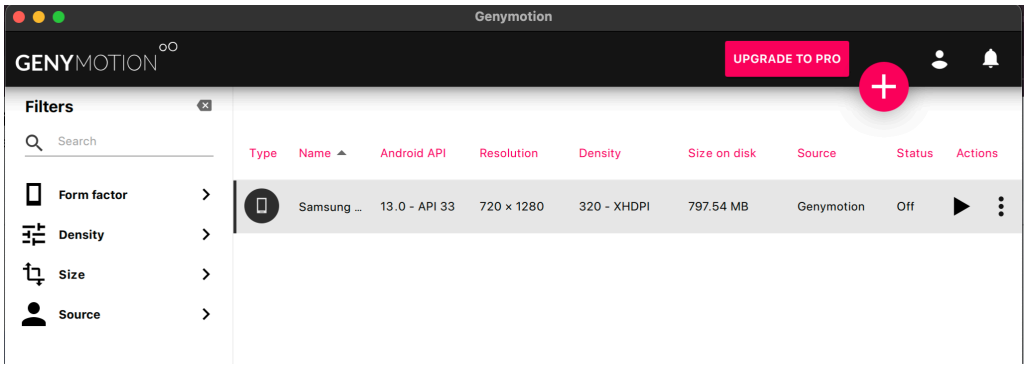
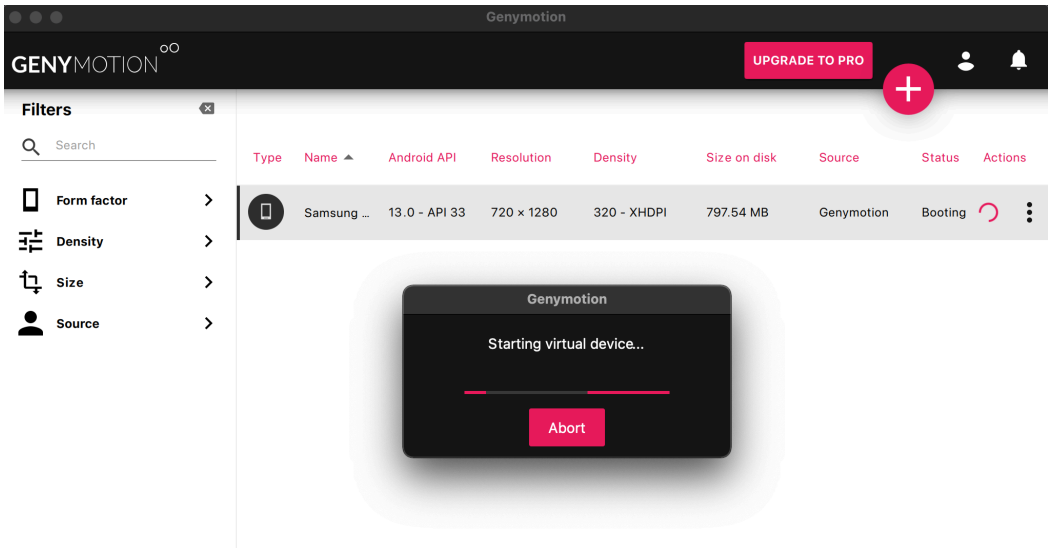
Mobile Application Dynamic Analysis with Frida and ADB
By: Tyler Dionne

Install:
https://www.genymotion.com/product-desktop/download/

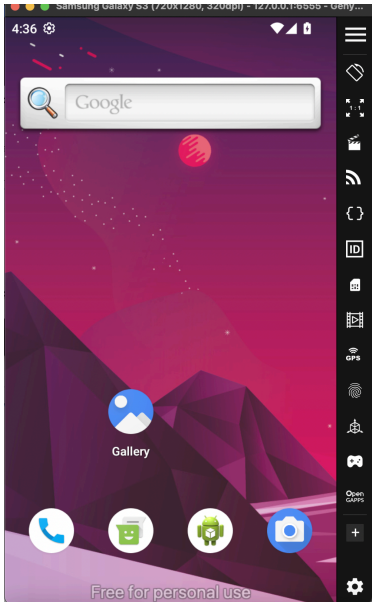
1. Open Genymotion with Emulated Android Phone:




2. Run the phone




3. We see:




4. Install adb on your host machine:

$ brew install android-platform-tools

```
==> Downloading https://dl.google.com/android/repository/platform-tools_r35.0.2-darwin.zip
######################################################################################################################## 100.0%
==> Installing Cask android-platform-tools
==> Linking Binary 'adb' to '/opt/homebrew/bin/adb'
==> Linking Binary 'etc1tool' to '/opt/homebrew/bin/etc1tool'
==> Linking Binary 'fastboot' to '/opt/homebrew/bin/fastboot'
==> Linking Binary 'hprof-conv' to '/opt/homebrew/bin/hprof-conv'
==> Linking Binary 'make_f2fs' to '/opt/homebrew/bin/make_f2fs'
==> Linking Binary 'make_f2fs_casefold' to '/opt/homebrew/bin/make_f2fs_casefold'
==> Linking Binary 'mke2fs' to '/opt/homebrew/bin/mke2fs'
🍺  android-platform-tools was successfully installed!
Warning: You are using macOS 12.
We (and Apple) do not provide support for this old version.
It is expected behaviour that some formulae will fail to build in this old version.
It is expected behaviour that Homebrew will be buggy and slow.
Do not create any issues about this on Homebrew's GitHub repositories.
Do not create any issues even if you think this message is unrelated.
Any opened issues will be immediately closed without response.
Do not ask for help from Homebrew or its maintainers on social media.
You may ask for help in Homebrew's discussions but are unlikely to receive a response.
Try to figure out the problem yourself and submit a fix as a pull request.
We will review it but may or may not accept it.
```

$ adb version

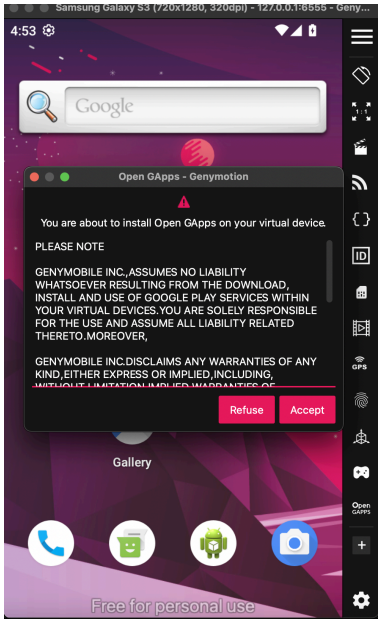
```
Android Debug Bridge version 1.0.41
Version 35.0.2-12147458
Installed as /opt/homebrew/bin/adb
Running on Darwin 21.1.0 (arm64)
```

5. Run the following command to see the device running:

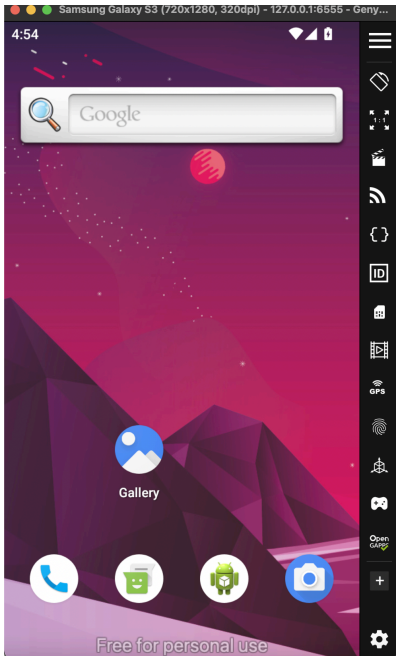
$ adb devices

```
List of devices attached
127.0.0.1:6555  device
```

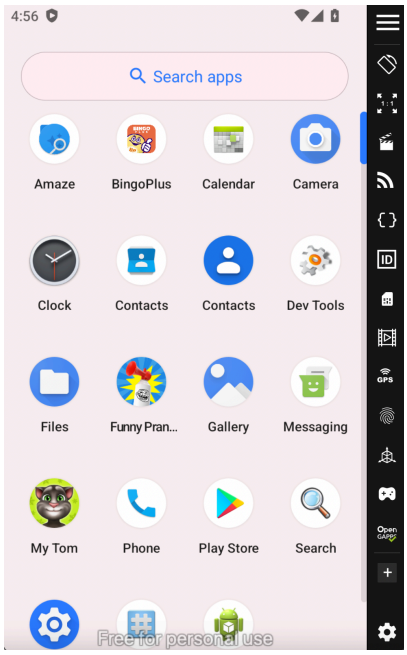
6. Install GApps from the sidebar menu to get access to google play store on the phone so we can run a process:




Restart the phone after installation:



7. Pick a process that you want to capture memory from and run it on the emulated phone: (Note: to get to the apps act like your cursor is your finger and swipe up)




I will choose Funny Pranks


8. Running the command:

$ adb shell ps | grep prank

We obtain the following information:

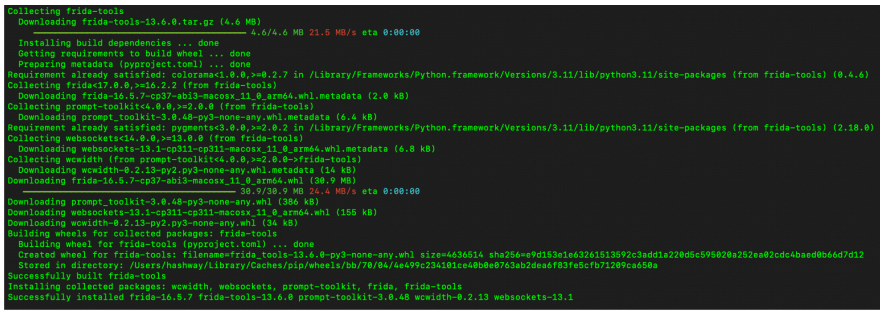
u0_a89 11513 1453 32698572 129388 do_epoll_wait 0 S com.airhorn.funny.prank.sounds

Showing us that our process is running with pid = 11513.

9. To capture the process memory we will use Frida. Frida is a dynamic instrumentation toolkit that allows you to interact with running processes. We can use it to capture process memory by attaching the target application and analyze the process memory data in real time with Frida.

Install Frida cli on host:

$ pip install frida-tools




$ pip install frida-tools
Download Frida Server. We have to make sure we download the right version for our Android architecture (ex. arm64, x86).

We can check the architecture of the phone we are emulating using adb:

$ adb shell getprop ro.product.cpu.abi

Which displays:

arm64-v8a

Meaning we should download the arm64 Frida server.

https://github.com/frida/frida/releases

Go to the webpage, hit show all assets and then scroll until you find something like:

frida-server-16.5.7-android-arm64.xz

And download it.

Unpack it:

$ unxz frida-server-16.5.7-android-arm64.xz

Now run the following command to push it to the phone:

$ adb push frida-server-16.5.7-android-arm64 /data/local/tmp/frida-server

Output should be something like:

$ frida-server-16.5.7-android-arm64: 1 file pushed, 0 skipped. 221.5 MB/s (56549216 bytes in 0.243s)

Now we can see if it is there:

$ adb shell file /data/local/tmp/frida-server

We get the following information confirming the file is there:

/data/local/tmp/frida-server: ELF shared object, 64-bit LSB arm64, dynamic (/system/bin/linker64), for Android 21, built by NDK r25b (8937393), stripped

Make it executable:

$ adb shell chmod +x /data/local/tmp/frida-server

10. Start the Frida server on emulator:

$ adb shell "/data/local/tmp/frida-server &"

Now we use Frida to attach to the target process.

Get the pid of the process:

$ adb shell ps | grep prank

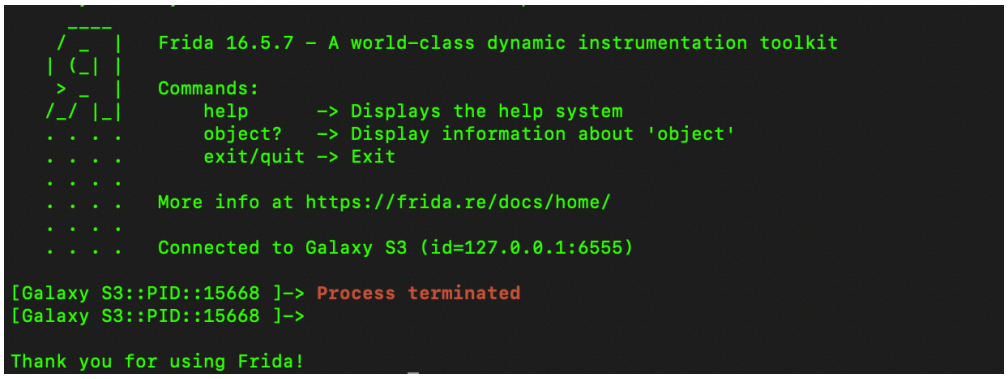
Now in one terminal:

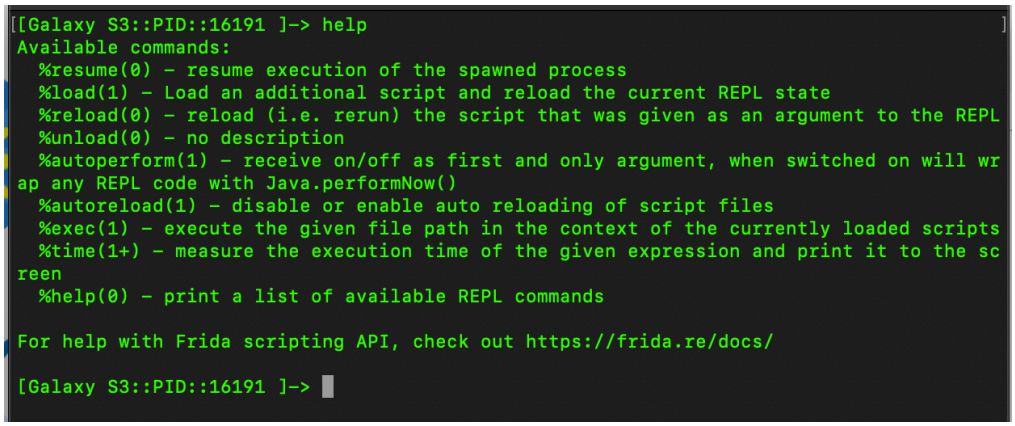
$ adb shell "/data/local/tmp/frida-server &"

In another terminal:

$ frida -U -p 15668

Should see this if successfully attached:




11. Frida script in javascript to perform dynamic analysis:

Save the following code under “memory_info.js”

```
// attach to the running process
console.log("Starting memory inspection...");

Java.perform(function () {
    // list loaded Java classes in the Android app
    var classes = Java.enumerateLoadedClassesSync();
    console.log("Loaded Java classes:");
    classes.forEach(function (className) {
        console.log(className);
```

```
    });

    // list the modules loaded in the process
    console.log("\nLoaded modules:");
    var modules = Process.enumerateModules();
    modules.forEach(function (module) {
        console.log("Module: " + module.name + " | Base address: " + module.base);
    });

    // read some memory for demonstration
    var address = ptr("0x12345678");  // example address
    try {
        var value = Memory.readU8(address);  // can try reading other types of data like readU32 or
readCString
        console.log("Memory value at address " + address + ": " + value);
    } catch (e) {
        console.log("Error reading memory: " + e.message);
    }
});
```

This script demonstrates how Frida scripting is used to interact with the application dynamically and how process memory can be analyzed. In this case we use an example memory address which will most likely result in an error but you could use a base address from one of the modules to find a valid memory address to inspect.

Get the pid of the process:

```
$ adb shell ps | grep prank
```

Now in one terminal:

```
$ adb shell "/data/local/tmp/frida-server &"
```

In another terminal:

```
$ frida -U -p 18752 -l memory_info.js
```

From this we get a massive amount of information printed to the console.

Some of this information is shown below:

```
     ____
[    / _  |   Frida 16.5.7 - A world-class dynamic instrumentation toolkit
    | (_| |
     > _  |   Commands:
    /_/ |_|       help      -> Displays the help system
    . . . .       object?   -> Display information about 'object'
    . . . .       exit/quit -> Exit
    . . . .
    . . . .   More info at https://frida.re/docs/home/
    . . . .
    . . . .   Connected to Galaxy S3 (id=127.0.0.1:6555)
Attaching...
Starting memory inspection...
Loaded Java classes:
jdk.internal.misc.SharedSecrets
sun.nio.fs.UnixChannelFactory$1
[Ljava.util.Formatter$Flags;
sun.security.util.ManifestEntryVerifier
java.time.ZonedDateTime
java.security.KeyFactory
java.time.format.DateTimeFormatterBuilder$3
java.lang.invoke.Transformers$GuardWithTest
[Ljava.util.concurrent.atomic.Striped64$Cell;
java.security.PrivateKey
sun.nio.fs.UnixFileStoreAttributes
[Ljava.security.Permission;
java.security.spec.EncodedKeySpec
java.time.format.DecimalStyle
java.util.stream.ReferencePipeline$5$1
java.lang.ThreadLocal$ThreadLocalMap
java.util.Collections$UnmodifiableSortedSet
sun.security.x509.SubjectInfoAccessExtension
java.util.logging.Handler
java.util.TreeMap$AscendingSubMap
java.io.UTFDataFormatException
```

```
com.android.internal.infra.AbstractRemoteService
android.security.keymaster.KeyCharacteristics
android.hardware.radio.V1_0.CellIdentityCdma
android.graphics.Paint$Join
android.graphics.drawable.Animatable
android.graphics.GraphicBuffer
android.security.keystore.KeyProperties$Digest
android.graphics.drawable.RotateDrawable$RotateState
android.provider.ContactsContract$BaseSyncColumns
android.telephony.NetworkScanRequest
android.animation.Animator$AnimatorListener
android.app.IAssistDataReceiver
android.content.pm.ActivityPresentationInfo
android.app.SystemServiceRegistry$85
android.content.pm.ProviderInfo
[Landroid.graphics.Matrix;
android.graphics.drawable.VectorDrawable$VGroup
android.view.HandlerActionQueue$HandlerAction
android.app.LocaleManager
android.graphics.YuvImage
android.media.MicrophoneInfo
android.location.Address
android.hardware.contexthub.V1_0.MemRange
android.os.IUserRestrictionsListener
android.telephony.data.TrafficDescriptor$Builder
android.hardware.SensorAdditionalInfo
android.graphics.drawable.RippleForeground$2
android.speech.tts.ITextToSpeechCallback
android.text.format.RelativeDateTimeFormatter$FormatterCache
android.view.InsetsController$InternalAnimationControlListener$$ExternalSyntheticLambda0
android.view.ViewRootImpl$6$$ExternalSyntheticLambda0
android.app.ActivityClient
android.view.inputmethod.SurroundingText$1
[Landroid.text.style.MetricAffectingSpan;
android.telephony.TelephonyCallback$CallDisconnectCauseListener
com.google.android.mms.MmsException
org.apache.http.conn.ssl.AndroidDistinguishedNameParser
com.android.internal.statusbar.IStatusBarService
```

```
com.appsflyer.oaid.OaidClient
com.google.android.gms.ads.internal.zzt
com.appsflyer.internal.av
y7.q
com.mbridge.msdk.e.x
r8.e5
androidx.lifecycle.g
vc.v
com.applovin.mediation.adapters.TaboolaMediationAdapter$2
com.applovin.exoplayer2.m.n$a
kf.o
com.bytedance.sdk.openadsdk.multipro.d.b
com.mbridge.msdk.e.y
com.ironsource.mediationsdk.demandOnly.o$a
vc.w
com.appsflyer.internal.aw
y7.r
androidx.lifecycle.h
k8.ps
com.mbridge.msdk.foundation.download.core.DownloadRequestQueue$ClassHolder
kf.p
k8.ri0
vc.x
com.appsflyer.internal.ax
y7.s
androidx.lifecycle.i
k8.pt
oc.i$a
[Lcom.applovin.impl.sdk.utils.d$a$a;
com.applovin.exoplayer2.d.g
u0.a
kf.q
com.bytedance.sdk.openadsdk.multipro.d.d
y7.t
com.ironsource.environment.ContextProvider$a
com.appsflyer.internal.ay
k8.pu
com.applovin.sdk.AppLovinAdDisplayListener
```

```
Module: libm.so | Base address: 0x710f7c7000
Module: libdl.so | Base address: 0x710e859000
Module: libbase.so | Base address: 0x710af03000
Module: libharfbuzz_ng.so | Base address: 0x71034c3000
Module: libminikin.so | Base address: 0x711330a000
Module: libz.so | Base address: 0x711899c000
Module: android.media.audio.common.types-V1-cpp.so | Base address: 0x710e662000
Module: audioclient-types-aidl-cpp.so | Base address: 0x7118cdd000
Module: audioflinger-aidl-cpp.so | Base address: 0x7107b15000
Module: audiopolicy-types-aidl-cpp.so | Base address: 0x7115f37000
Module: spatializer-aidl-cpp.so | Base address: 0x711318b000
Module: av-types-aidl-cpp.so | Base address: 0x70fc6f8000
Module: android.hardware.camera.device@3.2.so | Base address: 0x70fc705000
Module: libandroid_net.so | Base address: 0x710121f000
Module: libandroidicu.so | Base address: 0x711948b000
Module: libbattery.so | Base address: 0x7115e89000
Module: libnetdutils.so | Base address: 0x710f810000
Module: libmemtrack.so | Base address: 0x7118798000
Module: libandroidfw.so | Base address: 0x710af62000
Module: libappfuse.so | Base address: 0x710e892000
Module: libcrypto.so | Base address: 0x70f9073000
Module: libdebuggerd_client.so | Base address: 0x7103109000
Module: libbinder_ndk.so | Base address: 0x70fd9d3000
Module: libui.so | Base address: 0x710f683000
Module: libgraphicsenv.so | Base address: 0x711600f000
Module: libgui.so | Base address: 0x711608d000
Module: libhwui.so | Base address: 0x70fbc00000
Module: libmediandk.so | Base address: 0x7115fc8000
Module: libpermission.so | Base address: 0x710e5a2000
Module: libsensor.so | Base address: 0x7105284000
Module: libinput.so | Base address: 0x710eb92000
Module: libcamera_client.so | Base address: 0x7101553000
Module: libcamera_metadata.so | Base address: 0x70f8684000
Module: libprocinfo.so | Base address: 0x7103743000
Module: libsqlite.so | Base address: 0x70fb812000
Module: libEGL.so | Base address: 0x70f86c1000
Module: libGLESv1_CM.so | Base address: 0x7118727000
Module: libGLESv2.so | Base address: 0x710e4c3000
```

```
Module: libnms.so | Base address: 0x6d9abcb000
Module: gralloc.ranchu.so | Base address: 0x6dfde69000
Module: linux-vdso.so.1 | Base address: 0x711b131000
Error reading memory: access violation accessing 0x12345678
[Galaxy S3::PID::18752 ]->
[Galaxy S3::PID::18752 ]->
```