CORE SECURITY

2 threads, 1 app (Inyección en Dalvik VM)

Martín Balao | *mbalao@coresecurity.com*

Martín Fernández | *mfernandez@coresecurity.com*

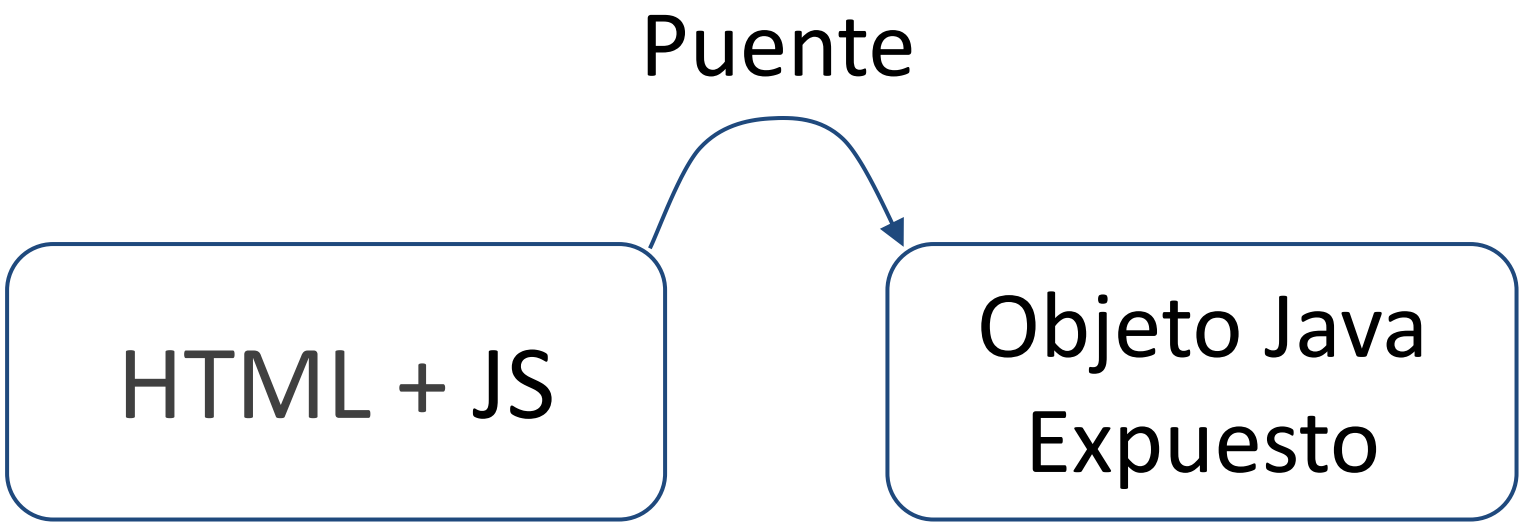
Octubre 2014

CORE SECURITY

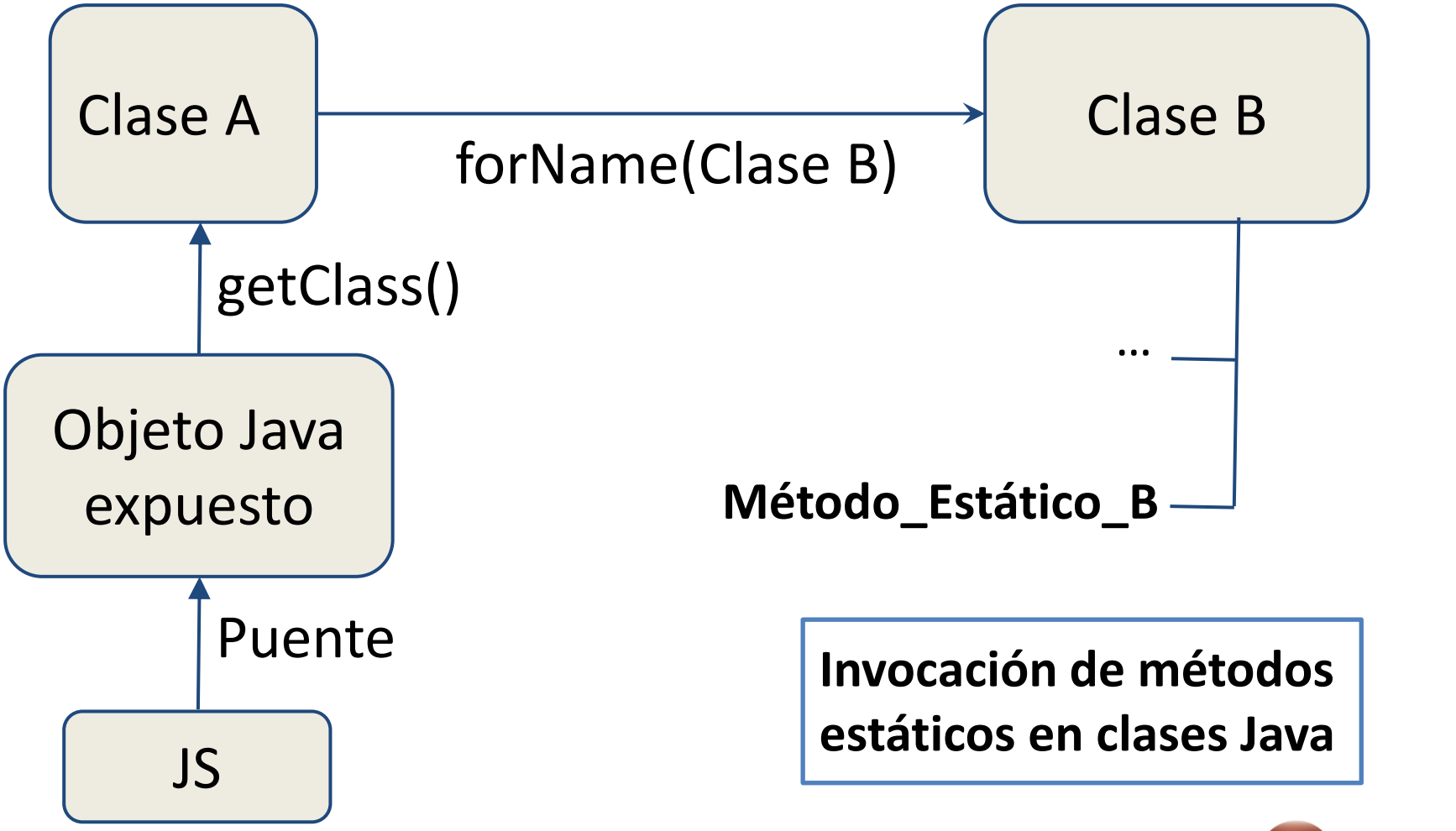
Introducción

¿Qué es un Web View?

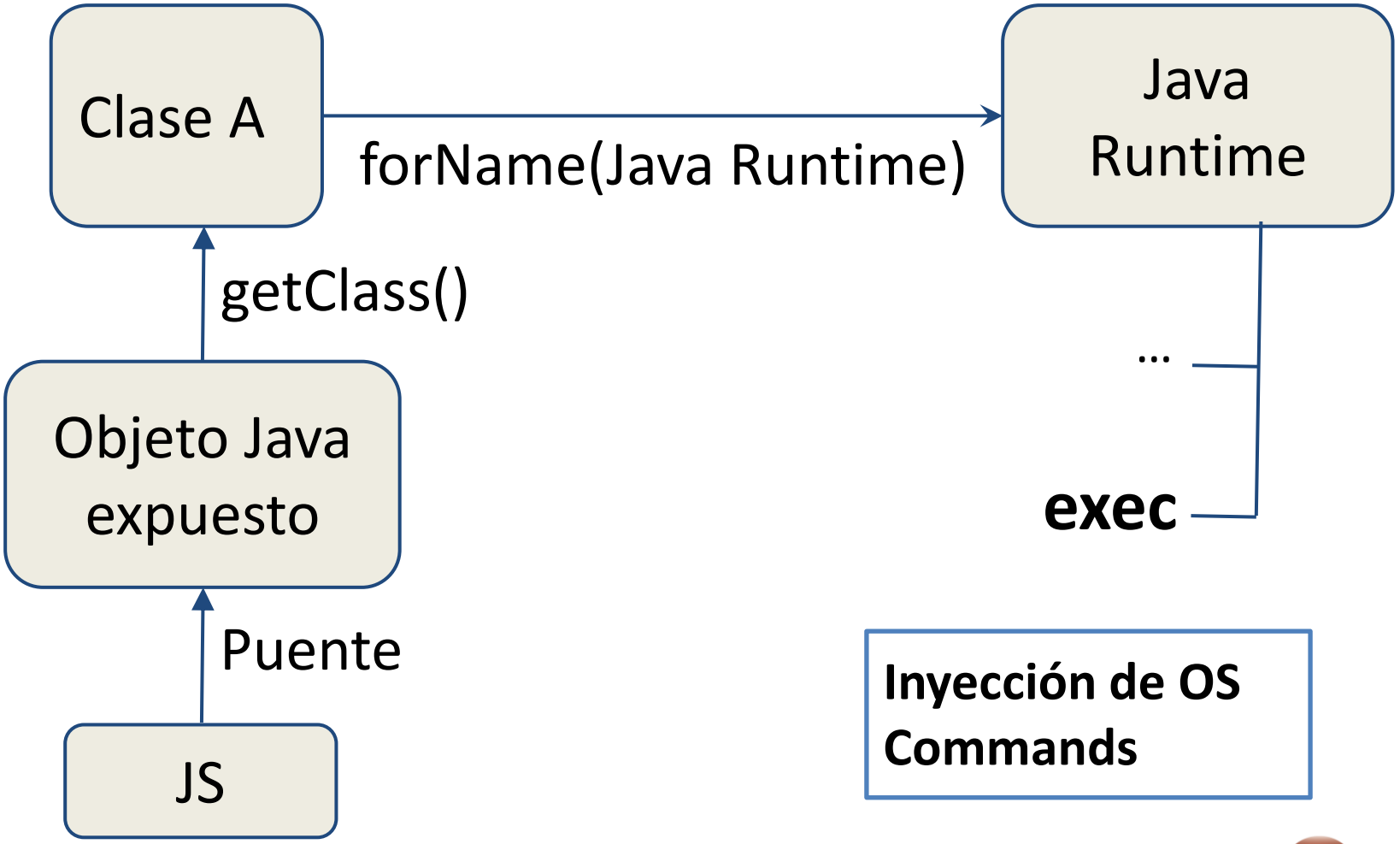
WebView + addJavascriptInterface



addJavascriptInterface exploit – Java Reflection



addJavascriptInterface exploit – OS commands



Limitaciones del exploit – Java Reflection

- Desde Javascript, sólo se pueden **invocar métodos Java** que tengan como argumento tipos nativos o String

- Por lo anterior:
  - No se pueden instanciar objetos
  - No se puede **interactuar con otras aplicaciones** del ecosistema Android

Limitaciones del exploit – OS Commands

- Hay un sand-boxing fuerte a nivel del sistema operativo
  - Permisos RWX en el sistema de archivos
- No se pueden instalar nuevas aplicaciones desde el usuario de la aplicación vulnerada
- ¿Cómo interactuar con otras aplicaciones del ecosistema Android?

Limitaciones del exploit – OS Commands

- Es posible levantar una segunda Dalvik VM pero no va a tener los permisos de la aplicación vulnerada: UID + PID

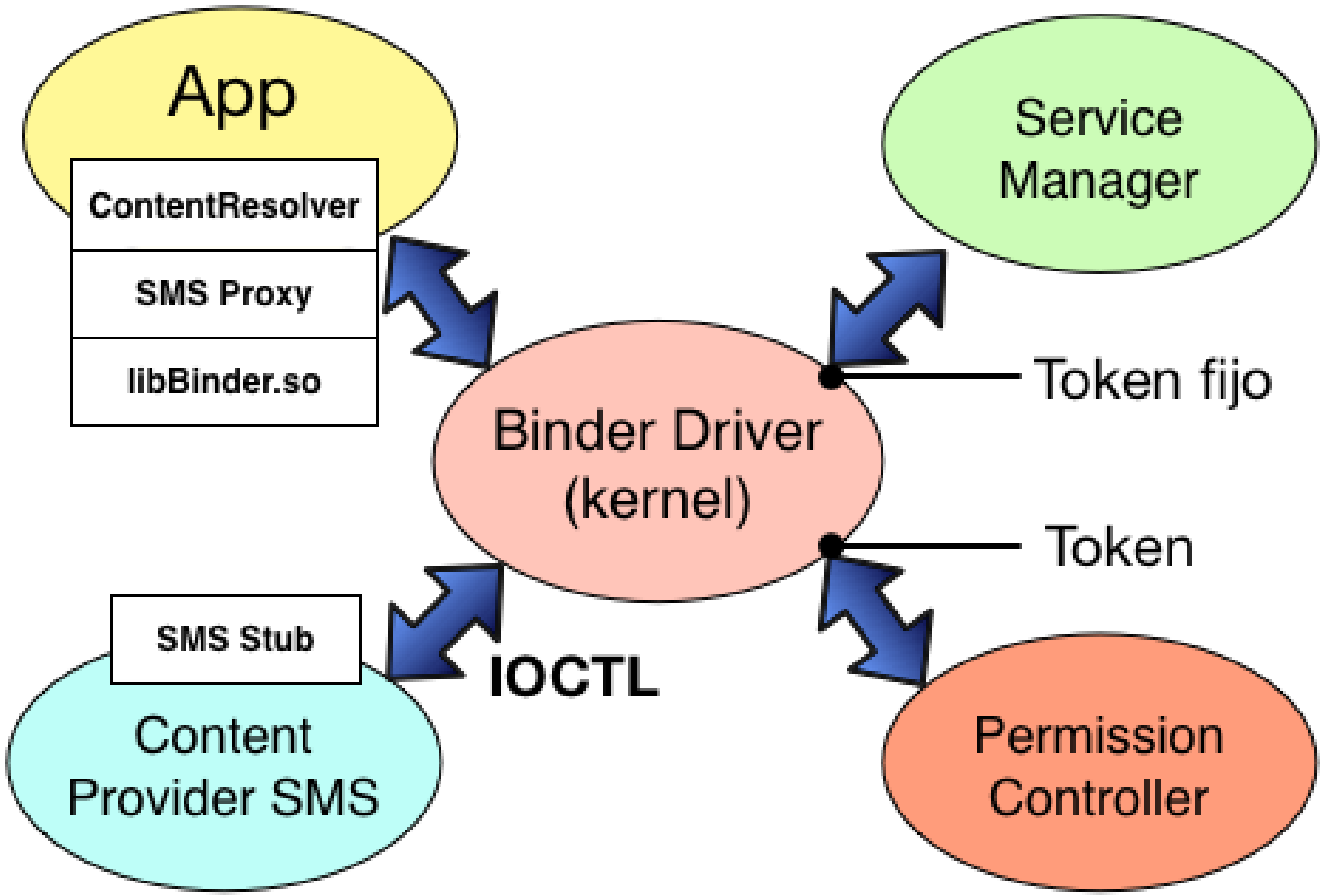
process_id_500 user_id_1234 app_vulnerable

process_id_432 user_id_1234 app_maliciosa

Comunicación en Android

- **Context.ContentResolver**

.query(URI(content://SMS))

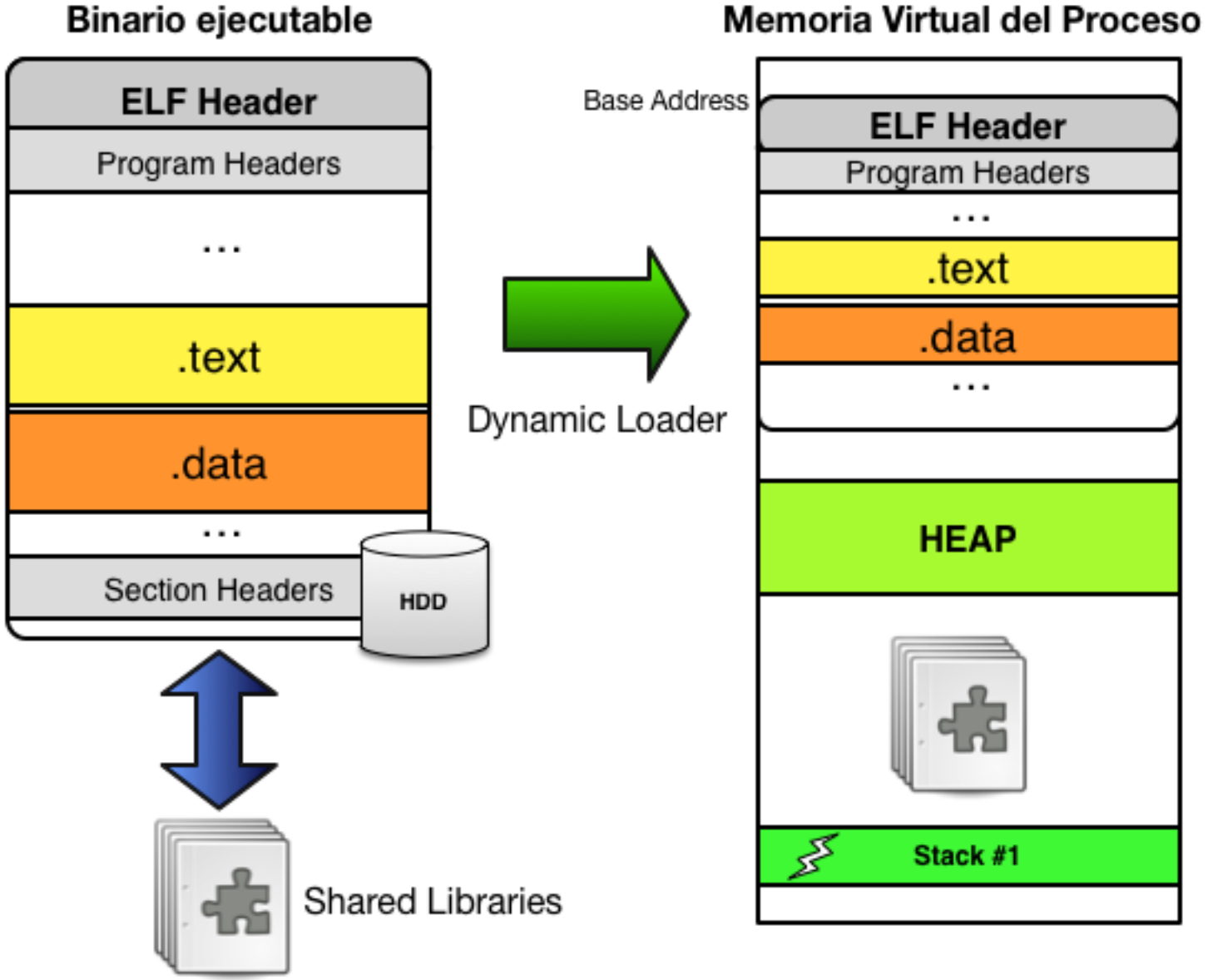


Objetivo

Usar los **permisos de la aplicación vulnerada** para **interactuar** con otras aplicaciones del ecosistema Android.

Inyección en Dalvik VM

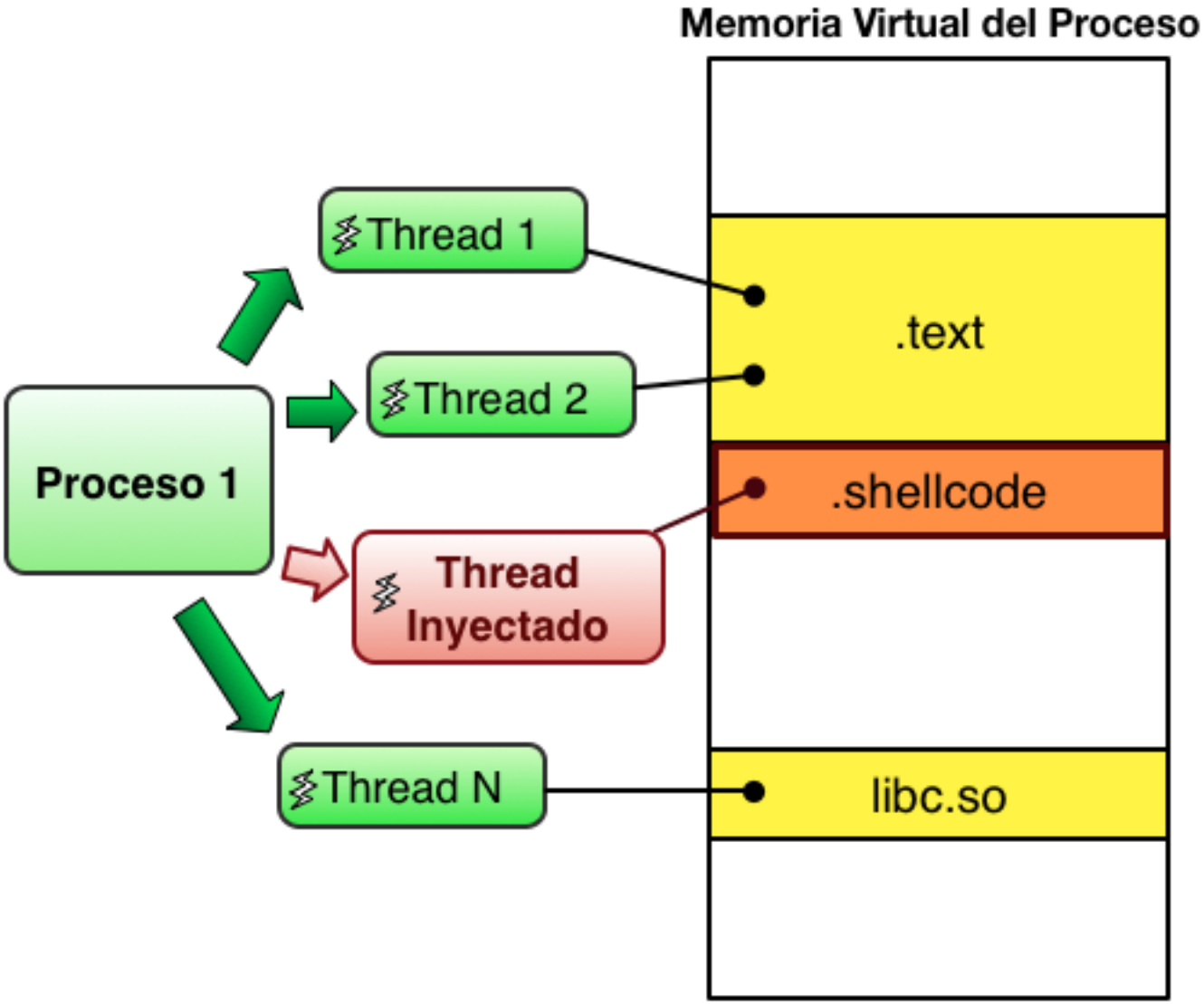
Ejecutables en disco y memoria



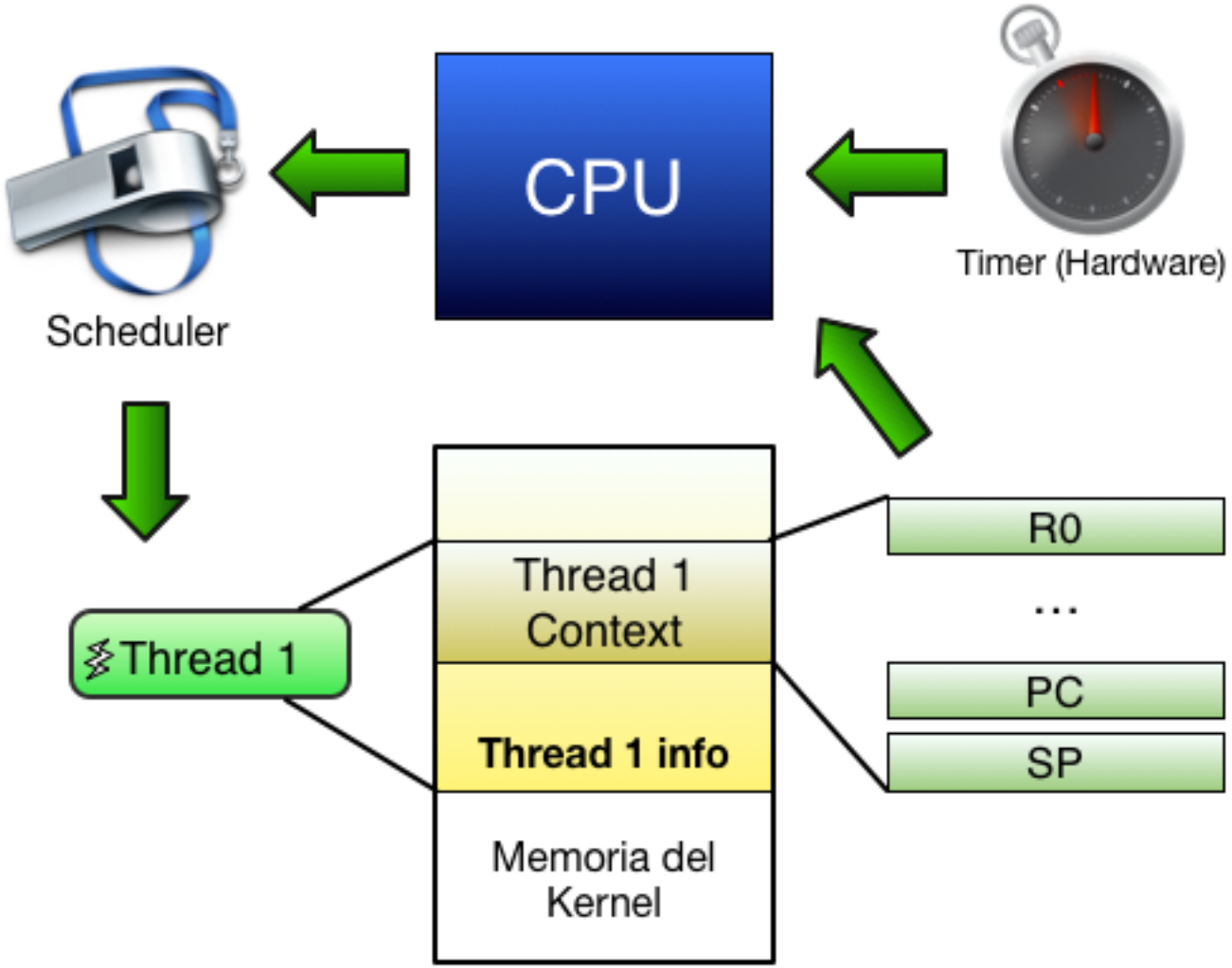
Mapa de memoria: cat /proc/<PID>/maps

- **00400000-0040b000 r-xp /bin/app**
- 00c9a000-00cbb000 rw-p [heap]
- 7fcada45a000-7fcada617000 r-xp /lib/libc-2.17.so
- 7fff9acc0000-7fff9ace1000 rw-p [stack]

¿Qué entendemos por inyección?



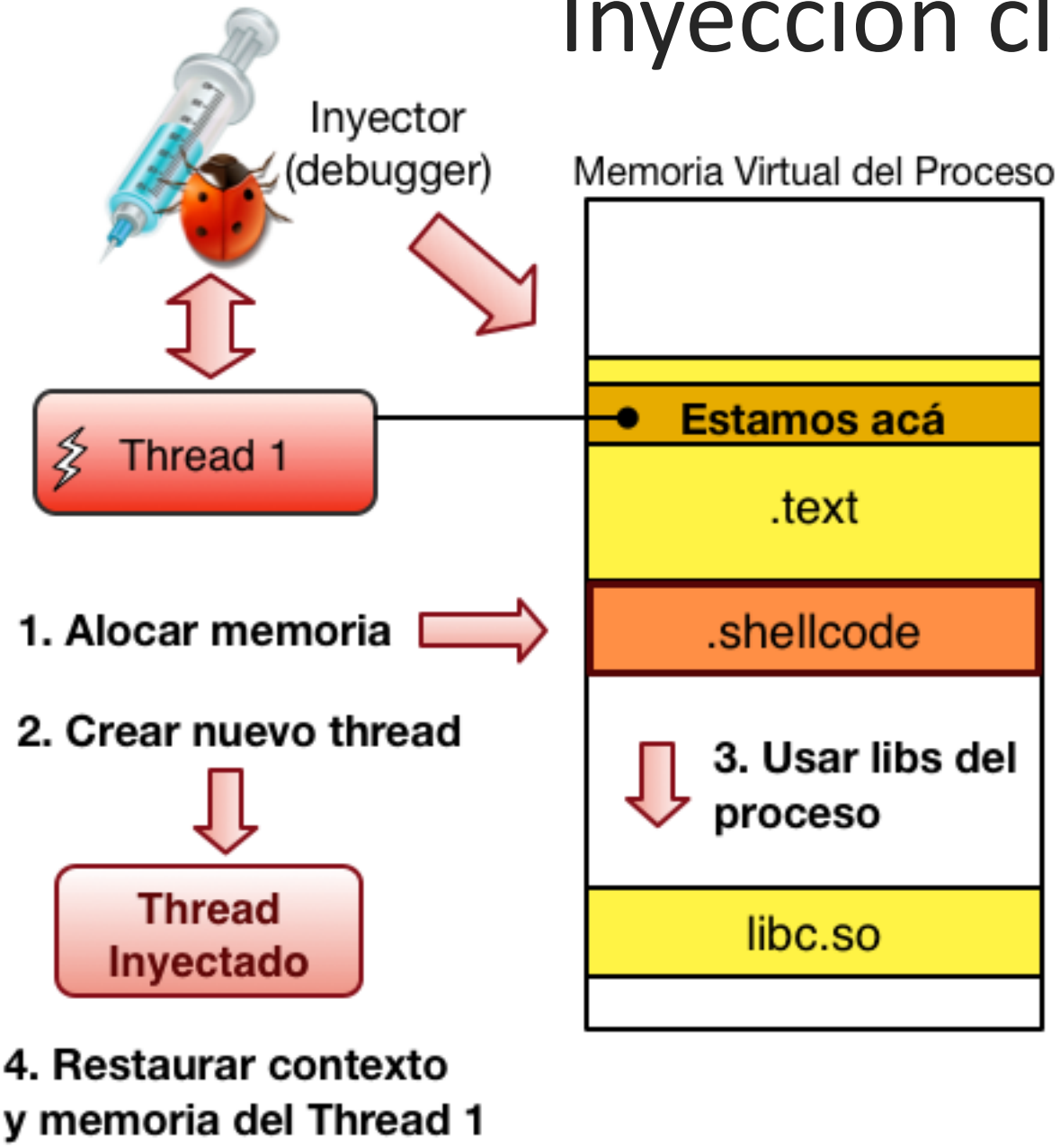
Cambio de contexto



Ptrace - Debugging

- Attacharse a un thread de un proceso
- Leer memoria
- Escribir memoria
- Modificar contexto de un thread (leer y escribir)
- Continuar proceso y handlear señales

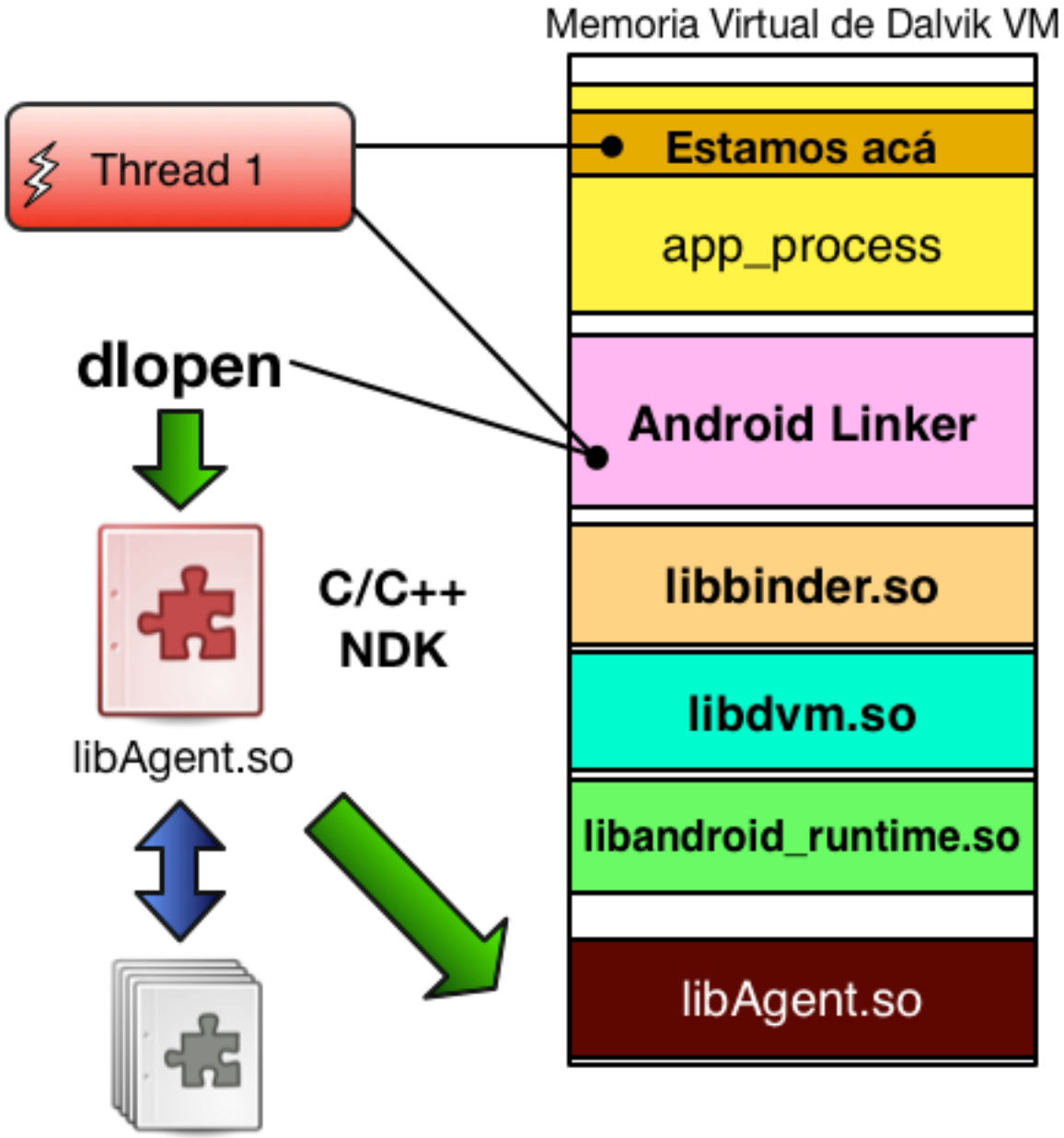
Inyección clásica



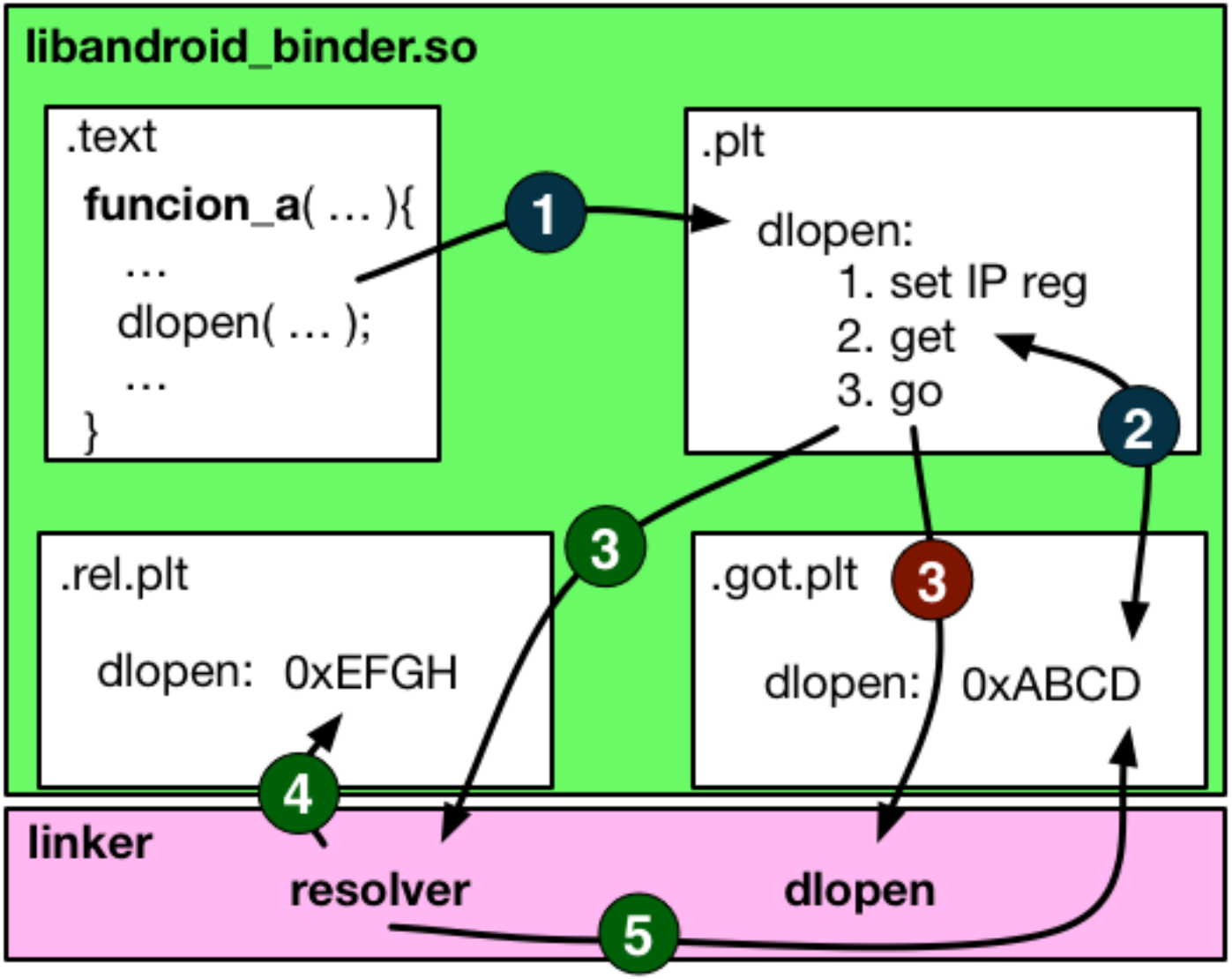
Problemas de la inyección clásica

- Compilar shellcode
- Alocar memoria
- Crear thread
- Resolver símbolos
  - Para usar funciones de otras libs

Inyección por dlopen



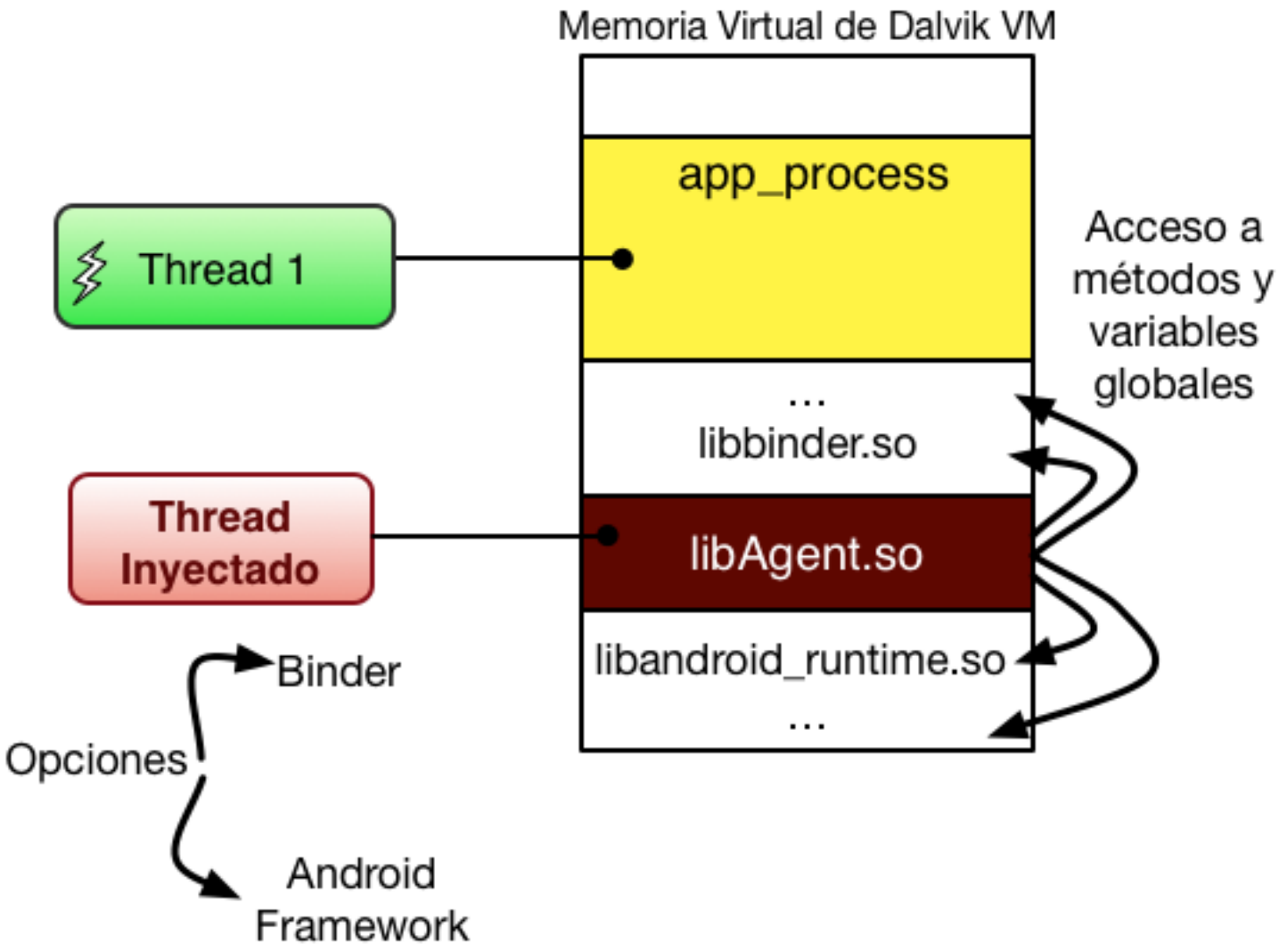
Encontrar dlopen



libAgent.so (C/C++, NDK)


```
#include <stdio.h>
#include <binder.h>
#include <jni.h>

__attribute__((constructor)) init_agent(){
    pthread_create(agent_main_loop);
    raise(SIGINT);
}
void agent_main_loop(){
    sleep(5); // restauración del thread secuestrado
    …
}
```

Después de la inyección

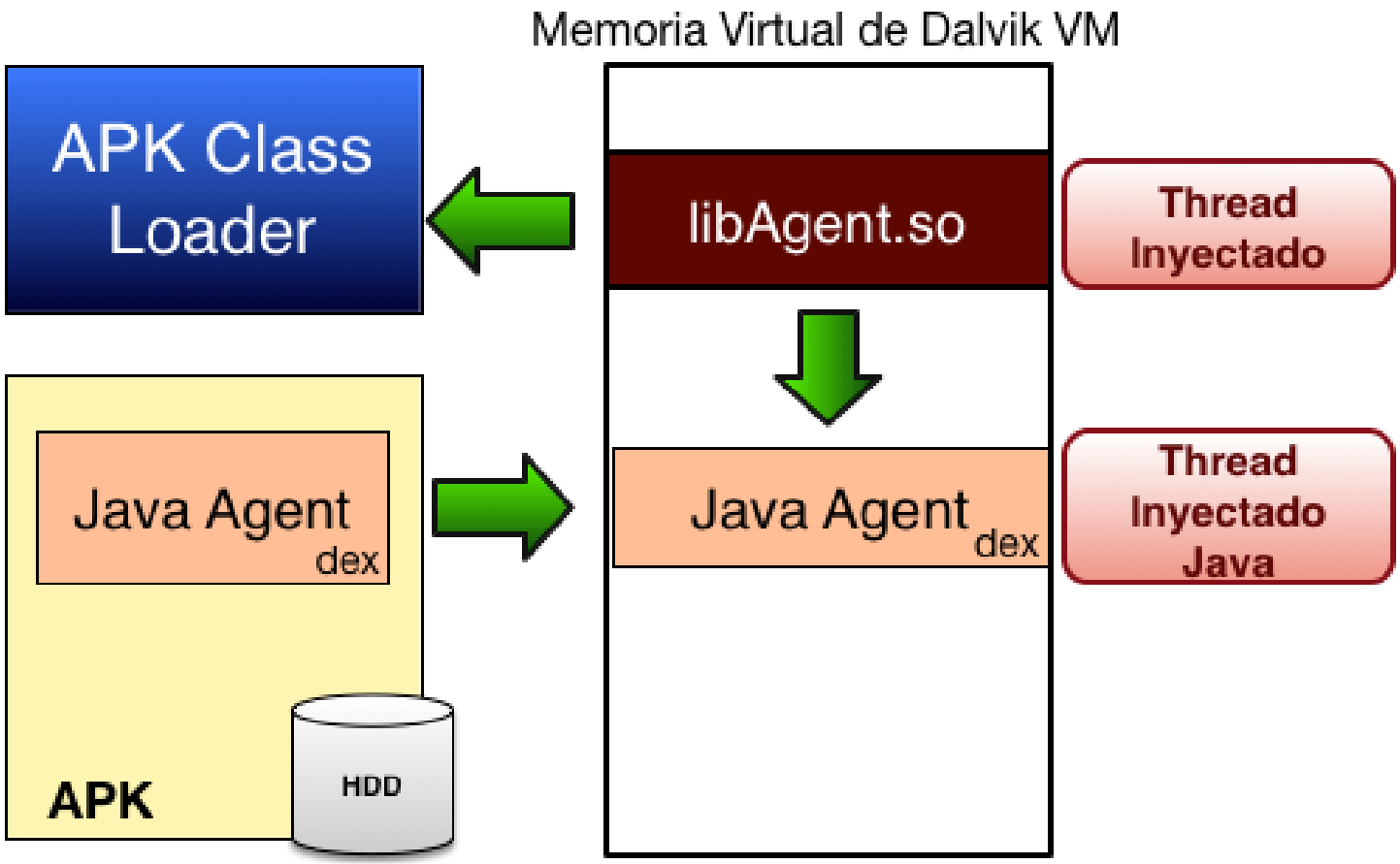


libAgent.so (C/C++, NDK)

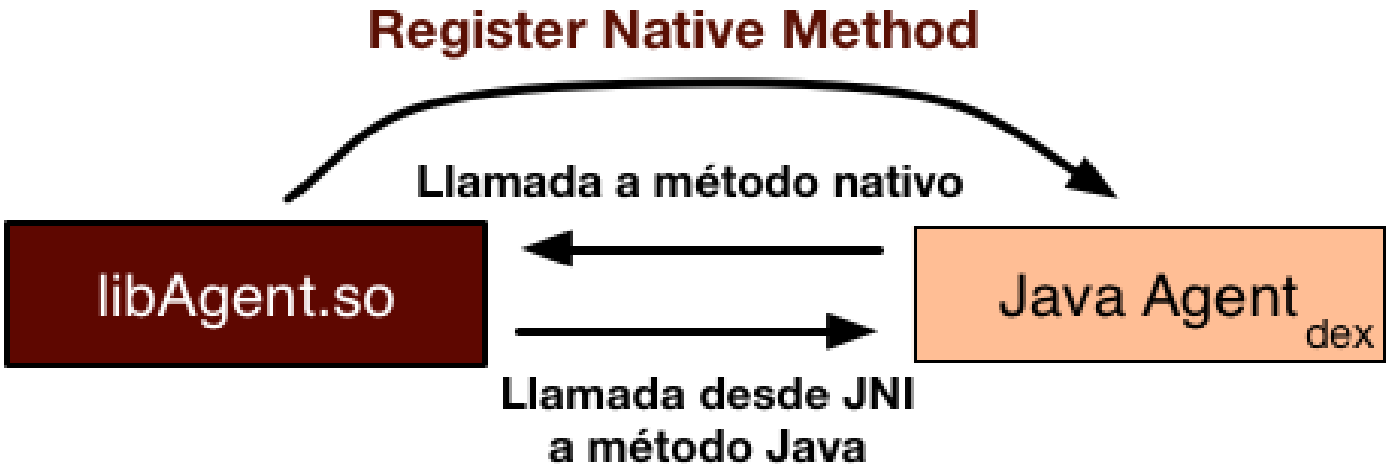

```
#include <jni.h>

JavaVM* java_vm = AndroidRuntime::getJavaVM();
java_vm->AttachCurrentThread(...);
java_vm->GetEnv(...);
```

- Instanciar objetos Java
- Invocar métodos de objetos Java
- Invocar métodos estáticos de clases Java
- Acceder a variables estáticas de clases Java

APK Load – Java Agent

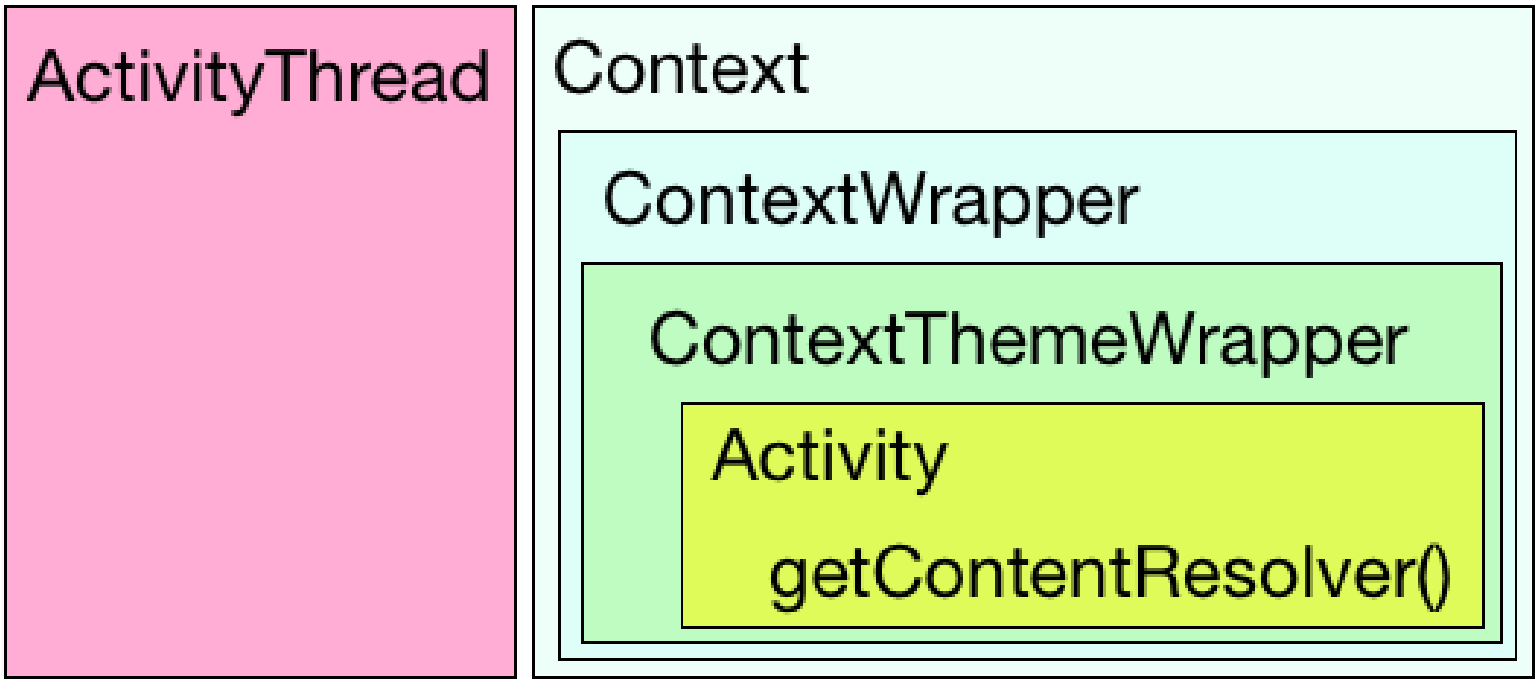


libAgent – Java Agent



- Desde libAgent puedo acceder a **todas las clases Java cargadas** (ver métodos, atributos, instanciar)
- Desde Java Agent solo las de la **SDK**
- Pasar referencias a objetos desde **libAgent** a **Java Agent** (ej. Activity Thread).

Context



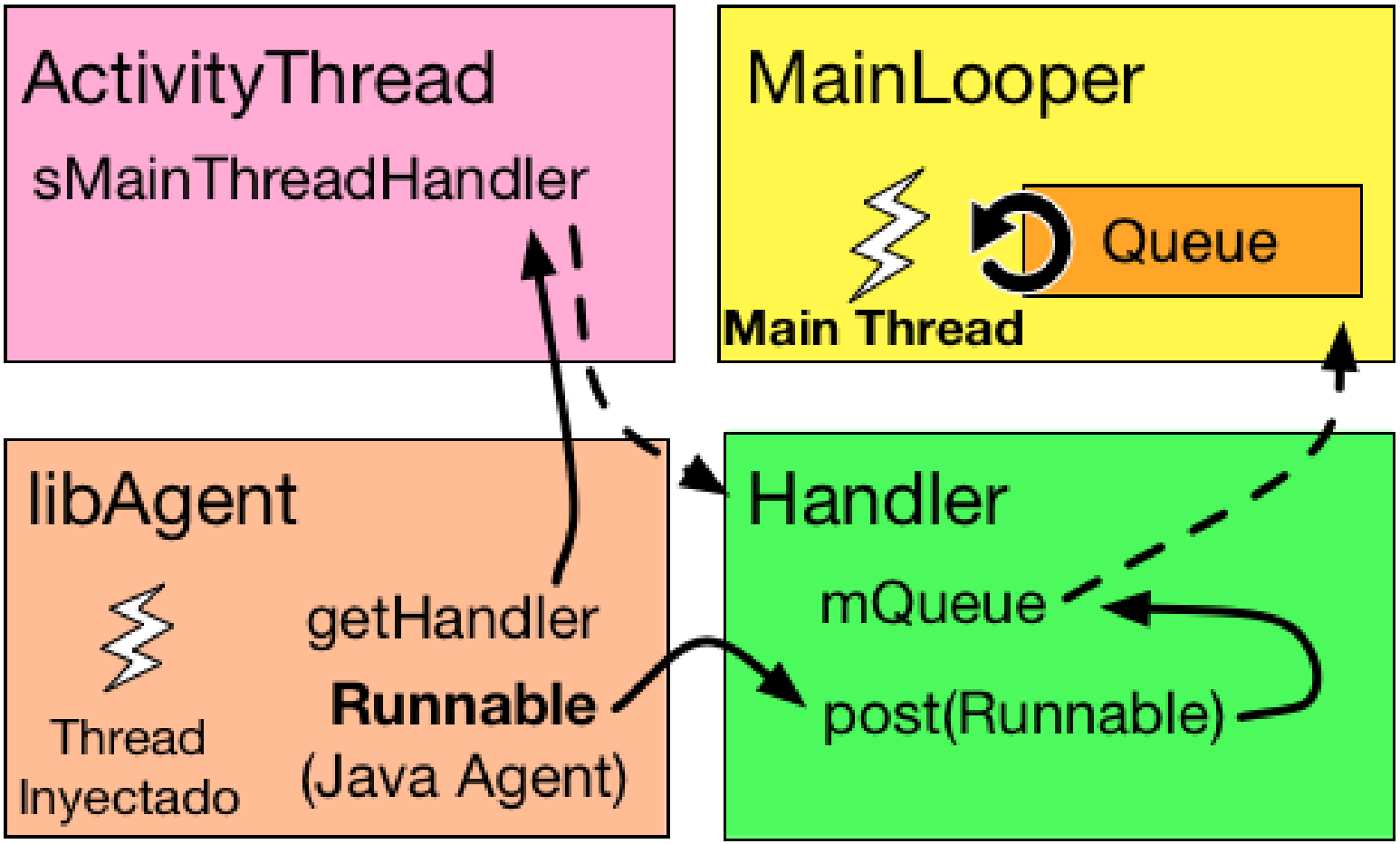
- En **Java Agent**, no podemos heredar de **Activity**: **¿Quién nos instancia?**

Activity Thread

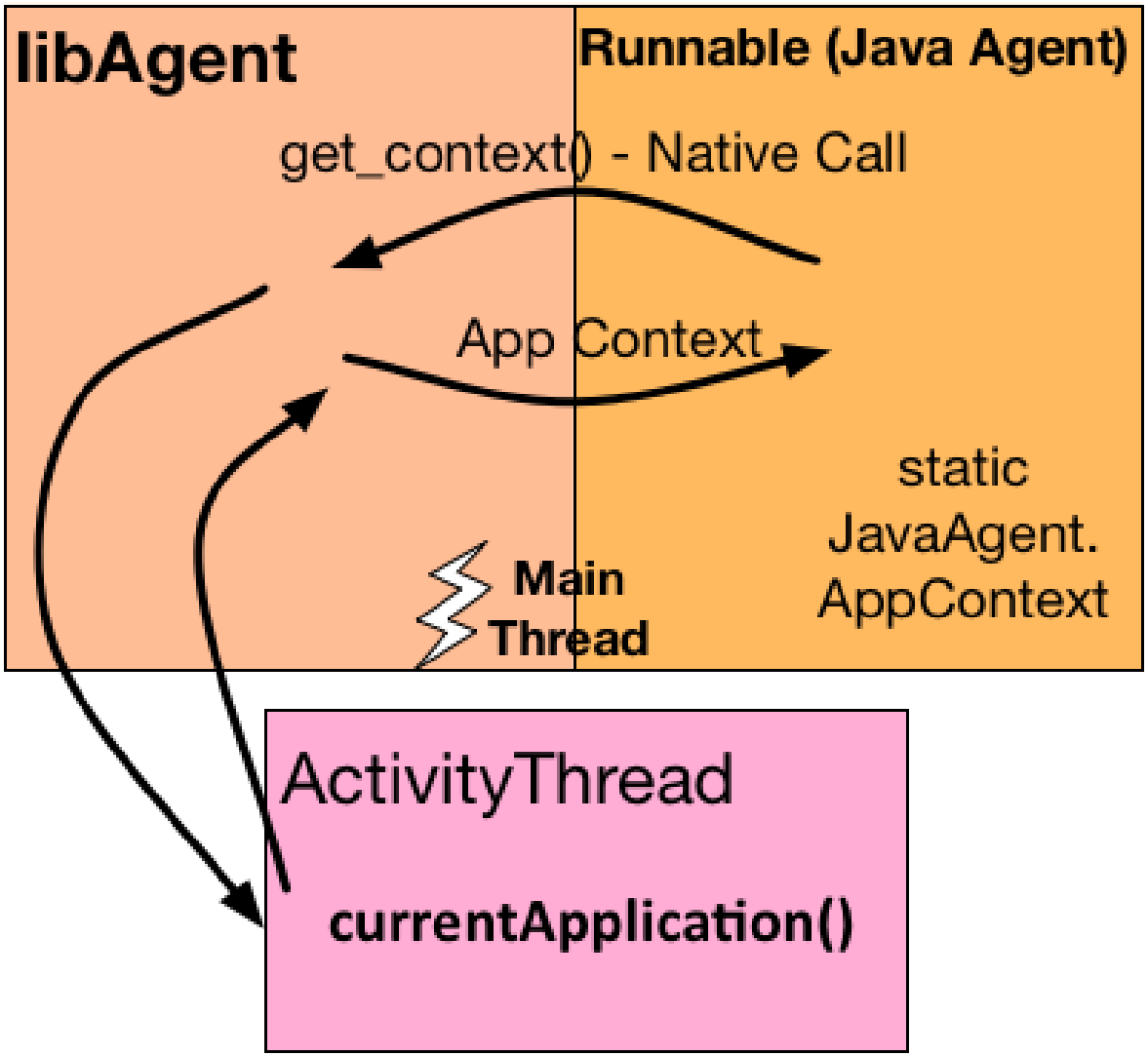
```
Class ActivityThread(){

    static Handler sMainThreadHandler


    public static Application currentApplication()

    public static void main(String[] args)

}
```

¿ NULL ?

Ejecución en el Thread Principal



Runnable



Java Agent - Runnable

```
Class JavaAgent{

    public static Context app_context;
    public native Context get_context();

    Class JavaAgentRunnable implements Runnable{

        public void run(){
            JavaAgent.app_context = get_context();
        }
    }
}
```

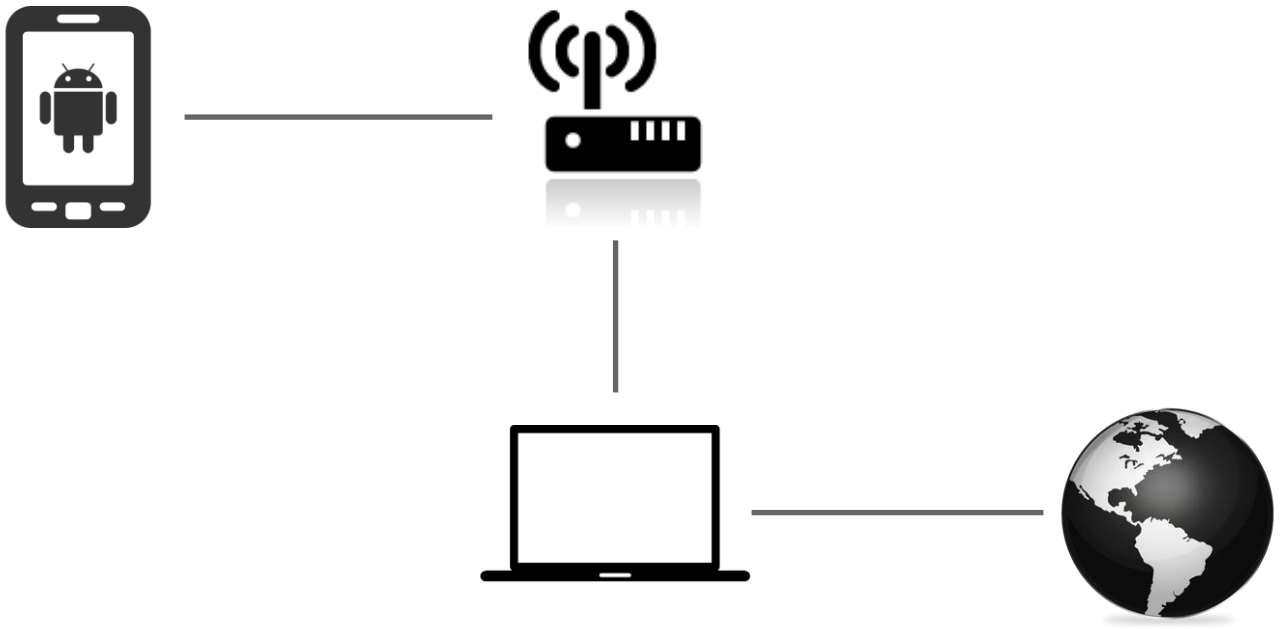
Java Agent – Main loop

Thread Inyectado Java

```
Class JavaAgent{

    public static Context app_context;
    public native Context get_context();

    public void main_loop(){

        JavaAgent.app_context.getContentResolver().
            query(…)

    }
}
```

Demo

Ambiente de ataque



Conclusiones

Conclusiones

- Si podemos debuggear, el PID como parte de la autenticación de una app puede ser *bypasseado*.
- APKs buildeados durante el desarrollo pueden terminar en producción (“Skip packaging and dexing…”).
- Una vulnerabilidad (ejecución sandboxeada) + una debilidad (privilege-escalation) pueden configurar un vector de ataque exitoso.
- No se limita a *addJavascriptInterface*: aplica para cualquier OS command injection.
- Target posible: apps a medida (ej. ambiente *enterprise*).

Trabajo futuro

- **Sniffear** comunicaciones a través de Binder de la aplicación vulnerada
- **Instrumentar** dinámicamente libs cargadas en la memoria del proceso (ej. librerías criptográficas y libdvm.so)
- **Fuzzear** driver de Binder

Trabajos relacionados y referencias

- WebView addJavascriptInterface Remote Code Execution - MWR Labs https://labs.mwrinfosecurity.com/blog/2013/09/24/webview-addjavascriptinterface-remote-code-execution/
- Putting JavaScript Bridges into (Android) Context - MWR Labs https://labs.mwrinfosecurity.com/blog/2014/06/12/putting-javascript-bridges-into-android-context/
- Introduction to Dynamic Dalvik Instrumentation - Collin Mulliner http://www.mulliner.org/android/feed/mulliner_ddi_summercon2013.pdf
- Android Binder – Android Interprocess Communication - Thorsten Schreiber http://www.nds.rub.de/media/attachments/files/2012/03/binder.pdf

¿Preguntas?
Gracias.