CONFERENCE

ZERONIGHTS 2014

NOVEMBER 13-14

Steroids For Your App Security Assessment

Marco Grassi
Mobile Security Researcher
VIAFORENSICS

➜ ~ whoami

- R&D Team Member @ viaForensics
- I work mainly on Android/iOS

The Problem

- We want to trace the code in mobile applications that we are analyzing
- We want to modify some behaviours and inputs that are out of our control
- We want to do it in the less time consuming way and as stealthy as possible (from the apps prospective).

Today we will focus on **Android** and **iOS**.

Windows Phone is gaining some traction. The concepts potentially apply to this platform as well, but we are very distant from the iOS/Android support and tools

Our Guide Principles to solve those Problems

- **Simplicity:** We want to keep our design as simple as possible and as open as possible to extensions in the future.
- **Reuse:** We want to use as much code as possible between different assessments and different platforms if possible.
- **Stealthiness:** We don’t want to be easily detected by the application under analysis.

How to solve our problems?

Why solving our problems statically will break our guide principles (Simplicity, Reuse, Stealthiness)?

We can potentially instrument applications directly by patching their code (or OS or kernel). (**APIMonitor** and others)

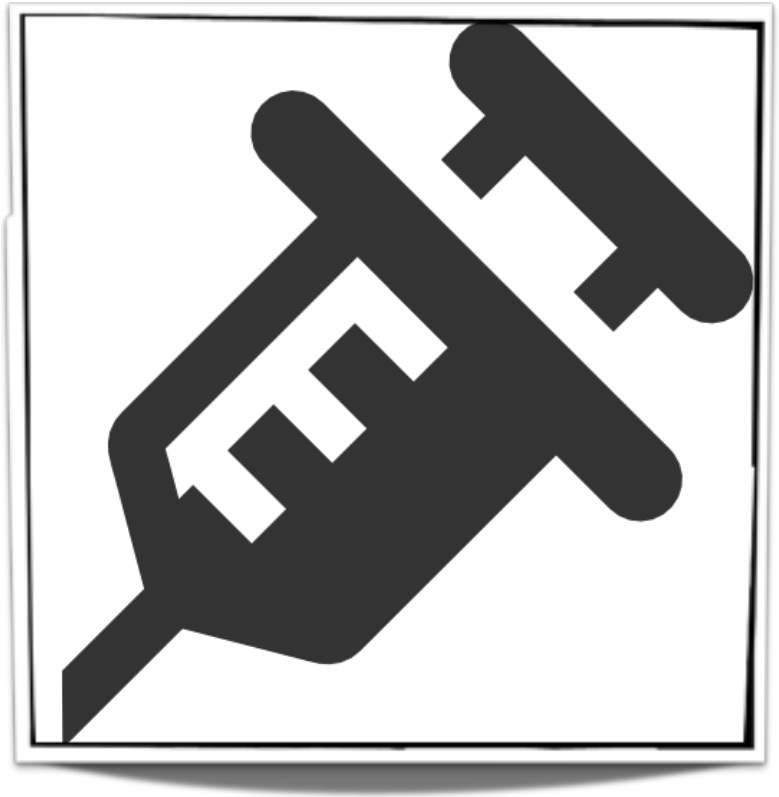
- This can be difficult in iOS applications or native code on Android ~~**(Simplicity)**~~
- Where it's feasible, like on Android Dalvik code for example, it will require instrumenting every single app. ~~**(Reuse)**~~
- It will trigger even simple tampering detection. ~~**(Stealthiness)**~~

For the reasons that we mentioned in the previous slides, we will experiment and try to solve those problems dynamically, by injecting at runtime our code, with various solutions and leveraging different frameworks and methodologies.

- **Simplicity**: We will write our injected code in high level languages
- **Reuse**: We can reuse both the tracing infrastructure that we make, that will be app independent, and also lot of the tracing code, for example if you hook into the “open” function into libc, you can reuse it to track file opening in all the applications
- **Stealthiness**: We don’t alter the application, and we have a more privileged access to the OS, so we can hide more easily.

Why the presentation is called Steroids for apps?

A “steroid” is just some custom code/ patch that we will “inject” in the system or process under analysis at runtime, leveraging various frameworks.
This additional code will be used to enhance our visibility, or to change the behaviours and inputs of the applications/system.



Malware Bazaar

In the talk we will make very small digressions to see how the malware is adopting those or similar techniques.

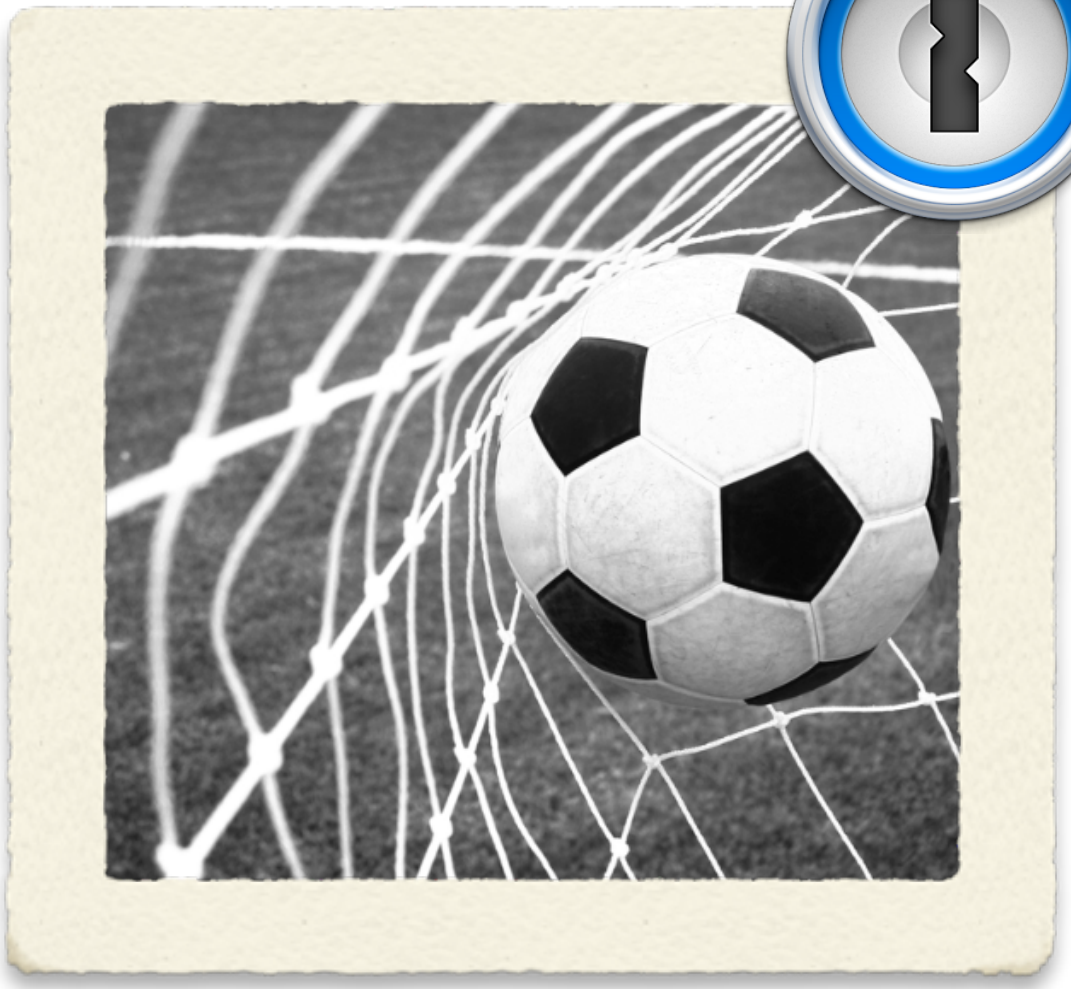
The slides related to **malware** will be marked with the bazaar **symbol on the left**.

The Test App

1Password
Password Manager

1Password (Android): Our Easy Goals

- Hide root on Android
- Hook into their cryptography code to improve our understanding.



What we will hook

```
public static boolean isDeviceRooted() {
    boolean v0 = (Root.checkRootMethod1())
                 ? true : false;
    return v0;
}
```

Trivial, we will patch to return always false

```
public static EncrKeyRec createEncrKeyRec(String masterPwd) throws E
    EncryptionMgr.showLogMsg("create encrKeyRec:  masterPwd=" + mast
    return EncrypStorageMgr.getEncrKeyRecProperties(masterPwd, new Encr
}
```

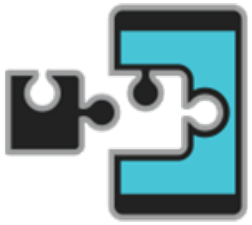
```
private static void showLogMsg(String logMsg) {
    MyPBKDF2Engine.showLog(false);
}
```

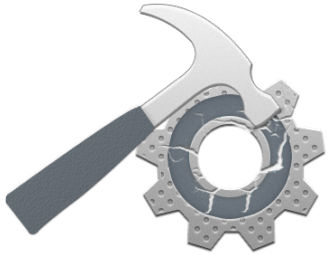
We will insert a hook here to print the logMsg

What is already available? What can we leverage?

Xposed Framework http://repo.xposed.info/	
Cydia Substrate http://www.cydiasubstrate.com/	
Cycript http://www.cycript.org/	
ddi / adbi https://github.com/crmulliner/	
Frida http://www.frida.re/	

Xposed Framework and Cydia Substrate





"Xposed is a framework for modules that can change the behavior of the system and apps without touching any APKs." - rovo89

"Modify behavior without patches or ROMs. Substrate makes it easy to modify software, even without the source code, and in a way that allows users to easily choose which changes they want. - saurik

Xposed Framework and Cydia Substrate

- They work similarly
- Essentially you hook a method, and you write some additional code to execute when that method is called

WorkFlow

1. Write your code and filtering metadata (for what processes to hook)
2. Deploy your code on the device
3. Reboot or Respring/restart zygote

Using Xposed to patch 1Password

More info at: https://github.com/rovo89/XposedBridge/wiki/Development-tutorial

```
public class OnePassword implements IXposedHookLoadPackage {

    @Override
    public void handleLoadPackage(LoadPackageParam lpparam) throws Throwable {
        if (!lpparam.packageName.equals("com.agilebits.onepassword")) {
            // We want to hook just 1Password and skip everything else
            return;
        }

        XposedHelpers.findAndHookMethod(
                // Class to hook
                "com.agilebits.onepassword.diagnostics.Root",
                lpparam.classLoader,
                // Method to hook
                "isDeviceRooted",
                // Hook
                new XC_MethodHook() {
                    @Override
                    protected void beforeHookedMethod(MethodHookParam param)
                            throws Throwable {
                        // We set the result to override the implementation
                        // And that the original method get called
                        param.setResult(Boolean.FALSE);
                    }
                }
        );
    }

}
```

```
public class OnePasswordTwo implements IXposedHookLoadPackage {

    @Override
    public void handleLoadPackage(LoadPackageParam lpparam) throws Throwable {
        if (!lpparam.packageName.equals("com.agilebits.onepassword")) {
            // We want to hook just 1Password and skip everything else
            return;
        }
        // Class to hook
        Class EncryptionMgr =
                XposedHelpers.findClass("com.agilebits.onepassword.mgr.EncryptionMgr",
                        lpparam.classLoader);
        // Method to hook
        Method showLogMsg =
                XposedHelpers.findMethodExact(EncryptionMgr,
                        "showLogMsg", String.class);
        // Hook
        XposedBridge.hookMethod(showLogMsg,
                new XC_MethodHook() {
                    @Override
                    protected void afterHookedMethod(MethodHookParam param)
                            throws Throwable {
                        // We log the first parameter, the msg
                        Log.d("EncryptionMgr", (String) param.args[0]);
                    }
```

Application	Tag	Text
	EncryptionMgr	Validating password:xposedmod
	EncryptionMgr	Done decrypt
	EncryptionMgr	validatePassword OK for:489 ms

Android Malware leveraging Xposed

- Hiding features, steal passwords from EditText by hooking methods
- http://blog.avlyun.com/1361.html (in Chinese)

```
otected void afterHookedMethod(XC_MethodHook$MethodHookParam arg6) {
  Object v0 = arg6.getResult();
  if(v0 != null && ((List)v0).size() != 0 && i.a() != null) {
      int v2;
      for(v2 = ((List)v0).size() - 1; v2 >= 0; --v2) {
          Object v1 = ((List)v0).get(v2);
          if(!i.a()[0] && (((ApplicationInfo)v1).packageName.equals("com.android.os.backup")
                  {
              ((List)v0).remove(v2);
          }

          if(!i.a()[1] && ((((ApplicationInfo)v1).packageName.equals("eu.chainfire.supersu")
                  (((ApplicationInfo)v1).packageName.equals("com.koushikdutta.superuser")) |
                  (((ApplicationInfo)v1).packageName.equals("com.qihoo.permroot")))) {
              ((List)v0).remove(v2);
          }

          if(!i.a()[2] && (((ApplicationInfo)v1).packageName.equals("de.robv.android.xposed.
                  ))) {
              ((List)v0).remove(v2);
          }
      }
  }
```

```
ArrayList v1 = new ArrayList();
Iterator v2 = arg8.iterator();
while(v2.hasNext()) {
    Object v0 = v2.next();
    FormField v3 = new FormField();
    v3.a(Integer.valueOf(0));
    v3.b(Integer.valueOf(0));
    v3.b(0);
    if((v0 instanceof EditText)) {
        v3.a(1);
        if(((EditText)v0).getInputType() == 129) {
            v3.b(1);
        }

String v4 = ((EditText)v0).getText().toString();
String v0_1 = ((EditText)v0).getHint().toString()
```

iOS Spyware leveraging Substrate

Commercial Spyware (iOS/Android and other platforms)

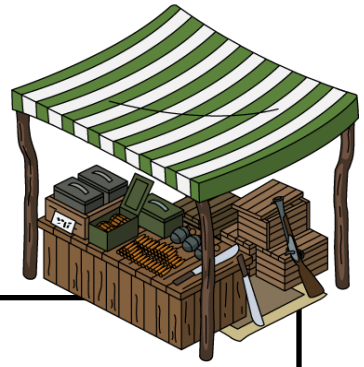
- It uses this framework to implement trivial “rootkit” functionalities on iOS, to hide the Cydia.app from a Jailbroken device to appear unjailbroken with a quick look.
- You can type **“4433*29342”** in the SpringBoard search bar, to hide or display the Cydia icon.
- The code in SpringBoard is hooked. When that magic code is detected Cydia is hidden or displayed.
- This hooking mechanism is deployed as a MobileSubstrate extension

Setting the hooks

```
        ; CODE XREF: _HideCydiaHookInitialize+15A↑j
R0, #(aSbsearchcontroller - 0x1EC0) ; "SBSearchController"
R0, PC  ; "SBSearchController"
_objc_getClass
R1, #(__ZL49$SBSearchController$searchBarSearchButtonClicked$P18SBSearchControllerP13objc_selectorP11UISearchBar+1 - 0x1ECE)
R1, PC ; $SBSearchController$searchBarSearchButtonClicked$(SBSearchController *,objc_selector *,UISearchBar *)
R2, #(__ZL49_SBSearchController$searchBarSearchButtonClicked$ - 0x1ED8) ; _SBSearchController$searchBarSearchButtonClicked$
R2, PC  ; _SBSearchController$searchBarSearchButtonClicked$
R3, #(paSearchbarsearchbuttonclicked - 0x1EE2)
R3, PC ; paSearchbarsearchbuttonclicked
R0, [SP,#0x70+var_18]
R0, [SP,#0x70+var_18]
R3, [R3] ; "searchBarSearchButtonClicked:"
R1, [SP,#0x70+var_68]
R1, R3
R3, [SP,#0x70+var_68]
R2, [SP,#0x70+var_6C]
R2, R3
R3, [SP,#0x70+var_6C]
_MSHookMessageEx
```

hooks when in springboard the search button is clicked when you are using the search functionality, to check if the string match the magic value and perform its actions

Actions performed



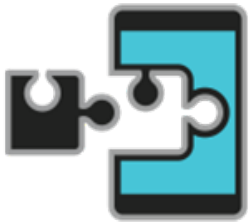
```
v21 = _objc_msgSend(a1, "isEqualToString:", CFSTR("4433*29342"));
if ( v21 )
{
  v1 = isIconHidden((int)CFSTR("/Applications/Cydia.app/Info.plist"));
  v20 = v1;
  _ZF = v1 == 0;
  v3 = 0;
  if ( !_ZF )
    v3 = 1;
  v19 = toggleIcon((int)CFSTR("/Applications/Cydia.app/Info.plist"), (v3 ^ 1) & 1);
  if ( v19 )
  {
    system("killall Cydia");
    system("killall backboardd");
    exit(1);
  }
```

Magic code to hide cydia from the search bar

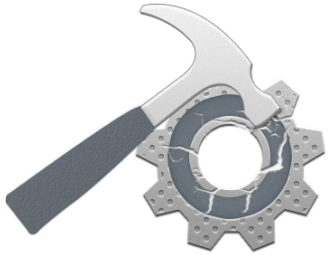
Edit the Cydia Info.plist

Refresh the UI

Summary

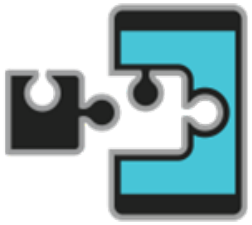


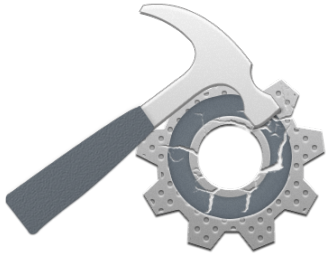
Xposed is open source and actively maintained. It's Android only and for now it supports only hooking Dalvik methods. There are a lot of modules developed on it to checkout.
It's probably the best framework for Android to invest in today, for it's active support and community.



Cydia Substrate supports iOS/ Android and it can hook pretty much everything, ObjC, Dalvik and native methods. On iOS the tweak community is very active.
Unfortunately it's closed source and so it can be risky to invest time developing tools on top of it, especially on Android.

Stuff to checkout





- https://github.com/Fuzion24/JustTrustMe (SSL Pinning kill switch)
- https://github.com/M66B/XPrivacy (manage sensitive data returned to the applications via Android APIs)

- iSECPartners Introspy Projects: - http://isecpartners.github.io/Introspy-Android/ - http://isecpartners.github.io/Introspy-iOS/
- SSL kill switch for iOS https://github.com/iSECPartners/ios-ssl-kill-switch

These first 2 frameworks are very good, but they are oriented to make more persistent changes that the ones we need for code tracing or small patches to a single application and the development cycle is quite slow, with **development**, **deployment** and **reboot**.

Can we do a little bit better, still using those frameworks?

MY SUGGESTION IS

CREATE YOUR OWN ABSTRACTION ON TOP OF THOSE FRAMEWORKS

- Define targets, hooks and actions generically
- Place them at runtime using reflection with your own module
- In this way you can define them in a structured file and restart zygote, and you will reuse the code!

My (first try) abstraction layer as example

Kankuro *- Puppet Master from the Sand Village (Naruto)*

He fights instrumenting bodies and puppets with chakra strings.

Very simple, Android only, can leverage Xposed or Cydia Substrate.
It will read the hook to deploy from a JSON file that we will push on the device.

How to Use and Example Hooks

define hooks and action in a
tracking_hooks.json
$ adb push tracking_hooks.json /data/local/tmp/
Let the hooking framework read the
hooks
from other processes
$ adb shell chmod 644 /data/local/tmp/tracking_hooks.json
Enable your module/restart zygote

The last one is a PITA!

```
{
   "com.spotify.music":
      [
         {
            "classCanonicalName": "android.app.Activity",
            "methodName": "onCreate",
            "hookType": "method",
            "hookAction": "logcat"
         }
      ],
   "com.agilebits.onepassword":
      [
         {
            "classCanonicalName": "com.agilebits.onepassword.mgr.EncryptionMgr",
            "methodName": "showLogMsg",
            "hookType": "method",
            "hookAction": "logcat"
         }
      ]
}
```

Hooks Output

We inserted our “probes” in the app, without having to write additional Java code, saving time. Now we can hook and logs intelligently the methods that we are interested into, tracking the flow of execution and printing parameters, or we can patch methods to return different values.

Tag	Text
MethodTracking	package:com.agilebits.onepassword - type:method - class: com.agileb its.onepassword.mgr.EncryptionMgr - method: showLogMsg - signature: null - args: [Ljava.lang.Object;@421667b0 - instance: null - retva l: null
MethodTracking	arg 0 - Validating password:wrongcode
MethodTracking	package:com.agilebits.onepassword - type:m lass: com.agileb its.onepassword.mgr.EncryptionMgr - method: ogMsg - signature: null - args: [Ljava.lang.Object;@42173430 - instance: null - retva l: null
MethodTracking	arg 0 - Failed pwd validation: pad block corrupted
MethodTracking	package:com.agilebits.onepassword - type:method - class: com.agileb its.onepassword.mgr.EncryptionMgr - method: showLogMsg - signature: null - args: [Ljava.lang.Object;@421b9780 - instance: null - retva l: null
MethodTracking	arg 0 - Validating password:methodtrack
MethodTracking	package:com.agilebits.onepassword - type:method - class: com.agileb its.onepassword.mgr.EncryptionMgr - method: showLogMsg - signature: null - args: [Ljava.lang.Object;@4215ba78 - instance: null - retva l: null
MethodTracking	arg 0 - Done decrypt
MethodTracking	package:com.agilebits.onepassword - type:method - class: com.agileb its.onepassword.mgr.EncryptionMgr - method: showLogMsg - signature: null - args: [Ljava.lang.Object;@4224ced8 - instance: null - retva l: null
MethodTracking	arg 0 - validatePassword OK for:552 ms

Limitations and problems

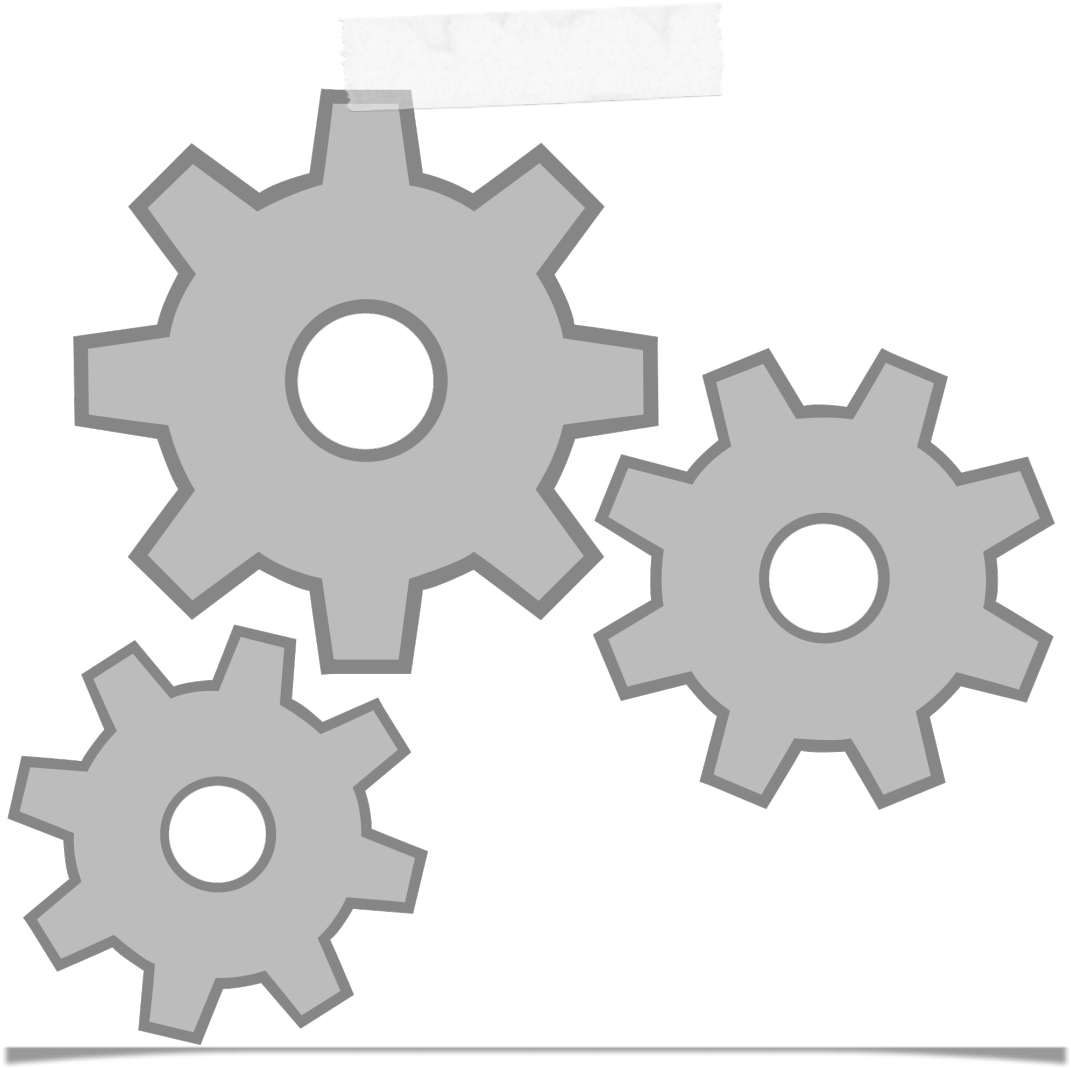
- Already pointed out: we must reboot/restart zygote/springboard if we modify our code
- Our injected code is running in the process context, so it inherits also its capabilities. Often those capabilities are not enough for us. Think of Android as an example: if our target process does not have Internet permission, we are limited in how we can exfiltrate our traces and informations
- We often have to leverage/hack logcat and world read/writable locations, we would like a more robust communication channel, platform independent

"Your own scripts get injected into black box processes to execute custom debugging logic. Hook any function, spy on crypto APIs or trace private application code." - *frida.re*

Seems exactly what we need :)
It's the one I want to push in this talk, because it should receive more attention!

How Does It Work?

- Injects Google V8 Javascript engine, on top of which the Javascript code that we will write run.
- Being inside the same process, our javascript code has full memory access and we can read or modify what we need.
- Frida offers javascript apis for us for hooking, communicating with the PC side of Frida software, and many other things.



My other (ongoing) Experiment…

Sasori - Sasori of the Red Sand (from Naruto)

He built **Kankuro's** fighting puppets because he also fight with puppets and chakra strings.
He transformed himself into a puppet.
He can transmigrate his heart into other puppets.

Based on Frida, support Android and iOS for now.

High Level Structure

Asking for pid list

```
./sasori.py --android -l
```

Can I haz PIDs?

Sure, …
com.agilebits.onepassword
has 18729 …

Injecting the Agent into a PID

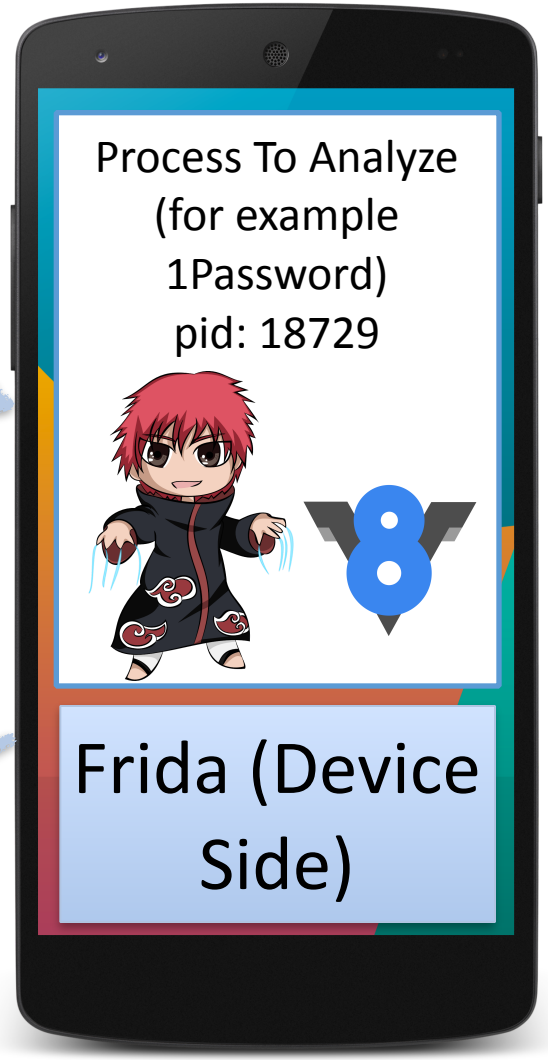
```
./sasori.py --android 18729
```

Inject into pid 18729

The agent is loaded

```
Loaded the agent press ENTER to stop
[*] platform: linux
[*] arch: arm
[*] pointer size: 4
[*] Found Dalvik Runtime!
```

Now we can communicate with the agent and we can modify and inspect the application!



Now we can hook and reimplement any method we want thanks to Frida APIs.

For example, we will hook the showLogMsg method that we already shown before:

```
<-Trace-> method: (com.agilebits.onepassword.mgr.EncryptionMgr.showLogMsg) args:
[Validating password:sasoritest]
<-Trace-> method: (com.agilebits.onepassword.mgr.EncryptionMgr.showLogMsg) args:
[Failed pwd validation: pad block corrupted]
<-Trace-> method: (com.agilebits.onepassword.mgr.EncryptionMgr.showLogMsg) args:
[Validating password:methodtrack]
<-Trace-> method: (com.agilebits.onepassword.mgr.EncryptionMgr.showLogMsg) args: [Done
decrypt]
<-Trace-> method: (com.agilebits.onepassword.mgr.EncryptionMgr.showLogMsg) args:
[validatePassword OK for:533 ms]
```

How Does It works?

A shared library is injected into the target process:

Android

```
[Debug] base: 0x67b02000 path: /system/lib/libwebviewchromium.so
[Debug] base: 0x6aa2a000 path: /system/lib/libcgdrv.so
[Debug] base: 0x6b035000 path: /data/local/tmp/.frida-49[..]/libfrida-agent.so
```

iOS 8.1 64bit

```
[Debug] base: 0x191048000 path: /System/Library/PrivateFrameworks/
IncomingCallFilter.framework/IncomingCallFilter
[Debug] base: 0x18f4dc000 path: /System/Library/PrivateFrameworks/FTAWD.framework/FTAWD
[Debug] base: 0x1053dc000 path: /private/var/root/.frida-a2[..]/libfrida-agent.dylib
```

Current structure

`sasori.py`

Command line interface written in Python that leverages Frida APIs. Print output and communicate with the agent running inside of the process.

`sasori-agent.js`

Agent written in Javascript, running on top of Google V8 JS engine inside our target process. It's completely independent from the client side and can be cross platform.

What improvements we had here?

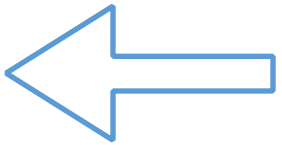
A LOT!

- We don’t have to reboot or restart zygote or respring or even restart the app anymore! **Our development time is dramatically reduced!**
- We have the code running on the device, written in javascript, completely **separated from the “server” logic** and **testable**! (yes if the instrumentation code size become important, we would love to be able to write software tests)
- The agent can be **crossplatform**, it can detect by itself what platform it’s running, and **code reuse** it very good.
- We can hook and modify pretty much everything in userspace, Dalvik, ObjC, native code.

What capabilities we have?

A very good bidirectional, platform independent communication channel with our agent.

```
def _process_message(self,
message, data, process):
…
print ‘[Debug] ‘ + message
```



```
send({'type': 'debug',
'message': 'base: ' +
module.base + ' path: ' +
module.path});
```

What capabilities we have?

```
script.post_message({"type":
"hook", [...]})
```

```
recv([type, ]callback)
…
Dalvik.perform(function () {
        var EncryptionMgr =
Dalvik.use("com.agilebits.one
password.mgr.EncryptionMgr");

EncryptionMgr.showLogMsg.impl
ementation = function (param)
{
                ...
        };
});
```

What capabilities we have?

- We can hook Dalvik, ObjC and native methods, pretty much everything.
- We have full memory access, read memory, write memory, scan memory (very interesting for game cheats :))
- We can know at runtime on what platform we are running, from our agent, for example with `Dalvik.available,` and know on what kind of architecture we are running.
- See http://www.frida.re/docs/javascript-api/ for a better understanding of what you can do.

What can be improved?

In Web Development, we can go with full-stack javascript.
Would it be cool to do it also in our dynamic analysis system, with Frida Node.js APIs ?
The idea already surfaced in Frida's website.
In this way we can potentially build a **full-stack javascript dynamic analysis platform**.

What can be improved?

ART runtime is not yet supported on Android (but it’s not even the other frameworks currently)

It needs some additional research on how to hook into the application before it starts, to be able to place the hooks **before** the app get a chance to execute.

Some bugfixes that will come naturally with more maturity of the Frida Android/iOS side code and contributions/testing.

Android Malware leveraging ptrace

On Linux/Android, Frida leverages ptrace to inject the shared library into the selected process.

Recently an Android Malware was found using ptrace to dynamically inject code in third party processes.

Samples: http://contagiominidump.blogspot.it/2014/10/android-chathook-ptrace.html
Writeup (Chinese) : http://blog.csdn.net/androidsecurity/article/details/27504615

```
v1 = a1;
if ( ptrace(16, a1, 0, 0) < 0 )
{
  perror("ptrace_attach");
  v3 = (int)"INJECT";
  v4 = "ptrace_attach error";
LABEL_5:
  _android_log_print(3, v3, v4);
  return -1;
}
waitpid(v1, 0, 2, v2);
if ( ptrace(24, v1, 0, 0) < 0 )
{
  perror("ptrace_syscall");
  v3 = (int)"INJECT";
  v4 = "ptrace_syscall error";
  goto LABEL_5;
}
```

Questions ?

“Using no way as way, having no limitation as limitation.”

Marco Grassi
@marcograss
MGrassi@viaforensics.com

VIAFORENSICS