Group 2 use a very particular approach to prevent the original source code being obtained unauthorized. Instead of using code obfuscation, they use an existing tool called DexUnShell to re-inforce the original apk file.

They first write the functional code in the normal way, and then compile the source code to an apk file. Then they write the proxy application (works as a shell and the decryption tool) from the template provided together by that tool. The proxy application is then compiled and get the dex file out.

To reassemble the proxy and actual application, DexUnShell is then applied.

The tool uses a byte operation and a key (“ivy”) to encrypt the apk byte and put the actual apk’s encrypted byte behind the proxy dex byte. To make the assembled dex still correct, the tool will recalculate the dex file header, checksum and SHA1.

The ouput dex is then put into the apk and replace the dex with in it.

At the runtime, the proxyapplication class and reinvoke class will recover the actual bytecode from the dex file and regenerate the actual application dynamically. Those who attempt to reverse engineer the apk will only get the proxyapplication which does not make any sense to what the application does.

To get the actual application source code, there is two ways to do that. First one is to dump the android memory when the proxy is regenerating the actual apk. The second way will be compile the reverse engineered proxy application, get the dex file, and compare it with the dex we get from the apk. The byte difference will be the encrypted bye of the original apk. And since the proxyapplication class have the decryption method, we can write a tool to do the decryption, and then do a de-compile on the original apk, which have no obfuscation.

Overall, this is a very effective protection to the source code, but not an obfuscation. To make the protection work better, they can add some code to the proxyapplication so that people cannot tell that it is not a proxy.

private byte[] sdkjfh(byte[] var1, String var2) {

char[] var7 = var2.toCharArray();

int var5 = var7.length;

for(int var3 = 0; var3 < var5; ++var3) {

char var6 = var7[var3];

for(int var4 = 0; var4 < var1.length; ++var4) {

var1[var4] = (byte)(var1[var4] ^ var6);

}

}

return var1;

}