Original Research Article

J. Computing & Int. Systems (2023) 329–336

SACRED HEART RESEARCH PUBLICATIONS

Journal of Computing and Intelligent Systems

Journal homepage: www.shcpub.edu.in

ISSN: 2456-9496

IMPLEMENTATION AND TESTING OF DEEP LINK IN AN ANDROID APPLICATION

Dhruti Ranjan Mohanty[#1], **P. Thiyagarajan** [#2]

Received on 30 JUL 2023, Accepted on 30 AUG 2023

***Abstract* — Deep linking has emerged as a vital technology in the field of mobile app integration and user experience. It enables seamless transitions between different digital contexts by allowing users to access specific content or features within apps directly. This paper explores the concept of deep linking, its implementation methods, challenges, and the impact of deep linking on user engagement and app discoverability. The paper begins by providing an overview of deep linking, outlining its purpose and potential benefits. It examines the technical aspects of deep linking, including the various approaches such as Uniform Resource Identifiers (URIs), custom URL schemes, Android App Links, and iOS Universal Links. The implementation methods for deep linking are discussed, emphasizing the importance of proper integration into the apps code base. This paper examines the idea of deep linking, its methods of implementation, the difficulties brought on by device fragmentation, and the problems with app installation status. User journeys will be further streamlined and the promise of connected digital experiences will be fully realized with further innovation and breakthroughs in deep linking technologies.**

Keywords - Deep Link, URI Scheme Handling, Android App Deep Linking, Deferred Deep Linking, Testing Deep Links

I INTRODUCTION

The current generation came into being in a time of quick technological development and easy access to knowledge. The experiences and behaviours of these users have been significantly shaped by Android applications. The ease, enjoyment, and access to a large digital universe offered by these programmes have become crucial parts of their everyday lives. These programmes enable users to lead more effective, organised, and balanced lives. A platform for customization and self-expression is provided by Android apps. People may use these programmes to share with the world their distinctive personalities, skills, and opinions. The process that the current generation connects and communicates with one another has changed as a result of Android applications like messaging apps and social media platforms [18]. These applications let users to communicate with friends, family, and co-workers anywhere in the world through instant messaging, phone and video chatting, and social networking capabilities. They have aided in the communication of ideas, supported online groups, and boosted interpersonal bonds. The way that today's generation obtains knowledge and receives information has been revolutionized by Android applications. Users are now able to easily access knowledge through Android applications, broadening their horizons and intellectual interests. Since their origin, when Android applications were isolated and only capable of basic functions, they have advanced significantly. The need to link apps and build a more connected environment was recognized by developers as technology improved. Deep linking, which enables precise navigation to particular app content, has become a crucial technique for seamless user experiences in the quickly developing world of mobile applications. Deep linking techniques have improved, but there are still issues with implementing and testing them effectively, which prevents this technology from being used to its full potential [13]. This paper aims to address the following problem: How can deep linking technologies be tested and implemented in Android apps to maximize user experience while ensuring consistent and reliable navigation?

Sub Problem 1: How to effectively integrate Deep Links considering variations in app architecture, framework compatibility, and evolving Android standards?

Sub Problem 2: What automated testing approaches and tools can be created to thoroughly check the functioning of deep links within an Android app, spanning a variety of usage situations, and requiring the least amount of human intervention?

Sub Problem 3: What comprehensive set of best practices and guidelines can be established for developers to follow during the implementation and testing phases of deep linking, ensuring consistency and high-quality user experiences?

This paper aims to empower developers to harness the potential of deep linking technology for creating seamless, personalized, and engaging user experiences across a wide range of Android devices and versions

* Corresponding author: E-mail: [1]dhrutiranjan2000@gmail.com, [2]thiyagu.phd@gmail.com

[1]M.Sc. Computer Science (Cyber Security), Rajiv Gandhi National Institute of Youth Development, Sriperumbudur, Tamil Nadu, India.

[2]Associate Professor & Head, Department of Computer Science (Cyber Security), Rajiv Gandhi National Institute of Youth Development, Sriperumbudur, Tamil Nadu, India.

II BASIC CONCEPTS OF DEEP LINK

Deep linking, which enables users to immediately access particular material within an app, was consequently included in Android applications. Deep linking makes it unnecessary for consumers to go through many stages and gives them immediate access to the material they want. It may be started from a variety of places, including websites, emails, texts, and other apps. User experience is improved through deep links, which open the related app and direct users to the necessary material. Deep linking improvements have improved how seamlessly various programs integrate with one another. Due to the possibility for cross-app communication, deep links inside one app can launch certain activities or visit specific websites. Deep linking increases app discoverability by making deep connections shareable across several platforms. Deep linking also has the benefit of enabling personalized experiences within apps, which increases engagement and personalisation. Deep linking can be implemented uniformly across a variety of platforms and devices, although fragmentation makes this difficult. Deep connections must continue to work after app upgrades or structural changes, which adds complexity to maintenance. To solve security concerns connected to deep linking, appropriate procedures for authentication and permission are required. Deep linking may expose users to security threats including unauthorized access and URL spoofing. If deep links are not deployed securely, user information may be compromised due to data leakage. Users need to be made aware of the dangers and developers must encourage thorough link authenticity checks. Deep linking has transformed how consumers engage with Android apps, despite these restrictions and security worries. In addition to fostering seamless app integration, it has enhanced user experience and app discoverability. Developers now depend on deep linking as a key technique for increasing engagement and offering individualized experiences. Deep linking is anticipated to develop further as technology advances, overcoming constraints and security issues. In the contemporary digital world, mobile applications have a big influence on every part of our lives. Android has emerged as the top platform among the several mobile operating systems, grabbing a sizable portion of the market. As they have developed over time to improve user experience and allow seamless connection with other apps and services, Android applications now offer a wide variety of features. Deep linking is one such innovation that has transformed how people interact with apps and information on Android devices. Deep links connect a specific link to a specific piece of content or feature within the app by using a unique URL structure or URI (uniform resource identifier). The operating system detects a deep link when a user clicks on it and sends them to the appropriate app content. Since 2008, both iOS and Android havesupported the most popular deep link. The mobile OS can register URI schemes with an app during installation if the programme wishes to be launched from the web. The format can vary depending on theplatform or app. For example:

<scheme>://<host>/<path>?<query-parameters>. As a result, the app is able to register and manage particular URLs that are activated from either inside or outside the app. The custom URL scheme often begins with a distinctive identifier selected by the app developer,such as “myapp://” or “mycustomscheme://”. This type of URL launches or comes to the foreground when it is clicked or triggered, enabling the connected app to manage and process the particular URL. But the problem with this scheme URL is that any other app or malicious app can register the same scheme and trick you into opening their app. As a result, they can steal your information. To address this problem, the Intent URL in 2013 and the App Link in 2015 came into picture. *(a) Intent URL:* It is limited to Android. Websites should call deep links in accordance with the intent URL, which is defined. In order to prevent misunderstanding, Intent URL includes the target app identification (i.e., package name) in the argument directly rather than invoking “myapp://path1”. *(b) App Link:* Comparing App Link to conventional scheme URLs, you can see an improvement in security. App Link employs domain verification, which necessitates hosting particular JSON metadata files on the website connected to the app, to establish the connection between a domain and an app. By ensuring that the programme only handles URLs from recognised domains, this verification lowers the possibility of spoofing or harmful usage. In some circumstances, the system may ask the user to approve the action when an app attempts to handle an app link URL for the first time. This adds an additional layer of protection and prevents deep connections from being handled by unintentionalor unauthorised apps. The “autoVerify” element found in the app’s manifest gives App Link users a way to validate URLs. In order to avoid hijacking or manipulation, this guarantees that the URLs linked to the app are routinely checked.

III RELATED WORK

Deep linking is a powerful technology that allows users to seamlessly navigate from a website or other apps directly to specific content within a mobile app. It greatly enhances the user experience by bypassing the app’s home screen and providing a more streamlined and targeted interaction. However, the implementation of deep linking is not without itschallenges. One of the main hurdles is compatibility across different mobile operating systems. Android and iOS, for instance, have their own distinct deep linking mechanisms, with android relying on URI schemes and iOS favouring Universal Links. This means that developers need to adopt differentapproaches and strategies to support deep linking on both platforms. Yifei Ma et al. [1] addressed Deep links are an essential tool for supporting programmed execution at specified app locations.By enabling users to go to specific areas or features inside the app, the release of deep links for applications is essential for enabling the programmable execution of apps. This feature improves the user experience and makes it possible to navigate the app ecosystem without any interruptions.

Xiaoxing Liu et al. noted that the primary method for using portable devices to access the Internet has evolved into a mobile application (app) [2]. Accessing a particular "content page" within an app necessitates navigating through various action within the app from the landing page. The navigation graph of Android apps can easily be built using a dynamic approach. Yizheng Liang et al.

addressed malicious URLs [3] that might accidentally or knowingly be included in the code or resources of Android applications. Researchers help preserve the security of the app ecosystem, stop fraudulent activity, and increase user confidence by discovering and reporting such URLs. Zhiqiang Yang et al. [4] tried to find out the transmission of sensitive data is secured or not, whether the transmission was intended for the user or not from an Android app. It is more likely to be user-initiated when a data-sharing feature is started or when a user grants authorization for a particular data transmission. Min Zhang et al. [5] focused on the component hijacking problem, one type of vulnerability that frequently affects Android applications. Attackers might use these weaknesses to their advantage by exploiting insecure applications, which can then leak private data and risk the security of Android devices' data. There are some reasons why relying solely on developers to fix the vulnerabilities is difficult. Elaine Chin et al. [6] analysed Android developers lunching of a webpage inside an application with the help of Web View. Web view and browser are two different things. Although this sophisticated interaction makes it easier for developers to support several platforms, it leaves programmes vulnerable to attack. Liu F. et al. addressed the subject of component hijacking [7] related to Android links, especially deep links, which is the main problem of this paper. The security issues associated with component hijacking, in which bad actors leverage flaws in deep link processing to lead users to unexpected or harmful app components, are highlighted by the authors. Component hijacking may occur as a result of incorrect deep link data validation, insufficient permission checks, or insecure deep link user input handling. De-Jun Wu et al. [8] addressed the possibility of malware cases rising due to the open environment of Android and the presence of third- party app stores. Researchers look for unusual or harmful behaviour in the manifest file, such as a high number of permissions or the inclusion of unidentified or hidden components. The manifest file for Android apps lists the permissions that are necessary. R. Dhaya et al. [9] addressed security flaws that have increased as a result of the growing use of Android-based applications, especially how these applications will handle links. Malicious links can be used to attack user devices, damage their security, and violate their privacy. Because of this, there is a demand for efficient static analysis approaches that can discover possible weaknesses in how links are handled in Android applications. Samuel Feldma et al. [10] focused on Android manifest files to identify potential malicious behaviour and patterns. The manifest's defined intents and intent filters specify the message-sending and receiving capabilities of components. Determine the communication channels the programme uses by analysing the intents and filters. Additionally, app versioning can complicate matters as changes to the app's deep linking structure may render existing deep links invalid, requiring careful handling and backward compatibility. Another significant issue is how to handle fall back scenarios when a user clicks on a deep link but doesn't have the corresponding app installed. In such cases, developers need to provide alternative options, such as redirecting the user to a mobile website or prompting them to install the app [15]. Furthermore, there can be instances where multiple apps on a device can handle a particular deep link, leading to the need for intelligent routing logic to determine which app should be launched based on user preferences or predefined rules. Maintaining app integrity is another critical aspect [16]. Deep links may become out-dated over time due to changes in the app's structure or modifications on the server side. Therefore, regular validation checks are necessary to ensure that deep links remain functional and up-to-date. Handling deep links within the app itself requires careful attention, including managing the app's state upon deep link invocation, retrieving the relevant data, and implementing security measures to prevent unauthorized access. Moreover, app store compliance is crucial, particularly with platforms like the Apple App Store, which have stringent guidelines for deep linking implementation. Adhering to these guidelines ensures the app's security and privacy measures are upheld. To overcome these implementation challenges,developers should follow best practices such as thorough testing across various devices and scenarios, clear documentation on deep linking implementation, robust error handling mechanisms, and providing user guidance on enabling or disabling deep link handling. By considering these factors and employing effective strategies, developers can successfully implement deep linking and provide users with a seamless and enhanced app experience.

IV ARCHITECTURE

In a mobile app, there are a number of architectural factors to consider while constructing a WebView object. See Fig. 1. A WebView object is first created by the native mobile app framework as the first step in the architecture. Providing features including page loading, JavaScript execution, and navigation, the WebView object serves as a container for presenting online information. The required parameters, including JavaScript enablement, cache management, and the definition of additional WebView attributes, are defined when it is created. It is then necessary to load a URL into the WebView object that has just been constructed. This entails launching a network request to get the web data related to the given URL [17]. In order to handle HTTP/HTTPS requests, control cookies, and cache resources, the WebView interacts with the network layer [20]. The web content is then shown in the WebView when the server's response has been processed. Handling WebView events is a part of the WebView architecture as well. The completion of a page load, navigation requests, form submissions, JavaScript call-backs, or failures are examples of events. Event delegates or listeners

that catch these events are built into the WebView object. Developers may respond appropriately since the WebView automatically launches the call-backs or event handler in the app logic when an event happens.

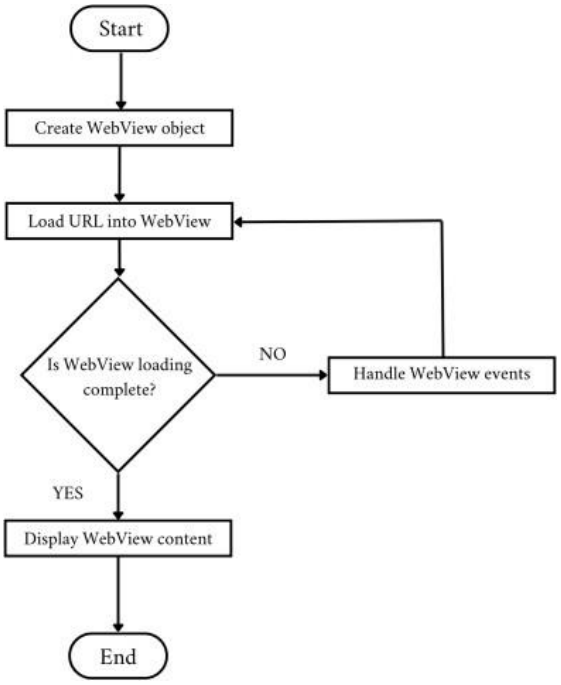

Fig. 1 WebView App Creation Framework

An event such as the conclusion of a page load may start additional processes or modify the user interface. The WebView component must be included in the app's user interface in order to display the WebView content. The WebView object offers ways to present web material in a specific layout or view. The UI is managed by the native mobile app framework, which also places the WebView component on the app's screen and supports user interactions with it. The rendered content of the WebView, including HTML, CSS, pictures, and media, is shown to the user, resulting in an intuitive surfing experience inside the app.

An activity is a key component of the user interface (UI) of an application when developing for Android [19]. Users can interact with the user interface on a single screen that it depicts. Activities often correlate to a particular feature or user process inside the programme, and they act as the entry points for users to access various portions of an app. The methods and callbacks in the activity class determine the behaviour and lifespan of the activity. When an activity becomes visible to the user but is not yet in the forefront, the onStart() function is invoked. It is generally employed to initiate or resume processes that ought to run only when an action is both visible and interactive. The activity loses attention and moves out of the foreground, which triggers the call to the onPause() function. Prior to the activity going into the background or maybe being deleted, there is a chance to preserve data, release resources, or carry out other clean-up tasks. See Fig. 2. The onStop() function is triggered when the user's view of the activity becomes invisible. It can be invoked when an activity is either ending or starting a new one. This function enables the execution of specific actions or clean-up tasks that are relevant when an activity is not visible. Upon termination of the activity, the onDestroy() function is called.

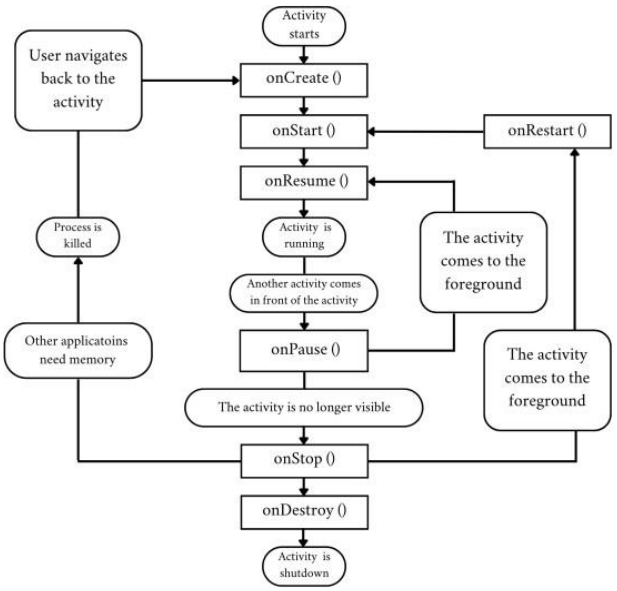

Fig. 2 Android Activity Lifecycle

This provides an opportunity to release resources or complete any necessary clean-up tasks before the activity is completely removed from memory. Because the system occasionally ends the activity without using this function, it is not always called. When an activity is relaunched after being stopped, the onRestart() function is invoked. It is a sign that after briefly stopping, the action is starting to move back into the forefront. Following this function are the onStart() and onResume() methods. If an activity is briefly deleted and then restored by the system, these techniques are used to save and restore its state. You can save any required data by using onSaveInstanceState() prior to the activity being paused or restarted. When the activity is being rebuilt, the function onRestoreInstanceState() is invoked, enabling you to restore the previously saved state. These lifecycle methods include hooks that let you carry out particular activities at various points in the lifespan of the activity. It's essential to comprehend and apply these techniques appropriately in order to manage resources, save data, and guarantee a smooth user experience while the activity switches between various stages.

V. METHODOLOGY

Before going to create a web-view app make sure to install and configure all the required software and tools. Choose a development environment based on your preference, such as Windows, macOS, or Linux. In this paper, Windows 10, an i7- 4th generation with 16 GB of RAM and a 256GB SSD, is used. Android Studio is used as IDE for app development. It included Android SDK which has necessary tools,

libraries, and emulators for testing. For testing and communicating with the Android emulator, PowerShell is used in this paper. Android Debug Bridge is another tool to connect the Android emulator for security testing.

A. WebView App Creation

There are several steps involved in creating a web-view application. The procedure is outlined in the following paragraphs. Building a web-view app requires a URL. In this paper a website is created and hosted on a free web hosting platform called 'netlify.com'. The URL used to create a web-view app is 'https://rgniyd.netlify.app'. To setup web- view in your app, open the layout file, i.e., *activity_main.xml*, and add a web-view element. Customise the web-view's attributes as needed. Open the *MainActivity.java* file and set up the web-view configuration. Retrieve the web-view instance from the layout file. Enable JavaScript and other web-view settings. Then load the initial URL into the web-view. To handle different web-view events, such as page loading, errors, and navigation, implement a web-view client. In order to handle certain URLs or deep links, override the *shouldOverrideUrlLoading* function. Set up your app to handle deep links or load only certain portions depending on the URL that was loaded. To go through the web-view history, you may optionally manage the Back-button behaviour using the *onBackPressed* function. Declare an intent filter for the activity that hosts the web-view. Make sure to add these three things in the intent filter i.e. action, category and data in *AndroidManifest.xml* file. Connect the Android device or set up an emulator to run the app. For the execution of the experiments conducted in this paper, a Nexus 5 smartphone model with a 2.3GHz CPU and 2GB RAM, running Android OS 6, was utilized. Build and run the app from Android Studio then release it into the preferred location. Test the web-view functionality by loading different web pages. Verify the deep links in order to open it in the application and ensure proper navigation.

B. Testing Deep Links

Perform a set of actions in order to test deep links. To start, choose the deep link URL you wish to test. It must follow the 'scheme://host/path?parameter' format [22]. As soon as you receive the URL, set up a test environment by making sure all the appropriate hardware, emulators, and target apps are loaded and ready for testing. The next step is to create the deep link by building the URL with the appropriate parameters or routesegments [23]. All necessary query parameters and route segments must be included. You have a few options for triggering the deep link once it is prepared. If a deep link is available through a website or another programme, one way to access it is to click on the deep link URL. The URL should open in the desired application when clicked. If you have access to the command-line interface, using the ADB command is an additional choice. You may transmit an intent with the deep link URL by connecting the device or emulator and executing the necessary ADB command. As an alternative, you may initiate the deep link programmatically by utilising the appropriate APIs if you have access to the app's source code or are constructing an app [24].Once the deep connection has been activated, watch to see how the destination app reacts. Make sure the deep link opens the appropriate page or executes the desired action in accordance with the deep link URL. It's important to investigate diverse possibilities when testing. Test a variety of deep linkages, including legitimate, improper, and edge instances. Check to see if the app processes them properly and offers proper feedback or error management. Testing the deep link undergoes two different methods. One is static analysis [11], and another one is dynamic analysis [12]. To assess the potential risks related to exploiting deep links, it is essential to consider theAndroid version on which the app is operating. This can be determined by examining the Android Manifest file to verify if the minSdkVersion is 31 or higher. By using ApkTool to decode the app and analyse the Android Manifest file, you can quicklyidentify whether deep links (with or without custom URL schemes) are established [25]. Look for intent- filter components in the Android Manifest file. Deep links must be confirmed in order to be regarded as App connections, even if they have the android:auto Verify="true" property. Check for any potential setup errors that can prevent thoroughverification.

VI RESULTS & DISCUSSION

The goal of this research project was to develop an Android-based Web-View app and assess its usability, performance, and user interface. The Web- View app provided users with a native app interface through which they could browse and interact with online information. The app's WebView component was successfully included, allowing the app's UI to render web information. In order to enable users to traverse through web pages, appropriate navigation controls, such as back and advance buttons, were developed. The programme let users to manually input URLs or click links inside the WebView to load online pages. To guarantee correct rendering and suitable presentation, the app was tested with a variety of web sites that contained various forms of material, including text, photos, videos, and interactive components. The software displayed online material properly, maintaining the original design and visual aspects. Fig. 3 shows the successful installation of the app in the Android emulator. I store different links in the Android message box to check the redirection. When the user clicks any one of these links for the first time, the android will show an ambiguous message (see in Fig.4); once the user clicks on the 'always' button, they give permission for the links to open in the specific application. We can see the successful launch of the app from the specific link. And Fig. 5-6, shows that if an app receives an attempt to open it with a scheme or host that is not specified in the intent filter of the manifest file, instead of launching the app, it redirects the request to the web browser of the emulator.

Fig. 3 Install Application in Device

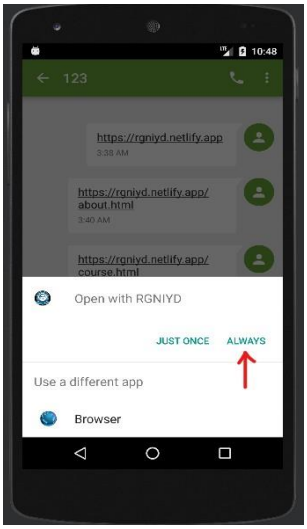

Fig. 4 Ambiguous Message Pop-Up

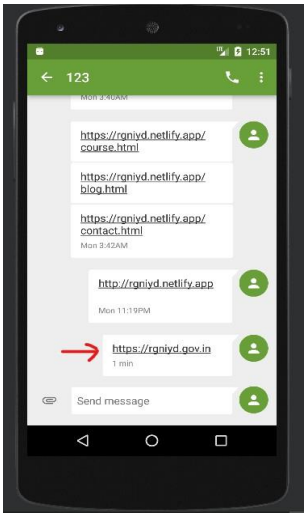

Fig. 5 Link with Different Host

VII CHALLENGES & BEST PRACTICES

In mobile marketing, deep linking is a crucial element that promotes involvement, persistence, and profits. The need to guarantee that deep links function properly in every medium is actually more important than ever. However, concerns with data transfer, security constraints, link routing, device compatibility, app installation status, and other factors may make its deployment more difficult [21]. To navigate these issues successfully, this section offers detailed insights.

A. Platform and Device Compatibility

Device fragmentation is indeed a significant challenge when it comes to deep linking, particularly in the context of the Android and iOS platforms. This fragmentation encompasses various aspects, including operating systems, screen sizes, resolutions, and hardware capabilities, which can make it difficult to ensure consistent and seamless deep linking experiences across different devices. For various platforms as well as for successive iterations of the same operating system, deep links must be set individually. Deep links may function flawlessly on one platform or OS version but malfunction on another as a result of this inconsistency. The two most popular mobile operating systems, iOS and Android, each offer a unique deep linking mechanism. Android uses Android App Links, but iOS utilises Universal Links.

These weren't always the norm, though. Prior to Android 6.0 Marshmallow, deep linking in Android mostly utilised unique URL schemes. But this presented a number of problems, including the possibility of URL scheme issues if numerous apps utilised the same scheme. To solve this, Android added Android App URLs in version 6.0, allowing URLs that are particular to an app to open instantly in that app, provided that app is installed. If an app enables Android App Links and is installed on the user's device, for instance, clicking a link in an email that begins with 'https://www.rgniyd.com/' will launch the app immediately. Maintaining compatibility across various Android versions presented a new problem for app developers with the advent of Android App Links. Although Android App Links are more dependable and secure, Android versions older than 6.0 do not support them. This indicates that in order to enable deep linking on these versions, programmes hoping for broad compatibility still need to use the earlier custom URL scheme technique. Developers use a variety of techniques to overcome the obstacles, including integrating contemporary Android programme Links or iOS Universal Links as well as unique URL schemes into your programme. This will allow your app to fall back to the specific URL scheme on earlier versions of the device while still using the better technique on devices that support it. Make sure your deep connections function properly in various settings by thoroughly testing them on a range of hardware and OS versions. In your testing matrix, incorporate both recent and historical releases of Android and iOS.

B. App Installation Status

The status of user installed the app or not can have a big impact on the behaviour of deep links. Deep links that point to a web browser or app store instead of the programme itself, if the app isn't installed, may impede the user's progress. Deferred deep links are one way for marketers to get around this problem. Although the programme must initially be installed, deferred deep links enable users to be routed to specific information. Once the app is installed, they save the data sent through the link and direct the user to the desired place. When a user interacts with a deep link, the term 'app installation status' refers to whether or not the user has your app loaded on their device. Depending on whether your app is already installed or not, numerous events might happen when a user clicks on a deep link. If installed, the deep link will open the appropriate in- app content immediately. If not, the link will often send the user to the app store where they may download your software, breaking the smooth user experience that deep linking is meant to offer. Link routing may be impacted by your app's status on a user's device, including whether it is installed, uninstalled, or operating in the background. It can be technically difficult to handle these cases effectively since each condition demands a separate routing path. Deferred deep linking implementation is a two- step procedure that involves both your marketing team and the developers of your programme. Your marketing team must comprehend the customer path and make plans for how to employ postponed deep links successfully. Deferred deep connections must work as planned across all platforms and devices, thus your developers must simultaneously include the technology into your programme.

C. Data Transmission Issue

Data transmission across deep networks may be a challenging operation. There may be a number of problems that prevent the data from going where it should or from being properly processed by the app. The data sent over a deep link must be accurately interpreted by the app. However, problems like improper data formatting or unknown special characters might lead to processing failures and invalidate the data. It might happen that the information added to a deep link is unreliable or inaccurate. Technical problems or mistakes in the deep link generation process might be to blame for this. When there are problems with data transfer, the user experience may suffer greatly. The personalised experience you had in mind might not be realised, or users might not be sent to the desired area inside the app. This can make users less engaged and aggravated, which might affect conversion rates and general app performance.

Maintain open channels of communication, include them in the planning process, and solicit their opinions on link routing tactics. To discover and resolve any possible link routing issues, test your deep connections across a variety of devices, operating systems, and app states. The marketing and QA teams should work together to make sure that the user experience supports your marketing objectives.

D. Security Restrictions

As OS makers take methods to stop harmful programmes from abusing deep links, security constraints may have an influence on deep linking. The user experience may be impacted by these restrictions by way of extra prompts. Following recommended deep linking procedures, such as making sure your website association files are properly configured and informing users as to why particular prompts may occur, will assist to prevent these problems [14]. Before being routed from a deep link to an app on various platforms, users are prompted to confirm. This extra step might obstruct the user's trip and perhaps affect conversion rates. Certain data cannot be sent through deep networks to reduce the threat of data breaches. The degree of customised service you may offer may be impacted by this. Contextual deep connections enable developers to provide consumers with a much more individualised and focused app experience immediately when they launch an app. Developers may use these connections to create features that send customers right to a unique welcome page after downloading or let them instantly upload a coupon. Additionally, they make it possible for marketers to compile data on the effectiveness of marketing initiatives and advertising campaigns.

VIII CONCLUSION & FUTURE WORK

Deep links are now a crucial component of creating modern mobile apps since they offer a smooth and practical approach for users to access particular app information. Deep links enable users to access the app's content directly from outside sources like websites, emails, or other applications by linking certain app screens or actions to specific URLs. Deeplinks' correctness and operation are ensured through thorough testing, providing a positive user experience. The establishment of deep links is an essential component of app development. Developers may make it simple for users to access particular material or carry out particular activities by specifying and tying deep links to specific app screens or actions. The deep link URLs must be registered in the app's manifest file, intent filters must be set up, and the app must handle incoming deep link intents. Designing a coherent and relevant URL structure is crucial for deep link deployment. It should give a direct route to particular screens or activities and represent the hierarchy of the app's content. The user experience is enhanced by a well- designed URL structure that enables consumers to comprehend the deep link's purpose. To handle deep link intents, the app's manifest file's intent filters must be properly configured. The host names, routes, and schemes that the app may handle are specified by the intent filters. Developers may make sure that the app reacts correctly when a deep link is opened by specifying intent filters. Deep links should direct visitors to the appropriate app panels or activities. The incoming deep link intents must be handled by developers, who must then take the appropriate steps, such as launching a certain activity or fragment or carrying out a particular operation based on the deep link data. Testing is necessary to guarantee proper operation, check deep link data, manage edge situations, and deliver a consistent user experience across all devices and circumstances. Deep connections may be used to increase app usage, boost user engagement, and boost user happiness by developers when they build and test them properly.

At the moment, deep links are usually defined statically in the manifest file of the app. In the future, apps could be able to create deep connections on the fly based on the context or preferences of the user thanks to improved dynamic deep linking capabilities. Because of this, deeper connection experiences might be more individualised and contextual. The security of deep link interactions is essential because they may provide access to sensitive information or open up app functionality. Future work may concentrate on enhancing security controls for deep link handling, including safeguards for user privacy, procedures for secure deep link source validation, and prevention of harmful deep link assaults. Establishing universal standards for deep link structures, naming conventions, and metadata formats could simplify deep link implementation and improve interoperability between apps and platforms. This would make it easier for developers to adopt and integrate deep linking capabilities into their apps.

XI REFERENCES

[1]. Yifei Ma et al., "Aladdin: Automating release of android deep links to in-app content". In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pages 139–140, Buenos Aires, 2017.

[2]. Yifei Ma, Xiaoxing Liu, Rui Du, Zhiqiang Hu, Yun Liu, Min Yu, and Guangtai Huang, "Droidlink: Automated generation of deep links for android apps". arXiv preprint arXiv:1605.0692, 2016.

[3]. Yizheng Liang and Xiaodong Yan002C "Using deep learning to detect malicious urls". In 2019 IEEE International Conference on Energy Internet (ICEI), pages 487–492, 2019.

[4]. Zhiqiang Yang, Ming Yang, Yanchao Zhang, Guofei Gu, Peng Ning, and Xiaofeng Steve Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection". In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 2013. Association for Computing Machinery.

[5]. Min Zhang and Heng Yin, "Appsealer: Automatic generation of vulnerability specific patches for preventing component hijacking attacks in android applications". In Network and Distributed System Security Symposium, 2014.

[6]. Elaine Chin and David Wagner, "Bifocals: Analyzing webview vulnerabilities in android applications". In Proceedings of the 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks, University of California, Berkeley, USA, 2014.

[7]. Liu F., Wang C., Pico A., Yao D., and Wang G.,"Measuring the insecurity of mobile deep links of android". In Proceedings of the USENIX Conference on Security Symposium., 2017.

[8]. De-Jun Wu, Chen-Hung Mao, Te-En Wei, Hao-Ming Lee, and Kuo-Pei Wu., "Droidmat: Android malware detection through manifest and api calls tracing". In 2012 Seventh Asia Joint Conference on Information Security, pages 62–69, Tokyo, Japan, 2012.

[9]. R. Dhaya and M. Poongodi., "Detecting software vulnerabilities in android using static analysis". In 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, pages 915–918, Ramanathapuram, India, 2014.

[10]. Samuel Feldman, David Stadther, and Bin Wang., "Manilyzer: Automated android malware detection through manifest analysis". In 2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems, pages 767–772, Philadelphia, PA, USA, 2014.

[11]. Brian Chess and Gary McGraw., "Static analysis for security". IEEE Security & Privacy, 2(6):76–79, November– December 2004.

[12]. Hossain Shahriar, Kun Qian, Md Anwarul Islam Talukder, David Lo, and Nirav Patel., "Mobile software security with dynamic analysis". In 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC), pages 223–224, Taipei, Taiwan, 2018.

[13]. Redfox Security Website Available: https://redfoxsec.com/blog/exploiting-android-webview-vulnerabilities/

[14]. Deep Link Exploitation: Introduction & Open/unvalidated Redirection. Available: https://medium.com/mobis3c/deep-link-exploitation-introduction- open-unvalidated-redirection-b8344f00b17b

[15]. Liu, F., Cai, H., Wang, G., Yao, D. D., Elish, K. O., And Ryder, B. G., "MR-Droid: A scalable and prioritized analysis of inter-app communication risks". In Proc. of MoST (2017).

[16]. Daoyuan Wu, Yao Cheng, Debin Gao, Yingjiu Li, and Robert H. Deng. 2018. SCLib: A Practical and Lightweight Defense against Component Hijacking in Android Applications. In Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY '18). Association for Computing Machinery, New York, NY, USA, 299–306.

[17]. P. Gadient, M. Ghafari, M. -A. Tarnutzer and O. Nierstrasz, "Web APIs in Android through the Lens of Security," 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada,2020, pp. 13-22.

[18]. Y. Liu, L. Li, P. Kong, X. Sun and T. F. Bissyandé, "A First Look at Security Risks of Android TV Apps," 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), Melbourne, Australia, 2021, pp.59-64.

[19]. A. Mendoza and G. Gu, "Mobile Application Web API Reconnaissance: Web-to-Mobile Inconsistencies & Vulnerabilities," 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2018, pp. 756-769.

[20]. A. Possemato and Y. Fratantonio, "Towards HTTPS Everywhere on Android: We Are Not There Yet," in Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), August 2020, pp. 343-360.

[21]. Yutian Tang, Yulei Sui, Haoyu Wang, Xiapu Luo, Hao Zhou, and Zhou Xu. 2020. All your app links are belong to us: understanding the threats of instant apps based attacks. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020). Association for Computing Machinery, New York, NY, USA, 914–926.

[22]. Exploiting Deep Links in Android - Part 1. Available: https://inesmartins.github.io/exploiting-deep-links-in-android-part1/index.html

[23]. Exploiting Deep Links in Android - Part 2. Available: https://inesmartins.github.io/exploiting-deep-links-in-android-part-2/index.html

[24]. Testing Deep Links. Available: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0028/

[25]. Best practices for Deeplinking in Android. Available: https://proandroiddev.com/best-practices-for-deeplinking-in- android-1dc1ea060c0c