Sean Trent
CMSC 414
V00707820

Piggybacking: Injecting Malicious Code into Trusted Android Apps
Mid-Semester Progress Report

Objective

The purpose of this project is to show the feasibility of cloning a known, trusted application for the Android platform and re-releasing the cloned application with malicious new functionality added to it.This project will explore what a maliciously crafted clone can reasonably expect to gain without the user's explicit consent. The cloned application will compromise the confidentiality, integrity, and accessibility of the system by distributing user data to an unauthorized eavesdropper, remotely executing commands, and remotely manipulating/deleting data respectively.

Approach

I am approaching this project with a small handful of tools. Their names, descriptions, and use in my project are listed here:

- Metasploit
    - Popular penetration testing framework
    - Open-source and maintained by Rapid7
    - So far utilized in my approach in two(2) ways:
        - Msfvenom.bat script is used to create a custom Android Package (APK) file that contains a "meterpreter". The payload in this APK is what will be "injected" into the trustworthy application clone.
            - Meterpreter - a payload of DLL injections of code that establishes a TLS/1.0 connection with a predetermined eavesdropper. This connection is used for transferring files as well as remote execution of commands on the victim machine.
        - Msfconsole.bat is used to access the connection established by the meterpreter. The console recognizes the various meterpreter commands and passes them on to the meterpreter process running on the victim machine.

- APKtool
    - Decompiler/compiler for Android Packages (APK)
    - Free tool sponsored by Sourcetoad
    - Used to reverse engineer the trustworthy APK, decompiling it into .smali files. The code in these files is then edited to include the payload meterpreter functionality. APKtool is then used to compile the edited files back into a functioning APK injected with the malicious payload.
        - Smali is the assembly language used by the android virtual machine

        - In my current approach, the APK generated by metasploit is also de-compiled using APKtool
- Jarsigner, Keytool, and Zipalign
    - Development tools provided by the Java/Android SDK
    - Jarsigner - Signs and verifies Java Archive (JAR) files
        - In my case, it signs my maliciously crafted APK files so they can be allowed to install on the target machine
    - Keytool - manages a keystore of keys and certificates
        - Used to create a keystore containing the key used by Jarsigner to sign my APKs
    - Zipalign - optimization tool
        - Used to assure smooth running of my APKs

The simplified breakdown of my approach so far is this:

1. Download source APK of trustworthy app
2. Create APK containing malicious payload
3. Decompile both APKs into their constituent assembly code files
4. Copy the assembly files containing code for the payload process into the list of files from the trustworthy app, giving the trustworthy app access to the payload code
5. Edit the code from the trustworthy app to ask for the permissions necessary for the attack
6. Edit the code from the trustworthy app to launch the payload process alongside its own startup process
7. Recompile the edited code from the trustworthy app into a clone APK that now contains the payload code
8. Download malicious clone APK on target Android device
9. Setup eavesdropper machine to listen for the connection request sent by payload code
10. Run malicious clone app on target Android device
11. Use connection between target device and eavesdropper machine established by payload code to steal data, edit files, etc.

Tasks: First 50% **[COMPLETED]**

- Pick an app to clone
    - Discord App - Communication/Chat VoIP application by Discord Inc.
    - After analysing the decompiled versions of a few popular apps with similar permissions (Snapchat, Facebook, Instagram), Discord seemed to have the most straightforward structure and fewest dependencies (added dependencies make the process of decompiling and recompiling code less successful). Aside from being the app I personally use the most out of the potential apps, Discord would provide the simplest and smoothest hacking experience.

Sean Trent
CMSC 414
V00707820

- Clone the app
    - I downloaded the Discord APK from www.apkmirror.com, a verified source of APKs from the original developers
    - Decompiling and recompiling the Discord APK and giving it my own signature creates a functional clone of the app
    - Even though building a convincing clone from scratch affords greater control and stability in the malicious app, rebuilding a professionally-made app of the same quality of Discord is way outside the scope of this project and irrelevant to the objective.
- Setup host to receive stolen data
    - I designed the payload to initiate a connection with my public IP address via a specific port number
    - I then configured my internet router to port forward from my public IP address and selected port to my actual IP address for my desktop computer
- Research and pick piggybacking method
    - After researching the known methods of injecting malicious code into Android apps, I settled on the method outlined above in the Approach section.

Tasks: Second 50% **[INCOMPLETE]**

- Inject malicious code
    - Even though I have achieved the desired functionality in a custom app containing nothing but a main activity to launch the payload process, I have yet to inject this functionality into the Discord APK successfully.
    - While I believe the method outlined in my Approach section will work, it has yet to produce a stable, runnable clone of the Discord app with the injected payload. Minor tweaking and debugging is still required.
- Deploy malicious clone
    - Naturally, I cannot deploy the cloned app if I do not yet possess it.
    - Certain security measures must be bypassed in order to successfully install anything containing the payload on an actual Android device. In order to bypass this built-in security during installation, the payload process must be ignored or rendered undetectable through a mix of social engineering and code hiding/disguising.
- Create presentation slides and demo video
    - I will create a presentation containing some of the research findings that equipped me to carry out the attack. I will also create a video of the malicious app carrying out its intended function of covertly sending data from my Android device to my desktop PC.