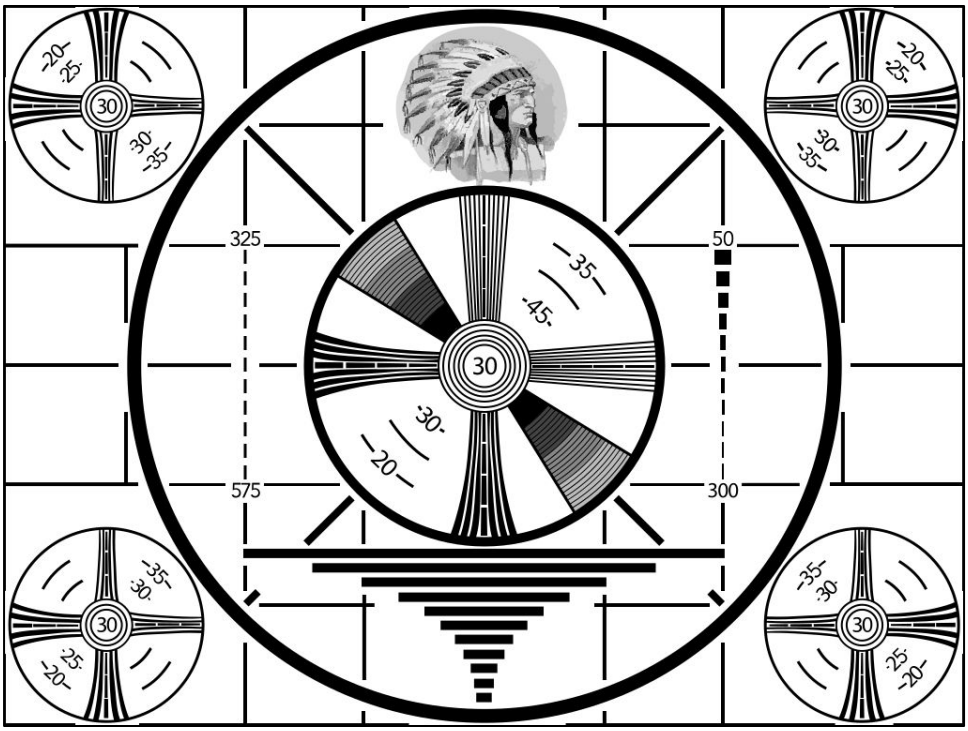
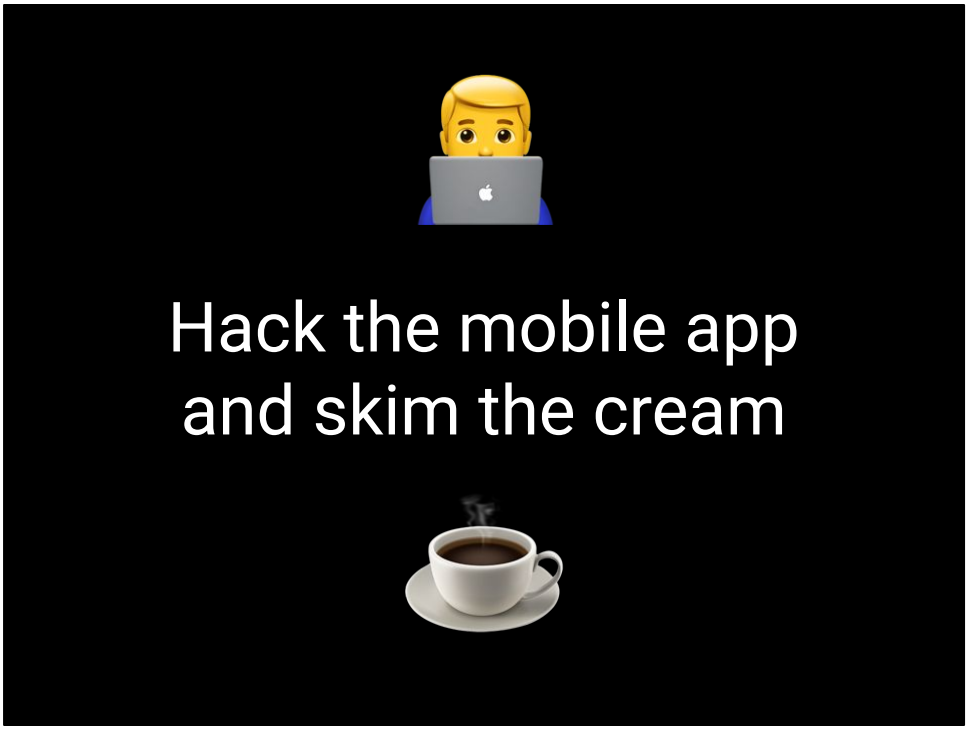

20
25
30
30
35
325
50
35
45
30
20
575
300
35
30
30
25
20

Hi. My name is Kamil Borzym. I am an iOS developer at Allegro in Poland. Allegro is the largest e-commerce company is Poland and 6th largest e-commerce company in Europe. You have Amazon and Ebay, we in Poland have Allegro.



Today, I would like to tell you a story about mobile insecurity and about drinking cocoa.

This presentation aims to raise security awareness of the mobile app developers.

The author never obtained any material benefits from using the techniques you are about to see.

Some names have been changed.

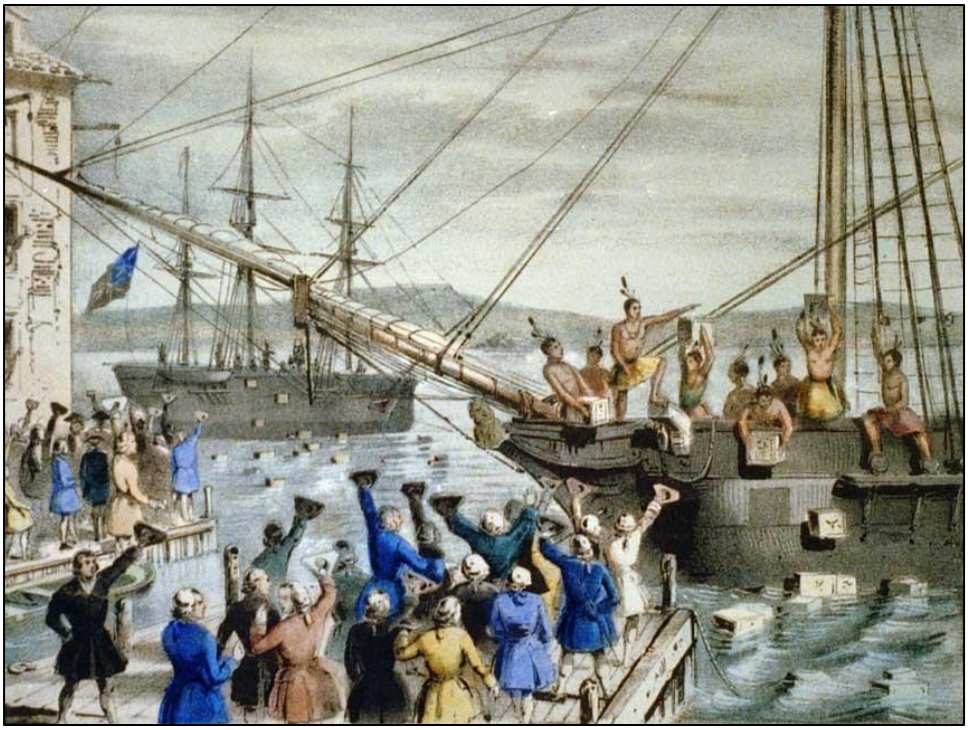
I hope I will raise your security awareness at least a little bit. But in case you find technical details obvious, I hope you will at least love the story around it.
Before going ahead, a little remark. I have never obtained any material benefits from using the techniques you are about to see, and I hope you won't either. Let's start.

Once upon the time in Boston, two mobile developers Connor and Murphy got really bored.

[flickr ??????]

Just to kill the time, they hit the nearby cafe to get something to drink...

[https://flic.kr/p/5g3pkQ]



… anything but exported tea.

[https://upload.wikimedia.org/wikipedia/commons/5/52/Boston_Tea_Party_Currier_colored.jpg]

They spotted a mobile app advertising. The app allowed you to buy a cup of cocoa. It was really tempting to install the app...

[https://flic.kr/p/ZzhkE1]

… because each new app user would get first cocoa cup for free.

Connor and Murphy started wondering...

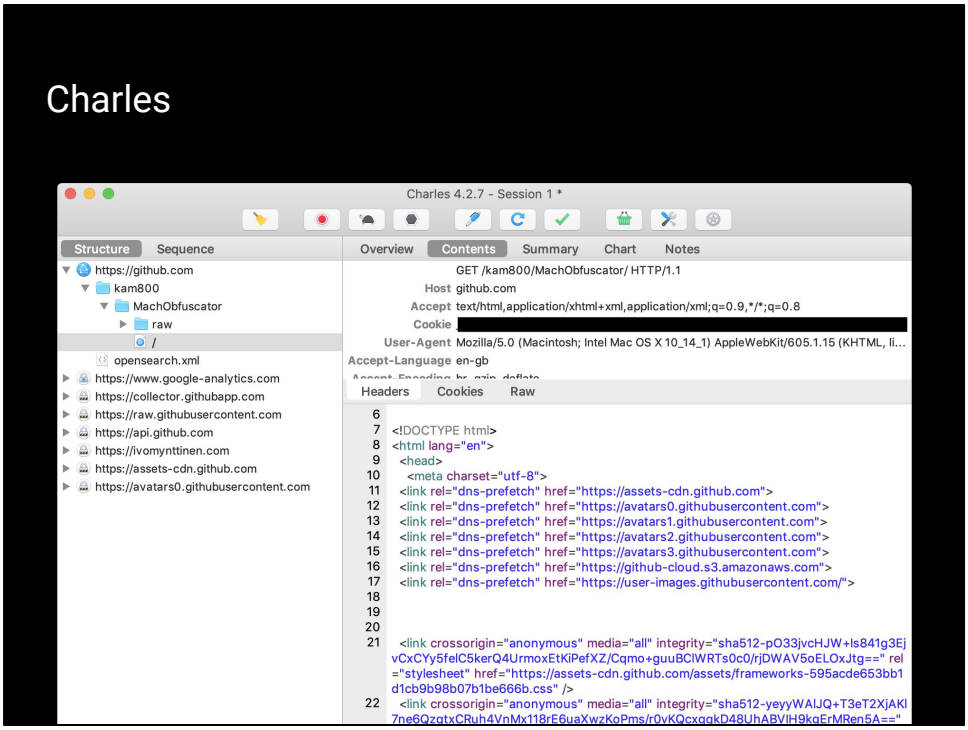
[https://flic.kr/p/qdjCuH]

… how to make every cocoa cup, the first cup.

They started with API reconnaissance.
When it comes to eavesdropping, there are a lot of tools to choose from.

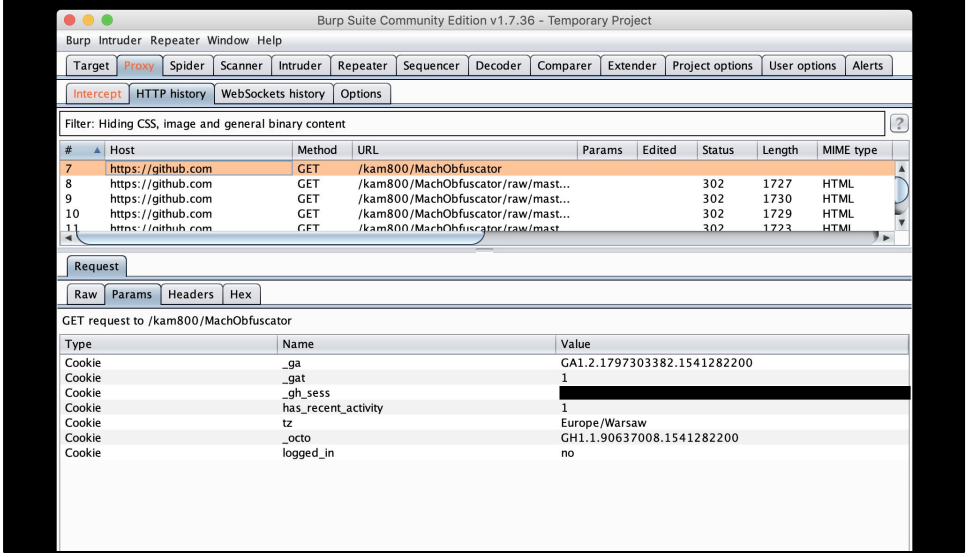
[https://flic.kr/p/8Rx2ov]

Charles



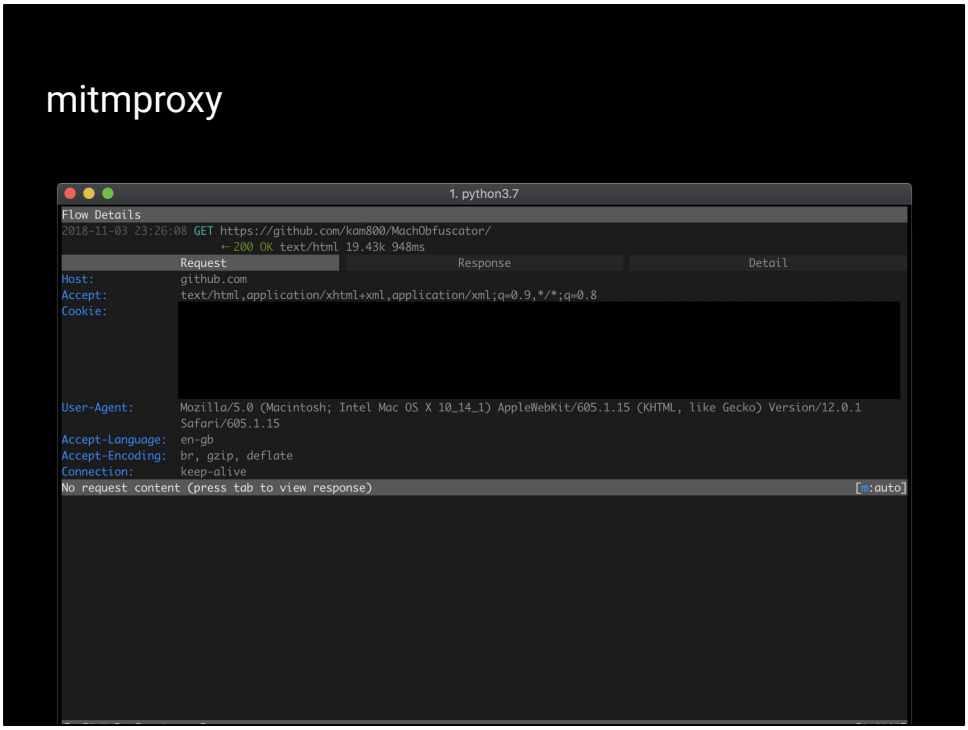
Charles is the most popular macOS http proxy tool. It is pretty straightforward.

Burp Suite



Burp Suite is a very popular multiplatform pentesting tool. A free community version is available.

mitmproxy



But Connor and Murphy use mitmproxy as a weapon of choice. Mitmproxy supports scripting and is opensourced.

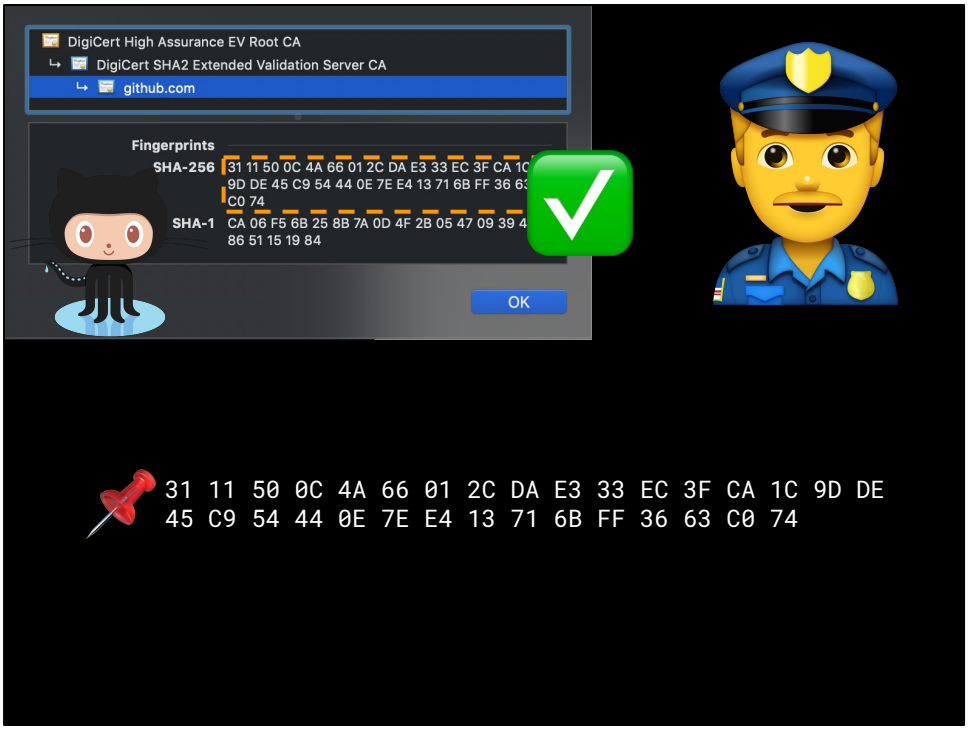
Let's see how Connor and Murphy reverse engineered networking of the CocoApp.

(DEMO)

[https://flic.kr/p/bsJKgb]

As an app developer you can protect your apps from eavesdropping.
One way to do this is by implementing custom certificate validation – so called SSL pinning.

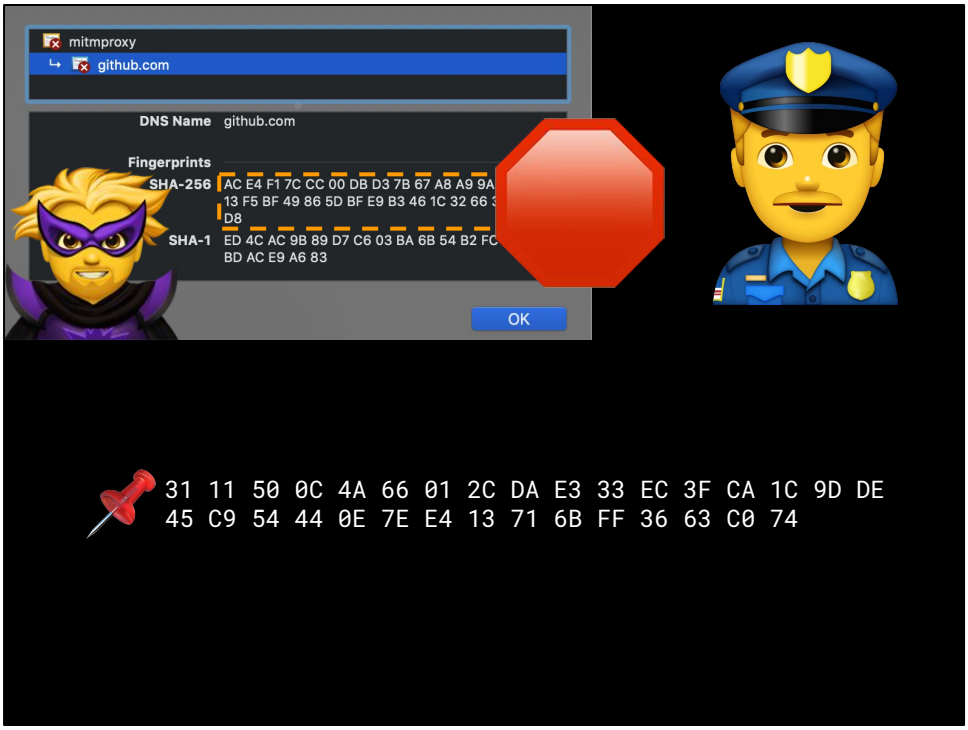
[https://flic.kr/p/4EXdy]



In the most simple form of SSL pinning, an app would contain a hardcoded certificate fingerprint.

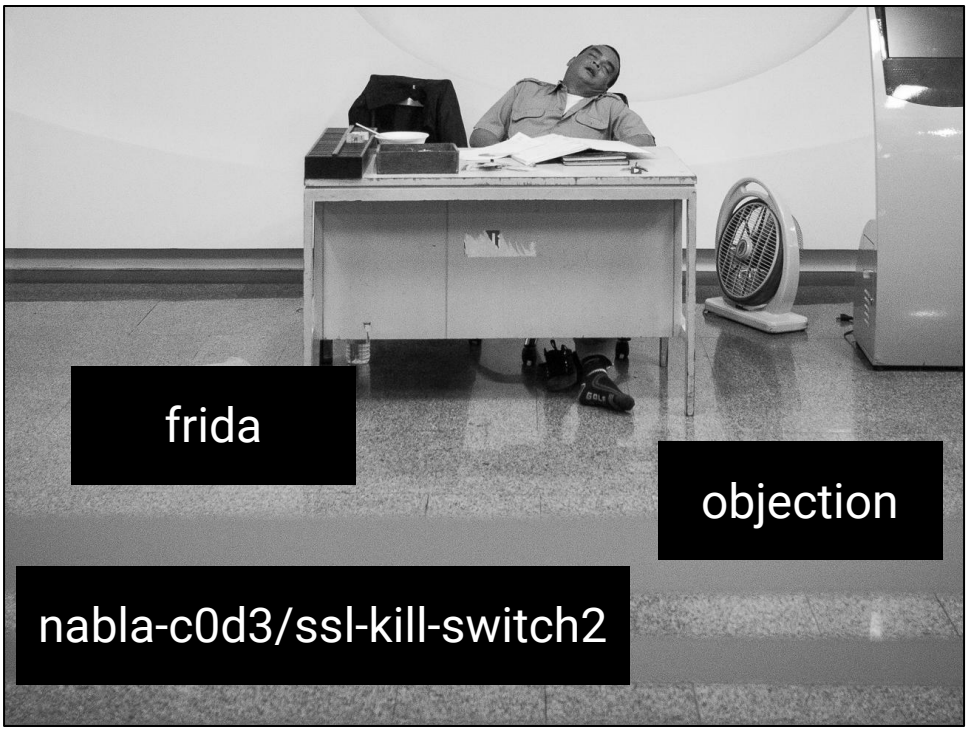
On each connection, the app could check the server certificate…

… and try to match its fingerprint before accepting the connection.



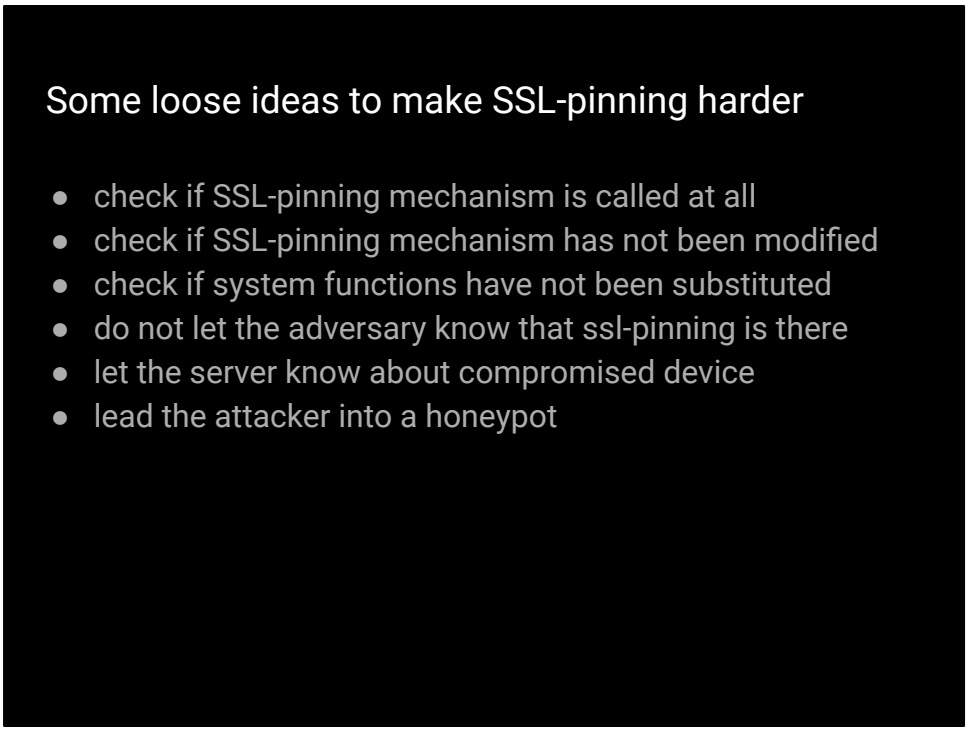
In case of some evil adversary trying to eavesdrop app connection, the certificate fingerprint wouldn’t match,

so the app could just drop the connection (but! only in the most simple form of SSL-pinning).



There are a lot of tools which allow you to easily bypass this most simple form of SSL-pinning.

[https://flic.kr/p/qKGKFC]

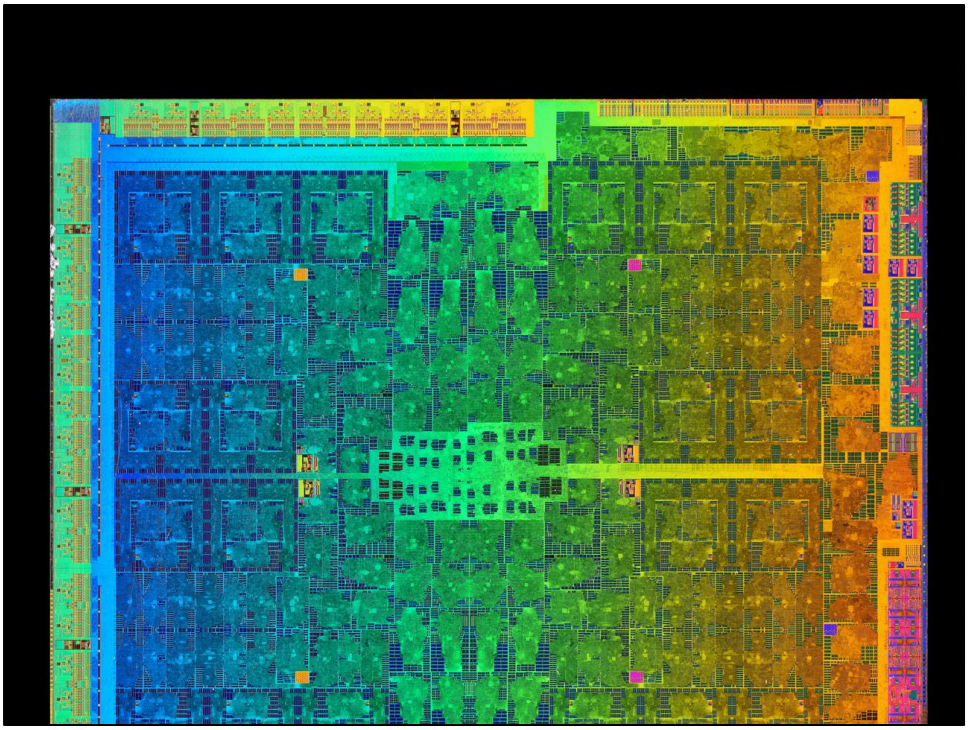


In case you would like to make your SSL-pinning harder, I got some loose ideas.

(READ THE SLIDE)

Connor and Murphy mastered the CocoApp API, but they found the man-in-the-middle technic a little cumbersome. They decided to rebuild the app a little bit to make the cocoa drinking even easier.

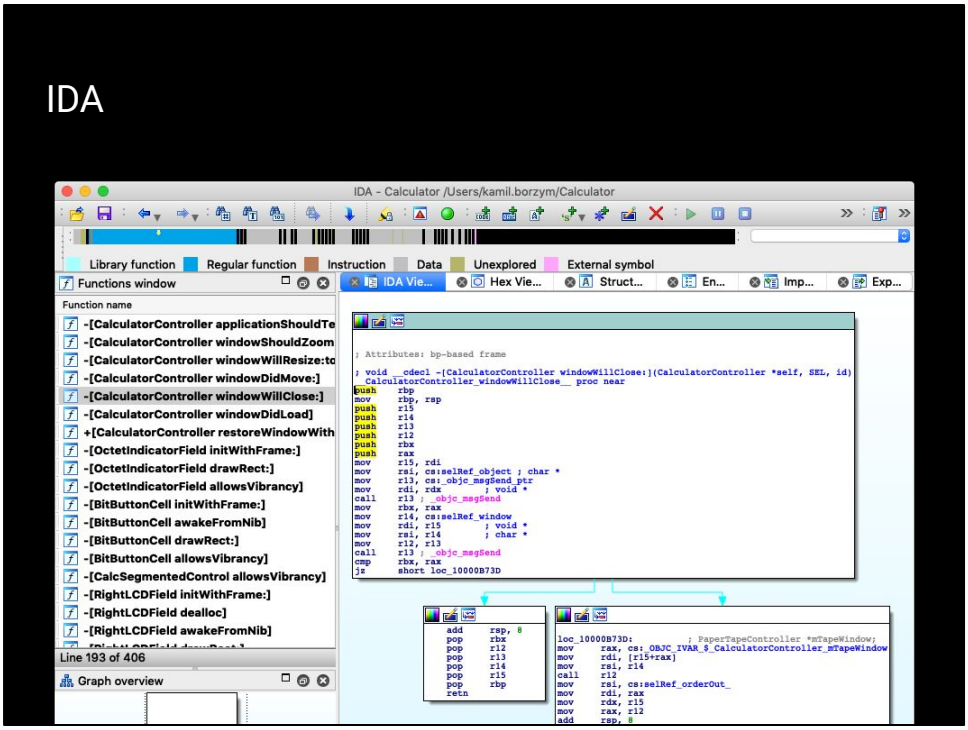
[https://flic.kr/p/dWQopf]



Connor and Murphy are not assembly experts, but they know basics and they have an awesome decompiler.

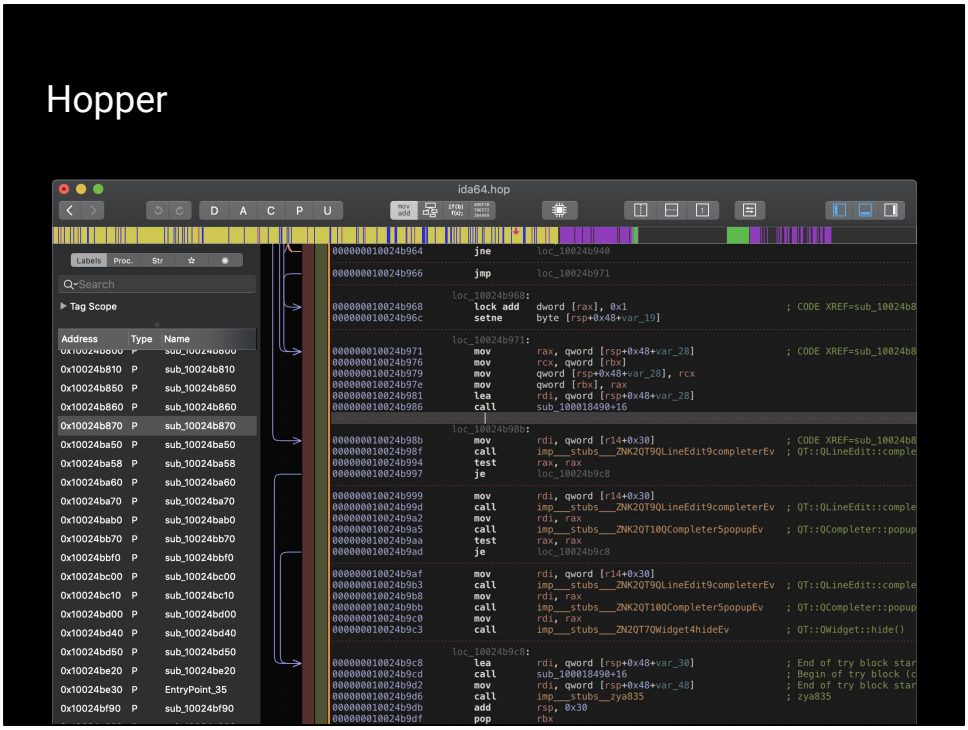
[https://flic.kr/p/XcAjn3]

IDA

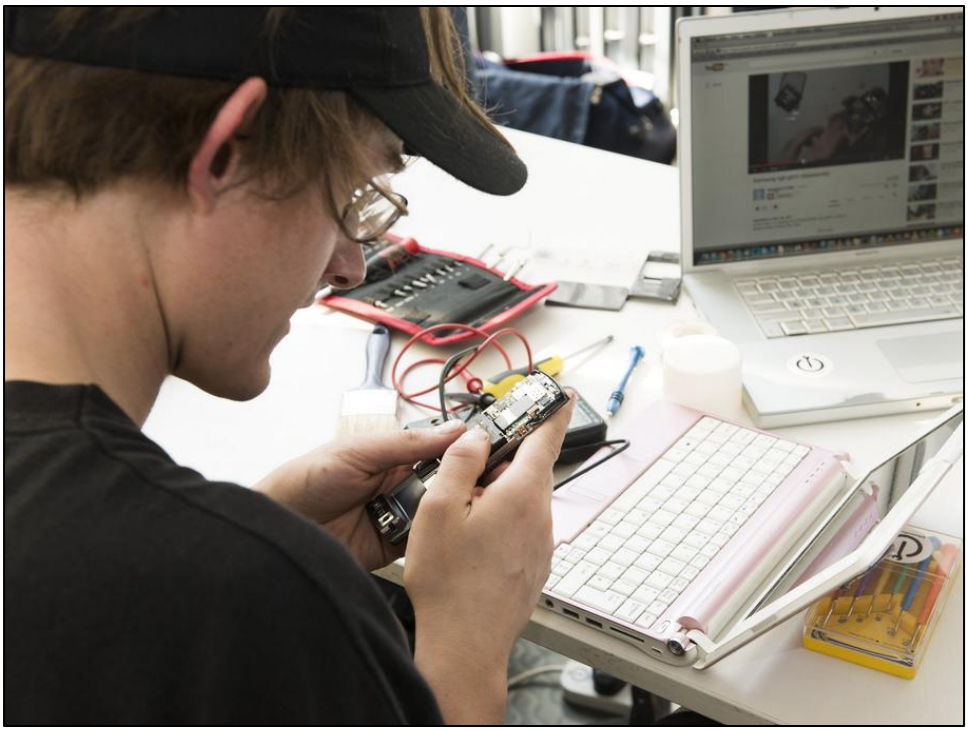


IDA is a multi-platform, multi-processor disassembler. It used to be considered the ultimate reverse engineering tool, but...

Hopper



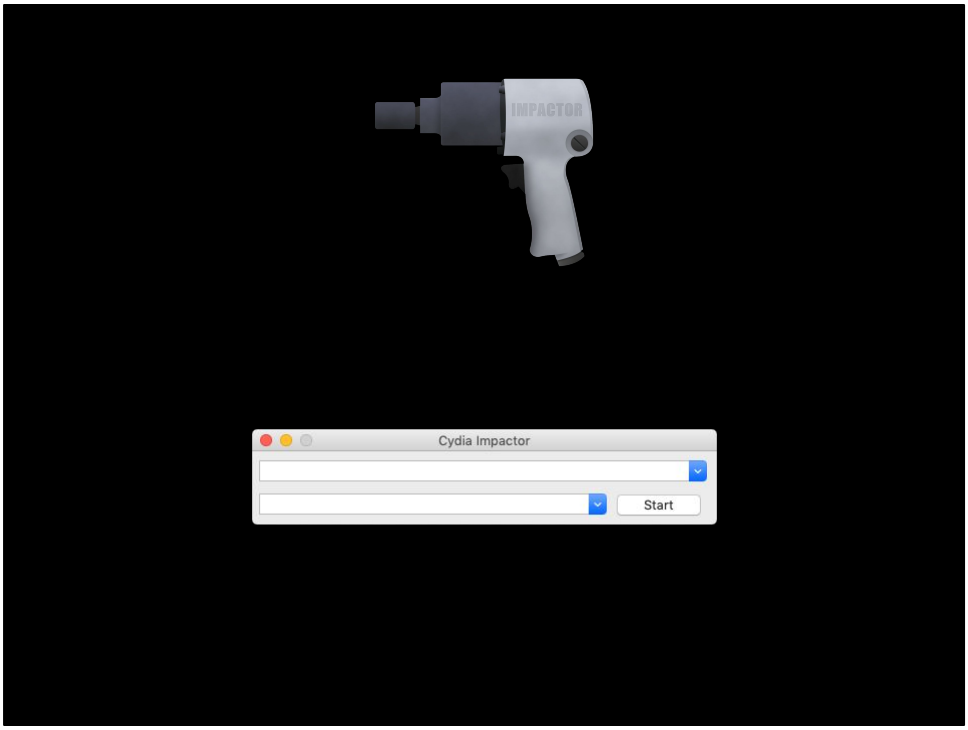
... Connor and Murphy use Hopper as a decompiler of choice. It has perfect support for iOS and macOS binaries, it matches their needs and is reasonably priced.



Let’s dive into bits of CocoApp.

(demo)

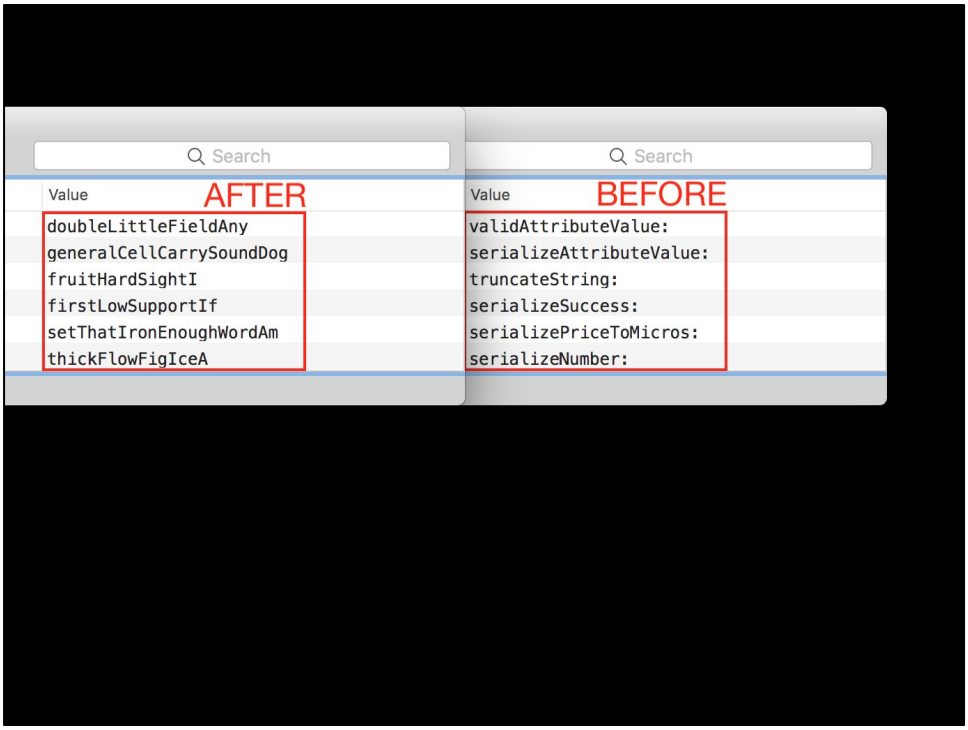
[https://flic.kr/p/n6Xi6E]



Such modified app can easily be installed using Cydia Impactor. Impactor doesn’t need jailbroken device, Xcode or developer account in order to work, just a regular Apple ID.
Now, imagine publishing this modified app on the Internet. This way...

… any creature could easily grab a cup of free cocoa.

Protecting against static reverse engineering is a very broad topic. Fortunately there is one thing that gives a relatively big advantage and is easy to apply. This thing is called...

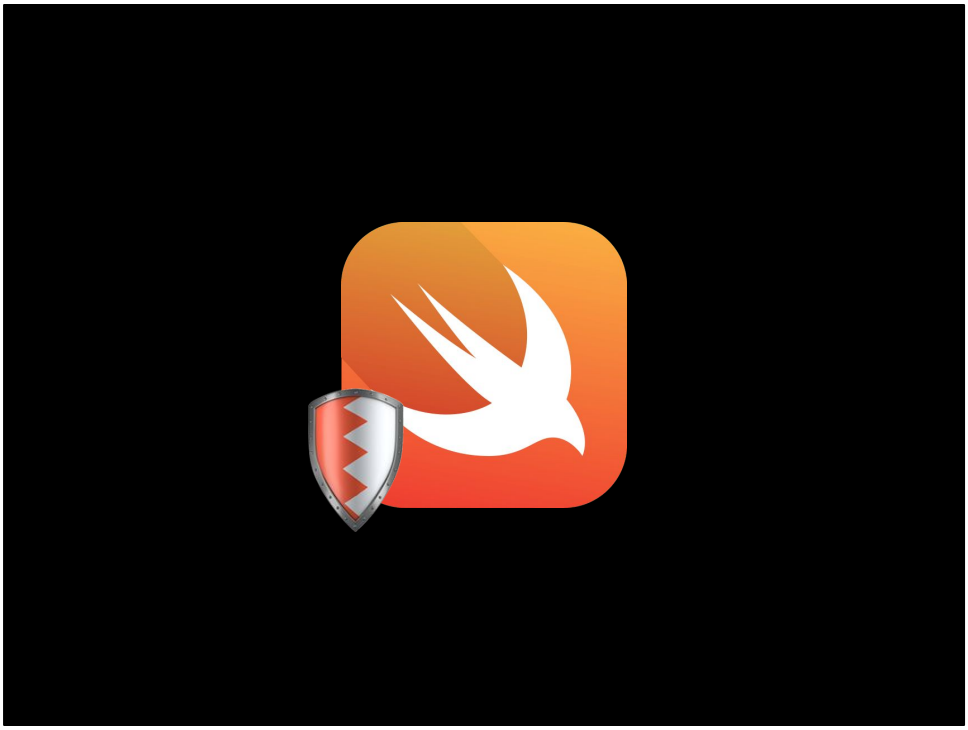


… symbols obfuscation. Symbols obfuscation is a process of replacing symbol names…

… with names that don't have much sense, but also don't impact the overall app behaviour.
This is not an ultimate protection, but it makes reading a large object-oriented assembly much harder.

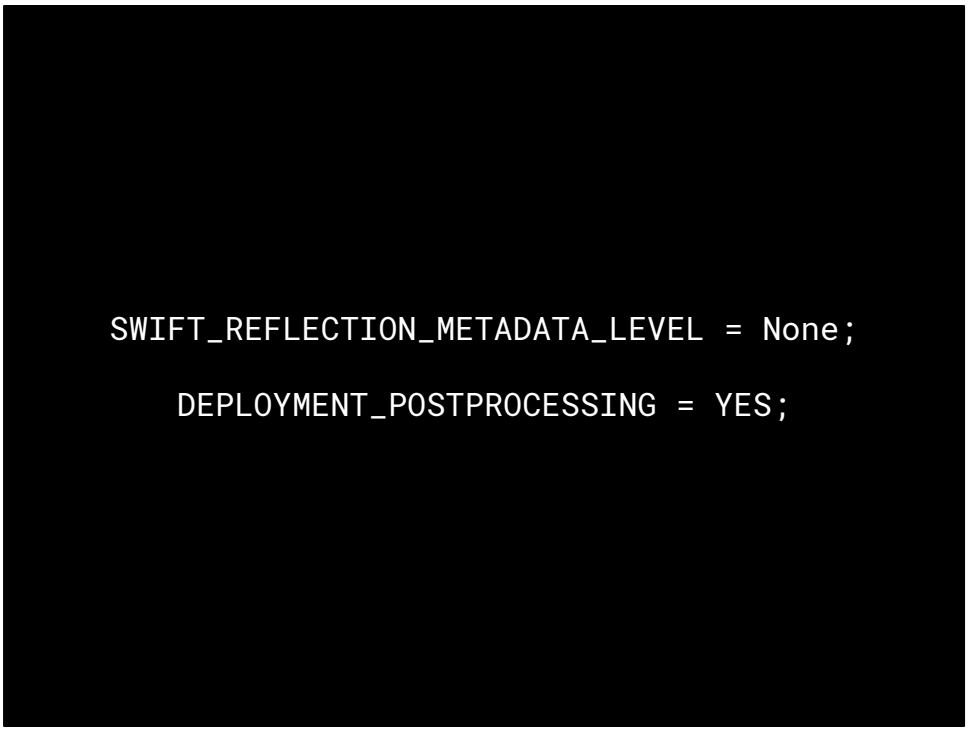
I encourage you to try using MachObfuscator. MachObfuscator obfuscates both Swift and Obj-C and it doesn’t need the source code at all. It obfuscates the compiled app in place. Additionally the crash symbolication isn’t affected, because dsyms are generated before the obfuscation.

(DEMO)



What in case of pure Swift application? Well, it should be harder to decompile Swift. Swift doesn't need all that dynamic stuff used by Obj-C. Is that always true? Let's take a look at swifty version of the CocoApp.

(DEMO)



To sum up:

- you can drop a lot of Swift metadata by setting SWIFT_REFLECTION_METADATA_LEVEL to None.
- you can also drop a lot of Mach-O symbols by enabling deployment postprocessing.

Let’s wrap up. All the stuff I told you today, can be read on the Internet. I just want you to remember two things after this presentation.

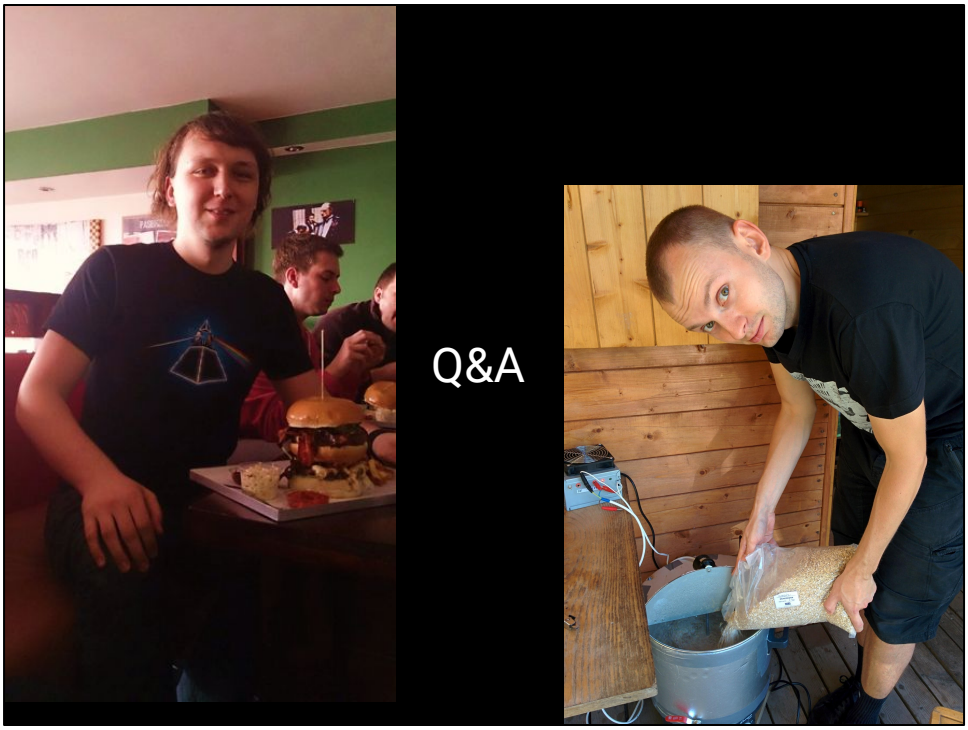
First: Be aware!

Don’t be an ignorant who skips security, but also don’t over engineer when increased security level is not needed.
Security level of your app should be chosen as a result of conscious decision, not just left to chance.
Most likely securing a cat pictures app isn’t worth it. On the other hand – unsecured banking app could make a lot of other attacks easier.

Second thing:

Assume the worst case scenario - this will help you to estimate the risk. Assume that your mobile app source code is publicly available, so all bundled keys and the app logic can be read. Never trust frontend.

Mind, that any protection can be broken. You can only make the attack harder, buying some additional time.



Thanks for your attention. In case you have any question, don’t hesitate to ask them now or chat with me later.
And this guy on the right is Krzysiek Kocel. We did all the research together, but he wasn’t able to be here today.