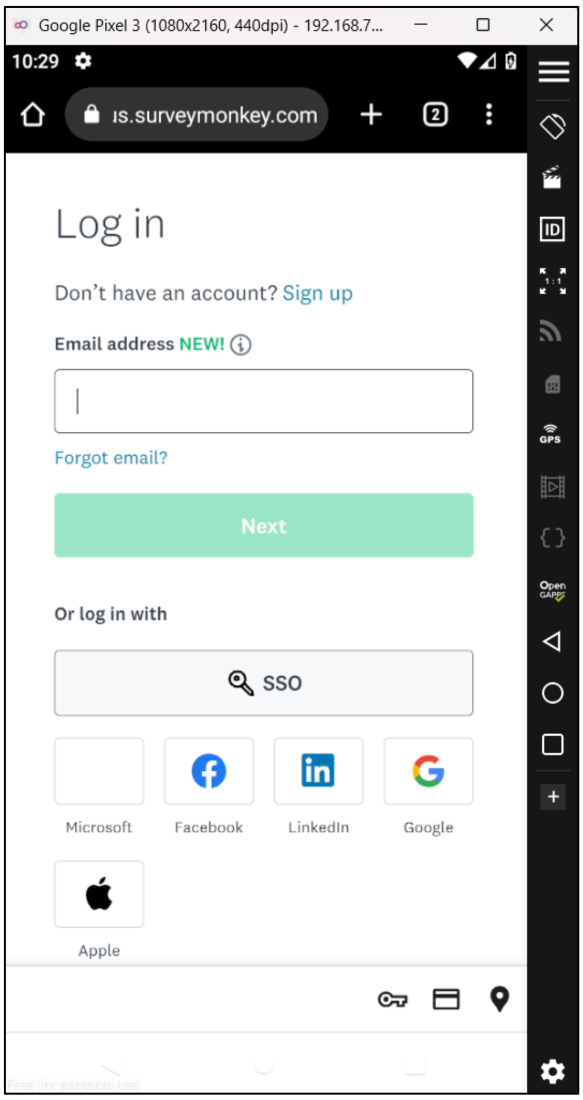
This doesn't let the app post to Facebook.

```
fbconnect://cct.com.scopely.monopolygo#granted
_scopes=user_friends%2Cemail%2Copenid%2Cpublic
_profile&denied_scopes=&signed_request=nYaZCxBtI6
_pvRHcVWx5LBhoVmKhqYjqsxIMpxZZwDM
.eyJ1c2VyX2lkIjoiMjE4MTAzNzA5ODc0ODQyNiIsImNvZGUi
OiJBUUNkWUxrczRvR0tXLTFyWW13SVR4ZS1KdFNfOWdY
MWRDT1JXcE9QUzh0NFNXN1hPcEpsYmNVS2pkNTE5azJ
kN2FKTFlKVFBOVVVIbHVMTnc3Wm45VWJQeDNnRWEzS3R
wbmp5RloyVkdEQ2hGOTNYTTkyalFWQjVmT0ZFSHZsMmlF
ZmFydzVPeXUyLXZqNWc2MzZtc0RpWVF3ZmVNaTJ6ZHhP
bTYzajg2Ti1CcHJwSDlGaS1oOEZsVjZVc0RfR2Jqd001X0c2
SXRHMzlKbkxuS0pxRkNaSHVLR2ZaZmgxUDFXQjZ2cEpNM
lBFcjBod0xNYngtWDZxRUZXNVNfLTMtRkFiU2dQeGRKSmN
OTy03bXdYU3g3RjRjemp1RkxrcXhvS1BJZFJ3d2VNakNxNn
l4TkpYUDd3Xy1XeXRxN200ZXNMSHJ5b3lyMlZUSk9kRTJWd
2lOSFdRNmxUZ3JjbngxbUhCUUwyRnhZTEUwZFhNdGoyZjN
VbFJrT18tMGFYemRMZmVMTloxS1E3Wi1XQXpHN3lXNlVjb
U8iLCJhbGdvcml0aG0iOiJITUFDLVNIQTI1NiIsImlzc3VlZF9h
dCI6MTY5MTQyOTgzOX0&graph_domain=facebook&access
_token=EAAEDOsLhrQsBO29Yfinw9ZCmRXbu4JDdO8UZ
BjYZCqfD0jFhuYLVikfmIMD6aoz5YifRZBlht1lZBZCu2trqBkY
R1ZCaJzp4eNZCHrpo1AGpG0h0ZCI1JBw8sP78QskmMFZC
rQ9pVJsOXyZAWhhUbrBtR1tm7QiGK1gR1b5yPulVU85ZBwQ
O2GHPZCCDNWlxBOTNsXl8MBxnylLn4Lm9Gh7qOyaP3Wk
hpN6fXMcPwhZCGSCZAoiYiw8ez9ywXoVg2LwAjAHZCjMZD
&data_access_expiration_time=1699205838&expires_in=
5183920&state=%7B%220_auth_logger_id%22%3A
%22517f9cc4-8d45-4a73-873f-64a0b6b741cf%22%2C%223
_method%22%3A%22custom_tab%22%2C%227_challenge
%22%3A%22rp24pj6jv202rv8c2e29%22%7D
```

Figure 4. Stealing OAuth Token Using Malicious Application

6.2 Experiment 2: Stealing token in the web view.

Another experiment was carried out on the SurveyMonkey website in the Web view. When a user attempts to log in using the Facebook Login option, they input their valid credentials, and then an attacker adjusts the redirect URL to "fbconnect". It matches with the intent, and it allows us to continue with Facebook. It was created by Facebook, enabling users to seamlessly proceed with their application. The authentication method aligns the intended purpose with the malicious application, hence allowing to choose the application to continue further. Upon selecting the malicious application, it will proceed to gather the token, so creating an opportunity for exploitation.



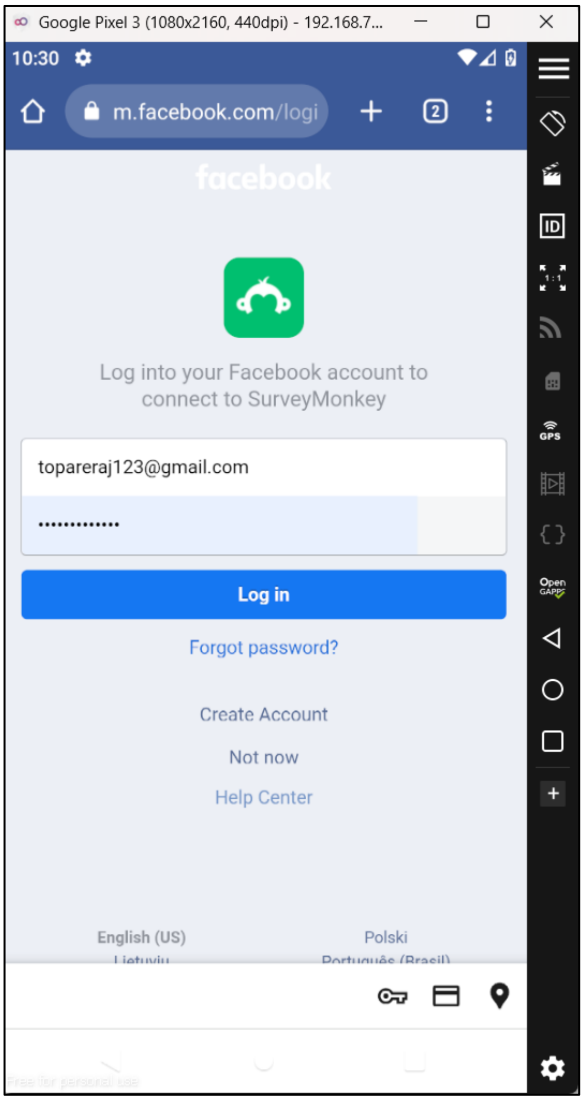


Figure 5. Signing In on SurveyMonkey Website Using Facebook Login on WebView in Genymotion.

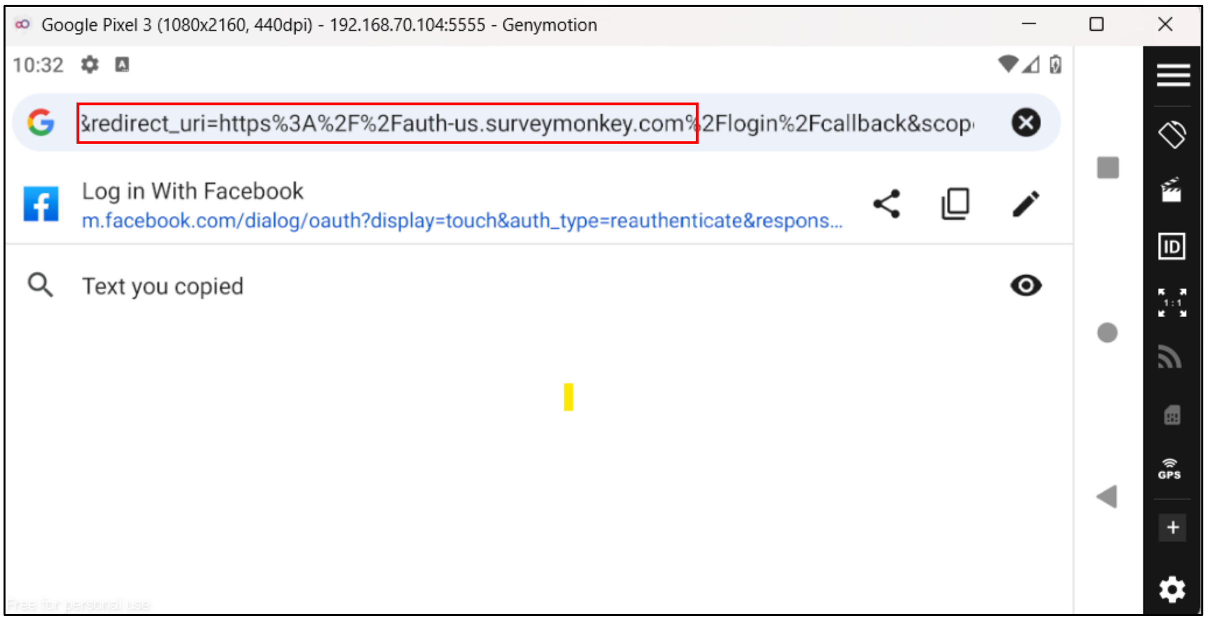
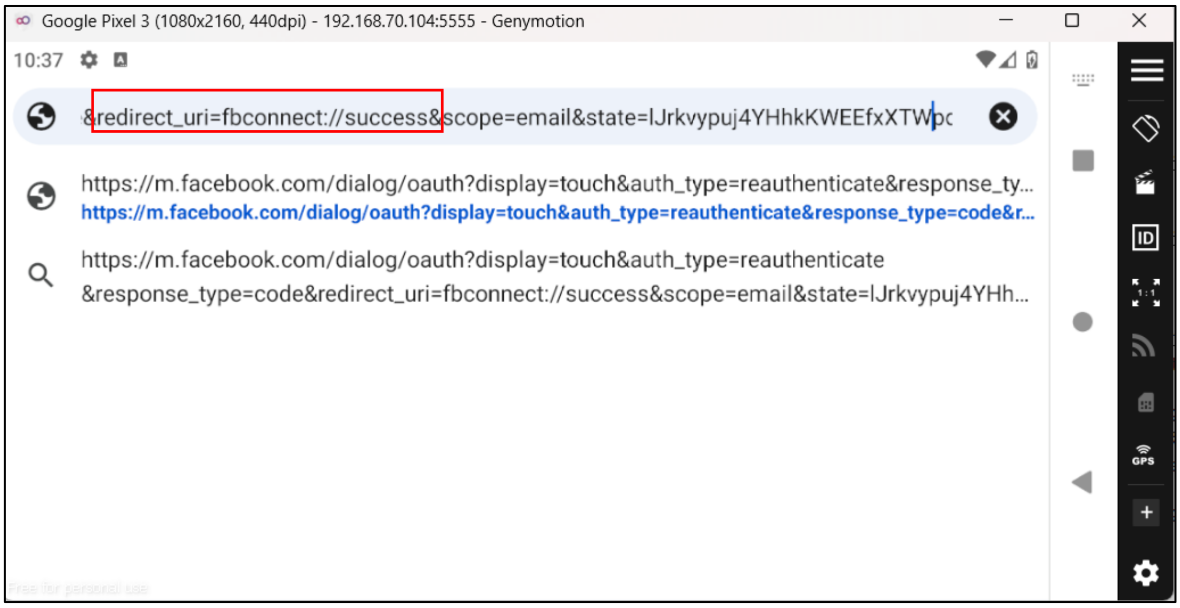

Google Pixel 3 (1080x2160, 440dpi) - 192.168.70.104:5555 - Genymotion
10:32
&redirect_uri=https%3A%2F%2Fauth-us.surveymonkey.com%2Flogin%2Fcallback&scop
Log in With Facebook
m.facebook.com/dialog/oauth?display=touch&auth_type=reauthenticate&respons...
Text you copied

Figure 6. Redirect URI during Facebook Login on SurveyMonkey Website.

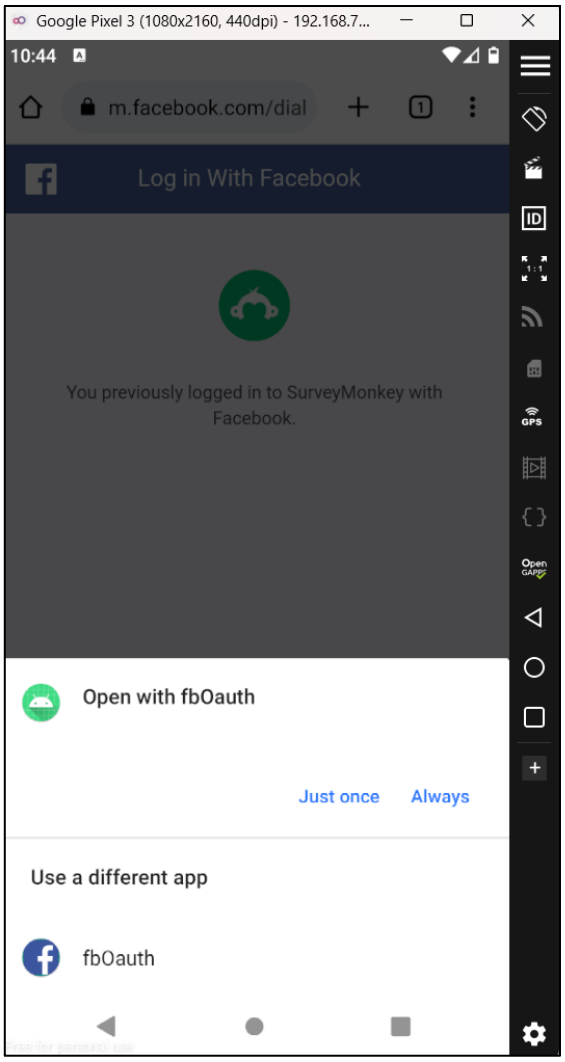


Google Pixel 3 (1080x2160, 440dpi) - 192.168.70.104:5555 - Genymotion

10:37

```
&redirect_uri=fbconnect://success&scope=email&state=lJrkvypuj4YHhkKWEEfxXTWpc
https://m.facebook.com/dialog/oauth?display=touch&auth_type=reauthenticate&response_ty...
https://m.facebook.com/dialog/oauth?display=touch&auth_type=reauthenticate&response_type=code&r...
https://m.facebook.com/dialog/oauth?display=touch&auth_type=reauthenticate
&response_type=code&redirect_uri=fbconnect://success&scope=email&state=lJrkvypuj4YHh...
```

Figure 7. Tampering with the redirect URI to match the intent.



Google Pixel 3 (1080x2160, 440dpi) - 192.168.7...

10:44

m.facebook.com/dial

Log in With Facebook

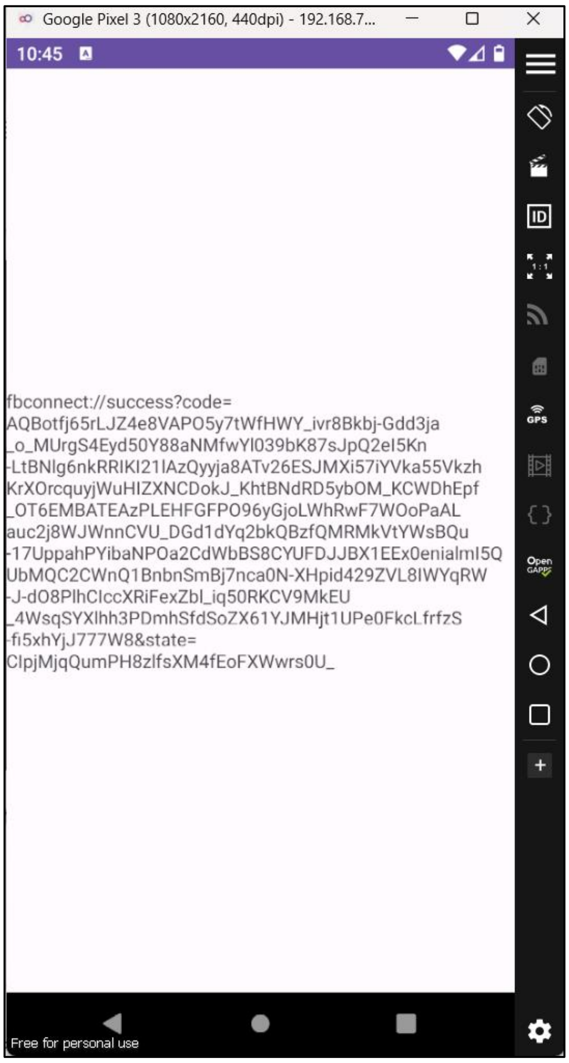
You previously logged in to SurveyMonkey with Facebook.

Open with fbOauth

Just once Always

Use a different app

fbOauth



Google Pixel 3 (1080x2160, 440dpi) - 192.168.7...

10:45

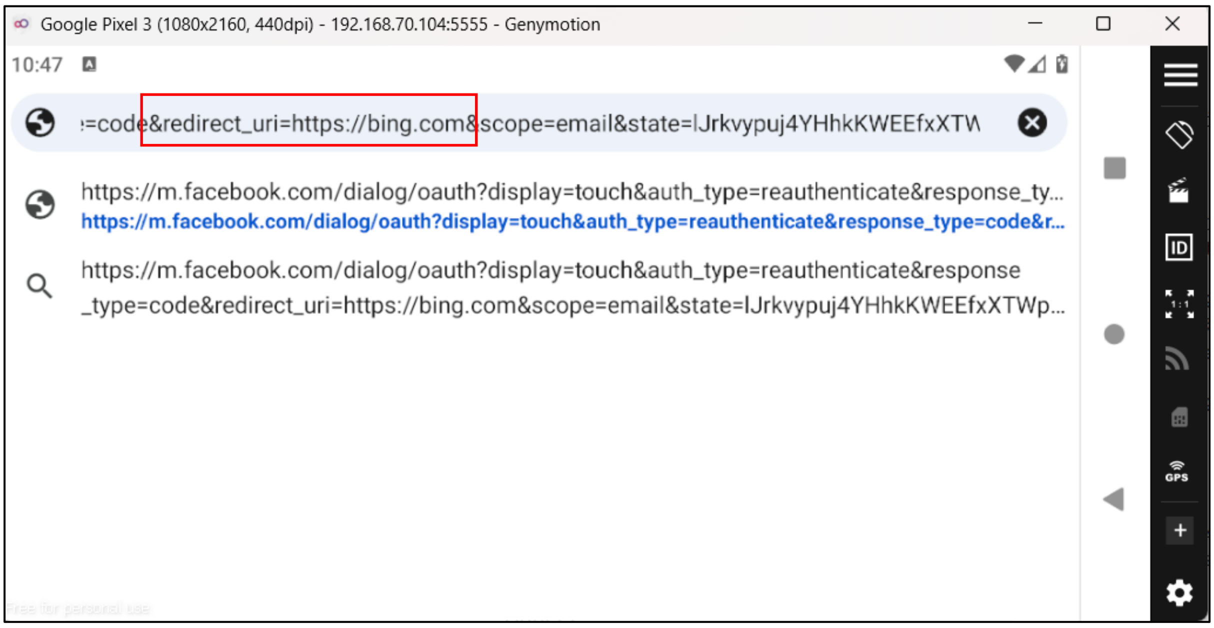
```
fbconnect://success?code=
AQBotfj65rLJZ4e8VAPO5y7tWfHWY_ivr8Bkbj-Gdd3ja
_o_MUrgS4Eyd50Y88aNMfwYl039bK87sJpQ2eI5Kn
-LtBNlg6nkRRIKI21lAzQyyja8ATv26ESJMXi57iYVka55Vkzh
KrXOrcquyjWuHIZXNCDokJ_KhtBNdRD5ybOM_KCWDhEpf
_OT6EMBATEAzPLEHFGFPO96yGjoLWhRwF7WOoPaAL
auc2j8WJWnnCVU_DGd1dYq2bkQBzfQMRMkVtYWsBQu
-17UppahPYibaNPOa2CdWbBS8CYUFDJJBX1EEx0enialmI5Q
UbMQC2CWnQ1BnbnSmBj7nca0N-XHpid429ZVL8IWYqRW
-J-dO8PlhClccXRiFexZbl_iq50RKCV9MkEU
_4WsqSYXlhh3PDmhSfdSoZX61YJMHjt1UPe0FkcL.frfzS
-fi5xhYjJ777W8&state=
ClpjMjqQumPH8zlfsXM4fEoFXWwrs0U_
```

Free for personal use

Figure 8. Capturing the OAuth Token

6.3 Experiment 3: Stealing token using open redirect method.

The following experiment was carried out using an alternate method- Open Redirect URI on the same website SurveyMonkey for which the token theft was performed using a malicious application. It showed that when modifying the redirect URL of Facebook authentication for SurveyMonkey, an authentication mechanism throws the invalid URL. As the URL did not match with the whitelisted domain such as SurveyMonkey.com therefore redirect method failed.


Google Pixel 3 (1080x2160, 440dpi) - 192.168.70.104:5555 - Genymotion

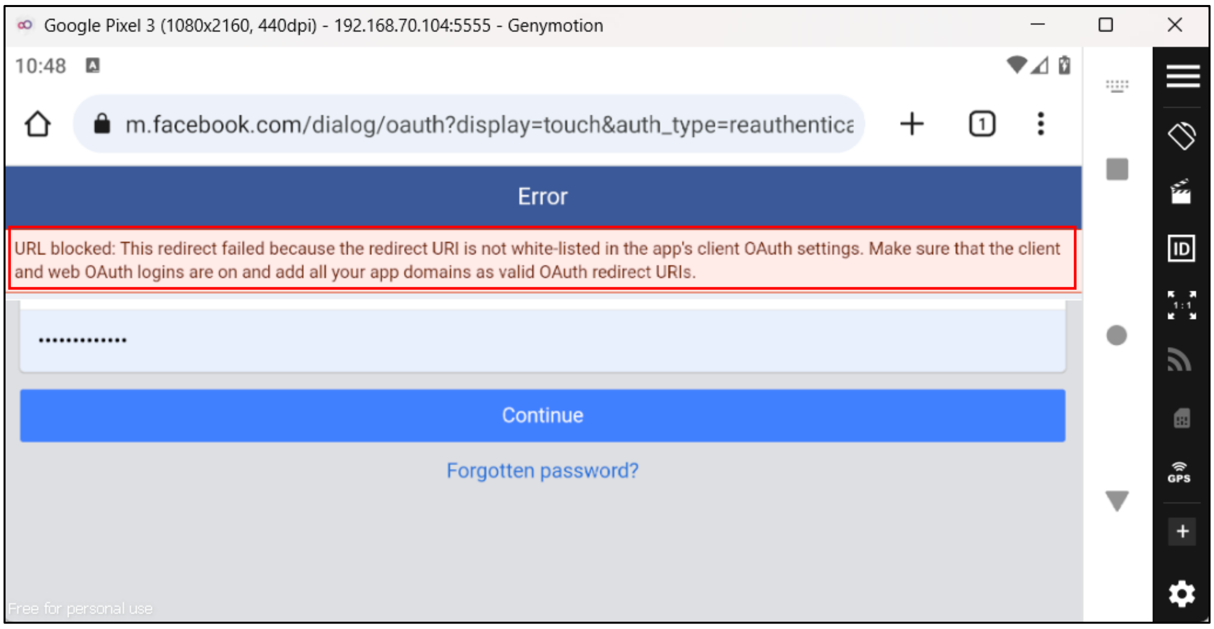
10:47

:=code&redirect_uri=https://bing.com&scope=email&state=lJrkvypuj4YHhkKWEEfxXTW

https://m.facebook.com/dialog/oauth?display=touch&auth_type=reauthenticate&response_ty...
https://m.facebook.com/dialog/oauth?display=touch&auth_type=reauthenticate&response_type=code&r...

https://m.facebook.com/dialog/oauth?display=touch&auth_type=reauthenticate&response
_type=code&redirect_uri=https://bing.com&scope=email&state=lJrkvypuj4YHhkKWEEfxXTWp...

Figure 9. Tampering URL using Open redirect method.


Google Pixel 3 (1080x2160, 440dpi) - 192.168.70.104:5555 - Genymotion

10:48

m.facebook.com/dialog/oauth?display=touch&auth_type=reauthentica

Error

URL blocked: This redirect failed because the redirect URI is not white-listed in the app's client OAuth settings. Make sure that the client and web OAuth logins are on and add all your app domains as valid OAuth redirect URIs.

••••••••••••••

Figure 10. An error of Failed attempt using the Open redirect method.

6.4 Discussion

After reviewing the case studies, it is observed that: when the coding scheme matches the redirect URI, the intended operation is executed, and the token is sent to the designated recipient. In the first two instances, the intent URL matches the expected parameters, resulting in successful execution; therefore, the malicious application receives the authentication token provided by Facebook. Whereas the third case represents a scenario in which the scheme does not match the expected configuration, resulting in the URL being blocked and the redirect method failing. Therefore, when an external application attempts to log in with Facebook credentials and successfully matches the predefined scheme, the subsequent login authentication token becomes vulnerable to interception.

7 Conclusion and Future Work

In summary, this research has provided insights into the critical issue of OAuth token theft in Android devices, revealing the weaknesses and possible risks linked to the implementation of OAuth in the mobile ecosystem. The research findings have shown that the security of OAuth tokens on Android devices is an issue of significant concern, as it may have implications for both user privacy and data integrity. It highlights the need for increased awareness and enhanced protective measures in mitigating the risk of token theft. By focusing on the authentication process and exploring potential vulnerabilities, this study contributes to a deeper understanding of the complexities involved in securing OAuth tokens on Android devices. The introduction of a novel technique for creating an Android application to capture OAuth tokens further emphasizes the need to enhance security measures to counter potential threats.

In the future, real-time behaviour monitoring can be implemented to detect and prevent malicious token capture attempts. When adopting OAuth, developers should prioritize the protection of users' privacy. This may be achieved by avoiding the usage of WebView-based OAuth and instead prompting users to install the native app of the service provider. If the web browser is the only available option, developers should include a procedure to identify and prevent malicious applications' potential hijacking of the authentication token.

References

Abdullah, H. and Zeebaree, S.R.M. (2021) ‘Android Mobile Applications Vulnerabilities and Prevention Methods: A Review’, Proceedings of 2021 2nd Information Technology to Enhance E-Learning and other Application Conference, IT-ELA 2021, pp. 148–153. Available at: https://doi.org/10.1109/IT-ELA52201.2021.9773615.

Armando, A., Carbone, R., Compagna, L., Cuellar, J., Pellegrino, G. and Sorniotti, A. (2011) ‘From Multiple Credentials to Browser-Based Single Sign-On: Are We More Secure?’, in S. and M.Y. and P.A. and R.C. Camenisch Jan and Fischer-Hübner (ed.) Future Challenges in Security and Privacy for Academia and Industry. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 68–79.

Bansal, C., Bhargavan, K., Delignat-Lavaud, A. and Maffeis, S. (2014) ‘Discovering concrete attacks on website authorization by formal analysis’, Journal of Computer Security, 22, pp. 601–657. Available at: https://doi.org/10.3233/JCS-140503.
Campbell, B., Mortimore, C. and Jones, M. (2015) ‘Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants’. Available at: https://doi.org/10.17487/RFC7522.

Chari, S., Jutla, C. and Roy, A. (2011) ‘Universally Composable Security Analysis of OAuth v2.0’. Available at: https://eprint.iacr.org/2011/526.

Chen, E.Y., Pei, Y., Chen, S., Tian, Y., Kotcher, R. and Tague, P. (2014) ‘OAuth Demystified for Mobile Application Developers’, in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. New York, NY, USA: Association for Computing Machinery (CCS ’14), pp. 892–903. Available at: https://doi.org/10.1145/2660267.2660323.

Cherrueau Ronan-Alexandre and Douence, R. and R.J.-C. and S.M. and de O.A.S. and R.Y. and D.M. (2014) ‘Reference Monitors for Security and Interoperability in OAuth 2.0’, in G. and C.-B.N. and F.S. and F.W.M. Garcia-Alfaro Joaquin and Lioudakis (ed.) Data Privacy Management and Autonomous Spontaneous Security. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 235–249.

Chin Erika and Wagner, D. (2014) ‘Bifocals: Analyzing WebView Vulnerabilities in Android Applications’, in H. and P.A. Kim Yongdae and Lee (ed.) Information Security Applications. Cham: Springer International Publishing, pp. 138–159.

Enck, W., Ongtang, M. and McDaniel, P. (2009) ‘Understanding Android Security’, IEEE Security & Privacy, 7(1), pp. 50–57. Available at: https://doi.org/10.1109/MSP.2009.26.

Ferry, E., Raw, J.O. and Curran, K. (2015) ‘Security evaluation of the OAuth 2.0 framework OAuth 2.0 framework 73’, Information & Computer Security, 23(1), pp. 73–101. Available at: https://doi.org/10.1108/ICS-12-2013-0089.

Fett, D., Küsters, R. and Schmitz, G. (2016) ‘A Comprehensive Formal Security Analysis of OAuth 2.0’, in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, pp. 1204–1215. Available at: https://doi.org/10.1145/2976749.2978385.

Li, W. and Mitchell, C.J. (2014) 'Security Issues in OAuth 2.0 SSO Implementations', Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8783, pp. 529–541. Available at: https://doi.org/10.1007/978-3-319-13257-0_34.

Lodderstedt, T., McGloin, M. and Hunt, P. (2013) 'OAuth 2.0 Threat Model and Security Considerations', Internet Engineering Task Force (IETF) RFC [Preprint]. Edited by T. Lodderstedt. Available at: https://doi.org/10.17487/RFC6819.

Luo, T., Hao, H., Du, W., Wang, Y. and Yin, H. (2011) 'Attacks on WebView in the Android System', in Proceedings of the 27th Annual Computer Security Applications Conference. New York, NY, USA: Association for Computing Machinery (ACSAC '11), pp. 343–352. Available at: https://doi.org/10.1145/2076732.2076781.

Mohsen, F. and Shehab, M. (2016) 'Hardening the OAuth-WebView Implementations in Android Applications by Re-Factoring the Chromium Library', in 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), pp. 196–205. Available at: https://doi.org/10.1109/CIC.2016.036.

Rahat, T. Al, Feng, Y. and Tian, Y. (2019) 'OAUTHLINT: An empirical study on oauth bugs in android applications', Proceedings - 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, pp. 293–304. Available at: https://doi.org/10.1109/ASE.2019.00036.

Rosen, S., Qian, Z. and Mao, Z.M. (2013) 'AppProfiler: A Flexible Method of Exposing Privacy-Related Behavior in Android Applications to End Users', in Proceedings of the Third ACM Conference on Data and Application Security and Privacy. New York, NY, USA: Association for Computing Machinery (CODASPY '13), pp. 221–232. Available at: https://doi.org/10.1145/2435349.2435380.

Shehab, M. and Mohsen, F. (2014) 'Towards enhancing the security of OAuth implementations in smart phones', Proceedings - 2014 IEEE 3rd International Conference on Mobile Services, MS 2014, pp. 39–46. Available at: https://doi.org/10.1109/MOBSERV.2014.15.

Shernan Ethan and Carter, H. and T.D. and T.P. and B.K. (2015) 'More Guidelines Than Rules: CSRF Vulnerabilities from Noncompliant OAuth 2.0 Implementations', in V. and M.F. Almgren Magnus and Gulisano (ed.) Detection of Intrusions and Malware, and Vulnerability Assessment. Cham: Springer International Publishing, pp. 239–260.

Sucasas, V., Mantas, G., Althunibat, S., Oliveira, L., Antonopoulos, A., Otung, I. and Rodriguez, J. (2018) 'A privacy-enhanced OAuth 2.0 based protocol for Smart City mobile applications', Computers & Security, 74, pp. 258–274. Available at: https://doi.org/https://doi.org/10.1016/j.cose.2018.01.014.

Sun, S.-T. and Beznosov, K. (2012) 'The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems', in Proceedings of the 2012 ACM Conference on Computer and Communications Security. New York, NY, USA: Association for Computing Machinery (CCS '12), pp. 378–390. Available at: https://doi.org/10.1145/2382196.2382238.

Wang, H., Zhang, Y., Li, J. and Gu, D. (2016) 'The Achilles Heel of OAuth: A Multi-Platform Study of OAuth-Based Authentication', in Proceedings of the 32nd Annual Conference on

Computer Security Applications. New York, NY, USA: Association for Computing Machinery (ACSAC '16), pp. 167–176. Available at: https://doi.org/10.1145/2991079.2991105.

Wang, H., Zhang, Y., Li, J., Liu, H., Yang, W., Li, B. and Gu, D. (2015) 'Vulnerability assessment of OAuth implementations in android applications', ACM International Conference Proceeding Series, 7-11-December-2015, pp. 61–70. Available at: https://doi.org/10.1145/2818000.2818024.

Wang, R., Chen, S. and Wang, X. (2012) 'Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services', in 2012 IEEE Symposium on Security and Privacy, pp. 365–379. Available at: https://doi.org/10.1109/SP.2012.30.

Xiao, Y., Bai, G., Mao, J., Liang, Z. and Cheng, W. (2017) 'Privilege Leakage and Information Stealing through the Android Task Mechanism', Proceedings - 2017 IEEE Symposium on Privacy-Aware Computing, PAC 2017, 2017-January, pp. 152–163. Available at: https://doi.org/10.1109/PAC.2017.29.

Zhou, Y. and Evans, D. (2014) 'SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities', in 23rd USENIX Security Symposium (USENIX Security 14). San Diego, CA: USENIX Association, pp. 495–510. Available at: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/zhou.