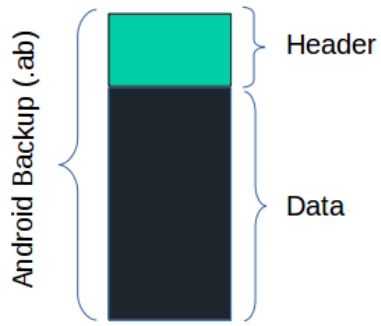
Exploiting Android backup.

Posted by Cristian R. April 28, 2018 in Android Pentesting



Here, we will see how to exploit an Android application that allows to be backed up, this is achieved when the application has the allowBackup flag set to true on the AndroidManifest.xml and can be exploited even on devices that have not been rooted. The Android Debug Bridge provides the ability to easily create an application backup and just as easy to restore it. We will see how to leverage this and other tools to exploit these kind of applications.

When you backup an Android application, you will get a multi-part file, the first 24 bytes represent the header and contains information about format, compression and encryption of the backup, the next part is the backup data, which is a compressed tar file that is optionally encrypted and includes shared preferences, files and databases, the backup also contains the apk (if the option is specified when backing up).

In order to exploit an Android application through backup, we need to:

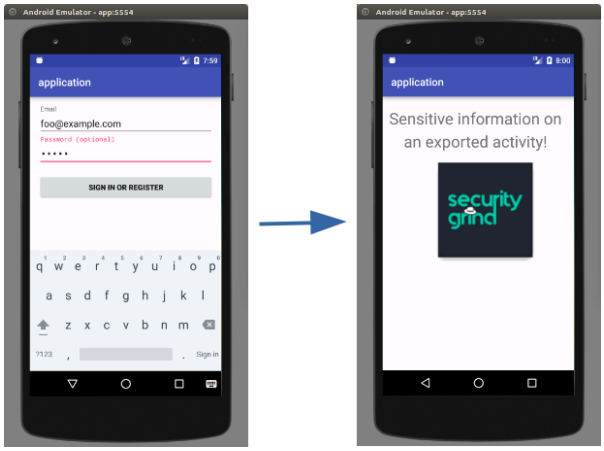
1. Get the backup.
2. Separate the backup data from the header.
3. Make the required changes to the backup data.
4. Repackage the modified data.
5. Get the header from the original Android backup file.
6. Prepend the header to the repackaged data.
7. Restore it into the device.

Each of the steps of this process is described below:

Getting the application backup

If the *allowBackup* flag is set to true on the application, getting an application backup is fairly easy, for this we will use the *backup* command of the Android Debug Bridge (adb). To do this, we need to run the Android virtual device where our vulnerable application ins installed, in this case we will be using an application created for this purpose which is simple, two activity app that looks like this:

cristian.rodriguez@securitygrind.com

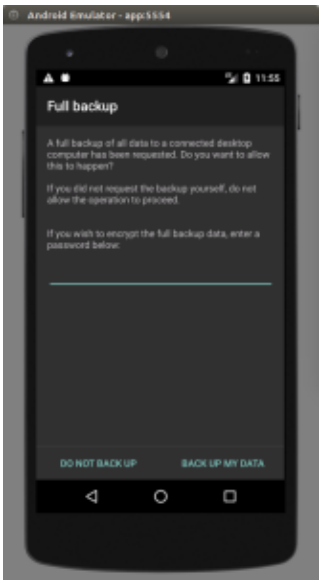


Once we connect to the Android virtual device through the adb, we proceed to create a backup of our application:

```
root@TuX: /opt/securitygrind/backup
root@TuX:/opt/securitygrind/backup# adb devices
List of devices attached
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
emulator-5554   device

root@TuX:/opt/securitygrind/backup# adb backup -f appbackup.ab com.securitygrind.application
Now unlock your device and confirm the backup operation...
```

You'll get a message saying to you need to unlock your device and confirm the backup operation, to do this, go back to your virtual device and hit the "BACK UP MY DATA" button, do not provide a password for encryption as this may complicate things a little bit more:



Once the backup finishes, you can go back to your workstation and see that the backup file was created:

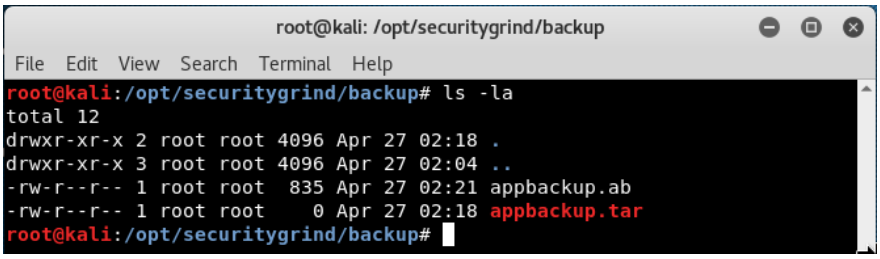
```
root@TuX: /opt/securitygrind/backup
root@TuX:/opt/securitygrind/backup# ls -la
total 12
drwxr-xr-x 2 root root 4096 abr 27 00:05 .
drwxr-xr-x 6 root root 4096 abr 27 00:05 ..
-rw-r----- 1 root root  835 abr 26 23:57 appbackup.ab
root@TuX:/opt/securitygrind/backup#
```

Separating the backup data from the header.

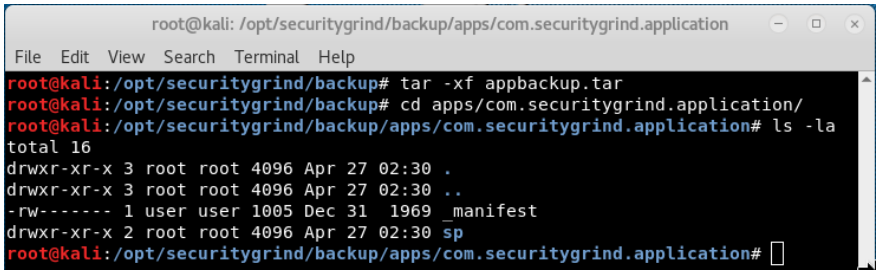
For this part we will be using *dd* to to separate the backup data from the header and then we'll use *openssl* with *zlib* support to decompress the backup data. Most *openssl* installations does not support *zlib*, so, it is recommended to fire up a virtual machine and use it to build an *openssl* installation that supports *zlib*, you

can do this by following the process described here. Once we have *zlib* support, we do the following:

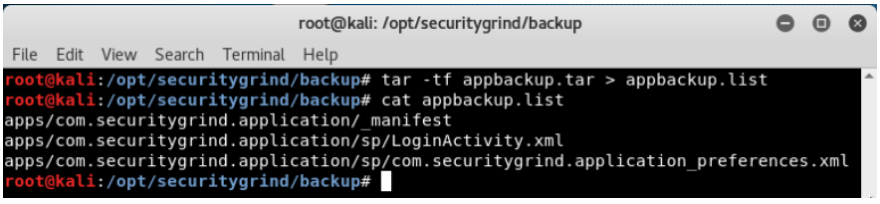
With the *dd* command shown above, we specify a block size of 24 bytes (bs=24) and tell it to skip the first block (this is the header), we pipe the result into *openssl* and use zlib to decompress it into a tar file (this is the backup data):



We can then extract the backup data *tar* file and inspect it's contents; we can see the *apps* folder containing another folder with the name of the backed up application and inside this one we see a *_manifest* file and a folder named *sp* (shared preferences).

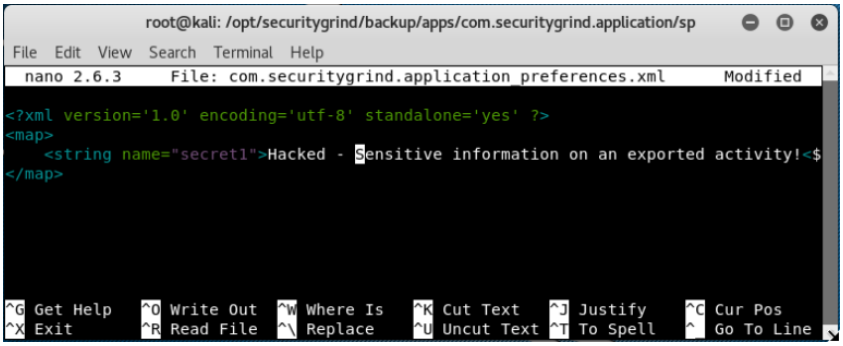


Now, we need to create a list of all the files contained in the *tar* file; this will be needed after we make the require changes to the backup data and proceed to re-package. To create this list we do the following:



Making the required changes to the backup data

We now proceed to make some changes to the backup data we extracted from the *tar* file. To do this, we choose one of the files within the *apps* folder, open it with a text editor (*nano* in this case) and make some changes to one of the files within the shared preferences folder.

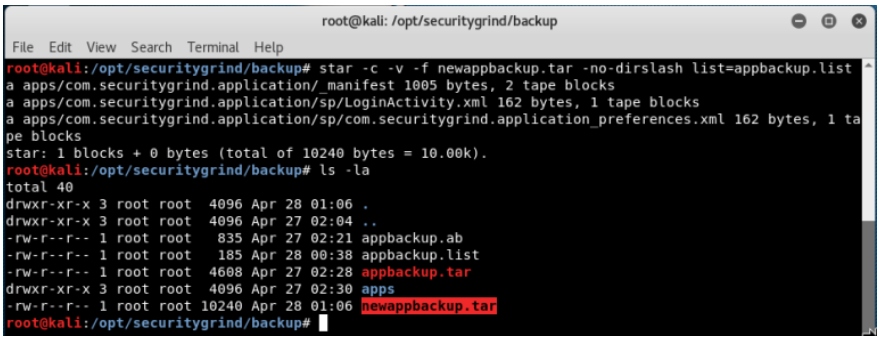


Repackage the modified backup data

Once the required changes are done, we need to repackage the *apps* folder into a *tar* file, to achieve this we can use the Android Backup Extractor; an utility based on the *BackupManagerService.java* that can be used to extract and repack Android backups created with *adb backup*. You can check the project code here and find the *deb* package here. Once you have downloaded the *android-backup-toolkit* zip file, decompress it and navigate to */android-backup-tookit/star-bin/star-ubuntu-lucid*; here you will find *deb* packages for both *amd64* and *i386* architectures, choose the one that applies to your case and install it with *dpkg*:

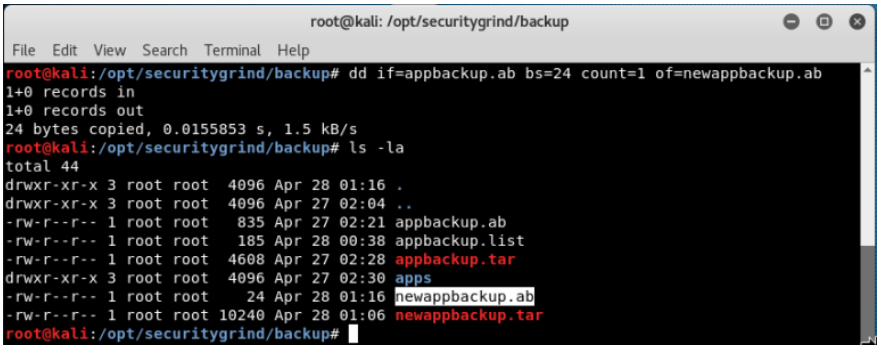
dpkg -i star_1.5final-2ubuntu2_amd64.deb

With the *star* command line is installed, we use it to repackage the *apps* folder into a new tar file containing the backup data with our previously done changes, to keep a strict order of files we set the *list* parameter to the list of files we previously created with *tar*, like this:



Getting the header from the original backup file and prepend it

We need to first get the header (first 24 bytes) from the original backup file, to do this we use the *dd* command again with a block size of 24 bytes, but with the count parameter instead of the skip parameter used the a previous step above, as follows:



Now, we proceed compress the previously repackaged data *tar* file using *openssl* with *zlib* and then we prepend the header into it , like this:

Restoring the modified backup file

Finally, we restore the modified backup file into the Android virtual device, to do this, we use the Android debug bridge again, only this time with the *restore* option:

We head to the Android virtual device and hit the “RESTORE MY DATA” button:

Once the restore is completed, you can open the application and see if the changes made any effect to the content or behavior of the application:

Here we can see that the changes we made to the shared preferences files indeed took effect.

Conclusion

In this article we saw how to exploit an Android application that allows backup by modifying one of the application resources (shared preferences). This risk is actually very easy to mitigate, you just need to make sure to set the *allowBackup* flag to false in the *AndroidManifest.xml* file.

1 Comment

seoshare May 18, 2020 at 1:07 am

Thanks.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name *

Email *

Notify me of follow-up comments by email.

Notify me of new posts by email.

I'm not a robot

reCAPTCHA
Privacy - Terms

Post Comment

Email

contact@securitygrind.com

Social