

Shallweplay CTF Google 2018

Introducción

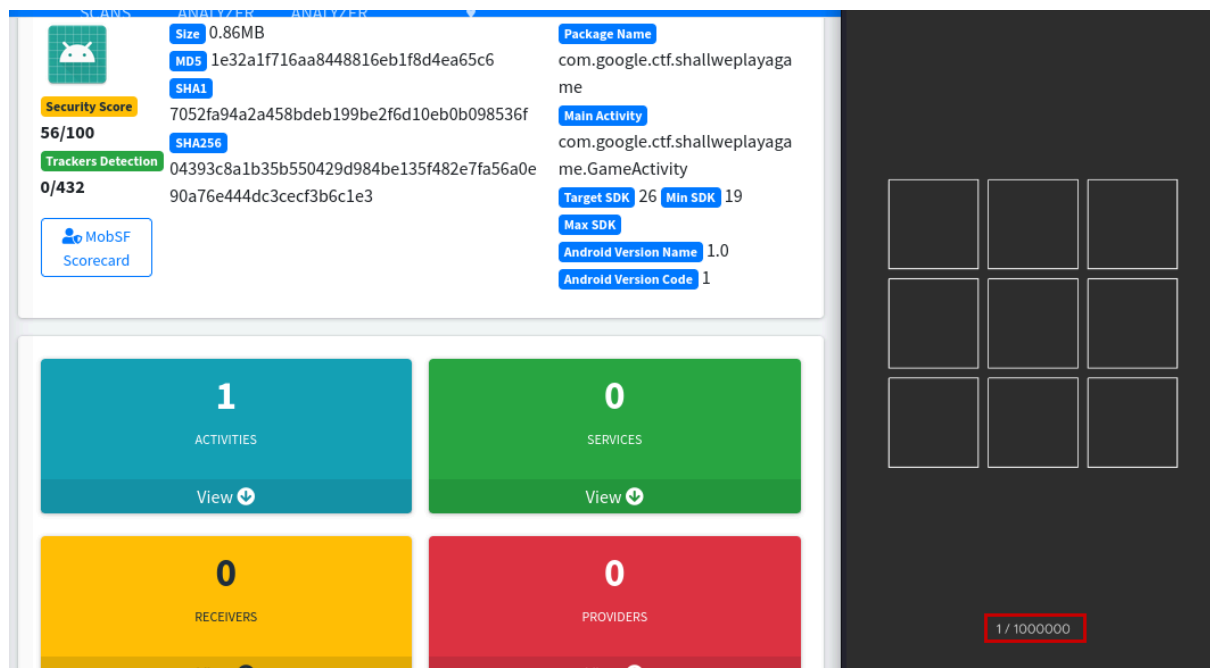
En este writeup, abordaremos el desafío "Shallweplay" del CTF de Google 2018.

La aplicación proporciona un juego de tres en raya y nuestra tarea es descubrir cómo obtener la bandera.

Análisis Inicial

Después de instalar y ejecutar la aplicación, realizamos un análisis inicial y observamos que no hay SSL pinning ni detección de root, lo que facilita el análisis. La aplicación parece requerir ganar el juego un número específico de veces para revelar la bandera.

Lanzamos MobSF para un primer análisis de la aplicación que nos servirá como punto de apoyo a la hora de buscar la bandera en el siguiente paso con el análisis estático.



Poca información podemos extraer del MobSF, principalmente que permite instalarse en versiones inseguras de android.

La aplicación solo cuenta con una actividad que será la que analizaremos estáticamente.

Análisis Estático

La principal información que encontramos relevante es la función n

```
void n() {
    for (int i = 0; i < 3; i++) {
        for (int i2 = 0; i2 < 3; i2++) {
            this.l[i2][i].a(a.EnumC0032a.EMPTY, 25);
        }
    }
    k();
    this.o++;
    Object _ = N._(2, N.e, 2);
    N._(2, N.f, _, this.q);
    this.q = (byte[]) N._(2, N.g, _);
    if (this.o == 1000000) {
        m();
    } else {
        ((TextView) findViewById(R.id.score)).setText(String.format("%d / %d", Integer.valueOf(this.o), 1000000));
    }
}
```

Como se puede ver, esta es la función a la que se llama cuando se consigue una victoria, dentro de esta función se comprueba el número de victorias (variable "o") y si se alcanza el número de victorias se llama a la función m.

```
void m() {
    Object _ = N._(0, N.a, 0);
    N._(0, N.c, _, 2, N._(1, N.b, this.q, 1));
    ((TextView) findViewById(R.id.score)).setText(new String((byte[]) N._(0, N.d, _, this.r)));
    o();
}
```

Técnicas de Bypass

Existen diversas técnicas para Bypassear esta comprobación:

- Mediante Frida, modificar la función onCreate de forma que al iniciar la aplicación se llame directamente a la función m
- Modificar las comprobaciones en smali y recompilar la aplicación de forma que no necesite un número tan elevado de victorias.
- Cambiar las condiciones del if, etc.

Mediante frida-ps buscamos el proceso relacionado con el ctf

```

(kali@kali)-[~/script]
$ frida-ps -Ua

```

PID	Name	Identifier
17408	Shall we play a game?	com.google.ctf.shallweplayagame

```

void m() {
    Object _
    N. (0, N.
    extVie
    ;
    ) {
        (int
        for (
        t
        )
        ;
        s.0++;
        Object _
        N. (2, N.
        this.q =

```

Una vez encontrado creamos un pequeño script de js para que Frida para ver qué ocurre si lanzamos directamente la función sobre escribiendo la función m().

Lamentablemente parece que la bandera no se muestra correctamente.

Probablemente sea que se necesite ejecutar el bucle todas las veces para mostrar la bandera correctamente, vamos a probar a volver a modificar la aplicación, esta vez para lanzar la función m el número de veces que sea necesario.

```

Java.perform(function() {
    // Obtener una referencia a la clase que contiene el método onCreate
    var MainActivity = Java.use('com.google.ctf.shallweplayagame.GameActiv


    // Interceptamos el método onCreate
    MainActivity.onCreate.overload('android.os.Bundle').implementation = f
    // Llamamos al método original onCreate
    this.onCreate(savedInstanceState);

    // Imprimir un mensaje para indicar que el método onCreate se ha l
    console.log('onCreate() hook!');

    // Llamar a la función 'm' al final del onCreate
    console.log('Calling m');
    try {
        this.m();
    } catch (error) {
        console.error('Error al llamar a la función m:', error);
    }

    console.log('Flag:', stringToShow);
});
});

```



```

t%BzY5t(X'a

```

La primera solución sería simplemente llamar a la función de victoria las veces necesarias en un bucle, tristemente resulta en un error por la memoria.

```
11628  n()      void    object = N(10, N.g, 0);
11629  n()      void    N(10, N.g, 0, this.g);
11630  toString()  void    this.g = (byte[]) N(10, N.g, 0);
11631  toString()  void    if (this.g == 1000000) {
11632  toString()  void    m();
11633  toString()  void    } else {
11634  toString()  void    ((TextView) findViewById(R.id.score)).setTextString(format("%d / %d", Integer.valueOf(this.g), 1000000));
11635  toString()  void    }
11636  toString()  void    }
11637  toString()  void    }
11638  toString()  void    void n() {
11639  toString()  void    for (int i = 0; i < 1000000; i++) {
11640  toString()  void    }
11641  toString()  void    }
Error al llamar a la función n: Error: java.lang.OutOfMemoryError: OutOfMemoryError[thrown while trying to throw OutOfMemoryError; no stack trace available]
ReferenceError: 'stringToShow' is not defined
    at <anonymous> (/home/kali/script/script.js:25)
    at apply (native)
    at ne (frida/node_modules/frida-java-bridge/lib/class-factory.js:673)
    at <anonymous> (frida/node_modules/frida-java-bridge/lib/class-factory.js:651)
[SM-N950F::com.google.ctf.shallweplayagame ]->
```

Vamos a intentar liberar recursos cada x iteraciones del bucle a ver si conseguimos mejores resultados.

```
Java.perform(function() {
    // Obtener una referencia a la clase que contiene el método onClick
    var MainActivity = Java.use('com.google.ctf.shallweplayagame.GameActivity');

    // Interceptamos el método onClick
    MainActivity.onClick.overload('android.view.View').implementation = function(view) {
        // Llamamos al método original onClick
        this.onClick(view);

        // Imprimir un mensaje para indicar que el método onClick se ha llamado
        console.log('onClick() hook!');

        // Llamar a la función 'n' 1,000,000 veces
        console.log('Calling n() 1,000,000 times');
        for (var i = 0; i < 1000000; i++) {
            try {
                console.log(i);
                this.n();
                if(i%1000==0){
                    console.log('Free memory');
                    Stalker.garbageCollect();
                    Java.performNow(function() {
                        var System = Java.use('java.lang.System');
                        System.gc();
                    });
                }
            } catch (error) {
                console.error('Error al llamar a la función n:', error);
            }
        }
    };
});
```

Volvemos a chocar con otro error, por este camino poco más se puede hacer ya que la clase N su código es cargado desde librerías nativas por lo que no es viable interceptar con Frida para modificar el flujo, vamos a analizar más en profundidad la aplicación..

```

11630
11631
11632
11633
11634
11635
11636
Process crashed: Trace/BPT trap
***
*** ** Build fingerprint: 'samsung/greatltexx/greatlte:9/PPR1.180610.011/N950FXXU7DSJ1:user/release-keys'
Revision: '10'
ABI: 'arm64'
pid: 20280, tid: 20280, name: hallweplayagame >>> com.google.ctf.shallweplayagame <<<
signal 6 (SIGABRT), code -6 (SI_TKILL), fault addr 0x0
Abort message: 'java_vm_ext.cc:542] JNI DETECTED ERROR IN APPLICATION: java_string = null'
x0 0000000000000000 x1 00000000000004f38 x2 0000000000000006 x3 0000000000000008
x4 fefeff79693e1667 x5 fefeff79693e1667 x6 fefeff79693e1667 x7 7f7f7f7f7f7f7f7f
x8 0000000000000083 x9 0000007a695f8828 x10 fffffff87ffffbdf x11 0000000000000001
x12 0000007fc6507978 x13 fffffff87ffffbdf x14 fffffff87ffffbdf x15 fffffff87ffffbdf
x16 0000007a6962f290 x17 0000007a6956e8e0 x18 0000007fc650714a x19 00000000000004f38
x20 00000000000004f38 x21 0000000000000083 x22 00000079e8814c00 x23 00000079c16df000
x24 0000000000000002 x25 0000000000000012 x26 0000000000000005 x27 00000079e8729013
x28 00000079e8728ec7 x29 0000007fc6507ed0
sp 0000007fc6507e90 lr 0000007a69561d28 pc 0000007a69561d54

backtrace:
#00 pc 00000000000021d54 /system/lib64/libc.so (offset 0x21000) (abort+124)
#01 pc 0000000000000108 <anonymous:0000007a6a47d000>
***
[SM-N950F::com.google.ctf.shallweplayagame ]->

```

```

package com.google.ctf.shallweplayagame;

/* loaded from: classes.dex */
class N {
    static final int[] a = {0, 1, 0};
    static final int[] b = {1, 0, 2};
    static final int[] c = {2, 0, 1};
    static final int[] d = {3, 0, 0};
    static final int[] e = {4, 1, 0};
    static final int[] f = {5, 0, 1};
    static final int[] g = {6, 0, 0};
    static final int[] h = {7, 0, 2};
    static final int[] i = {8, 0, 1};

    static {
        System.loadLibrary("rary");
    }

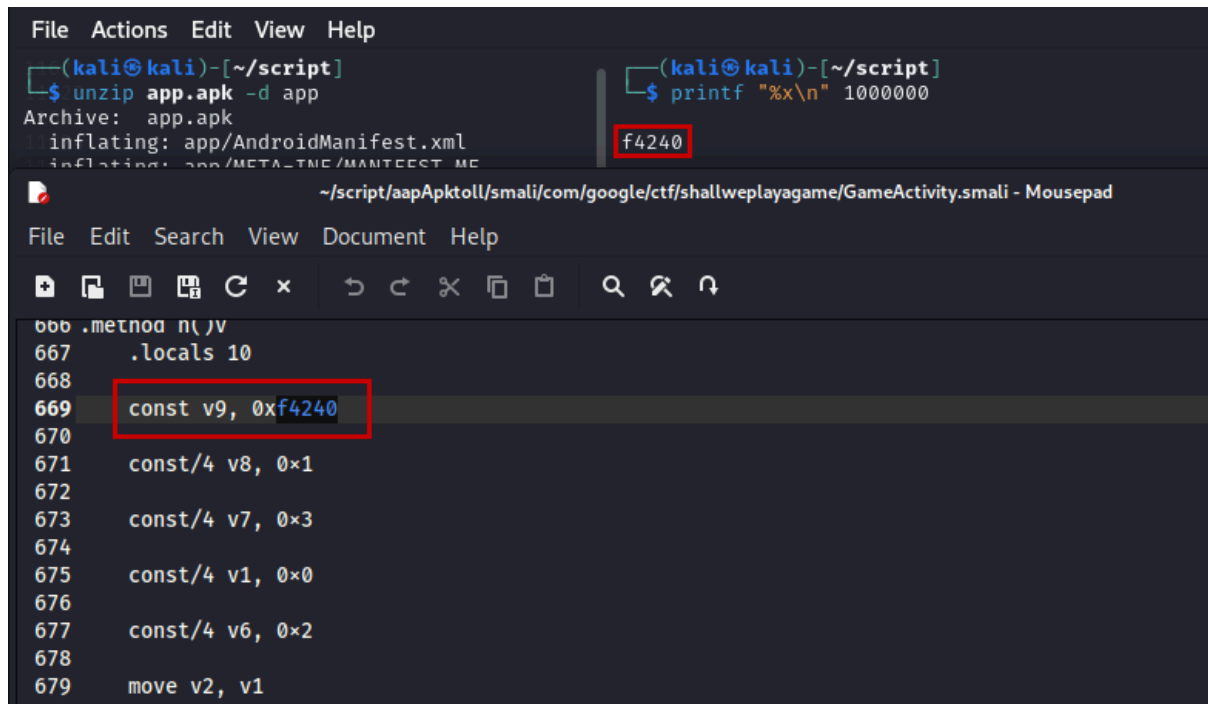
    /* JADX INFO: Access modifiers changed from: package-private */
    public static native Object _(Object... objArr);
}

```

Vamos por tanto a intentar modificar a más bajo nivel actuando sobre el código de smali.

Para ello decompilamos la aplicación utilizando apktool y buscamos en el código smali de la actividad.

Buscamos el valor de la comprobación (1000000 en hexadecimal) y de esta forma podremos encontrar la parte del bucle que nos interesa iterar.

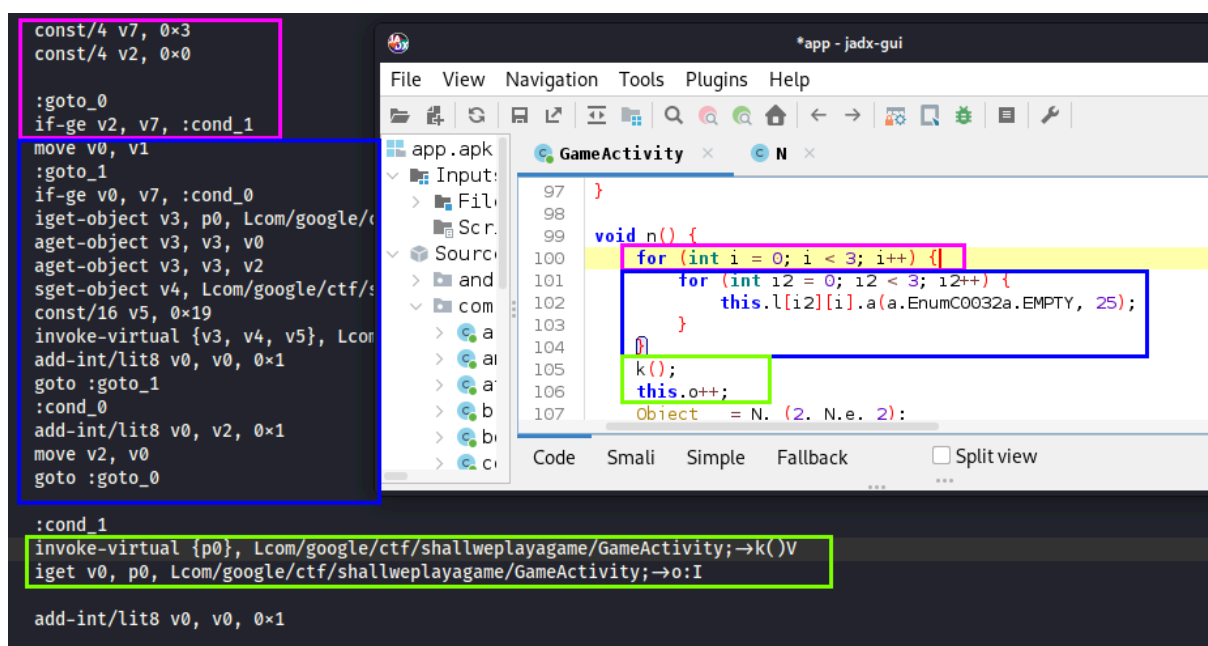


```
File Actions Edit View Help
(kali@kali)-[~/script]
$ unzip app.apk -d app
Archive: app.apk
  inflating: app/AndroidManifest.xml
  inflating: app/META-INF/MANIFEST.MF

(kali@kali)-[~/script]
$ printf "%x\n" 1000000
f4240

~/.script/aapApkoll/smali/com/google/ctf/shallweplayagame/GameActivity.smali - Mousepad
File Edit Search View Document Help
. .metnod n(JV
667 .locals 10
668
669 const v9, 0xf4240
670
671 const/4 v8, 0x1
672
673 const/4 v7, 0x3
674
675 const/4 v1, 0x0
676
677 const/4 v6, 0x2
678
679 move v2, v1
680
```

Es hora de ir analizando el código smali para ir relacionándolo con las partes de java.



```
const/4 v7, 0x3
const/4 v2, 0x0

:goto_0
if-ge v2, v7, :cond_1
move v0, v1
:goto_1
if-ge v0, v7, :cond_0
iget-object v3, p0, Lcom/google/ctf/shallweplayagame/GameActivity;
aget-object v3, v3, v0
aget-object v3, v3, v2
sget-object v4, Lcom/google/ctf/shallweplayagame/GameActivity;
const/16 v5, 0x19
invoke-virtual {v3, v4, v5}, Lcom/google/ctf/shallweplayagame/GameActivity;
add-int/lit8 v0, v0, 0x1
goto :goto_1
:cond_0
add-int/lit8 v0, v2, 0x1
move v2, v0
goto :goto_0
:cond_1
invoke-virtual {p0}, Lcom/google/ctf/shallweplayagame/GameActivity;
iget v0, p0, Lcom/google/ctf/shallweplayagame/GameActivity;
add-int/lit8 v0, v0, 0x1

void n() {
    for (int i = 0; i < 3; i++) {
        for (int i2 = 0; i2 < 3; i2++) {
            this.l[i2][i].a(a.EnumC0032a.EMPTY, 25);
        }
        k();
        this.o++;
    }
}
```

En principio la parte que debemos repetir es la que empieza a partir de cond_1. como primera idea, vamos a cambiar la parte de la condición de forma que si no se cumple la condición para obtener la flag, vuelva al inicio del bucle y repita el proceso hasta que consigamos la flag.


```
(kali@kali)-[~/script]
$ apktool b apkApktool -o apkMod.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.7.0-dirty
I: Checking whether sources has changed ...
I: Checking whether resources has changed ...
I: Copying raw resources ...
I: Building apk file ...
I: Copying unknown files/dir ...
I: Built apk into: apkMod.apk

(kali@kali)-[~/script]
$ keytool -genkey -v -keystore key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias Gari
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter keystore password:
Re-enter new password:
What is your first and last name? [Unknown]:
What is the name of your organizational unit? [Unknown]:
What is the name of your organization? [Unknown]:
What is the name of your City or Locality? [Unknown]:
What is the name of your State or Province? [Unknown]:
What is the two-letter country code for this unit? [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
[no]: yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
for: CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
[Storing key.jks]

(kali@kali)-[~/script]
$ jarsigner -keystore key.jks apkMod.apk Gari

Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter Passphrase for keystore:
jar signed.

Warning:
The signer's certificate is self-signed.

(kali@kali)-[~/script]
$ zipalign -v 4 apkMod.apk apkModAligned.apk
Verifying alignment of apkModAligned.apk (4)
```

Volvemos a ejecutar y obtenemos la bandera:

CTF{ThLssOfInncncIsThPrcOfAppls}

Conclusión

Este writeup destaca la importancia de la ingeniería inversa y la modificación de aplicaciones para la resolución de desafíos en CTFs. La combinación de técnicas de análisis estático y dinámico nos permitió superar los obstáculos y obtener la bandera.


```

move-result-object v0
check-cast v0, [B
check-cast v0, [B
input-object v0, p0, Lcom/google/ctf/
iget v0, p0, Lcom/google/ctf/shallwe
if-ne v0, v9, :cond_1
invoke-virtual {p0}, Lcom/google/ctf/
:goto_2
return-void

:cond_2
const v0, 0x7f070055
invoke-virtual {p0, v0}, Lcom/google
move-result-object v0
check-cast v0, Landroid/widget/TextV
const-string v2, "%d / %d"
new-array v3, v6, [Ljava/lang/Object
iget v4, p0, Lcom/google/ctf/shallwe
invoke-static {v4}, Ljava/lang/Integ
move-result-object v4
aput-object v4, v3, v1
invoke-static {v9}, Ljava/lang/Integ

```



CTF{ThLssOfInncnclsThPrcOfAppls}