Intro To Android Game Hacking

Android OS, APK, ELF, ARM, and Hypervisors

What Is Android?

Android is a branch of the linux operating system which is developed and maintained mainly by google.

https://en.wikipedia.org/wiki/List_of_Linux_distributions

What about Architecture? ARM? x86?

Most mobile phones use ARM processors and thus the kernel and native binaries must be compiled to so variant of ARM instructions. However there are a very few amount of phones which use x86 processors.

Based upon unity game engine stats, only 1.7% of phones have x86 processors.

https://web.archive.org/web/20170808222202/http://hwstats.unity3d.com:80/mobile/cpu-android.html

Why does Architecture matter?

Running Android OS in virtualized environments and on x86 phones requires a port of the Android kernel. Android x86!

Emulation of ARM instructions is much more computationally expensive than native execution. In addition, emulation of an entire kernel, and additional kernel drivers compiled to ARM native would be excruciatingly slow. Seven to ten times as slow as the native instruction!

Android x86 Continued...

- Android x86 has an x86_64 based kernel and kernel modules.
- It emulates native ARM binaries if it must via a closed source library by the code name “libhoudini”.
- Android x86 can be installed in vmware/virtualbox/QEMU-KVM.

Android x86 Continued… “libhoudini”

libhoudini is a proprietary ARM translation layer for x86-powered Android devices. It allows an app that has NDK binaries for ARM, but not x86, to still run on x86 hardware, albeit not as quickly as it would with native x86 binaries.

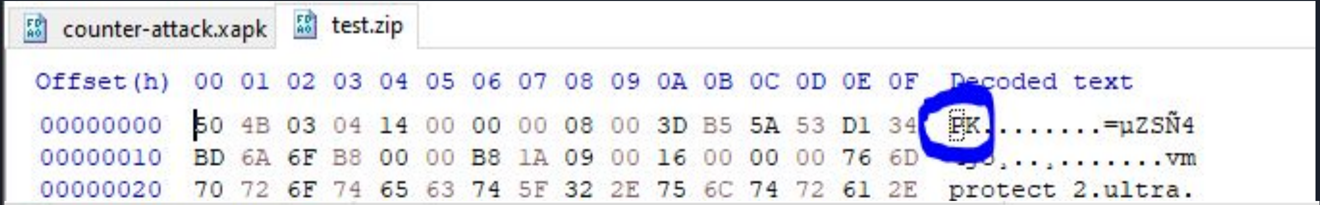
libhoudini == ARM interpreter

Side note: Although libhoudini is closed source there has been some very interesting research done upon it. If you would like to learn more about libhoudini you can watch this very nice black hat talk here:

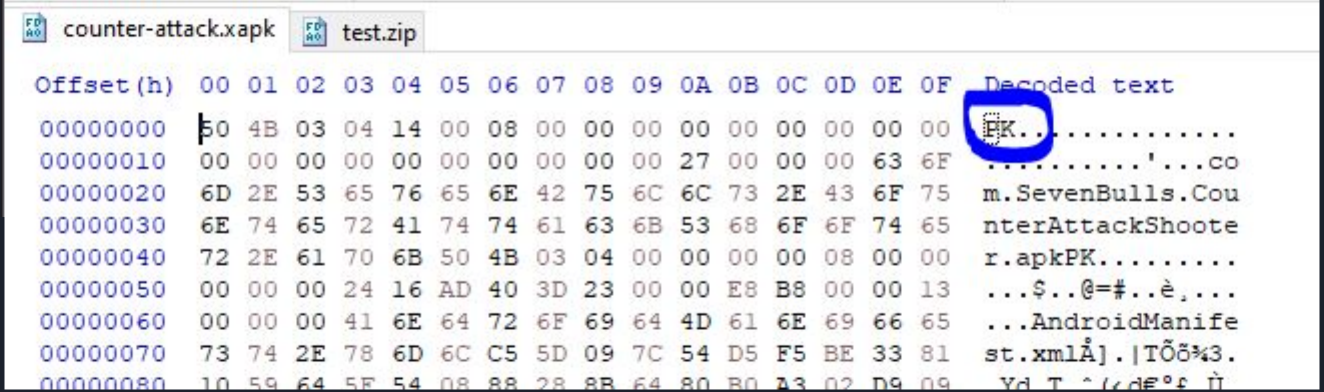
https://www.youtube.com/watch?v=9oQ5XjA1aq0

Enough about Android OS, more about APK file format!

The XAPK and APK file format are simply using the ZIP file format to compress contents. Comparing the first few bytes (typically referred to as magic bytes like MZ for PE files) APK and ZIP files have “PK”.

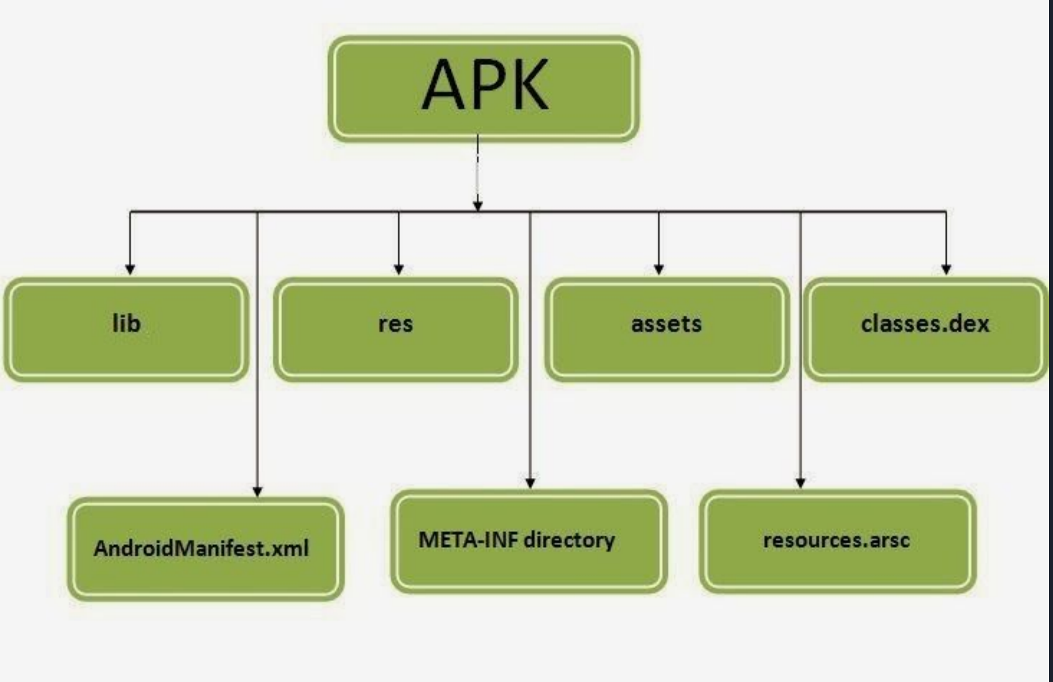


```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000   50 4B 03 04 14 00 00 00 08 00 3D B5 5A 53 D1 34  PK........=µZSÑ4
00000010   BD 6A 6F B8 00 00 B8 1A 09 00 16 00 00 00 76 6D  ½jo¸..¸.........vm
00000020   70 72 6F 74 65 63 74 5F 32 2E 75 6C 74 72 61 2E  protect_2.ultra.
```



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000   50 4B 03 04 14 00 08 00 00 00 00 00 00 00 00 00  PK..............
00000010   00 00 00 00 00 00 00 00 00 00 27 00 00 00 63 6F  ..........'...co
00000020   6D 2E 53 65 76 65 6E 42 75 6C 6C 73 2E 43 6F 75  m.SevenBulls.Cou
00000030   6E 74 65 72 41 74 74 61 63 6B 53 68 6F 6F 74 65  nterAttackShoote
00000040   72 2E 61 70 6B 50 4B 03 04 00 00 00 00 08 00 00  r.apkPK.........
00000050   00 00 00 24 16 AD 40 3D 23 00 00 E8 B8 00 00 13  ...$..@=#..è¸...
00000060   00 00 00 41 6E 64 72 6F 69 64 4D 61 6E 69 66 65  ...AndroidManife
00000070   73 74 2E 78 6D 6C C5 5D 09 7C 54 D5 F5 BE 33 81  st.xmlÅ].|TÕõ¾3.
00000080   10 59 64 5F 54 08 88 28 8B 64 80 B0 A3 02 D9 09  .Yd_T.^(‹d€°£.Ù.
```

APK and XAPK File Format Continued...



Good reference for the file format:

http://pavan2pyati.blogspot.com/2013/12/android-apk-file-format.html

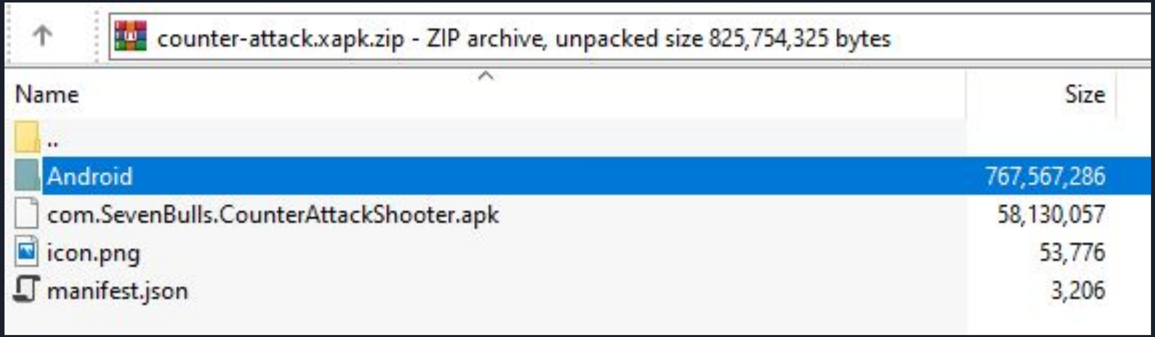
Main things we care about in the APK:

- Shared objects/native binaries (libs folder)
- .dex files (all of the java code is compiled into these!)
- Strings.xml if you are into that ;)

XAPK vs APK

XAPK is just an expansion of APK. An XAPK contains a regular APK as well as a folder containing “obb” files. These are just RAW data files which can be interpreted however the programmer of the APK wants. Usually big assets go into these, like textures and videos.

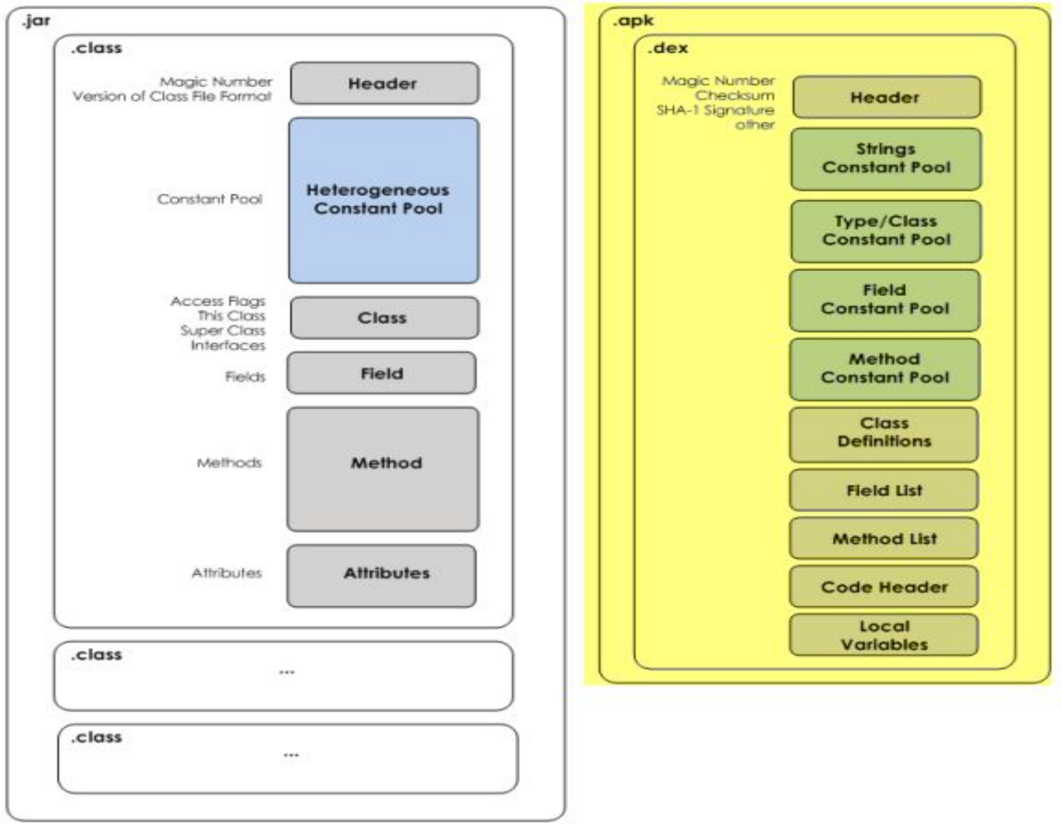
There is also a manifest.json file which contains relative paths to these obb files.



counter-attack.xapk.zip - ZIP archive, unpacked size 825,754,325 bytes

Name	Size
..	
Android	767,567,286
com.SevenBulls.CounterAttackShooter.apk	58,130,057
icon.png	53,776
manifest.json	3,206

APK File Format Continued…. DEX Files



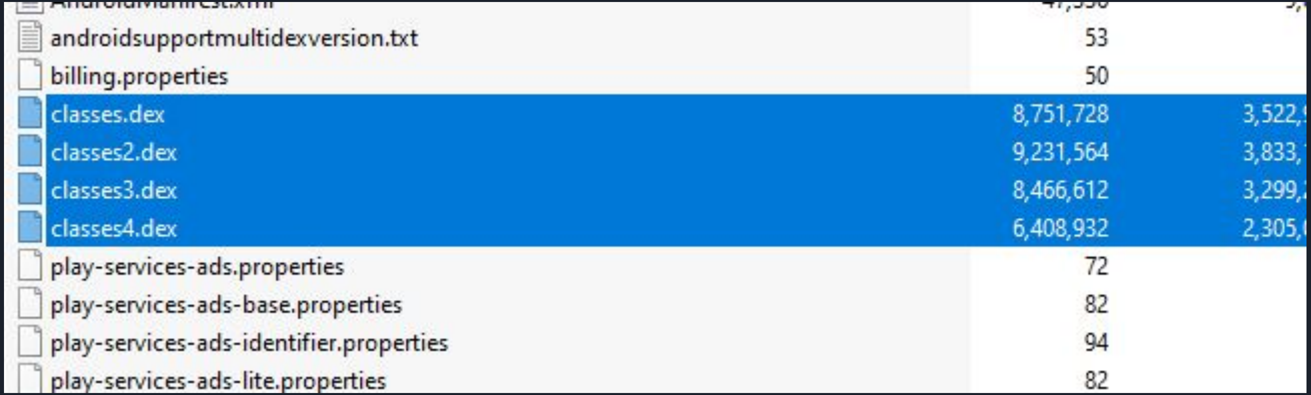
DEX files can be related to java class files which are generated by javac! Android uses a specific java virtual machine called the DALVIK virtual machine or DVM for short. The DVM is NOT THE JVM!

DVM is designed specifically for android applications and tries to use less memory, as well as load faster.

DEX Files Continued...

There can be multiple DEX files. This is called multidex'ing and is required if your application has more than 64k methods. https://developer.android.com/studio/build/multidex#about

The 64k method limit is due to the file format/header field which only uses 2 bytes (unsigned short) to detail how many methods are in a given DEX file. This means there is a limit of 2^16 methods per-DEX file.

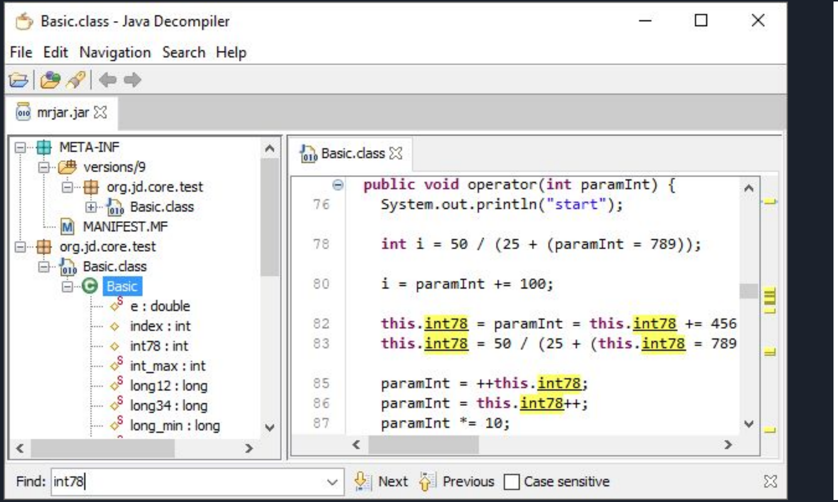


androidsupportmultidexversion.txt	53	
billing.properties	50	
classes.dex	8,751,728	3,522,
classes2.dex	9,231,564	3,833,
classes3.dex	8,466,612	3,299,
classes4.dex	6,408,932	2,305,
play-services-ads.properties	72	
play-services-ads-base.properties	82	
play-services-ads-identifier.properties	94	
play-services-ads-lite.properties	82	

DEX Files Continued… Decomposition

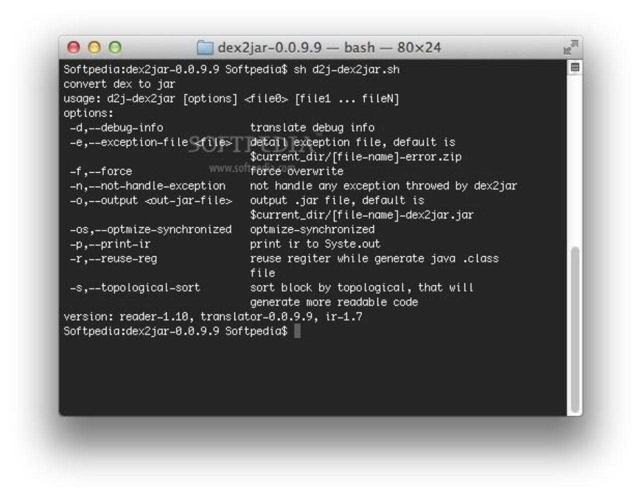
- There is a tool called “dex2jar” which is packaged by default into Kali linux. https://www.kali.org/tools/dex2jar/
- It can be used to decompose a dex file and create a JAR file containing .class files.
- There are already tools to examine .class files! (JD-GUI)
- JD-GUI is a drag-and-drop JAR/class inspector

The process of decomposition of DEX files to java source code takes this path: unzip apk → extract dex(s) → dex2jar → jd-gui



```
public void operator(int paramInt) {
  System.out.println("start");

  int i = 50 / (25 + (paramInt = 789));

  i = paramInt += 100;

  this.int78 = paramInt = this.int78 += 456
  this.int78 = 50 / (25 + (this.int78 = 789

  paramInt = ++this.int78;
  paramInt = this.int78++;
  paramInt *= 10;
```

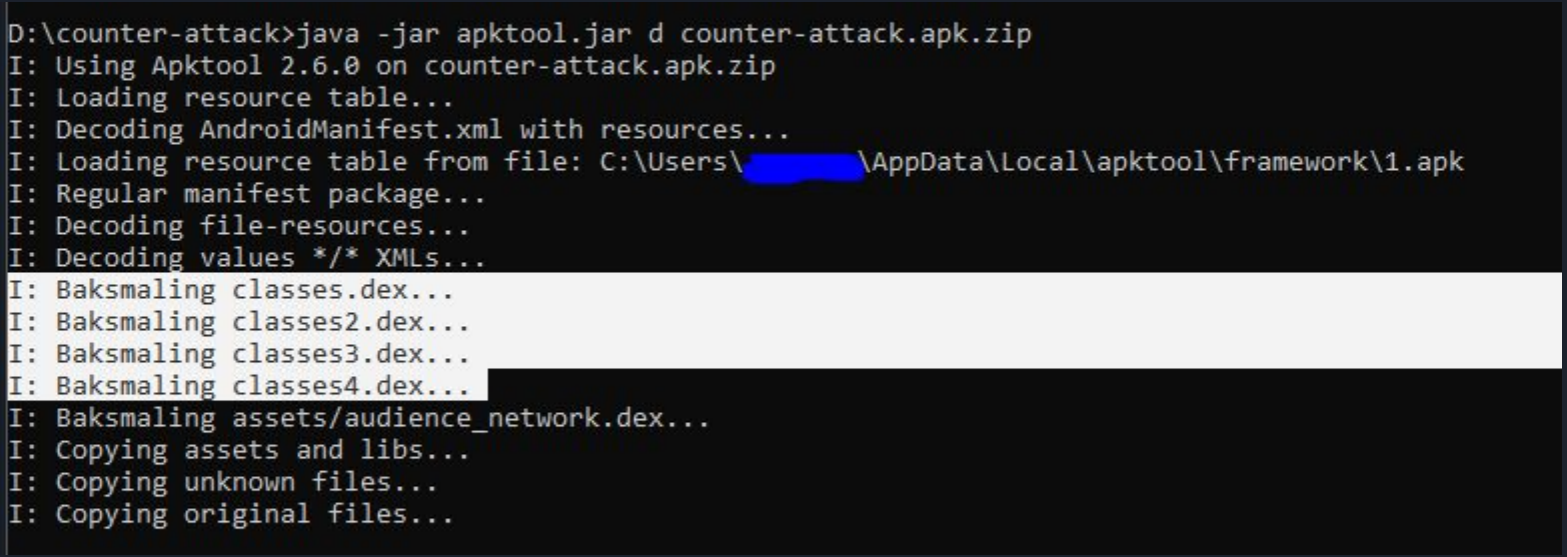
Find: int78



```
Softpedia:dex2jar-0.0.9.9 Softpedia$ sh d2j-dex2jar.sh
convert dex to jar
usage: d2j-dex2jar [options] <file0> [file1 ... fileN]
options:
 -d,--debug-info              translate debug info
 -e,--exception-file <file>   detail exception file, default is
                              $current_dir/[file-name]-error.zip
 -f,--force                   force overwrite
 -n,--not-handle-exception    not handle any exception throwed by dex2jar
 -o,--output <out-jar-file>   output .jar file, default is
                              $current_dir/[file-name]-dex2jar.jar
 -os,--optmize-synchronized   optmize-synchronized
 -p,--print-ir                print ir to Syste.out
 -r,--reuse-reg               reuse regiter while generate java .class
                              file
 -s,--topological-sort        sort block by topological, that will
                              generate more readable code
version: reader-1.10, translator-0.0.9.9, ir-1.7
Softpedia:dex2jar-0.0.9.9 Softpedia$
```

What about byte patching DVM byte code?

- Smali and baksmali! An assembler/disassembler for DEX/DVM bytecode!
- Open source! https://github.com/JesusFreke/smali
- Even better, there is a suite of these tools, including dex2jar, called apktool! Its also open source https://ibotpeaches.github.io/Apktool/



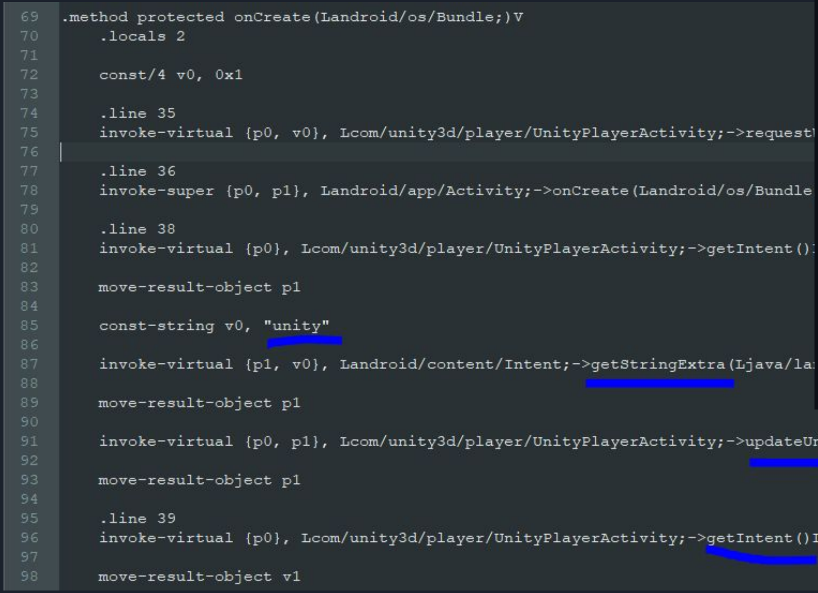
```
D:\counter-attack>java -jar apktool.jar d counter-attack.apk.zip
I: Using Apktool 2.6.0 on counter-attack.apk.zip
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\[redacted]\AppData\Local\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Baksmaling classes2.dex...
I: Baksmaling classes3.dex...
I: Baksmaling classes4.dex...
I: Baksmaling assets/audience_network.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Byte Patching DVM Byte Code Continued… APK Entry Point.

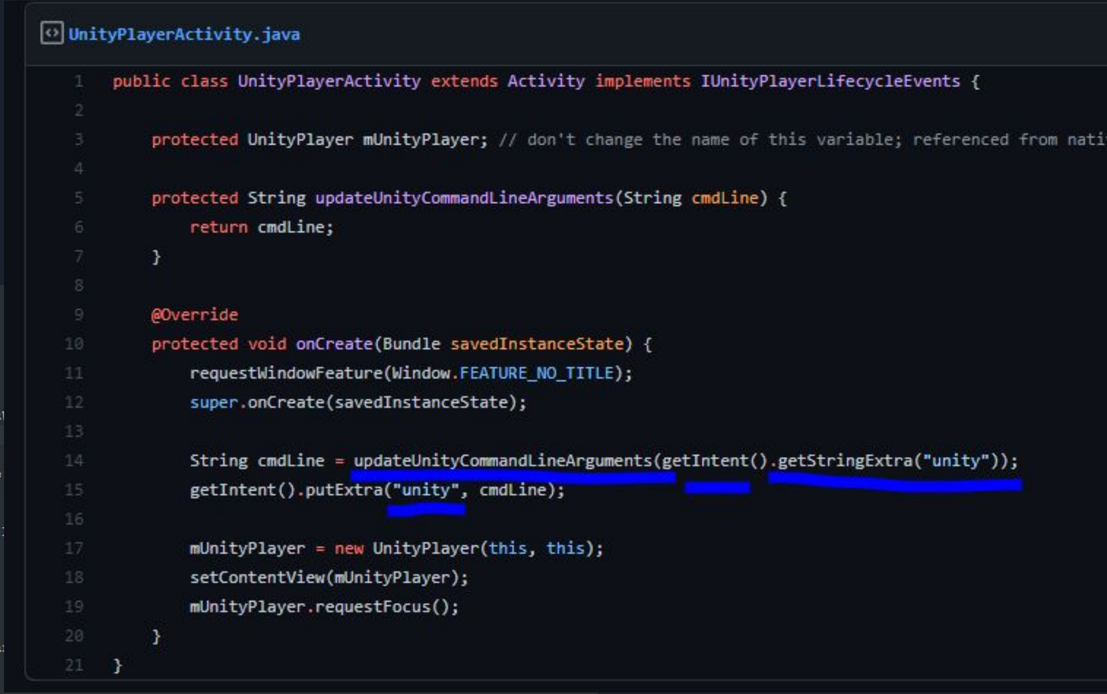
(Note: the game/apk I am referring to is called “counter attack”. Its a rip off mobile game of counter strike…)

Every APK has a designated entry point. It can different entry points depending upon certain actions such as accepting an invitation from facebook, etc… Counter attack entry point is:

“com.unity3d.player.UnityPlayerActivity”



```
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 2

    const/4 v0, 0x1

    .line 35
    invoke-virtual {p0, v0}, Lcom/unity3d/player/UnityPlayerActivity;->request

    .line 36
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle

    .line 38
    invoke-virtual {p0}, Lcom/unity3d/player/UnityPlayerActivity;->getIntent()

    move-result-object p1

    const-string v0, "unity"

    invoke-virtual {p1, v0}, Landroid/content/Intent;->getStringExtra(Ljava/la

    move-result-object p1

    invoke-virtual {p0, p1}, Lcom/unity3d/player/UnityPlayerActivity;->updateUnityCommandLineArguments(Ljava/lang/String;)Ljava/lang/String;

    move-result-object p1

    .line 39
    invoke-virtual {p0}, Lcom/unity3d/player/UnityPlayerActivity;->getIntent()Landroid/content/Intent;

    move-result-object v1
```



UnityPlayerActivity.java

```java
public class UnityPlayerActivity extends Activity implements IUnityPlayerLifecycleEvents {

    protected UnityPlayer mUnityPlayer; // don't change the name of this variable; referenced from nati

    protected String updateUnityCommandLineArguments(String cmdLine) {
        return cmdLine;
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        super.onCreate(savedInstanceState);

        String cmdLine = updateUnityCommandLineArguments(getIntent().getStringExtra("unity"));
        getIntent().putExtra("unity", cmdLine);

        mUnityPlayer = new UnityPlayer(this, this);
        setContentView(mUnityPlayer);
        mUnityPlayer.requestFocus();
    }
}
```

Patching Continued... Entry Point Continued... Loading Shared Objects...

From the slide before we saw that “com.unity3d.player.UnityPlayerActivity” (onCreate) is the entry point of the APK. If we look close at the code we see the following line:

“mUnityPlayer = new UnityPlayer(this, this);”

This calls the constructor of the UnityPlayer class… This constructor does many things, but most importantly it makes a call to “loadNative”. This is a wrapper function for “System.loadLibrary”, which is a Java wrapper that calls the native function “dlopen”.

The call stack would look like this:

- com.unity3d.player.UnityPlayerActivity.onCreate (java)
- com.unity3d.player.UnityPlayer.ctor (java)
- com.unity3d.player.UnityPlayer.loadNative (java)
- System.loadLibrary (java)
- … internal java shit here like doLoad/ClassLoader … (kernel32.LoadLibraryA vs libc.dlopen)
- dlopen/LoadLibraryA (libc/native binary or ntdll.dll)

Lets Byte Patch! Load before libmain.o!

A power move is to load before any other shared objects. If there was an anti cheat, we would want to be the first to load so we could place hooks on everything. So lets patch in a call to “System.loadLibrary” right before the call to load libmain.o!

We can do it in this method: com.unity3d.player.UnityPlayerActivity.onCreate

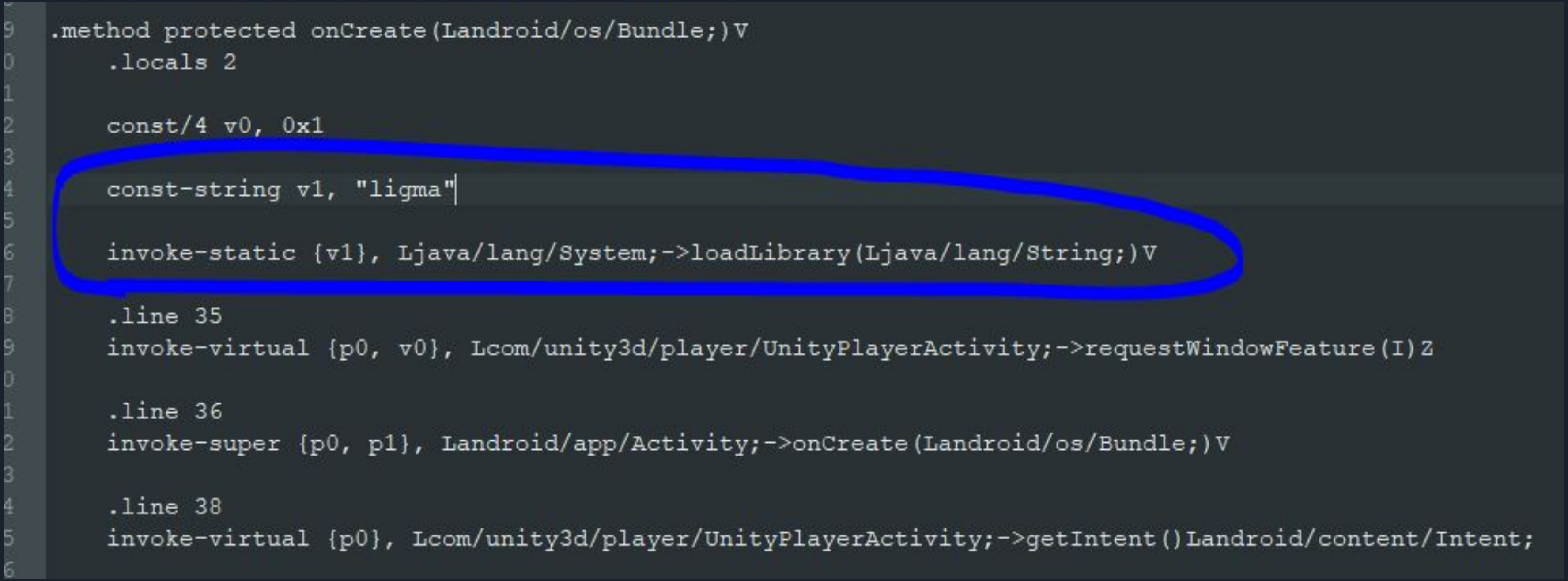
The following lines of smali will load a shared object by the name “libligma.so”

```
    const-string v1, "ligma"

    invoke-static {v1}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V
```

Note: the “lib” is prepended to the shared object name, and the .so is appended to the string.

I.E: ligma → libligma.so, main → libmain.so

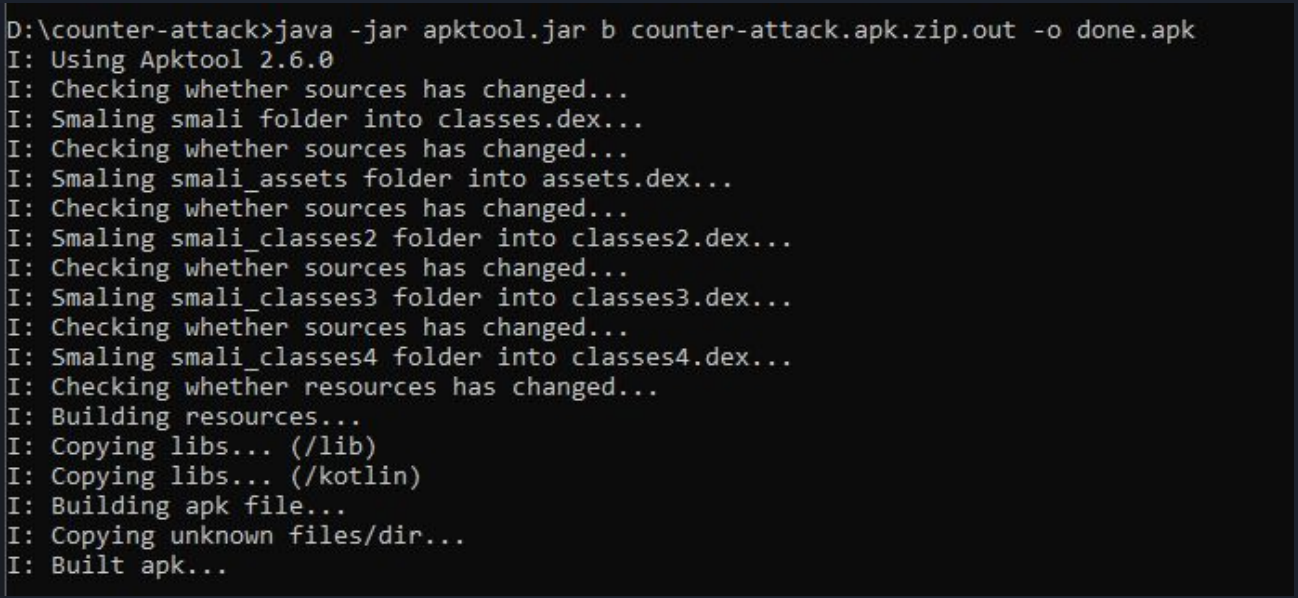
System.loadLibrary(“ligma”) injected!



```
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 2

    const/4 v0, 0x1

    const-string v1, "ligma"

    invoke-static {v1}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V

    .line 35
    invoke-virtual {p0, v0}, Lcom/unity3d/player/UnityPlayerActivity;->requestWindowFeature(I)Z

    .line 36
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

    .line 38
    invoke-virtual {p0}, Lcom/unity3d/player/UnityPlayerActivity;->getIntent()Landroid/content/Intent;
```

Smali assembling and APK reconstruction with applied patches...

Once you have made the desired patches to the APK you can simply use apktool to reassemble the smali files back into DEX files and then from a folder back into an APK.

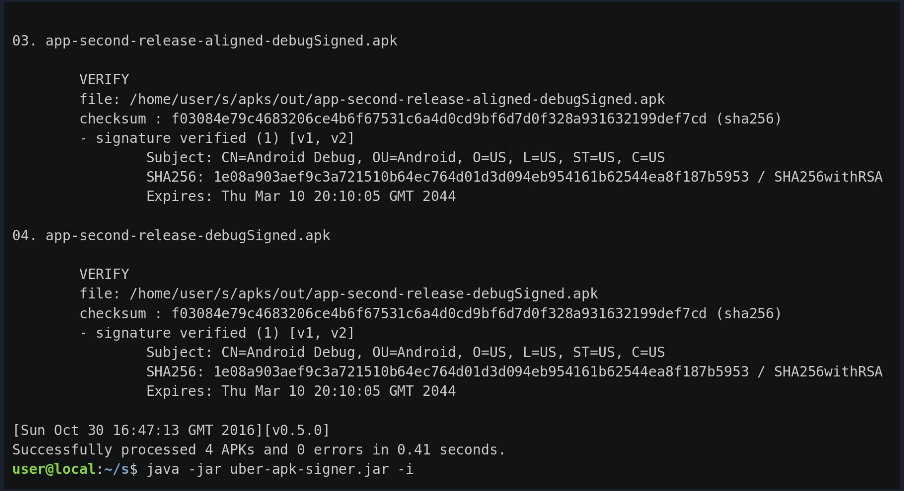


```
D:\counter-attack>java -jar apktool.jar b counter-attack.apk.zip.out -o done.apk
I: Using Apktool 2.6.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether sources has changed...
I: Smaling smali_assets folder into assets.dex...
I: Checking whether sources has changed...
I: Smaling smali_classes2 folder into classes2.dex...
I: Checking whether sources has changed...
I: Smaling smali_classes3 folder into classes3.dex...
I: Checking whether sources has changed...
I: Smaling smali_classes4 folder into classes4.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Copying libs... (/kotlin)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```

APK reconstruction continued... Signing the APK

APK Signer from github will re-sign the patched apk. A link to it can be found here:

- https://github.com/patrickfav/uber-apk-signer
- https://asciinema.org/a/91092 (video showing what apk signer looks like)



```
03. app-second-release-aligned-debugSigned.apk

        VERIFY
        file: /home/user/s/apks/out/app-second-release-aligned-debugSigned.apk
        checksum : f03084e79c4683206ce4b6f67531c6a4d0cd9bf6d7d0f328a931632199def7cd (sha256)
        - signature verified (1) [v1, v2]
                Subject: CN=Android Debug, OU=Android, O=US, L=US, ST=US, C=US
                SHA256: 1e08a903aef9c3a721510b64ec764d01d3d094eb954161b62544ea8f187b5953 / SHA256withRSA
                Expires: Thu Mar 10 20:10:05 GMT 2044

04. app-second-release-debugSigned.apk

        VERIFY
        file: /home/user/s/apks/out/app-second-release-debugSigned.apk
        checksum : f03084e79c4683206ce4b6f67531c6a4d0cd9bf6d7d0f328a931632199def7cd (sha256)
        - signature verified (1) [v1, v2]
                Subject: CN=Android Debug, OU=Android, O=US, L=US, ST=US, C=US
                SHA256: 1e08a903aef9c3a721510b64ec764d01d3d094eb954161b62544ea8f187b5953 / SHA256withRSA
                Expires: Thu Mar 10 20:10:05 GMT 2044

[Sun Oct 30 16:47:13 GMT 2016][v0.5.0]
Successfully processed 4 APKs and 0 errors in 0.41 seconds.
user@local:~/s$ java -jar uber-apk-signer.jar -i
```

DVM Byte Patching Steps (Summary)

- Unpack apk via apktool
- Navigate to desired smali file
- Patch smali code
- Pack apk via apktool
- Sign apk via apksigner

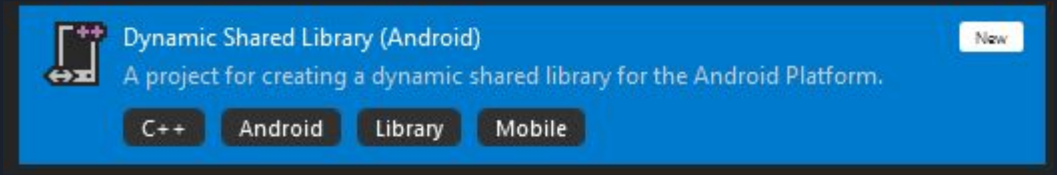
- Optional: include additional shared objects in the “lib” folder.

The C++ Android development kit (NDK)

Let's switch gears to development of shared objects for Android. I will only be explaining how to do so for Windows users.

- Download Visual studios 2019
- Open “Visual Studios Installer”
- Click “Modify” on Visual Studios 2019
- Click the “Mobile Development with C++” and install it.

You should be able to create a new shared object project for Android!



Dynamic Shared Library (Android)

A project for creating a dynamic shared library for the Android Platform.

Introduction to NDK (JNI)

Java and native binaries have a library they use to call between the two. JNI stands for “java native interface”.

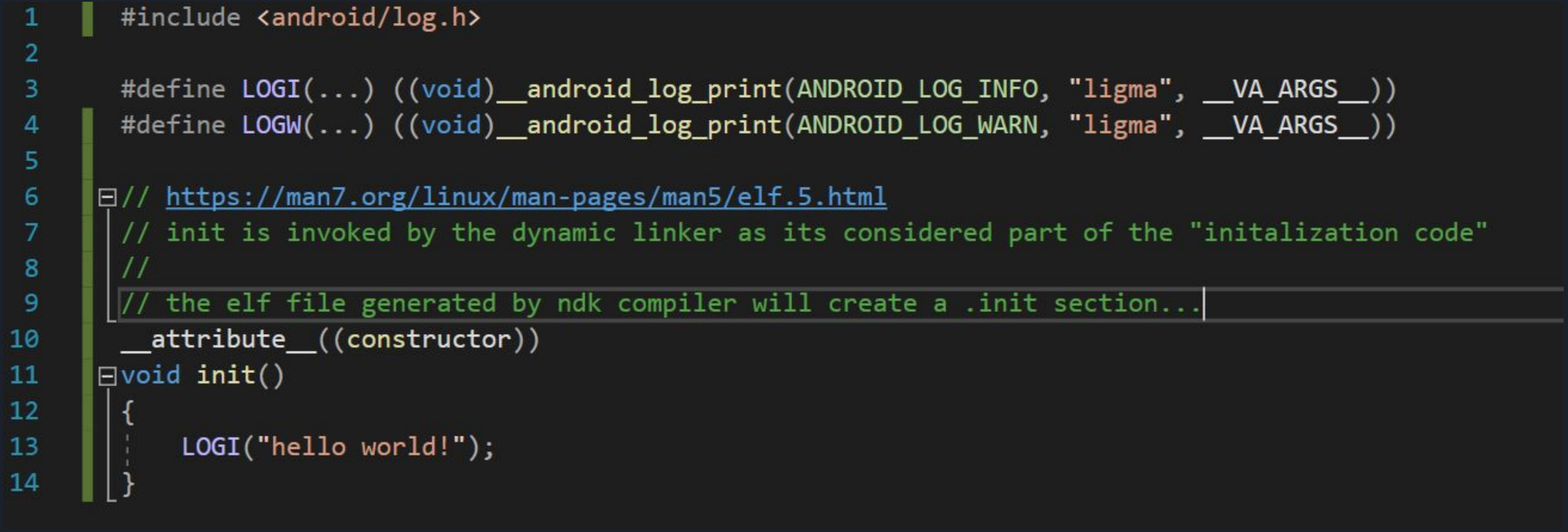
Main things to look for when reverse engineering:

- JNIEnv::FindClass
- JNIEnv::RegisterNatives
  - jint RegisterNatives(jclass clazz, const JNINativeMethod* methods, jint nMethods)
  - JNINativeMethod is a structure that contains 2 strings (char pointers) and 1 function pointer...

Great resource here: https://www.fer.unizg.hr/_download/repository/jni.pdf

Let's create libligma.so!

Since our code is dynamically loaded we do not have an entry point. Rather we must declare a function as a “constructor” so the dynamic linker will invoke our code during initialization. If this is too complicated just think of “init” as “main”.



```
#include <android/log.h>

#define LOGI(...) ((void)__android_log_print(ANDROID_LOG_INFO, "ligma", __VA_ARGS__))
#define LOGW(...) ((void)__android_log_print(ANDROID_LOG_WARN, "ligma", __VA_ARGS__))

// https://man7.org/linux/man-pages/man5/elf.5.html
// init is invoked by the dynamic linker as its considered part of the "initalization code"
//
// the elf file generated by ndk compiler will create a .init section...
__attribute__((constructor))
void init()
{
    LOGI("hello world!");
}
```

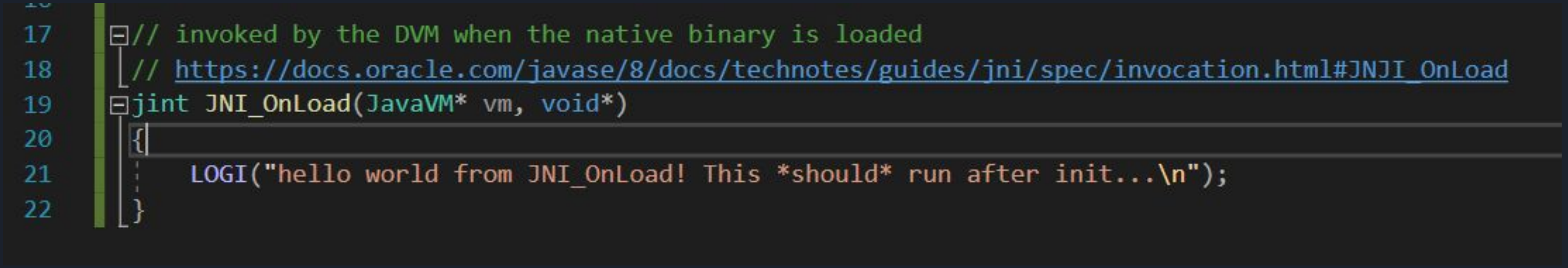
JNI_OnLoad, called from the DVM!

You can also export a function by the name of JNI_OnLoad and the DVM will invoke it when it loads the shared object.... This is the proper way of creating an entry point.

“The VM calls JNI_OnLoad when the native library is loaded (for example, through System.loadLibrary). JNI_OnLoad must return the JNI version needed by the native library.

In order to use any of the new JNI functions, a native library must export a JNI_OnLoad function that returns JNI_VERSION_1_2. If the native library does not export a JNI_OnLoad function, the VM assumes that the library only requires JNI version JNI_VERSION_1_1. If the VM does not recognize the version number returned by JNI_OnLoad, the VM will unload the library and act as if the library was never loaded.”

Resource from oracle:
https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/invocation.html#JNJI_OnLoad

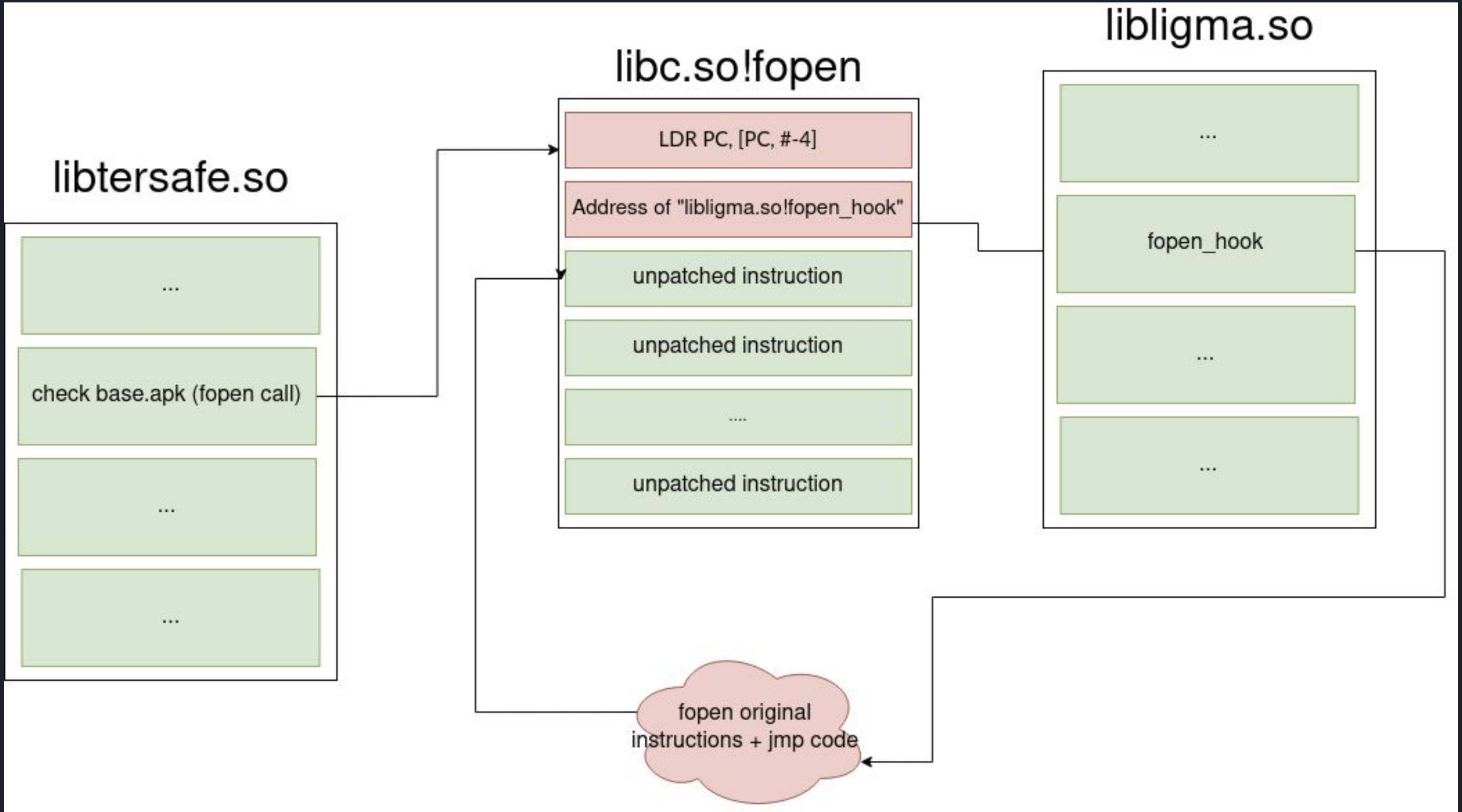


```
// invoked by the DVM when the native binary is loaded
// https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/invocation.html#JNJI_OnLoad
jint JNI_OnLoad(JavaVM* vm, void*)
{
    LOGI("hello world from JNI_OnLoad! This *should* run after init...\n");
}
```

Basics of ARM: Inline Hooks (Shithooks)

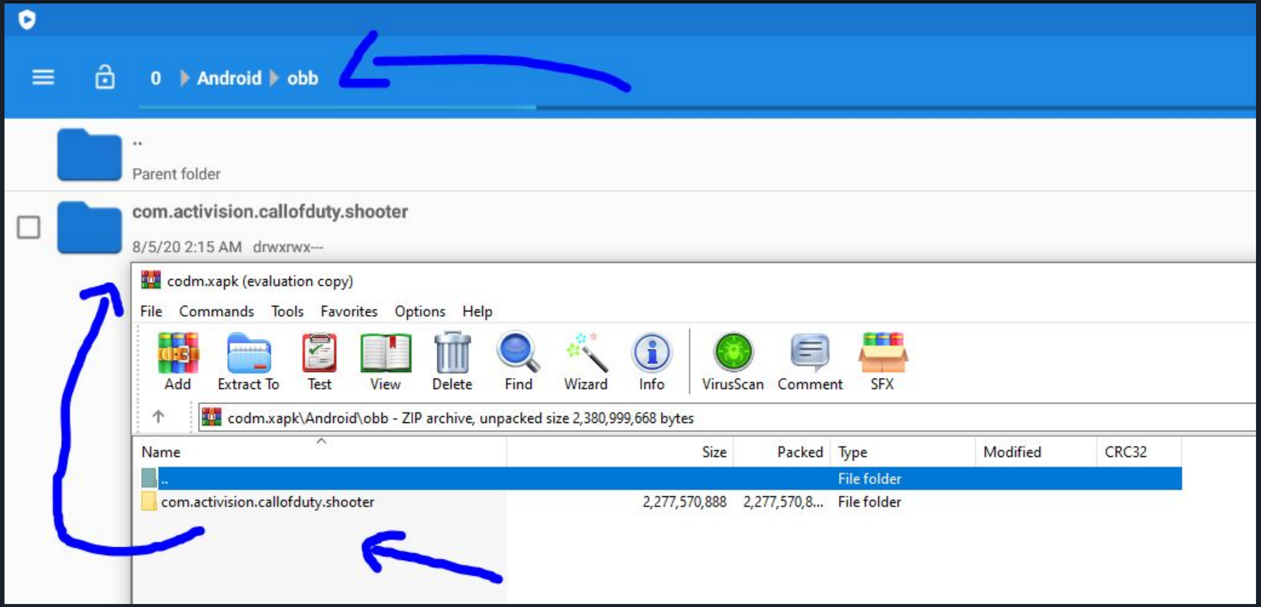
- Fixed sized instruction set.
    - Thumb instructions are 2 bytes
    - ARM instructions are 4 bytes
- Register:
    - 13 general-purpose registers R0-R12
    - One Stack Pointer (SP)
    - One Link Register (LR)
    - One Program Counter (PC)
    - One Application Program Status Register (APSR)
- Fixed length instructions allow for a programmer to not require an instruction length disassembler to create a hooking library
    - Function A calls Function B
        - First instruction of Function B is changed to load the process counter with an address
            - LDR PC, [PC, #-4] // Machine Code: 0xE51FF004
            - 4 byte address is patch directly below the LDR PC.
            - Original instructions can be copied into a new malloc page with mprotect to make it executable
            - Final LDR PC can be used to jmp back to the original function after the frist 2 instructions...

libc.so!fopen hook example



ADB, Logcat and installing XAPK’s

Once you have patched the apk with the new shared object and smali code to load it you can then push the APK to the Android-x86 instance and install it via ADB. Keep in mind that we are not actually installing from an XAPK! We will need to copy the OBB files ourselves.



com.activision.callofduty.shooter

8/5/20 2:15 AM drwxrwx---

codm.xapk\Android\obb - ZIP archive, unpacked size 2,380,999,668 bytes

Name	Size	Packed	Type	Modified	CRC32
..			File folder		
com.activision.callofduty.shooter	2,277,570,888	2,277,570,8...	File folder		

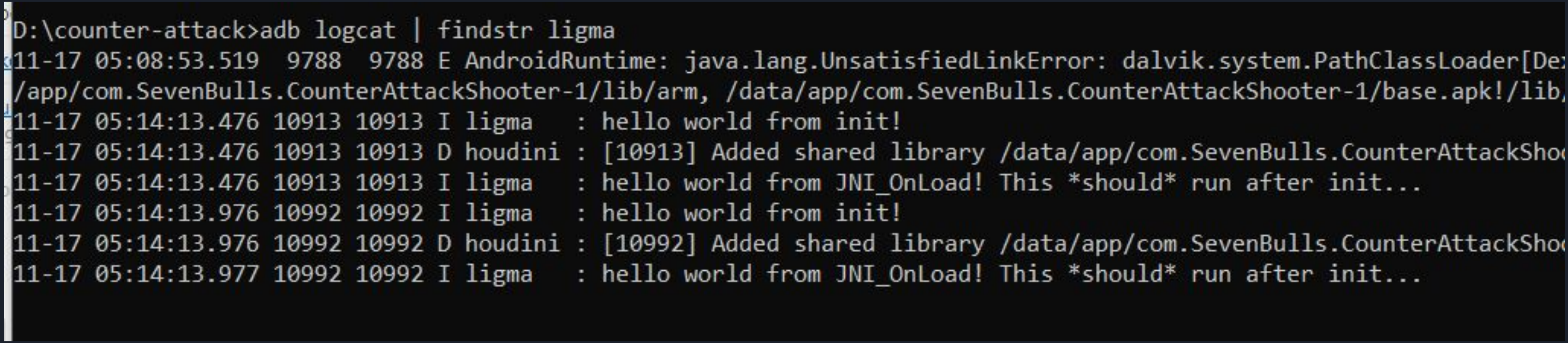
First run, viewing Logcat

Executing the following ADB command will filter logs for our LOGI macro:

Windows: adb logcat | findstr “ligma”

Linux: adb logcat | grep “ligma”

We also see that houdini has loaded our shared object!



```
D:\counter-attack>adb logcat | findstr ligma
11-17 05:08:53.519  9788  9788 E AndroidRuntime: java.lang.UnsatisfiedLinkError: dalvik.system.PathClassLoader[De
/app/com.SevenBulls.CounterAttackShooter-1/lib/arm, /data/app/com.SevenBulls.CounterAttackShooter-1/base.apk!/lib
11-17 05:14:13.476 10913 10913 I ligma   : hello world from init!
11-17 05:14:13.476 10913 10913 D houdini : [10913] Added shared library /data/app/com.SevenBulls.CounterAttackSho
11-17 05:14:13.476 10913 10913 I ligma   : hello world from JNI_OnLoad! This *should* run after init...
11-17 05:14:13.976 10992 10992 I ligma   : hello world from init!
11-17 05:14:13.976 10992 10992 D houdini : [10992] Added shared library /data/app/com.SevenBulls.CounterAttackSho
11-17 05:14:13.977 10992 10992 I ligma   : hello world from JNI_OnLoad! This *should* run after init...
```

Debugging Your Code? Hypervisors?

There are many applications that bundle a hypervisor with a pre-configured version of android-x86. Some of these applications are listed below:

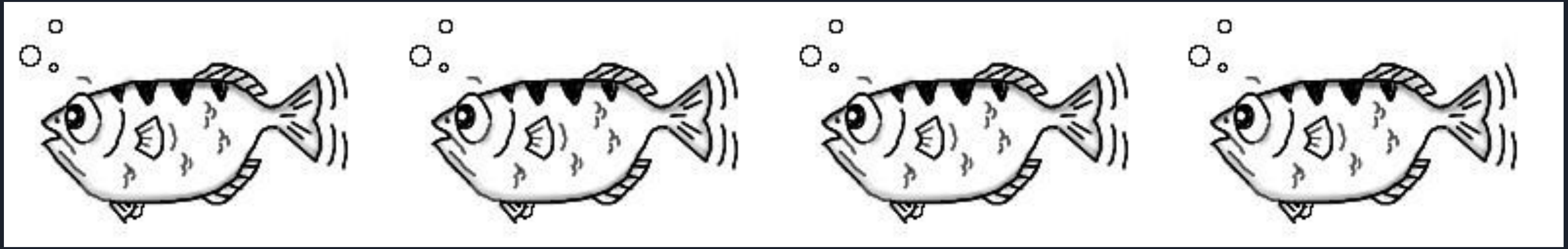
- Bluestacks
- Memu
- LDPlayer
- Genymotion

Alternatively you can create your own custom configuration of android-x86. I have done such a thing myself with QEMU/KVM (linux only).

However for this slideshow I will be using Memu for windows 10.

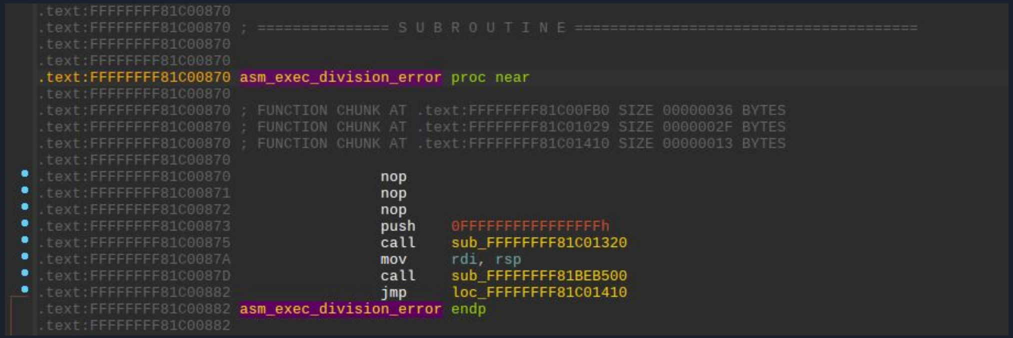
Hypervisor/Virtual Machine runs an x64 Kernel!

- Most Virtual Machines/Hypervisors have internal GDB stubs which you can use to debug code running inside of the virtual machine without the guest “knowing” (this is subjective and another topic of its own)
- VMWare GDB stub allows you to read/write physical memory, virtual memory, control registers, general purpose registers, etc.
- Thus we can use the GDB stub to read/write to process memory inside of the virtual machine from our host operating system.
    - These are called “External cheats” and dont require even unpacking the APK.



Memory Introspection Of Guest Virtual Machines

- GDB stub does not parse task structures or ANY operating system structures
- It simply gives us basic Read/Write primitives
  - Meaning it is up to us to implement a library/code to make it more useful
- First step, finding the kernel’s base address…
  - IDTR! (Interrupt descriptor table register)
  - Contains the base address of the interrupt descriptor table
    - Interrupt descriptor table contains linear virtual addresses of interrupt routines
    - These interrupt routines are within the kernels module.
    - Vector 0 (division by 0) aka: vmlinux!asm_exc_divide_error is inside of vmlinux.ko…
- Once we have the kernels base address we can then parse the ELF file for segments
- We can then start looking for task structures and subsequently the games process/loaded modules...

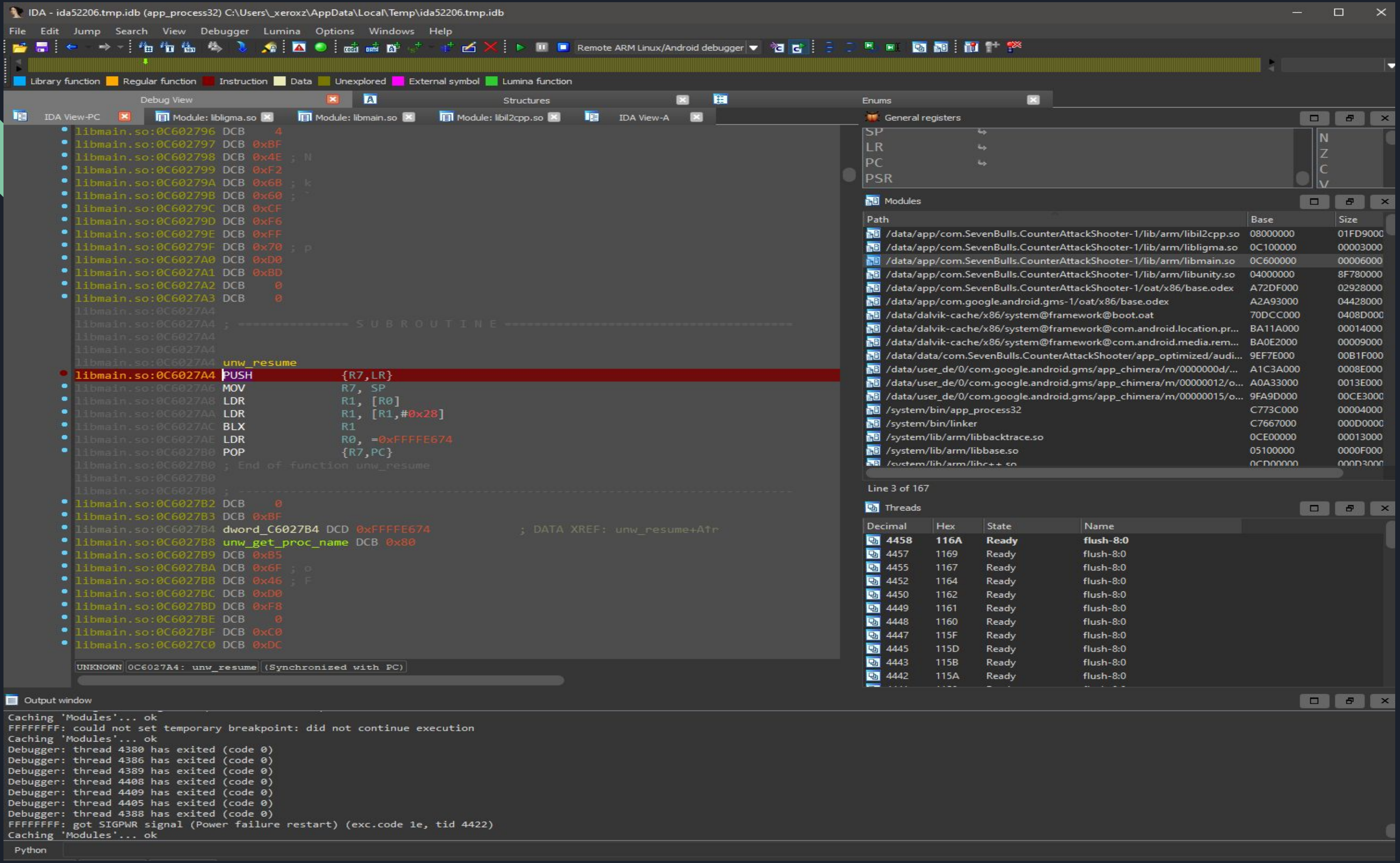


```
.text:FFFFFFFF81C00870
.text:FFFFFFFF81C00870 ; =============== S U B R O U T I N E =======================================
.text:FFFFFFFF81C00870
.text:FFFFFFFF81C00870
.text:FFFFFFFF81C00870 asm_exec_division_error proc near
.text:FFFFFFFF81C00870
.text:FFFFFFFF81C00870 ; FUNCTION CHUNK AT .text:FFFFFFFF81C00FB0 SIZE 00000036 BYTES
.text:FFFFFFFF81C00870 ; FUNCTION CHUNK AT .text:FFFFFFFF81C01029 SIZE 0000002F BYTES
.text:FFFFFFFF81C00870 ; FUNCTION CHUNK AT .text:FFFFFFFF81C01410 SIZE 00000013 BYTES
.text:FFFFFFFF81C00870
.text:FFFFFFFF81C00870                 nop
.text:FFFFFFFF81C00871                 nop
.text:FFFFFFFF81C00872                 nop
.text:FFFFFFFF81C00873                 push    0FFFFFFFFFFFFFFFFh
.text:FFFFFFFF81C00875                 call    sub_FFFFFFFF81C01320
.text:FFFFFFFF81C0087A                 mov     rdi, rsp
.text:FFFFFFFF81C0087D                 call    sub_FFFFFFFF81BEB500
.text:FFFFFFFF81C00882                 jmp     loc_FFFFFFFF81C01410
.text:FFFFFFFF81C00882 asm_exec_division_error endp
.text:FFFFFFFF81C00882
```

How To Debug Your Application?

Below is a summary of the steps required to debug. You can find a more detailed blog post about such a thing here if you are interested: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/debugging-android-libraries-using-ida/

- We an use ADB to push files to the Memu instance.
- We can use ADB to install our patched APK.
- We can use ADB as a shell to execute a remote GDB server.
- We can then use IDA Pro to connect to they remote GDB server.
- We can then load symbols for our shared object and set breakpoints.



```
libmain.so:0C602796 DCB    4
libmain.so:0C602797 DCB 0xBF
libmain.so:0C602798 DCB 0x4E ; N
libmain.so:0C602799 DCB 0xF2
libmain.so:0C60279A DCB 0x6B ; k
libmain.so:0C60279B DCB 0x60 ; `
libmain.so:0C60279C DCB 0xCF
libmain.so:0C60279D DCB 0xF6
libmain.so:0C60279E DCB 0xFF
libmain.so:0C60279F DCB 0x70 ; p
libmain.so:0C6027A0 DCB 0xD0
libmain.so:0C6027A1 DCB 0xBD
libmain.so:0C6027A2 DCB    0
libmain.so:0C6027A3 DCB    0
libmain.so:0C6027A4
libmain.so:0C6027A4 ; =============== S U B R O U T I N E =======================================
libmain.so:0C6027A4
libmain.so:0C6027A4
libmain.so:0C6027A4 unw_resume
libmain.so:0C6027A4 PUSH            {R7,LR}
libmain.so:0C6027A6 MOV             R7, SP
libmain.so:0C6027A8 LDR             R1, [R0]
libmain.so:0C6027AA LDR             R1, [R1,#0x28]
libmain.so:0C6027AC BLX             R1
libmain.so:0C6027AE LDR             R0, =0xFFFFE674
libmain.so:0C6027B0 POP             {R7,PC}
libmain.so:0C6027B0 ; End of function unw_resume
libmain.so:0C6027B0
libmain.so:0C6027B0 ; ---------------------------------------------------------------------------
libmain.so:0C6027B2 DCB    0
libmain.so:0C6027B3 DCB 0xBF
libmain.so:0C6027B4 dword_C6027B4 DCD 0xFFFFE674          ; DATA XREF: unw_resume+A↑r
libmain.so:0C6027B8 unw_get_proc_name DCB 0x80
libmain.so:0C6027B9 DCB 0xB5
libmain.so:0C6027BA DCB 0x6F ; o
libmain.so:0C6027BB DCB 0x46 ; F
libmain.so:0C6027BC DCB 0xD0
libmain.so:0C6027BD DCB 0xF8
libmain.so:0C6027BE DCB    0
libmain.so:0C6027BF DCB 0xC0
libmain.so:0C6027C0 DCB 0xDC
UNKNOWN 0C6027A4: unw_resume (Synchronized with PC)
```

Path	Base	Size
/data/app/com.SevenBulls.CounterAttackShooter-1/lib/arm/libil2cpp.so	08000000	01FD9000
/data/app/com.SevenBulls.CounterAttackShooter-1/lib/arm/libligma.so	0C100000	00003000
/data/app/com.SevenBulls.CounterAttackShooter-1/lib/arm/libmain.so	0C600000	00006000
/data/app/com.SevenBulls.CounterAttackShooter-1/lib/arm/libunity.so	04000000	8F780000
/data/app/com.SevenBulls.CounterAttackShooter-1/oat/x86/base.odex	A72DF000	02928000
/data/app/com.google.android.gms-1/oat/x86/base.odex	A2A93000	04428000
/data/dalvik-cache/x86/system@framework@boot.oat	70DCC000	0408D000
/data/dalvik-cache/x86/system@framework@com.android.location.pr...	BA11A000	00014000
/data/dalvik-cache/x86/system@framework@com.android.media.rem...	BA0E2000	00009000
/data/data/com.SevenBulls.CounterAttackShooter/app_optimized/audi...	9EF7E000	00B1F000
/data/user_de/0/com.google.android.gms/app_chimera/m/0000000d/...	A1C3A000	0008E000
/data/user_de/0/com.google.android.gms/app_chimera/m/00000012/o...	A0A33000	0013E000
/data/user_de/0/com.google.android.gms/app_chimera/m/00000015/o...	9FA9D000	00CE3000
/system/bin/app_process32	C773C000	00004000
/system/bin/linker	C7667000	000D0000
/system/lib/arm/libbacktrace.so	0CE00000	00013000
/system/lib/arm/libbase.so	05100000	0000F000
/system/lib/arm/libc++.so	0CD00000	000D3000

Line 3 of 167

Decimal	Hex	State	Name
4458	**116A**	**Ready**	**flush-8:0**
4457	1169	Ready	flush-8:0
4455	1167	Ready	flush-8:0
4452	1164	Ready	flush-8:0
4450	1162	Ready	flush-8:0
4449	1161	Ready	flush-8:0
4448	1160	Ready	flush-8:0
4447	115F	Ready	flush-8:0
4445	115D	Ready	flush-8:0
4443	115B	Ready	flush-8:0
4442	115A	Ready	flush-8:0

```
Caching 'Modules'... ok
FFFFFFFF: could not set temporary breakpoint: did not continue execution
Caching 'Modules'... ok
Debugger: thread 4380 has exited (code 0)
Debugger: thread 4386 has exited (code 0)
Debugger: thread 4389 has exited (code 0)
Debugger: thread 4408 has exited (code 0)
Debugger: thread 4409 has exited (code 0)
Debugger: thread 4405 has exited (code 0)
Debugger: thread 4388 has exited (code 0)
FFFFFFFF: got SIGPWR signal (Power failure restart) (exc.code 1e, tid 4422)
Caching 'Modules'... ok
```

Game Engines, Unity

Unity is a game engine which allows you to easily develop games for almost every platform imaginable. It is a very very popular game engine.

Games are written in C# or javascript and trans-compiled into C++, which is then compiled into native. This process is called IL-2-CPP.

- C#
- C++
- Native instructions (native binaries)

https://docs.unity3d.com/Manual/IL2CPP.html

Generating an SDK for the game

Unity creates a file by the name of “global-metadata.dat” which contains all sorts of information for all of the symbols in the il2cpp.so shared object. We can use publicly available tools to generate C++ header files for every single function in the entire game!

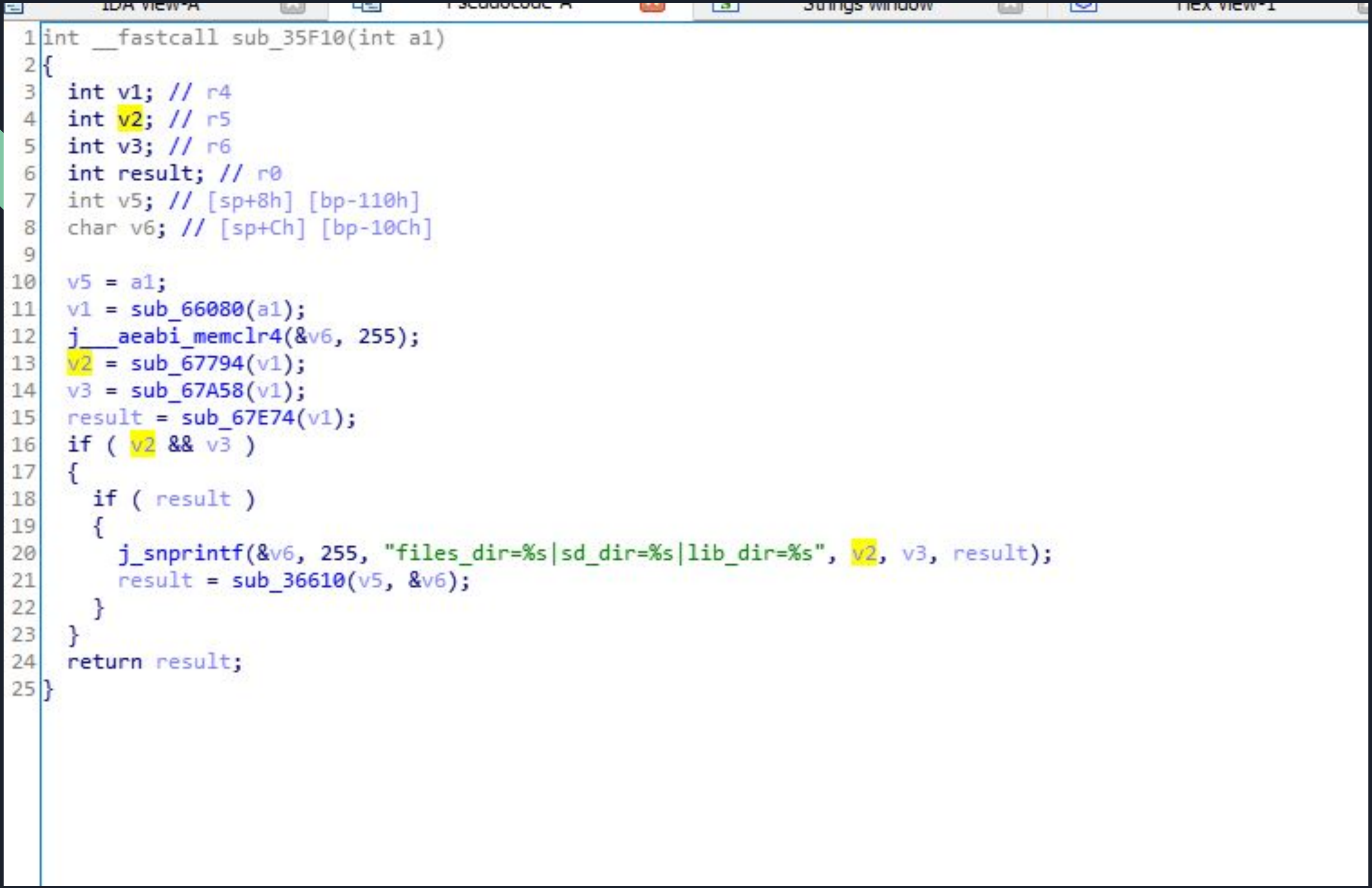
- Il2CppDumper (https://github.com/Perfare/Il2CppDumper)
- Il2CppSDKGenerator (https://github.com/kur0yama/Il2CppSDKGenerator)

Mobile Anti Cheats/Protections

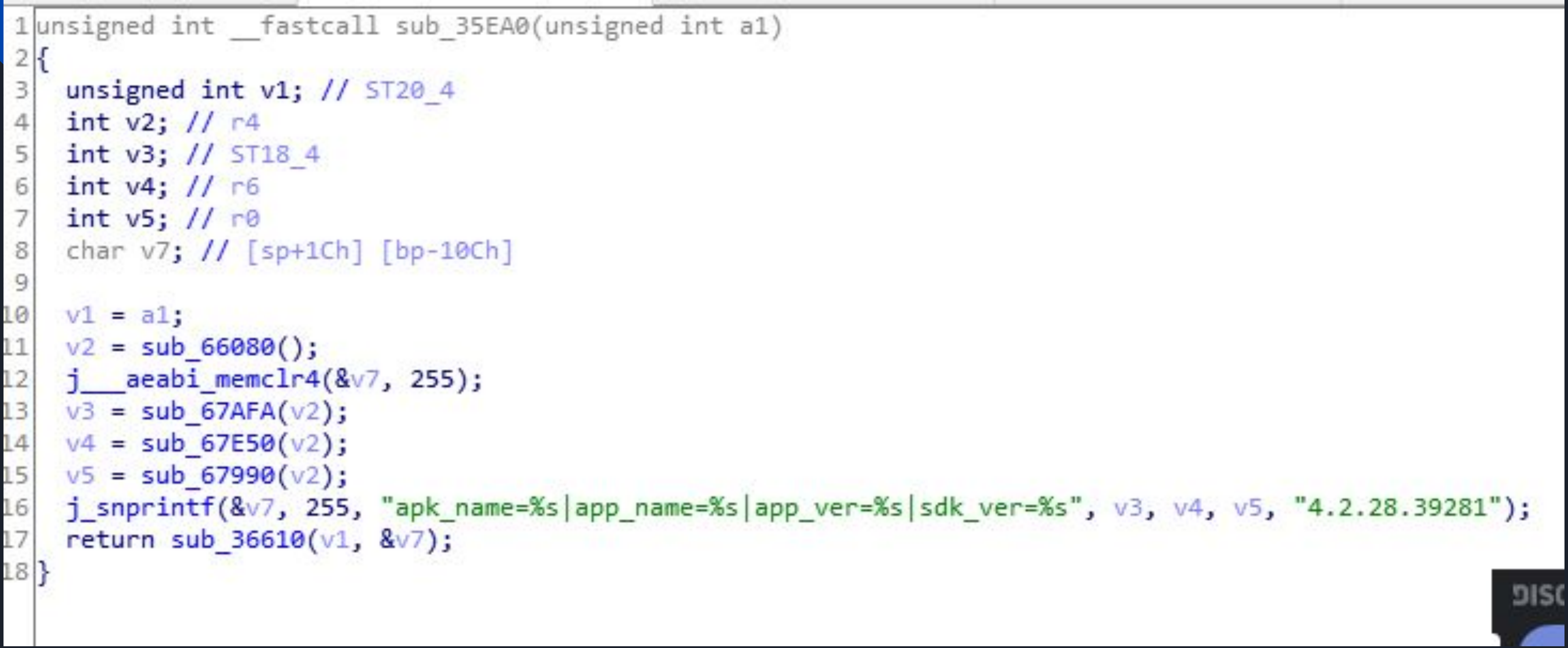
- Tencent
  - Shared objects in the libs/ directory
    - libtersafe.so
    - libtprt.so
- Anti Cheat Tool Kit (ACTK)
  - Compiled into libil2cpp.so
  - Strings in libil2cpp.so
  - Public SDK documentation.
  - Client sided/can be patched easily
- BeeByte Obfuscator
  - global-metadata.dat method renaming.
  - 40$ on unity asset store

Tencent Anti Cheat Emulation

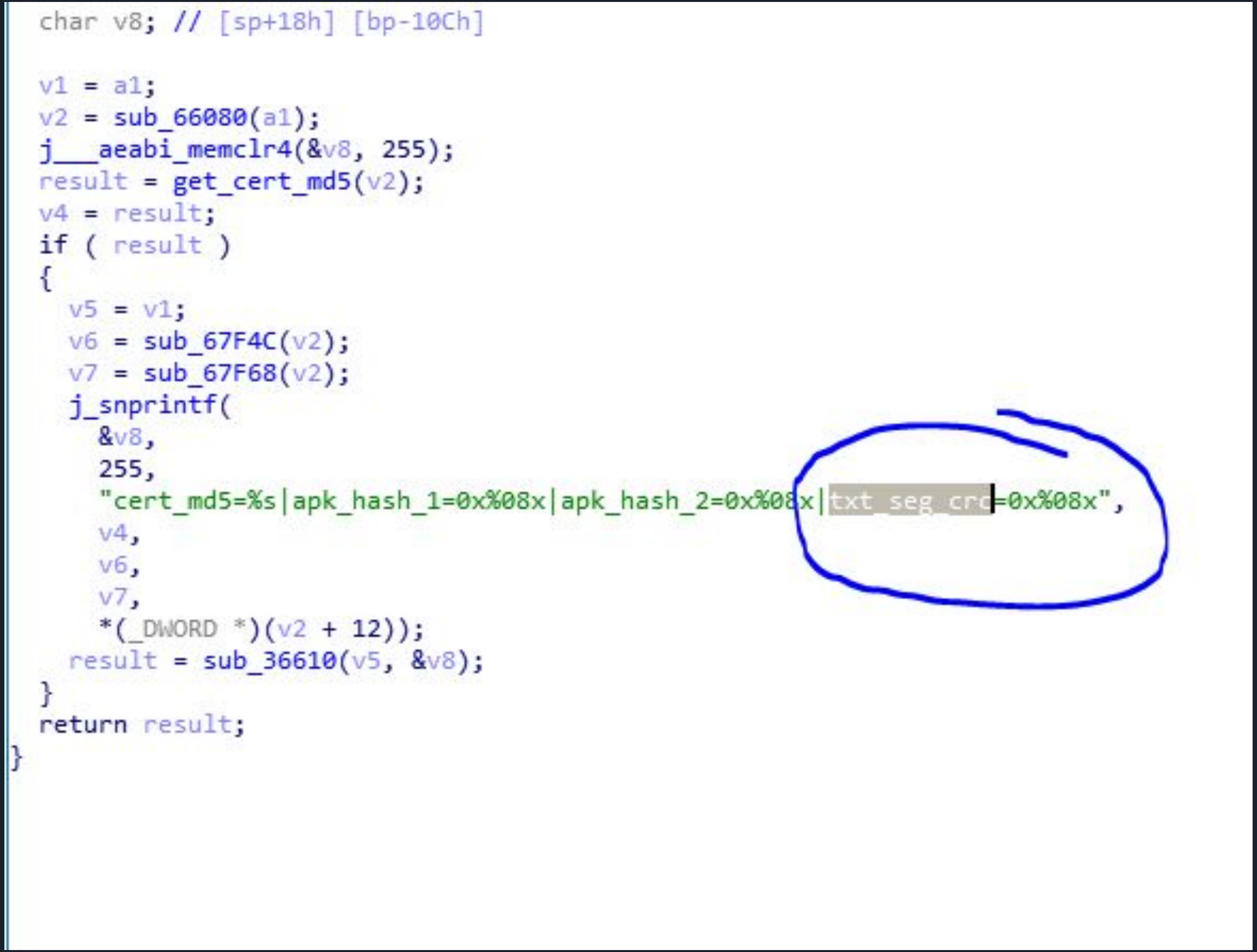
- A global report structure exists and can be filter however one would like.
- The next three slides show the serialization of this global data structure



```c
int __fastcall sub_35F10(int a1)
{
  int v1; // r4
  int v2; // r5
  int v3; // r6
  int result; // r0
  int v5; // [sp+8h] [bp-110h]
  char v6; // [sp+Ch] [bp-10Ch]

  v5 = a1;
  v1 = sub_66080(a1);
  j__aeabi_memclr4(&v6, 255);
  v2 = sub_67794(v1);
  v3 = sub_67A58(v1);
  result = sub_67E74(v1);
  if ( v2 && v3 )
  {
    if ( result )
    {
      j_snprintf(&v6, 255, "files_dir=%s|sd_dir=%s|lib_dir=%s", v2, v3, result);
      result = sub_36610(v5, &v6);
    }
  }
  return result;
}
```



```
unsigned int __fastcall sub_35EA0(unsigned int a1)
{
  unsigned int v1; // ST20_4
  int v2; // r4
  int v3; // ST18_4
  int v4; // r6
  int v5; // r0
  char v7; // [sp+1Ch] [bp-10Ch]

  v1 = a1;
  v2 = sub_66080();
  j___aeabi_memclr4(&v7, 255);
  v3 = sub_67AFA(v2);
  v4 = sub_67E50(v2);
  v5 = sub_67990(v2);
  j_snprintf(&v7, 255, "apk_name=%s|app_name=%s|app_ver=%s|sdk_ver=%s", v3, v4, v5, "4.2.28.39281");
  return sub_36610(v1, &v7);
}
```



```
  char v8; // [sp+18h] [bp-10Ch]

  v1 = a1;
  v2 = sub_66080(a1);
  j___aeabi_memclr4(&v8, 255);
  result = get_cert_md5(v2);
  v4 = result;
  if ( result )
  {
    v5 = v1;
    v6 = sub_67F4C(v2);
    v7 = sub_67F68(v2);
    j_snprintf(
      &v8,
      255,
      "cert_md5=%s|apk_hash_1=0x%08x|apk_hash_2=0x%08x|txt_seg_crc=0x%08x",
      v4,
      v6,
      v7,
      *(_DWORD *)(v2 + 12));
    result = sub_36610(v5, &v8);
  }
  return result;
}
```

Tencent Conclusions

- Complete emulation of the anti cheat modules is possible however time consuming
- Easier to filter upload data when it is serialized
- “State of the art anti cheat” (for mobile)

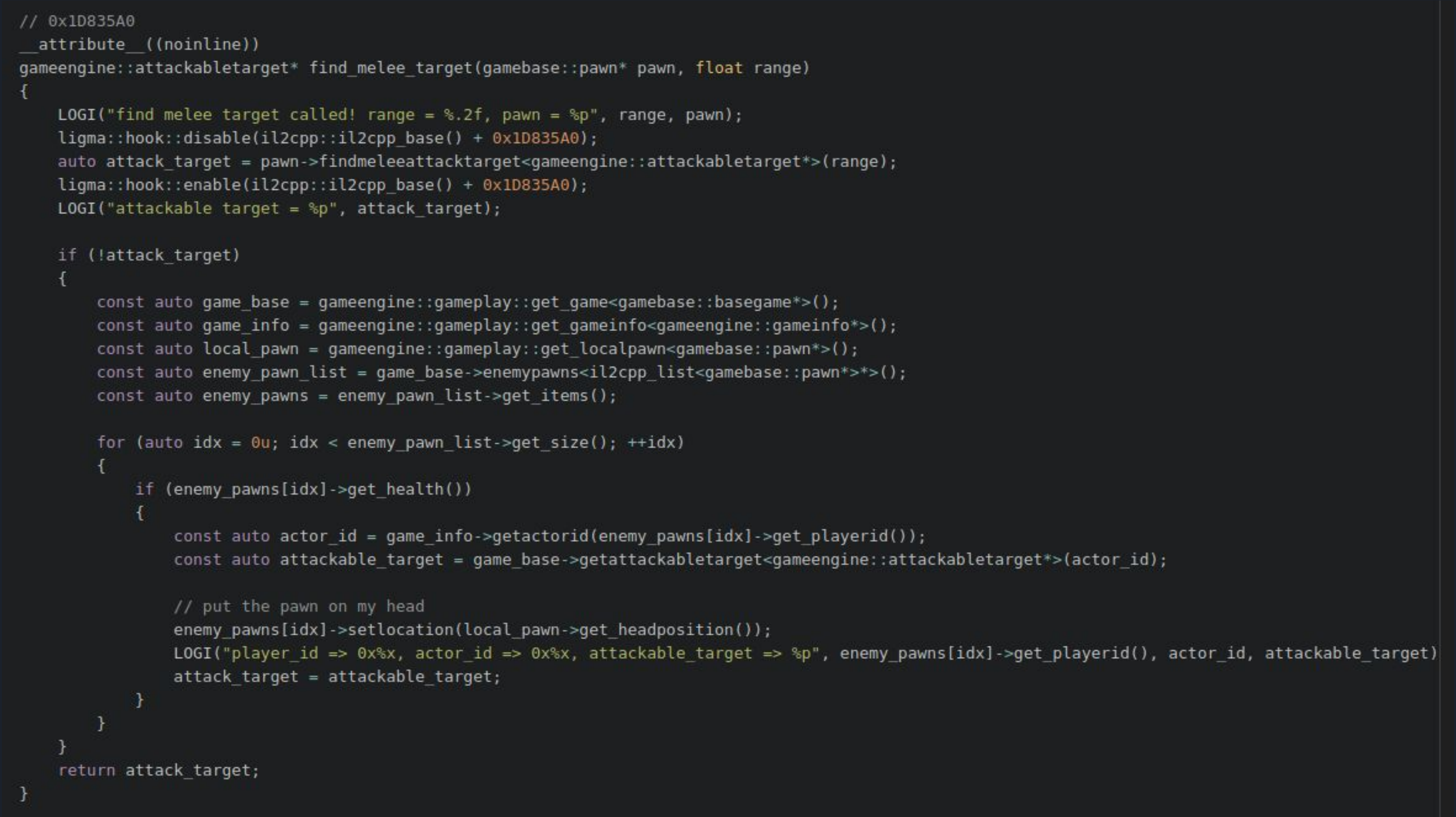
Anti-Cheat Tool Kit (ACTK)

- Client side only
- Can be completely disabled with 2 function calls
  - https://codestage.net/uas_files/actk/api/class_code_stage_1_1_anti_cheat_1_1_detectors_1_1_injection_detector.html#aa40763670bfeb15744b06e46649c4498
  - https://codestage.net/uas_files/actk/api/class_code_stage_1_1_anti_cheat_1_1_detectors_1_1_obscured_cheating_detector.html#aa40763670bfeb15744b06e46649c4498
  - https://codestage.net/uas_files/actk/api/class_code_stage_1_1_anti_cheat_1_1_detectors_1_1_speed_hack_detector.html#aa40763670bfeb15744b06e46649c4498

Proof Of Concept: CODM Cheat

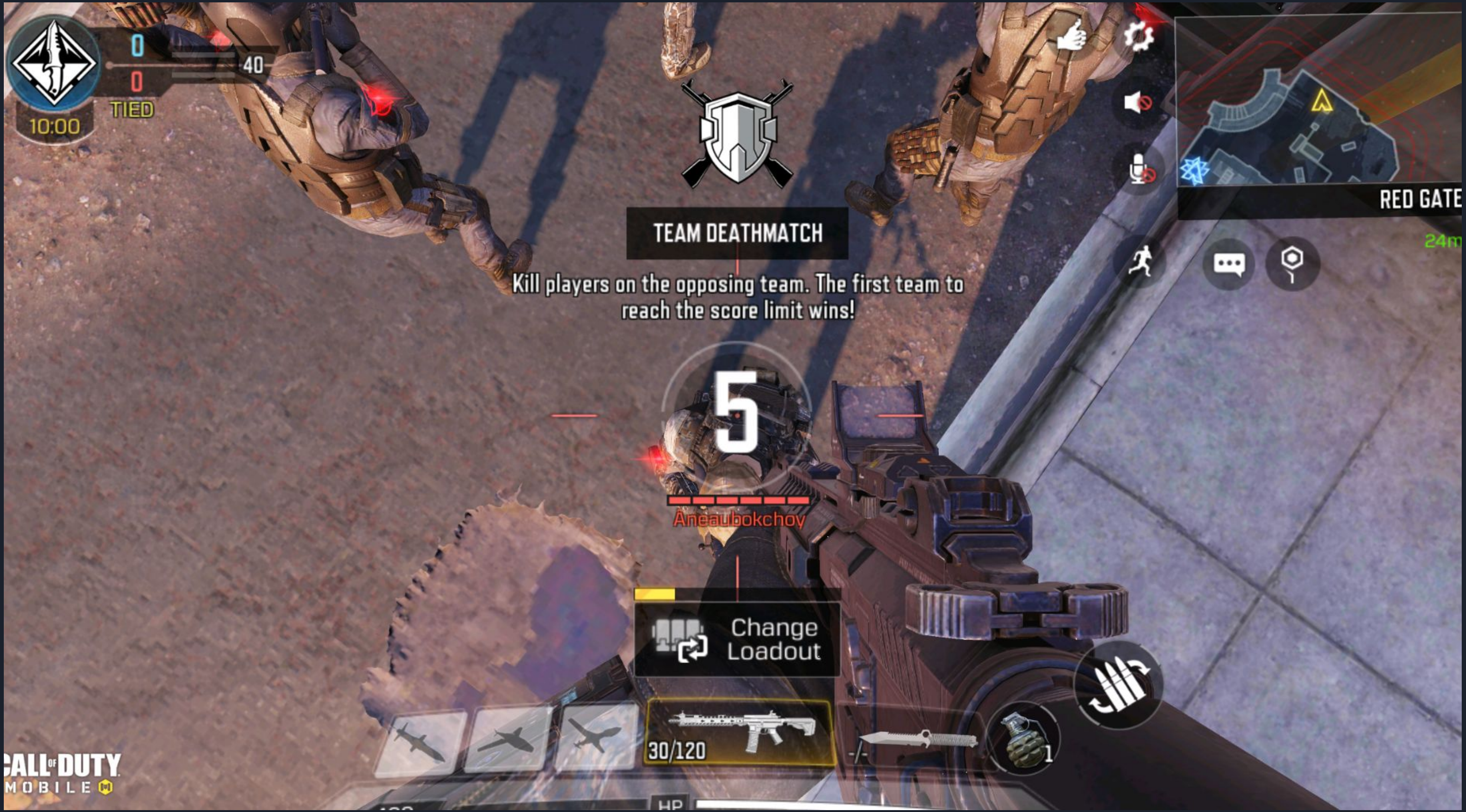
- SDK Showcase
- Teleporting
- Teleport Enemies to Knife
- Anti Cheat Bypass And Filter

Proof Of Concept: Insta Knife

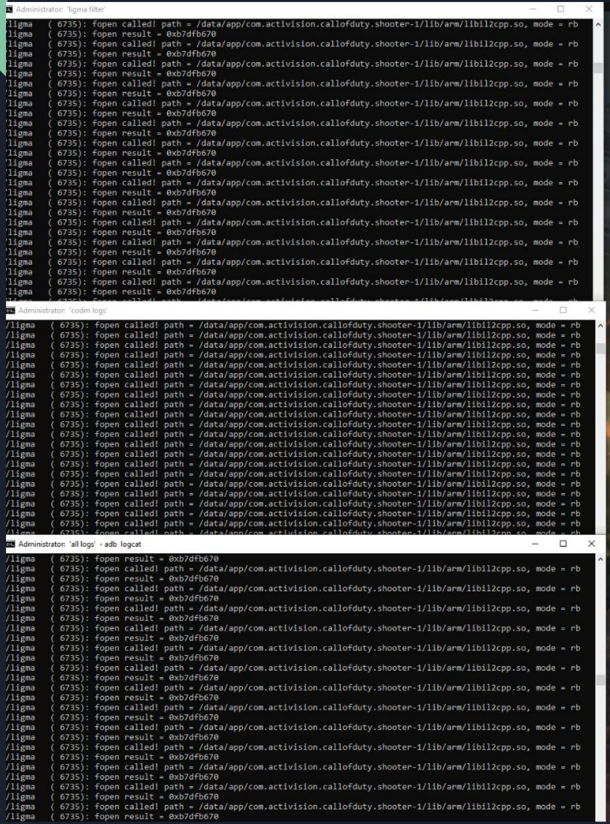


```
// 0x1D835A0
__attribute__((noinline))
gameengine::attackabletarget* find_melee_target(gamebase::pawn* pawn, float range)
{
    LOGI("find melee target called! range = %.2f, pawn = %p", range, pawn);
    ligma::hook::disable(il2cpp::il2cpp_base() + 0x1D835A0);
    auto attack_target = pawn->findmeleeattacktarget<gameengine::attackabletarget*>(range);
    ligma::hook::enable(il2cpp::il2cpp_base() + 0x1D835A0);
    LOGI("attackable target = %p", attack_target);

    if (!attack_target)
    {
        const auto game_base = gameengine::gameplay::get_game<gamebase::basegame*>();
        const auto game_info = gameengine::gameplay::get_gameinfo<gameengine::gameinfo*>();
        const auto local_pawn = gameengine::gameplay::get_localpawn<gamebase::pawn*>();
        const auto enemy_pawn_list = game_base->enemypawns<il2cpp_list<gamebase::pawn*>*>();
        const auto enemy_pawns = enemy_pawn_list->get_items();

        for (auto idx = 0u; idx < enemy_pawn_list->get_size(); ++idx)
        {
            if (enemy_pawns[idx]->get_health())
            {
                const auto actor_id = game_info->getactorid(enemy_pawns[idx]->get_playerid());
                const auto attackable_target = game_base->getattackabletarget<gameengine::attackabletarget*>(actor_id);

                // put the pawn on my head
                enemy_pawns[idx]->setlocation(local_pawn->get_headposition());
                LOGI("player_id => 0x%x, actor_id => 0x%x, attackable_target => %p", enemy_pawns[idx]->get_playerid(), actor_id, attackable_target)
                attack_target = attackable_target;
            }
        }
    }
    return attack_target;
}
```

Proof Of Concept: Teleporting



Proof Of Concept: Anti Cheat Bypass



Administrator: 'ligma filter'

```
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
ligma   ( 6735): fopen result = 0xb7dfb670
```

Administrator: 'codm logs'

```
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
```

Administrator: 'all logs' - adb logcat

```
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
/ligma   ( 6735): fopen called! path = /data/app/com.activision.callofduty.shooter-1/lib/arm/libil2cpp.so, mode = rb
/ligma   ( 6735): fopen result = 0xb7dfb670
```

Conclusions & Future Works?

- Rendering/Drawing?
  - Opengl-ES (ES meaning Embedded systems) is a pain in the ass!
  - Easier to compile java classes to smali and embed them in the apk and register them so that C++ can call into them to draw...
    - This is another presentation of its own
- GDB Stub library/reading process memory from the host?
- Desktop Games?
- Other non-game applications?
- Fuzzing networking protocols (lots of signed integers in unity! Subtract a negative number?)