Josh Howard

Wei Zhong

CSCI U599

4/25/19

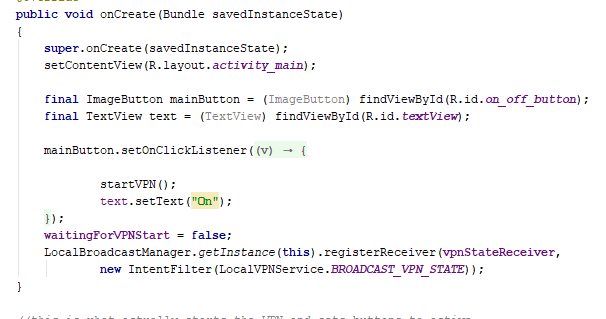
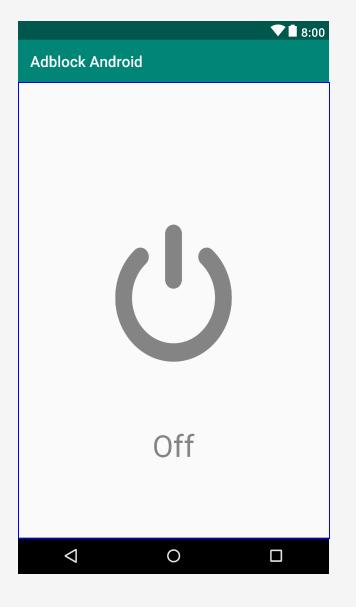
Adblocking On Android

Adblocking on a personal computer is quite common place these days. From browser extensions, secondary programs and custom router settings to some web browsers having the functionality built into them by default. Said adblockers help users limit wasted bandwidth, reduce unnecessary clutter on websites, and make a generally better user experience overall. Adblocking on the Android platform, however, isn’t nearly as common or clear.

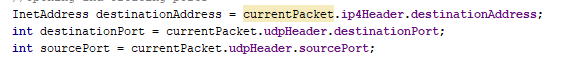
Adblocking on an Android device isn’t nearly as easy as it easy for a personal computer. With a PC, there are many ways one can employ an AdBlock. The main three are deleting, hiding, and blocking the connection (AdBlock Help, n.d.). With the first one, deleting, the blocker looks for certain elements within the HTML code that is loaded on your browser. It then deletes the blocks of code responsible for showing the ad. The second, hiding, works in generally the same way but instead of deleting the HTML elements, it hides them from the user. The third, blocking the connection, is the one we’ll be using and talking about for the Adblock Android application. This one works by looking at the outgoing address in the packets sent out by your device. It then compares the address to a list of known ad filters. If the address is on the filter list, the out going request is blocked. Doing this, the ad contend never loads in the first place. Letting the webpage contend load quicker and saving bandwidth in the process.

The first major problem of an adblocker on Android is reading the network traffic. Without rooting your device, there is no immediate way to view the packets to and from your device. As I wanted the process to be as simple, so rooting was a route I didn’t want to take. The process I ended up going with was using a VPN. This allowed me to look at the packet headers sent and received so that I could check addresses. I was able to find a pre-coded lite packet interceptor called “VpnService” (hexene, 2019) that was built on top of VpnService, an Android class for applications to build their own VPN Solutions. (Android Developers, n.d.)

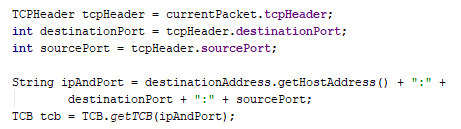
Once I found the “VpnService”, I needed to update it as it used old packages that were no longer completely available in the Android classes. As stated earlier, I wanted the process to be as simple as possible. So the U.I. I created was a simple ImageButton of a power icon, and an on/off text that changed whenever the process was started.

  
 “The layout of the app” (left) “The code creating the button using drawable files” (right)

Once the button is pressed the VPN is created and will start running. The VPN will filter all traffic from the phone through it. It will dissect the packets headers into a readable and usable format, then open a connection to the address through it and send the request out. Once the request is answered, the VPN will route the packets through itself and back to your phone.

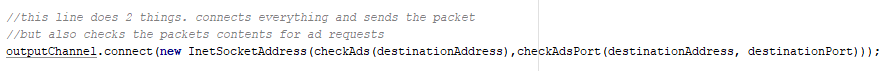


“The UDP portion of the code dissecting the packets”



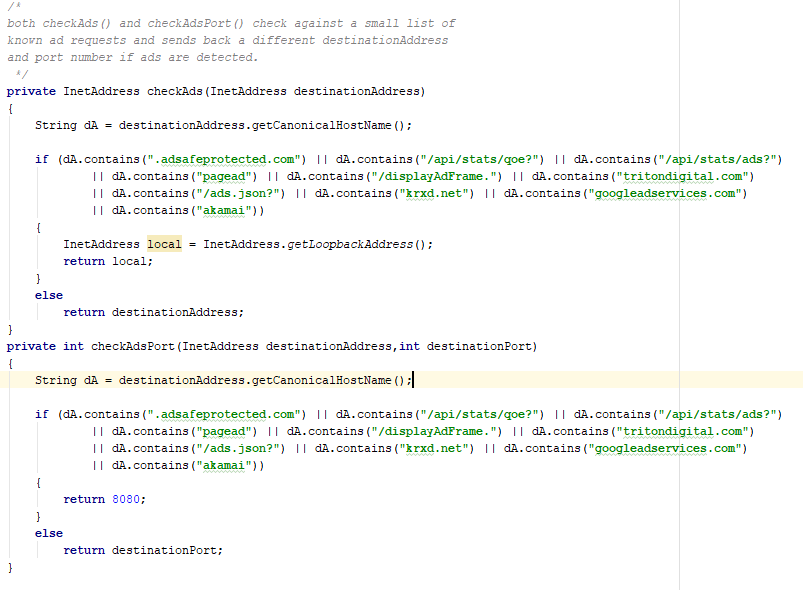
“The TCP portion of the code dissecting the packets”

The VPN then goes onto creating the packets to send out a request for. The next important stage for the Adblocking portion of this is the filtering. Whenever the VPN tries to make a connection to the desired address, a checkAds function is called within the connection function that checks for ads. If any known ad address is present, the checkAds function will return the address to the local host. Doing this implements a loopback address, which, will give nothing back in place of the ad that was supposed to load.



“Calling the function that checks ads when calling to connect”

The check ads function works about the same way as a web browser full adblocker would. It will take the webpage address and check it against a list of known filters for ads. This application though only uses around 10 filters. This is due to the fact that when I tried to load the external filter list <https://easylist.to/>, the device would crash due to the shear size of the list and the way it was implemented. If the destinationAddress contained any of the rejected filter characters, the function will get and return a LoopbackAddress. If not will return the original passed in destinationAddress. Due to the fact that the .connect() function requires an address and a port number, a separate checkAdsPort function was made. This function has and does the same filtering that the checkAds function does, but returns either the current devices port number, or the passed in destinationPort.



“The functions that check the address and returns address and ports”

Even though this adblocker doesn’t have the complete capabilities as one on a personal computer, I still think this was successful. This shows that adblocking is technically possible on an android device that anyone can use. This method does not require advanced techniques as rooting your device or creating your own VPN, setting up ports, and running a separate adblocking application.

# References

*AdBlock Help*. (n.d.). Retrieved from help.getadblock.com: https://help.getadblock.com/support/solutions/articles/6000087914-how-does-adblock-work-

Android Developers. (n.d.). *VpnService*. Retrieved from developer.android.com: https://developer.android.com/reference/android/net/VpnService

hexene. (2019, Janurary 23). *hexene/LocalVPN*. Retrieved from github.com: https://github.com/hexene/LocalVPN

Kolowich, L. (2015, October 1). *How Ad Blocking Ads*. Retrieved from Hubspot Blog: https://blog.hubspot.com/marketing/how-ad-blocking-works

Pelican. (n.d.). *EasyList*. Retrieved from EasyList: https://easylist.to/

*Writing Adblock Plus Filters*. (n.d.). Retrieved from Adblock Plus: https://adblockplus.org/en/filters