Security Analysis

Acutus

Rutvik Sanghavi
Bachelor of Science in Computer Science
University of Saskatchewan
January 20, 2019

Overview

This document talks about the types of security used within the jailbroken application Acutus. From this point on in the document a security for this type of application will be mentioned as a DRM (Digital Rights Management). In general, there are many types of DRM. Some are more effective than others, I will do my best to explain the type of DRM that has been used in Acutus and what type of attacks can be used to bypass it.

Purpose of the Analysis

This security analysis for the jailbroken application Acutus was undertaken to display the common practices of DRM within the jailbreak community and how hackers have bypassed these DRM's. I mainly hope to encourage better practices by the developers of this community and show them how to avoid some of these attacks.

Methods

1. Status Check of Package

With the common practice of jailbreaking, the dpkg (Debian Package) software is included by default. It is a low level tool to manage packages on Unix-based operating systems. It also has a useful function that provides us with the status of an installed package. In the very initial versions of Acutus, I had used a compiled postinst[1] to utilize this function to make sure the package was installed from the correct source.

```
$ dpkg -s dpkg
Package: dpkg
Status: install ok installed
Priority: required
Section: Packaging
Installed-Size: 828
Maintainer: Jay Freeman (saurik) <saurik@saurik.com>
Architecture: iphoneos-arm
Version: 1.14.25-9
Depends: bash, bzip2, coreutils-bin, diffutils, findutils, gzip, lzma, ncurses, tar
Description: package maintainance tools from Debian
Name: Debian Packager
Homepage: http://wiki.debian.org/Teams/Dpkg
```

FIGURE 1. EXAMPLE OF DPKG -S FUNCTION

Any strings used within the source files show up as a huge combined string within the binary of the application. In the case of Acutus the file within the DynamicLibraries folder labelled Acutus.dylib. Basically this file can be split into 7 parts:

1. Path of the libraries used
2. Raw strings
3. Used function calls
4. Defined variables and object classes
5. Literal name of the library and frameworks used

[1] A script that is to run after the installation of the package. Read more about it here

6. Another set of defined variables and object classes
7. Parts 1-6 will repeat depending on the number of architectures it was built for

FIGURE 2. EXAMPLE OF A BINARY DYLIB FILE (PART 1)

Knowing this information a hacker could take advantage of this easily by creating a shell script that simply echoes the information that the application uses to check. The only thing left after creating the shell script is to use a hex editing software to replace the *dpkg* string within the binary file. Now the trick with that is to name the shell script the same length as the name we will be replacing in the binary file. In our case *dpkg* is 4 letters so we can name our shell script, *crak*. The key with editing using a hex editing program is to keep the length of the binary file unchanged.

2. File Existence Check

After the failure of the status check approach I thought to use obscurity and confusion to my advantage by checking for many unnecessary files and hiding the checked files path behind obscure strings. This approach checked for the postinst file, and checked if it came from the right source. If so creates a new file and the name is arbitrary. Then the second check if that file was not spoofed but the key here was to create random arbitrary strings around the main paths to confuse the hacker.

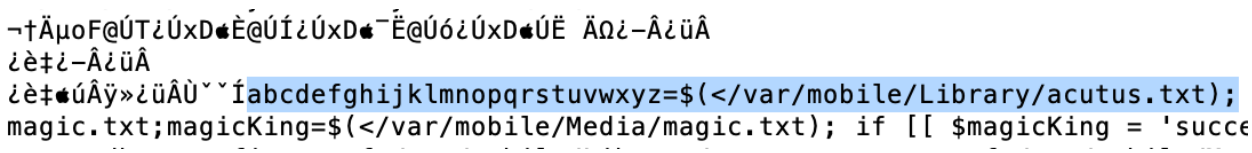

FIGURE 3. EXAMPLE OF AN OBSCURED STRING

Now to be very clear, Figure 3 is not the best example as to even the novice of a developer can tell the path apart from the gibberish. It was mainly to show with extra gibberish, the path, and any type of string can be well hidden. As to no surprise to any developer reading this, it can be easily bypassed with the same trick as shown above in the ‘status check of package’ method. Simply by hex editing the string to any known file (literally any file) and the check would pass. Normally with the file existence method, a file within the dpkg folder is checked, for example with Acutus this file would be, “*/var/lib/dpkg/info/com.thebigboss.acutus.list*”. The hex editing trick only works when the file stays the same length, so to specifically avoid that problem a hacker changes that path to this, “*/var/lib/dpkg/info/////////////////////cydia.list*”. The length of the

file path this way stays the same and the file check could be tricked to true using any files within the limit of the length of the original path.

3. Server Check using Cydia API

The server check was implemented in the later versions of Acutus, but I will not be particularly going in-depth for this method. However, I will speak in detail about the server check in a different application when it was significantly improved and is worth talking about. In brief, this approach in Acutus takes the Unique Device ID (UDID) from a private library provided by Apple, and it is called libMobileGestalt. This UDID is sent to a server where a simple Cydia API is called upon which checks if this particular UDID was used to purchase the said jailbroken application. If Cydia sends us that it was “purchased with this UDID”, then we can return a success string of our choice back to the application, otherwise return a fail string. After the return of a success or a fail string by the server, the application checks and acts on the outcome appropriately. Sadly, and more specifically with Acutus, a hacker was able to use the URL string from the binary to send bunch of random UDID’s to the server in an attempt to know the fixed success and the fail string. This approach was obviously improved upon in my future jailbroken applications which is why I would not like to provide any weak examples or go in-depth when I can easily do so with better examples when analyzing another application of mine.

References

https://iphonedevwiki.net/index.php/Tweak_DRM

I have greatly contributed to the "*Types of DRM*" section at this URL which re-iterates what I have mentioned in this analysis but with a generic application and more so generic examples.