Mobile Application Security

By:
Ethan Tuning
Collin Nolen
Gavin Rouse

Mobile phones have evolved greatly since their conception in 1983. The cell phone started out as a simply way to call someone on the go, now to a device capable of doing some pretty intense computing. Gaming, photo editing, social media, email, and many other things are just an example of what is possible on a mobile phone of today. The power that these phones possess today rival some of the most powerful computers of the past. Mobile phones have surpassed every medium of “computing” device as far as overall usage goes in 2014. It is 2017 and the gap is only getting larger. With all of this growth, evolution, and power, there arises a new issue regarding mobile devices(phones, or now tablets), that is security. Mobile devices cannot just do all of these wonderful things on their own, but when paired with the right software we can see the full potential of the mobile platform. This software, known as an application or app, can handle some extremely sensitive data. Credit card info, GPS location, passwords, and much more, are all examples of the data that these apps could be, and most likely are, handling at any given time. So how are the developers of apps integrating features to keep apps secure? How are hackers exploiting apps to get this valuable info.? How can we, as developers, and consumers ensure our data is secure? In this paper we will explore the vulnerabilities of mobile application, how developers can mitigate exploits, and examples of just how attackers go about exploiting poor application development.

In what ways are mobile applications vulnerable? Well to start out we have to realize that when modeling out threats on the mobile platforms there are a few things to consider. For one, the most natural migration of applications to appear on the mobile platform is web applications. We see all the time that a company or individual will have

a web app as well as a mobile version. With this we have to realise that with the mobile platform that these applications now have to take in account more client side threats. Another thing we have to look at is how exactly people are obtaining these applications. With computers it is easy to install anything from about anywhere. Most software is downloaded and installed via the internet. Most people, even those that are not computer savy, can look at a website and see that it may be a little fishy, or something feels a little off. Most harmful websites go unexplored by the everyday user, and resulting in not a lot of harmful software being downloaded to your computer. But, with mobile applications there is a rather large sense of trust that users put into the application marketplaces. We all download and install app after app onto our devices daily. What most of us don't realize is that even the app marketplaces can be filled with harmful software, or contain apps that have not been properly tested with regards to security. People can upload close to anything to Google Play Store. It is a little harder to get an app onto the App Store on ioS, but there is still some risk. Even apps realised from large, trustworthy companies can have vulnerabilities within them and could result in an exploit being taken advantage of by an attacker. I even find myself not really caring what I download and install onto my phone, because there is a sense of trust in the Google Play Store. There has been a lot of research done by Penn State, NC State, IBM, and a few other organizations and companies regarding mobile app security. A specific study by IBM showed that about 95% of mobile apps are vulnerable to attacks in some way. Also about 40% of enterprises release applications that have not been properly tested for security related issues, despite having huge budgets on these

applications. Other studies also had similar results. So what are the actual vulnerabilities within these applications? Well there are many ways to attack software, and attackers will do anything to get what they want. With mobile applications it is super easy for a malicious user to download the app to their device and reverse engineer it into its original source code. It is more easily done with the Android platform, but still possible with ioS. With the source code in their hands, attackers will run through every line of code until they find something they can exploit. Then they will try to construct an attack on other users that may have installed that particular application. These attacks can range from denial of service(DoS), SQL injection, request manipulation, etc. After researching studies as well as reading OWASP's top mobile threats, I have compiled a list of the most common vulnerabilities on the mobile platform. First, misuse of file permissions. Developers will mistakenly elevate a file's permissions within their app runtime. This could lead to attackers manipulating the file and having it be ran when it should not be ran. Also with this, the OS does not encrypt data when reading or writing to the SD card, so it is up to developers to take this into account and take the necessary steps in locking this down. Second, apps can sometimes write sensitive user data to log files, appmanifest, xml, etc. and in a world where phones are on us at any given time an attacker could physically steal a phone and have all that data at their fingertips. Or they could even go about accessing that data remotely in some way. Third, developers often disable certificate validation causing server-side authentication to be bypassed. This can lead to man-in-the-middle attacks, because your network traffic will be easily viewed by anyone sniffing traffic. Fourth, oftentimes apps will use poor user

authentication techniques that rely on IMEI number, MAC addresses and others. Sometimes these can be tampered with and can lead to information of a user to be released. Finally, on the Android side of things there is something called intents. Intents are basically a way that apps interact with one another. There are two types, explicit and implicit. With implicit intents, there is possibility for malicious apps to request data from another app and retrieve vary valuable info. This can also be manipulated to make sensitive OS calls that could disclose even your GPS location. As you can see there are many ways that applications can be exploited and these are only a few. There are many more ways as well. So how do developers deal with all of these ways attackers can exploit things?

When it comes to mitigating insecurities and vulnerabilities within a mobile application, there are three main steps that developers should take in order to produce more secure applications. They are: information gathering, static analysis, and dynamic analysis.

Before actual code development occurs the first step of creating a secure application should occur. This is the information gathering stage. This part of the process is all about researching the application itself and the supporting infrastructure for the application. This step is important because without a good understanding of each aspect of the application and the supporting infrastructure and technologies around the application, a developer will not be able to properly identify when the application behaves in a way that it shouldn't or why it is behaving that way. This step of the process is often skipped entirely, or at the very least, not completed with the proper

length and depth. Developers will often dive headfirst into a project before they have a good understanding of the application or the infrastructure surrounding the application. This is a poor practice, as it leads to increased chances of insecure code being written and insecure practices being followed. In this step, a developer should research and identify as much useful information about the application and the infrastructure around it. Examples of things that should be researched are: will the application need to support 3G, 4G, wifi connections, or all three? Will the application need to support commerce transactions? What hardware components within the phone will the application be interacting with, such as the camera, the bluetooth, GPS, microphone, or USB. What frameworks will the application use? Will the application be interacting with other applications on the phone, such as contacts, email, autocorrect services, or social networks. These are just a few examples of the vast amounts of things to be considered and researched in this step. It is important for a developer to understand these things before beginning to develop the application, and if they don't understand something, they need to research it and ensure that they can properly, and most importantly, securely design and build an application and that any supporting infrastructure and technologies are also used securely.

The second step in creating a secure application is static analysis. Static analysis is the process of analyzing the source code of the application attempting to identify insecurities and vulnerabilities. Performing each of the three steps and doing them in the proper order is also important. This is the case because it will be very difficult for a developer to look at the source code and be able to identify insecurities in the

application if they don't fully understand each part of the application and how it should be used in order to stay secure. Therefore, if a developer does not correctly perform the information gathering step, they will not be able to correctly perform the static analysis either. When performing the static analysis, some examples of things to analyze and review are: what permissions is the application requesting, and does it really need the permissions that it gets? What resources does the application request to use, and does it actually use them and do it securely? What libraries is the application using? Are the libraries up to date, or do they have any known insecurities that can't be avoided and therefore, that library shouldn't be used? Does the application authenticate the user? Is the authentication done properly and securely? Is any sensitive data being saved on the phone, and is properly and securely encrypted? Does the application log any data, and if it does, is there any sensitive data being logged that could be exploited by an attacker? These are just a few examples of things to be reviewed during static analysis. There is much more that should also be analyzed and reviewed to ensure the security of the application.

The third and final step to help ensure security in mobile applications is dynamic analysis. In dynamic analysis, a developer should use all of the information that they collected during the first two steps to perform an informed and in-depth vulnerability assessment. Essentially, the developer is going to take on the role of an attacker and attempt to exploit any insecurities the application may have at this stage of the process. Again, it is important that the first two steps are done first and done correctly so that the developer can know exactly what an attacker might attempt to exploit in the application.

Something that an attacker might try, and that a developer should also try in order to prevent it from working is fuzz testing. Fuzz testing is when a user inputs massive amounts of random data into the application in an attempt to find coding errors or security loopholes, or to cause the application to crash. Another example is brute force attacks against keys, pins, and hashes. If a user is being authenticated on a mobile application through a pin, if there is nothing in place to keep an attacker from trying multiple pins in a row, like a timeout or account lockout, an attacker could easily try a large number of pins quickly and potentially gain access to an account that isn't theirs. Or if they successfully use a brute force attack on a hash used to encrypt data they could gain access to all of that data. Also, an attacker will look for any unencrypted data storage that could help them to exploit the application or the data itself could be sensitive data that should have been encrypted. One last example of something an attacker might try, and the developer should try also, is dumping the device or application memory in an attempt to obtain any sensitive information that could be used to further exploit the application. As with each other step, these are only a few examples of things to try, but there are many more things that should be considered.

If each of these three steps is done in the correct order and done correctly, developers will start to produce much more secure applications. Of course this is not a perfect system, it is still entirely possible that vulnerabilities will not be caught during this process and could be released with the application, but this process should minimize that chance as much as possible.

Mobile applications are unique in that the application can be heavily proded by malicious users under a very fine magnifying glass in search of vulnerabilities to take advantage of. Knowing how these users are able to get at the application's code is key in being able to know what vulnerabilities to look out for, and how to made sure the application is secure.

For Android applications, any user that downloads an application can get the source code of that application. This is possible because the Android applications are coded in Java then translated into machine independent java bytecode. Because of this, converting the java bytecode back into plain Java is relatively easy. This poses a problem for developers as any user has full access to the source code for you application so malicious users can look and identify vulnerabilities within the application to plan their attack. IOS applications can't be converted back into source code as cleanly as their Android counterparts. IOS applications are converted straight into machine code with heavy optimization which destroys much of the structure of the source code. However, variables names and hard coded values can still be scraped from converting the machine code back into a readable language, and can still be used in search of vulnerabilities within the application.

The process in which a user would convert an Android application back into source code isn't the most challenging which makes having secure coding practices much more important as vulnerabilities can be detected by a much wider audience. The first step of converting an Android app back into source code begins at the Google Play Store where a user will download an APK Extractor application. APK stands for Android

Package Kit and it is Android's package file format that is used to install applications and middleware. The APK Extractor will extract the APK file that was used to install the application onto a device and put it in the filesystem of that device that can be retrieved when plugging it into a computer.

Once the APK file exists on a computer, unzipping the package is required to view the packaged filed. To do this on a Windows machine, you would simply rename the '.apk' file to a '.zip' file and extract the file. The unpackaged file will now contain multiple files and folders, this is the file structure that all Android applications use. While many of the files you can simply open up in an editor to view the code, these files are Android specific files that determine how the application works with the device. To view the actual source code of the application the 'classes.dex' file needs to be converted to a readable file.

Converting the 'classes.dex' file into a readable file starts by downloading a tool called d2j-dex2jar, and as the name suggests, the tool converts '.dex' files into '.jar' files. This application lives on github and is available free to anyone. Once d2j-dex2jar is downloaded and set up properly the '.dex' file is converted by using the command line command that specifies the dex2jar function, and the file location. Once complete the '.dex' file will be converted. The jar file is not yet readable so we rely on another program to view the source code.

To view the source code of the jar file, we need a program called jd-gui. Jd-gui is also found on github and is free for anyone to download. After downloading and

following the instructions for setting up, viewing the source code is simply opening up the jd-gui application and opening the jar file from within.

Viewing the source code of an Android application is very simple as previously outlined, amplifying the need for mobile applications to have security as a main focus when developing. Attackers will target applications with the easier to target vulnerabilities which are very easy to find within the mobile application realm. Not just with getting the source code of an application either; monitoring the communication traffic between an application and outside resources can reveal vulnerabilities within an application as well.

While monitoring the communication between an application and servers is common place in web development, it's often overlooked in mobile application development. This can lead to unchecked vulnerabilities that an attacker can monitor and take advantage of. Unlike obtaining the source code of an application, both IOS and Android apps can have network traffic monitored. The process of configuring a monitor on to an application is simple enough that security should be integrated into the application from the start.

The first step to monitoring a mobile application's traffic is to download a sniffer application that allows you to view the data being transferred from an application to the server, for this example the sniffer application will be Burp Suite, but there are many other applications that can monitor the traffic. Setting up the Burp Suite, a user would create a proxy listener and linking their IP address of the computer where the Burp Suite is being used. Next the user would set up a certificate through the Burp Suite that

will be downloaded, and installed onto a mobile device that will use the application being monitored. Following this, a proxy will need to be set up within the mobile device using the port number and IP address that were used in setting up Burp Suite proxy listener. Once configured, in Burp Suite under the intercept tab, turn on intercept which will intercept any traffic coming from the mobile device and stop the request. The user can look at the details of the request and reject the request, or forward the request. Once the request is forwarded, the user will be able to look at the response from the server in its entirety.

Along with application's source code being vulnerable to being viewed, the network traffic of mobile applications can be sifted through to gain valuable data malicious users are looking for to capitalize on vulnerable applications. Knowing how these vulnerabilities can be discovered is key to being able to test mobile applications to catch these vulnerabilities before they can be exploited. These tools are available to not only attackers, but to developers alike, use these tools to better understand the vulnerabilities because they will be.