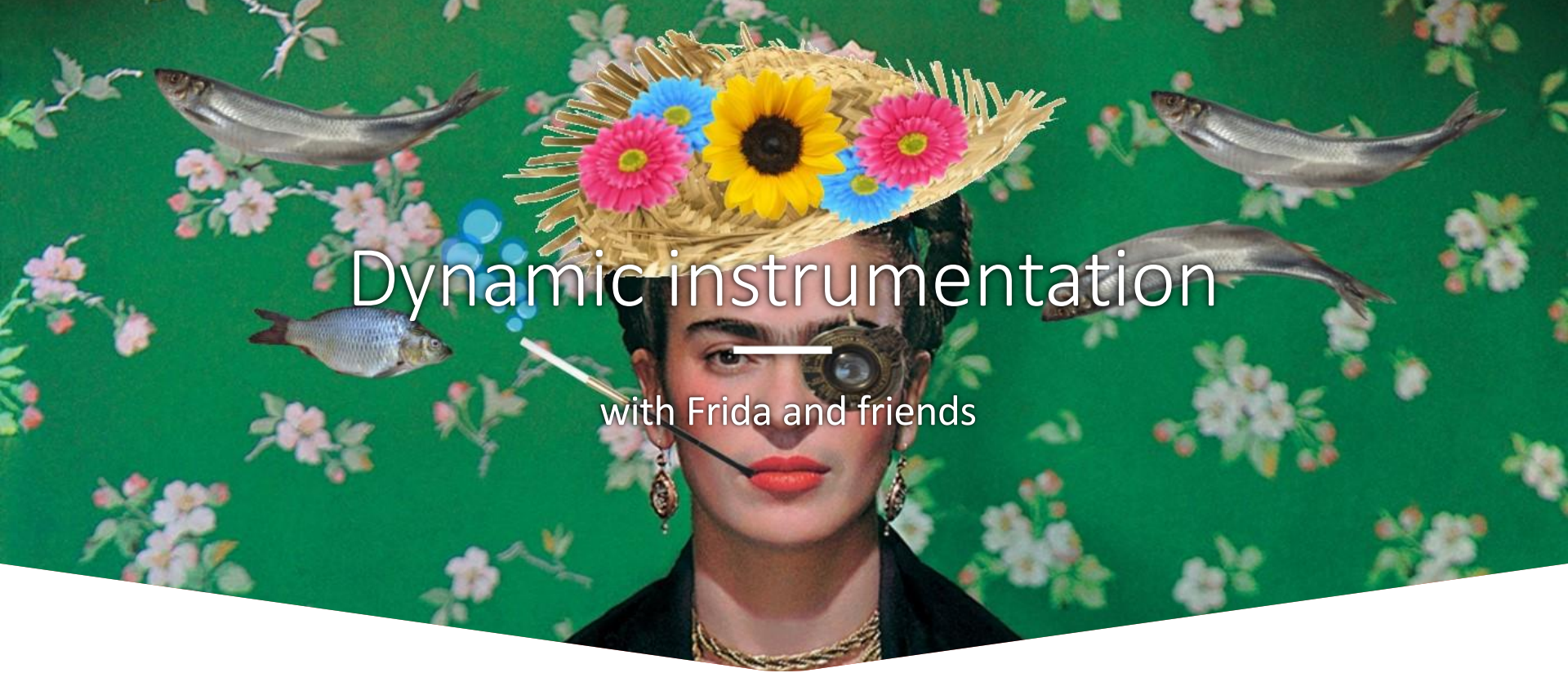


Dynamic instrumentation

with Frida and friends

Martin Schwaighofer and Thomas Wimmer

2018-03-21 @ fhlug.at

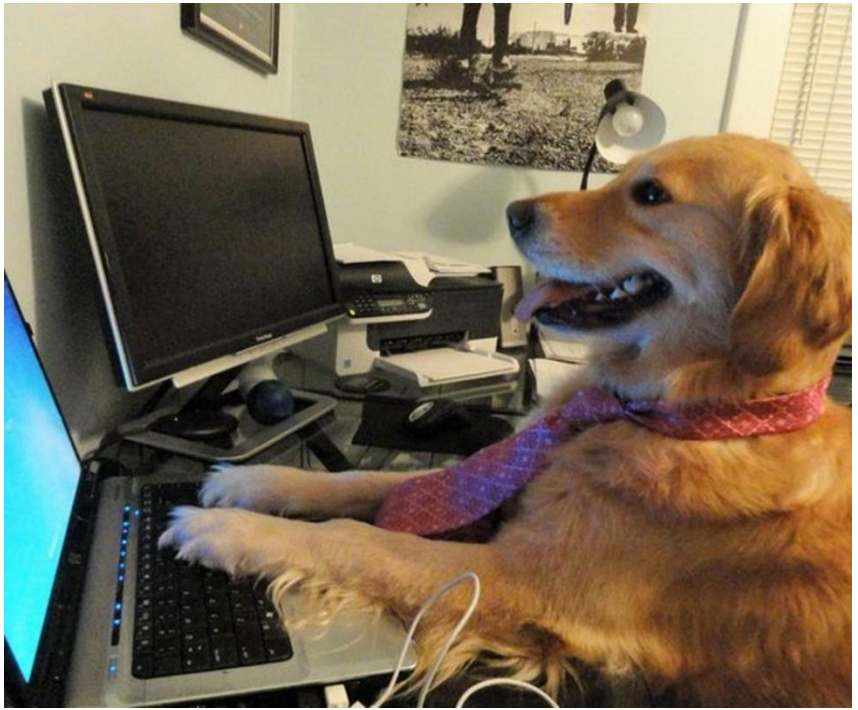
netcetera

What is Frida

- in-process binary instrumentation framework
- lets you inject a JavaScript engine into any process have control over
- enables live editing of injected code
- multi-platform and multi-arch
- built from several independently useful components
- offers rich scriptable APIs with bindings for many languages

Goals of this talk

- create awareness of this kind of tooling
- give an opportunity to play with it
- help getting past initial difficulties by sheer numbers
  - at least someone here should get something to work
- show of Thomas's personal project (which is really cool)

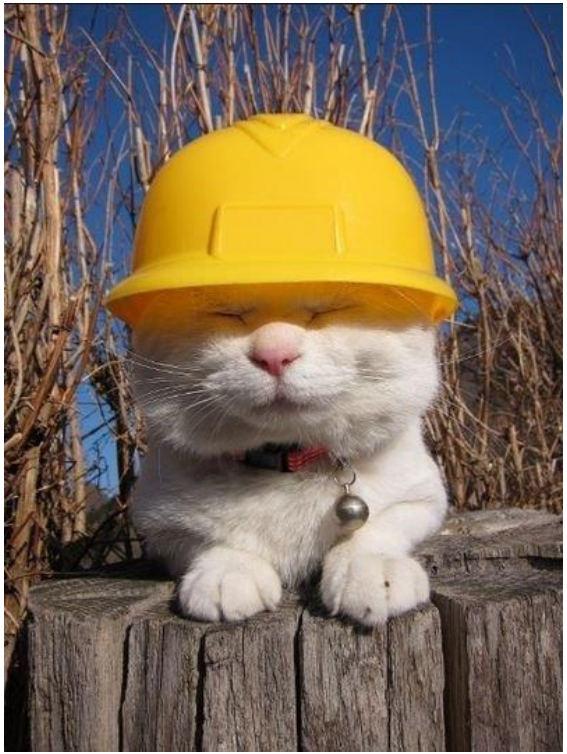


Structure of this talk

- live coding session where you can get your feet wet by participating
- some more theoretical stuff about what Frida is and how it works
- some tools built on Frida and alternative tools you might want to check out
- the story of Thomas reverse engineering the android app for a smart bulb
- at any point during the talk feel free to
  - interrupt and ask questions or
  - start trying something on your own

Safety Instructions

- depending on the context the things you can use this stuff to “cheat” or “violate the terms of services” or “commit fraud” so act responsibly
- do not break things for other people
- try not to get accounts you care about banned/blacklisted by getting caught (SafetyNet on Android, Warden.exe/VAC and so on)



Let’s install Frida

The easiest way to install is installing the python bindings via PyPi.

```
pip install Frida
```

Actually on a fresh Ubuntu install it would be

```
sudo apt install python-pip
pip install [--user] Frida
```

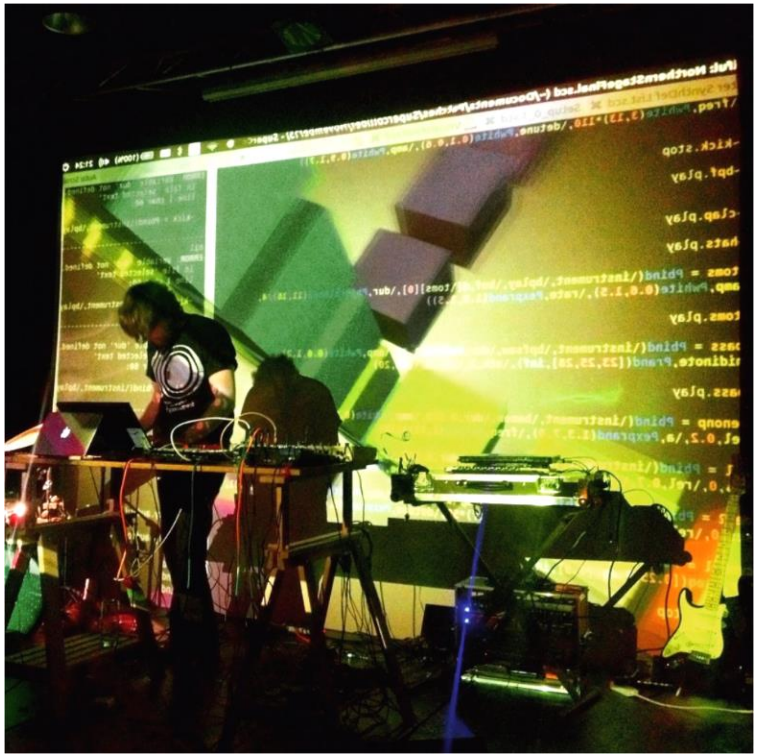
and then it will prompt you to set `ptrace_scope=0` on first launch after each reboot.

Live coding time

We will instrument this harmless little program:

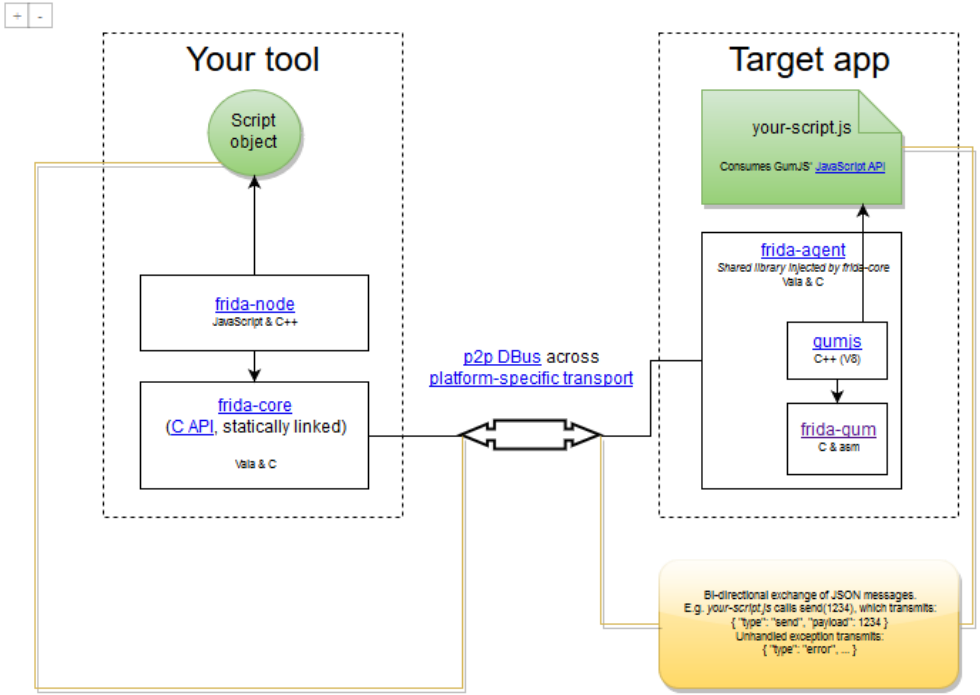
```
#include <stdio.h>
#include <unistd.h>
void f (int n){
  printf ("Number: %d\n", n);
}

int main (int argc, char * argv[]){
  int i = 0;
  printf ("f() is at %p\n", f);
  while (1){
    f (i++);
    sleep (1);
  }
}
```



We are cheating on the live coding part and so can you.
Find our notes at: https://mschwaig.github.io/2018/03/21/live-coding-notes-on-dynamic-instrumentation-with-frida

Frida's architecture



Source:
https://www.frida.re/docs/hacking/

Breaking into a process

In injected mode Frida uses the ptrace system call meant for debuggers to insert breakpoints into code, but instead inserts the following logic:

- Allocate a page
- Put some bootstrapping code on that page
- Jump to the bootstrapping code, which
  - Creates a thread that executes the agent code
  - Revert the modifications you made to break in and bootstrap

Getting invited into a process

When ptrace is not available or you want you attach to the application when it starts up there is a special gadget .so file available, which bootstraps and starts running agent code as soon as the dynamic linker calls the constructor function of the .so file.

The only thing you need to do is either

- add some code into the target application, which loads the .so file or
- use *LD_PRELOAD* to tell the dynamic linker to load the gadget .so before the application code

Frida on jailed/unrooted devices

- The gadget approach works on IOs as well as Android
- With this approach you can use Frida without rooting/jailbreaking your phone
- Under iOS the process also needs to be marked as debuggable if you want to use interceptor API
- You might prefer starting out with an emulator anyways where at least for Android you can easily have root and therefore run Frida server

Use cases for Frida*

- interactively inspect some binary that you want to reverse engineer or dump it after it did some fancy unpacking
- add logging to code that is in production at a customer site to track down some tricky bug
- fake tricky error conditions like a lot of dropped TCP packets without polluting your production code with testing code
- see your own app from a hackers perspective

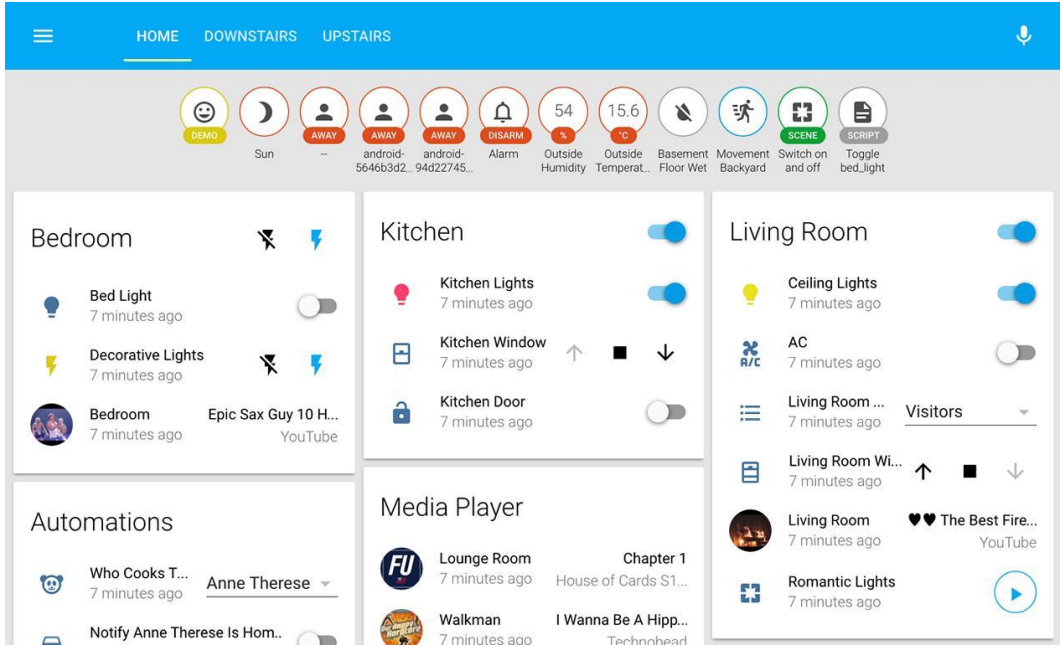
*blatantly copied from my own talk announcement

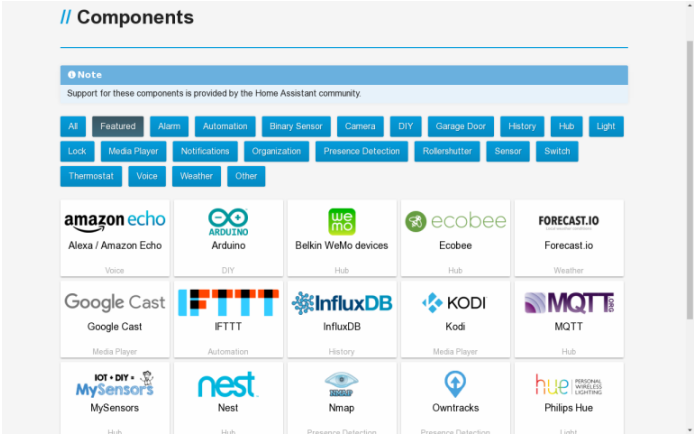
Alternatives to Frida

- Regular debugging tools like **`strace`**, **`dtrace`** and **`valgrind`**
- Objection: mobile exploration toolkit for Android/IOs based on Frida
  https://github.com/sensepost/objection
- Xposed framework: instrument Java on Android with tight ART integration
  - Check out these Xposed-based root/emulator bypasses for banking apps:
    https://github.com/Razer2015
- Great collection of Frida-based tools and resources:
  https://github.com/dweinstein/awesome-frida


FRIDA & Android
A real world use case

Motivation





home-assistant.io

Motivation

Philips Hue -> EUR 47,99

There are some cheaper generic alternatives to those expansive bulbs.
Let’s hack one of those so that we can better integrate it with home asisstant.

The code for the Android part is again available at https://mschwaig.github.io/2018/03/21/live-coding-notes-on-dynamic-instrumentation-with-frida#update-inspecting-network-requests-by-an-android-app
We do not disclose our target App.

Step #1 MITM

Setup:

- Android Emulator
- Android SDK Platform Tools
  https://developer.android.com/studio/releases/platform-tools.html
- APK for App we want to attack
  https://apkpure.com/some.generic.bulb.app
- mitmproxy
  https://mitmproxy.org/

Step #2 Check Network Stack

Setup:

- Frida Server
  https://www.frida.re/docs/installation/
- Frida CLI
  https://www.frida.re/docs/installation/

Reference:

- Frida JavaScript API
  https://www.frida.re/docs/javascript-api/

Step #3 Check for OkHttp Request execution

Setup:

- See Step #2

Reference:

- OkHttp Recipes -> How to send a request with OkHttp ?
  https://github.com/square/okhttp/wiki/Recipes

- Frida JavaScript API
  https://www.frida.re/docs/javascript-api/

Step #4 Log OkHttp Request & Response objects

Setup:

- See Step #3

Reference:

- OkHttp JavaDoc
  https://square.github.io/okhttp/3.x/okhttp/
- Frida JavaScript API
  https://www.frida.re/docs/javascript-api/

Step #5 Disable certificate pinning

Setup:

- See Step #1 & #4

Reference:

- OkHttp Source
  https://github.com/square/okhttp/blob/master/okhttp/src/main/java/okhttp3/internal/connection/RealConnection.java#L313

- Conscrypt Source
  https://android.googlesource.com/platform/libcore/+/38375a4/crypto/src/main/java/org/conscrypt/OpenSSLSocketImpl.java#579

Some defenses against Frida

- Explicit detection algorithms for what Frida does (just google Frida detection)
- Checksumming code of modules you control can detect Interceptor modifying them
- Avoid/obfuscate/encrypt critical data passing though the most exposed boundaries
- Avoid blindly relying on system functionality/heaps of dependencies
- Obfuscate your code well/consider obfuscation early in the design process
- Inline utility functions an attacker would LOVE to hook

Caveats for defenses

You cannot ‘secure’ your application against this, you can only make it ‘obscure’ and difficult to attack. Some people who want to do this:

- Banking apps: here is your bank account, make transactions, except if you are malware
- video streaming platforms: watch this video, but do not upload it to the internet
- social media platforms: watch ads and click things, except if you are a bot
- high-profile games: play this game but, except if you cheat/are a bot

In a lot of other scenarios you actually need not worry about these kinds of attacks. Just have a secure design and give up as soon as somebody gets inside the application process. If you decide to mitigate attacks like that, then make smart tradeoffs with regards to how you spend your time. This is why it is valuable to know the attackers perspective.

Main sources of inspiration

Sam Rubenstein at Bsides Knoxville 2016: https://youtu.be/RINNW4xOWL8

Ole Andre at r2con 2017: https://youtu.be/sBcLPLtqGYU

Francesco Tamagni at r2con 2017: https://youtu.be/URyd4bcV-Ik

There are some more talks at:

https://www.frida.re/docs/presentations/

Bonus Slide

One unexpected use of this kind of tech is community-patches for video games.

Dark Souls was famously broken on PC, but Durante fixed a lot of the graphical issues by instrumenting function calls to the DirectX APIs. Another guy, Nwks originally made it run at 60 FPS. This was possible by patching the binary.

Durante's blog:
http://blog.metaclassofnil.com/