

# Kamil Borzym

@kam800

allegro

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



# Hack the mobile app and skim the cream



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.



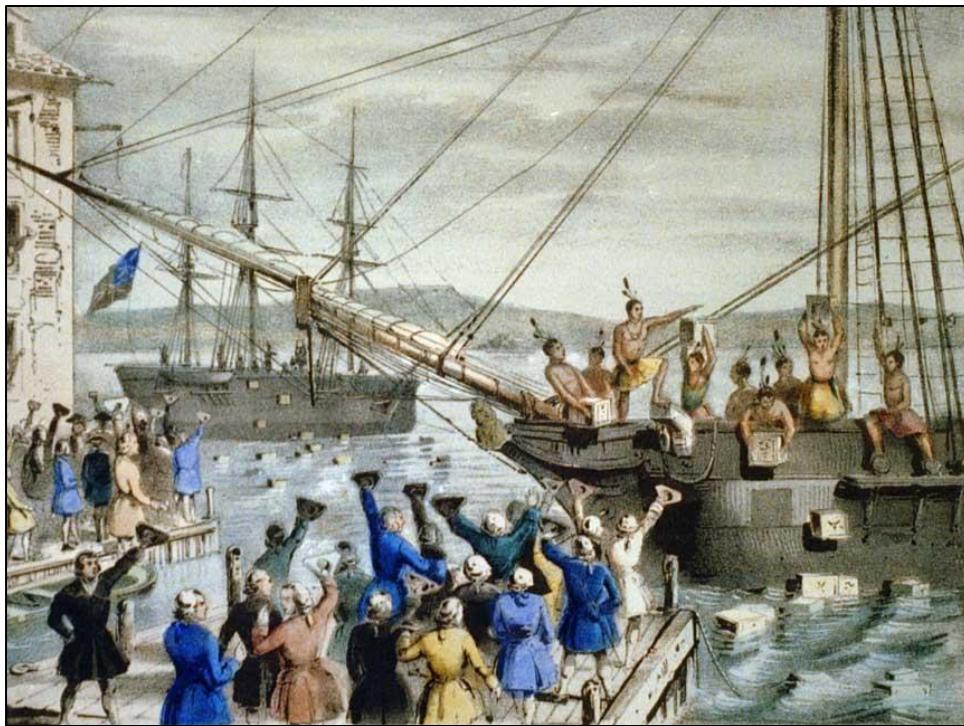
Once upon the time in Boston, two mobile developers Connor and Murphy got really bored.

[flickr ??????]



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

[<https://flic.kr/p/5q3pkQ>]



... anything but exported tea.

[[https://upload.wikimedia.org/wikipedia/commons/5/52/Boston\\_Tea\\_Party\\_Currier\\_colored.jpg](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...

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

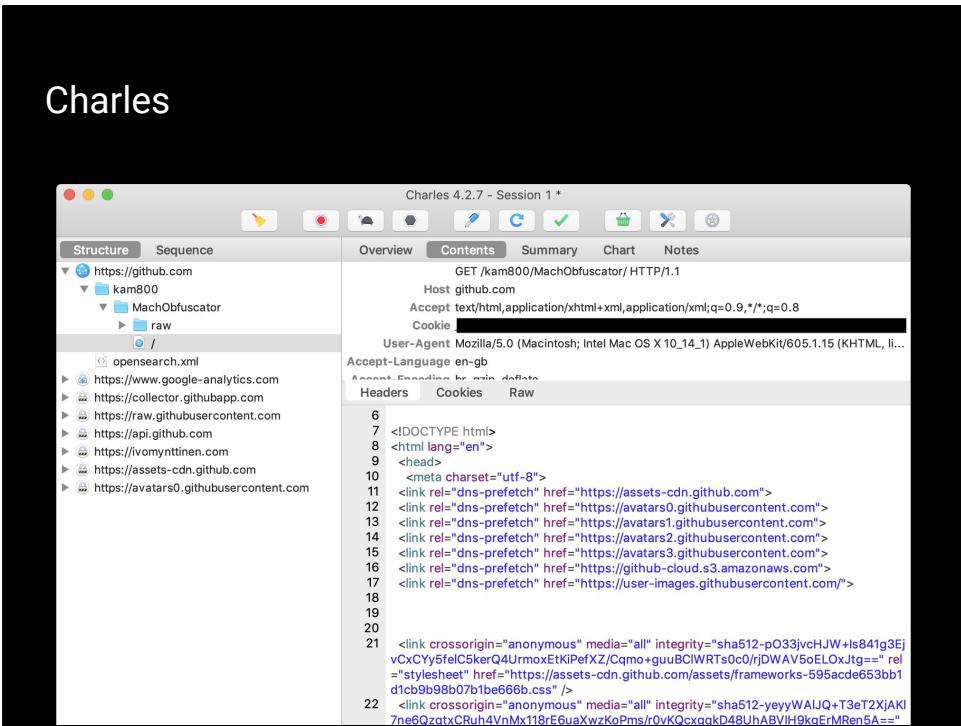


They started with API reconnaissance.

When it comes to eavesdropping, there are a lot of tools to choose from.

[\[https://flic.kr/p/8Rx2ov\]](https://flic.kr/p/8Rx2ov)

Charles



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

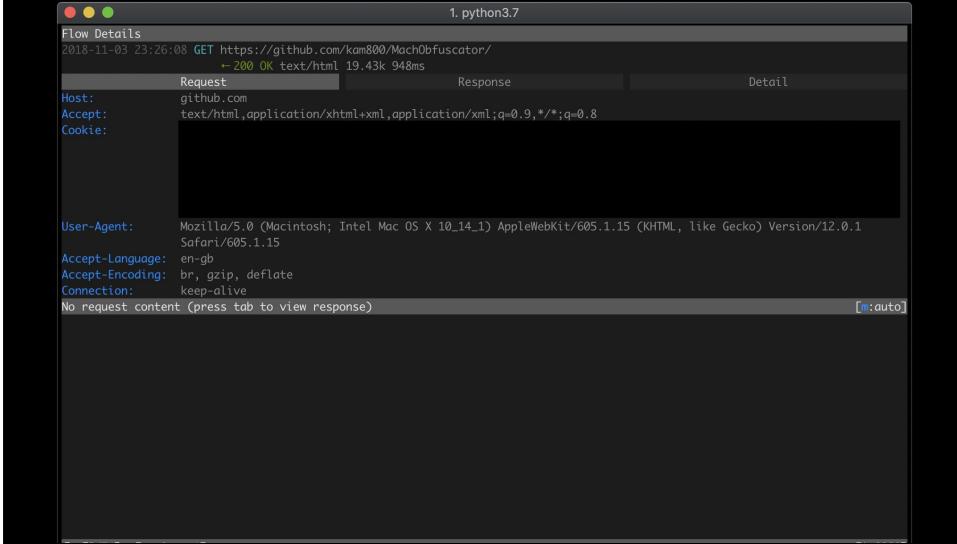
## Burp Suite

The screenshot shows the Burp Suite interface with the following details:

- Toolbar:** Burp Intruder, Repeater, Window, Help.
- Menu Bar:** Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, Alerts.
- Sub-Menu:** Intercept, HTTP history, WebSockets history, Options.
- Table Headers:** #, Host, Method, URL, Params, Edited, Status, Length, MIME type.
- Table Data:** A list of captured requests from https://github.com, mostly 302 status codes with varying lengths and MIME types (HTML).
- Request Tab:** Selected tab.
- Request Details:** Raw, Params, Headers, Hex tabs.
- Cookie Table:** Type, Name, Value columns.
- Cookie Values:** \_ga (GA1.2.1797303382.1541282200), \_gat (1), \_gh\_sess (1), has\_recent\_activity (1), tz (Europe/Warsaw), \_octo (GH1.1.90637008.1541282200), logged\_in (no).

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

## mitmproxy



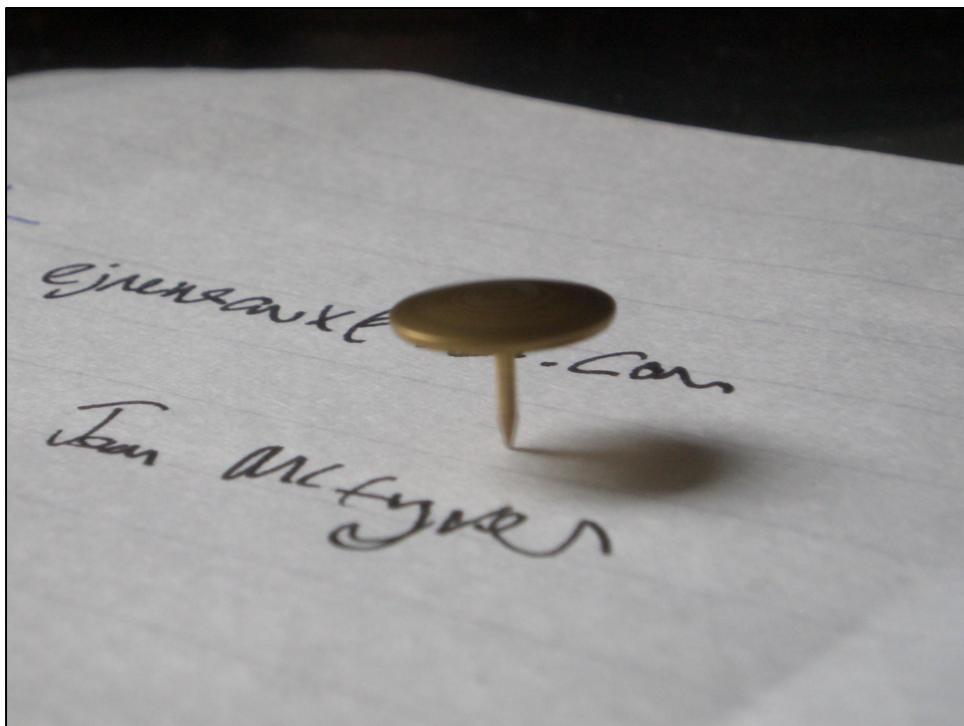
But Connor and Murphy use mitmproxy as a weapon of choice. Mitmproxy supports scripting and is opensourced.



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

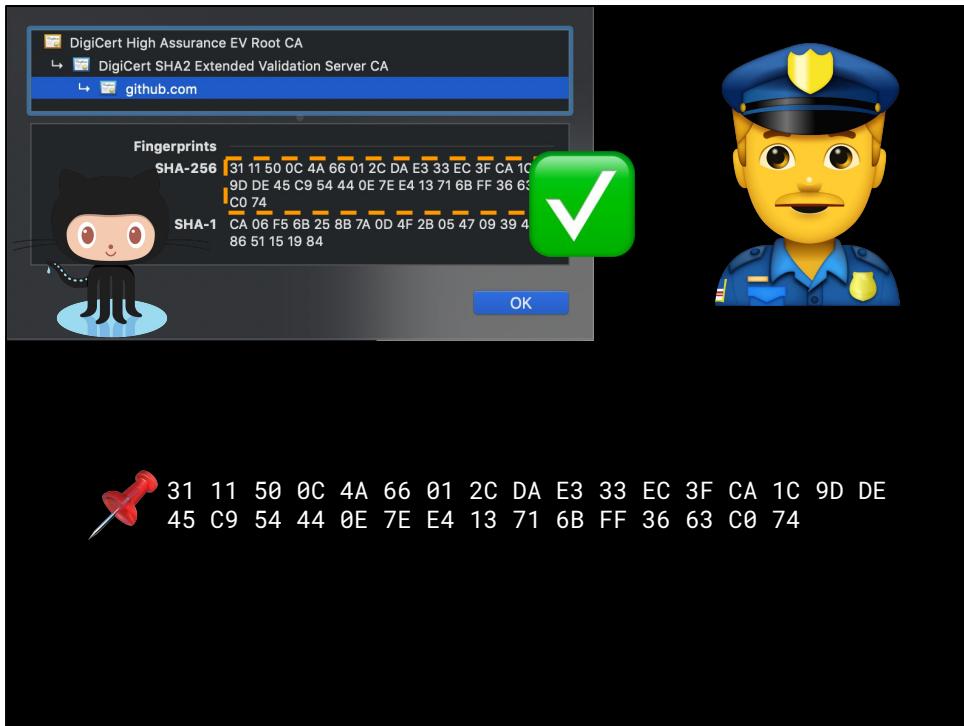
(DEMO)

[<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.

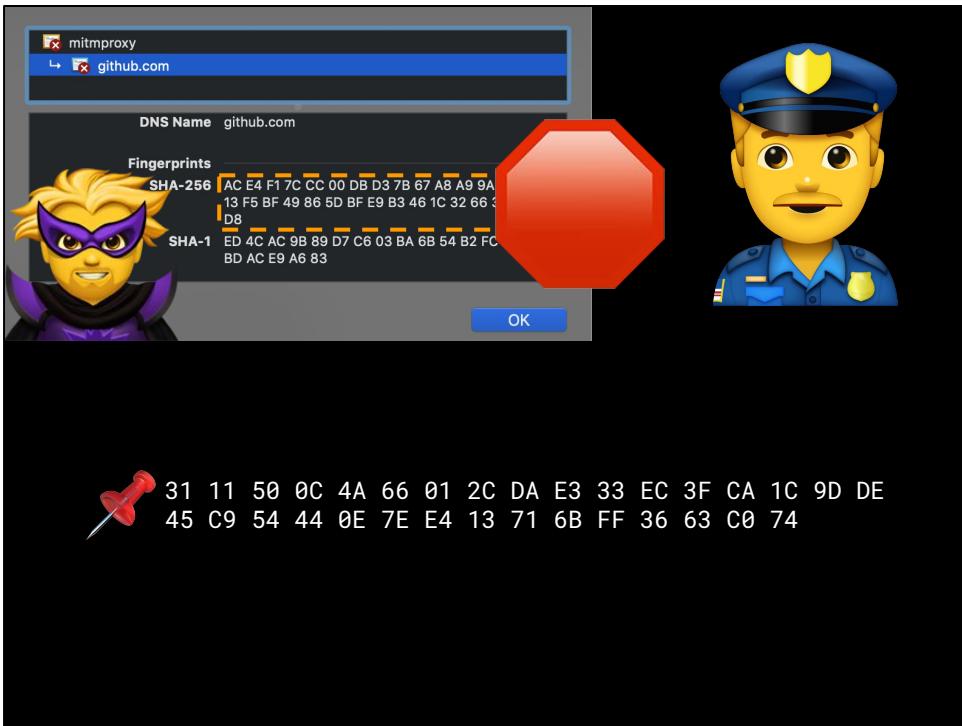
[<https://flic.kr/p/4EXdy>]



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

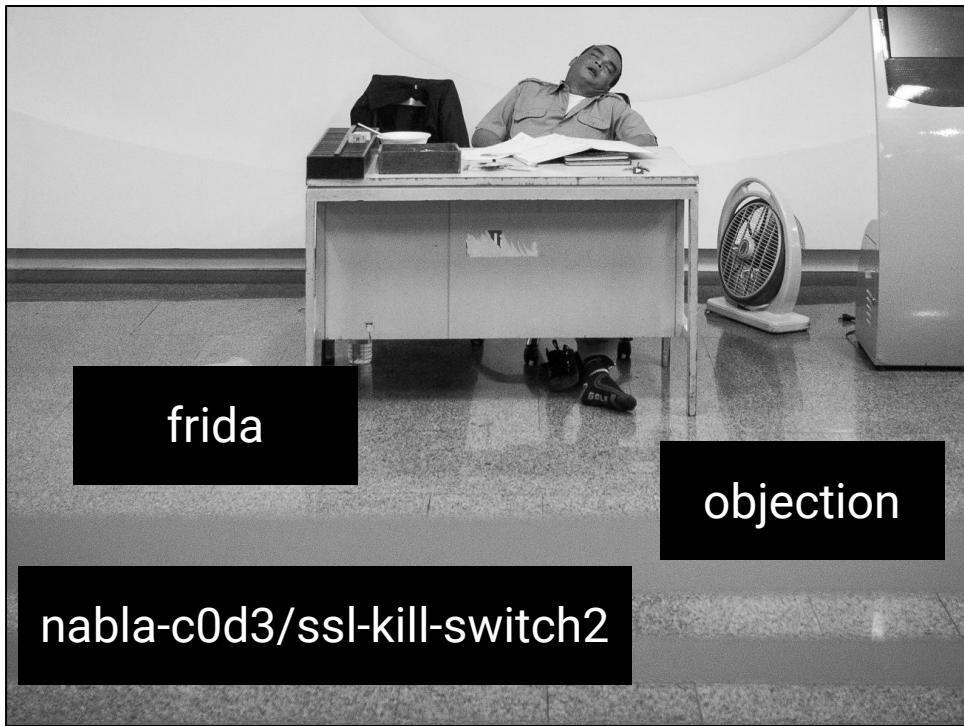
On each connection, the app could check the server certificate...

... 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.

[\[https://flic.kr/p/qKGKFC\]](https://flic.kr/p/qKGKFC)

## Some loose ideas to make SSL-pinning harder

- check if SSL-pinning mechanism is called at all
- check if SSL-pinning mechanism has not been modified
- check if system functions have not been substituted
- do not let the adversary know that ssl-pinning is there
- let the server know about compromised device
- lead the attacker into a honeypot

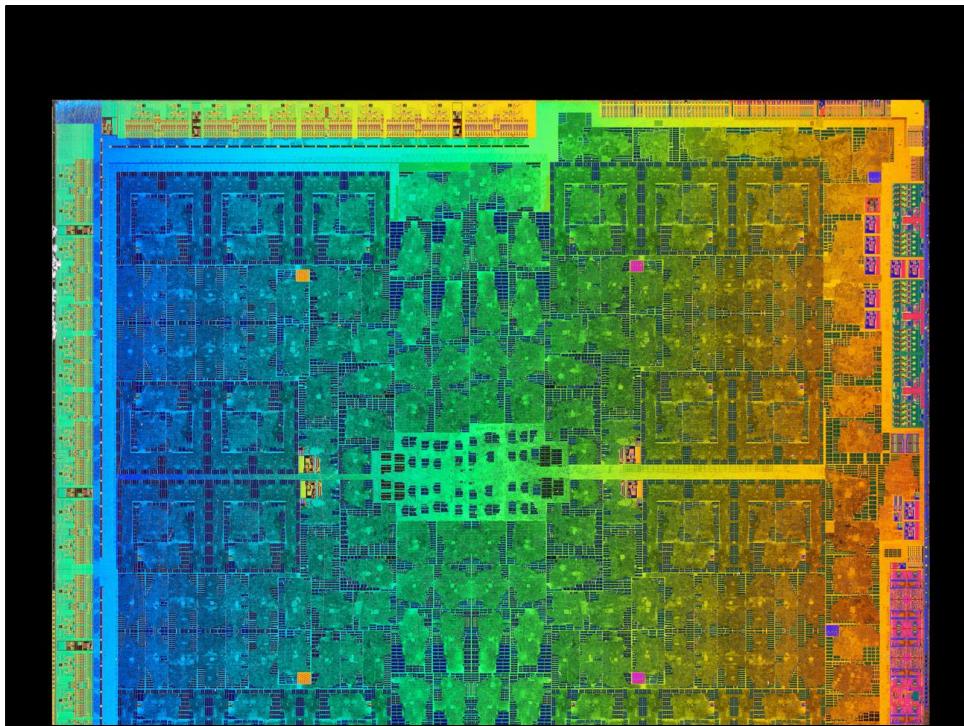
In case you would like to make your SSL-pinning harder, I got some loose ideas.

(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.

[\[https://flic.kr/p/XcAjn3\]](https://flic.kr/p/XcAjn3)

# IDA

The screenshot shows the IDA Pro interface with the following details:

- Functions window:** Lists various Objective-C methods:
  - [CalculatorController applicationShouldTerminate:]
  - [CalculatorController windowShouldZoom]
  - [CalculatorController windowWillResize:toSize:withAnimator:]
  - [CalculatorController windowDidMove:]
  - [CalculatorController windowWillClose:]
  - [CalculatorController windowDidLoad:]
  - +[CalculatorController restoreWindowWithCoder:]
  - [OctetIndicatorField initWithFrame:]
  - [OctetIndicatorField drawRect:]
  - [OctetIndicatorField allowsVibrancy]
  - [BitButtonCell initWithFrame:]
  - [BitButtonCell awakeFromNibFromNib:]
  - [BitButtonCell drawRect:]
  - [BitButtonCell allowsVibrancy]
  - [CalcSegmentedControl allowsVibrancy]
  - [RightLCDField initWithFrame:]
  - [RightLCDField dealloc]
  - [RightLCDField awakeFromNib:]
- Assembly view:** Displays the assembly code for the `-[CalculatorController windowWillClose:]` method.

```
; Attributes: bp-based frame
void _objc_msgSend__(CalculatorController *self, SEL, id)
{
    push    rbp
    mov     rbp, rsp
    push    r15
    push    r14
    push    r13
    push    r12
    push    r11
    push    r10
    push    rax
    mov     r15, rdi
    mov     r13, cs:_objc_msgSend_ptr
    mov     r12, rdx
    mov     r11, void *
    call    r13
    mov     rbx, rax
    mov     r10, cs:_objc_msgSend_
    mov     rdi, r15
    mov     rsi, r14
    mov     r11, char *
    call    r13
    cmp     rbx, rax
    jne    short loc_10000B73D
}
```
- Data view:** Displays the assembly code for the `loc_10000B73D:` label.

```
add    rsp, 8
loc_10000B73D:
    mov    rax, cs:_OBJC_IVAR_$_CalculatorController.mTapeWindow
    mov    rax, cs:_OBJC_IVAR_$_CalculatorController.mTapeWindow
    mov    rdi, [r15+rax]
    mov    rsi, r14
    mov    r11, r12
    call    r12
    mov    rsi, cs:_objc_orderOut_
    mov    rdi, rax
    mov    rdx, r15
    mov    rax, r12
    add    rbp, 8
```
- Graph overview:** Shows a small graph of nodes and edges.

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

# Hopper

ida64.hop

Labels Proc. Str ⚡ ●

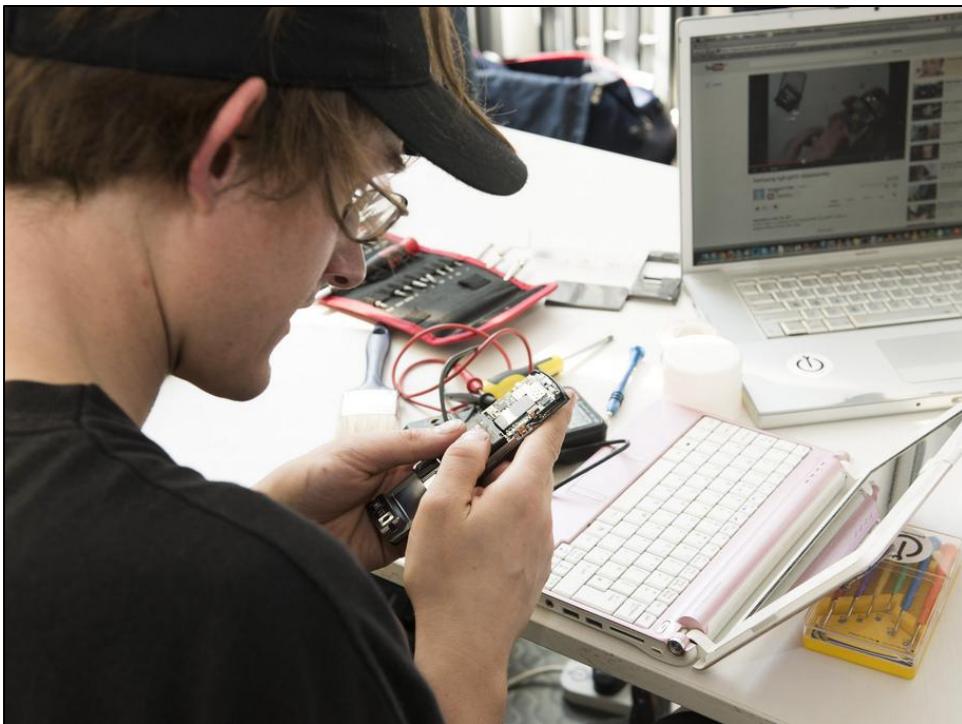
Q>Search

Tag Scope

Address	Type	Name
0x10024b800	P	sub_10024b800
0x10024b810	P	sub_10024b810
0x10024b850	P	sub_10024b850
0x10024b860	P	sub_10024b860
0x10024b870	P	sub_10024b870
0x10024b850	P	sub_10024b850
0x10024b858	P	sub_10024b858
0x10024b860	P	sub_10024b860
0x10024b870	P	sub_10024b870
0x10024b850	P	sub_10024b850
0x10024b858	P	sub_10024b858
0x10024b860	P	sub_10024b860
0x10024b870	P	sub_10024b870
0x10024bab0	P	sub_10024bab0
0x10024bab0	P	sub_10024bab0
0x10024bab70	P	sub_10024bab70
0x10024bbf0	P	sub_10024bbf0
0x10024bc00	P	sub_10024bc00
0x10024bc10	P	sub_10024bc10
0x10024bd00	P	sub_10024bd00
0x10024bd40	P	sub_10024bd40
0x10024bd50	P	sub_10024bd50
0x10024be20	P	sub_10024be20
0x10024be30	S	EntryPoint_35
0x10024bf90	P	sub_10024bf90

000000010024b964 jne loc\_10024b940  
000000010024b956 jmp loc\_10024b971  
loc\_10024b968:  
000000010024b968 lock add dword [rax], 0x1 ; CODE XREF=sub\_10024b8  
000000010024b96c setne byte [rsp+0x48+var\_19]  
loc\_10024b971:  
000000010024b971 mov rax, qword [rsp+0x48+var\_28] ; CODE XREF=sub\_10024b8  
000000010024b976 mov rcx, qword [rbx]  
000000010024b979 mov qword [rsp+0x48+var\_28], rcx  
000000010024b982 mov rdx, qword [rcx+0x48+var\_28]  
000000010024b981 lea rdx, qword [rsp+0x48+var\_28]  
000000010024b986 call sub\_100015490+16  
loc\_10024b988:  
000000010024b988 mov rdi, qword [r14+0x30] ; CODE XREF=sub\_10024b8  
000000010024b98c call imp\_stubs\_ZN2QT9LineEdit9completerEv ; QT:QLineEdit::comple  
000000010024b994 test rax, rax  
000000010024b997 je loc\_10024b9c8  
000000010024b999 mov rdi, qword [r14+0x38]  
000000010024b99d call imp\_stubs\_ZN2QT9LineEdit9completerEv ; QT:QLineEdit::comple  
000000010024b9a2 mov rdi, rax  
000000010024b9a5 call imp\_stubs\_ZN2T10Completor5popupEv ; QT:QCompleter::popup  
000000010024b9a8 test rax, rax  
000000010024b9ad je loc\_10024b9c8  
000000010024b9af mov rdi, qword [r14+0x38] ; CODE XREF=sub\_10024b8  
000000010024b9b3 call imp\_stubs\_ZN2QT9LineEdit9completerEv ; QT:QLineEdit::comple  
000000010024b9b8 mov rdi, rax  
000000010024b9bc call imp\_stubs\_ZN2T10Completor5popupEv ; QT:QCompleter::popup  
000000010024b9e0 mov rdi, rax  
000000010024b9e3 call imp\_stubs\_ZN2QT7QWidget4hideEv ; QT:QWidget::hide()  
loc\_10024b9c8:  
000000010024b9c8 lea rdi, qword [rsp+0x48+var\_30] ; End of try block star  
000000010024b9cd call 000000010024b9a16 ; Begin of try block (c  
000000010024b9d2 mov rdi, qword [rsi+0x48+var\_48] ; End of try block star  
000000010024b9d6 call imp\_stubs\_2y@035 ; sy@035  
000000010024b9d9 add rsp, 0x30  
000000010024b9df pop rbx

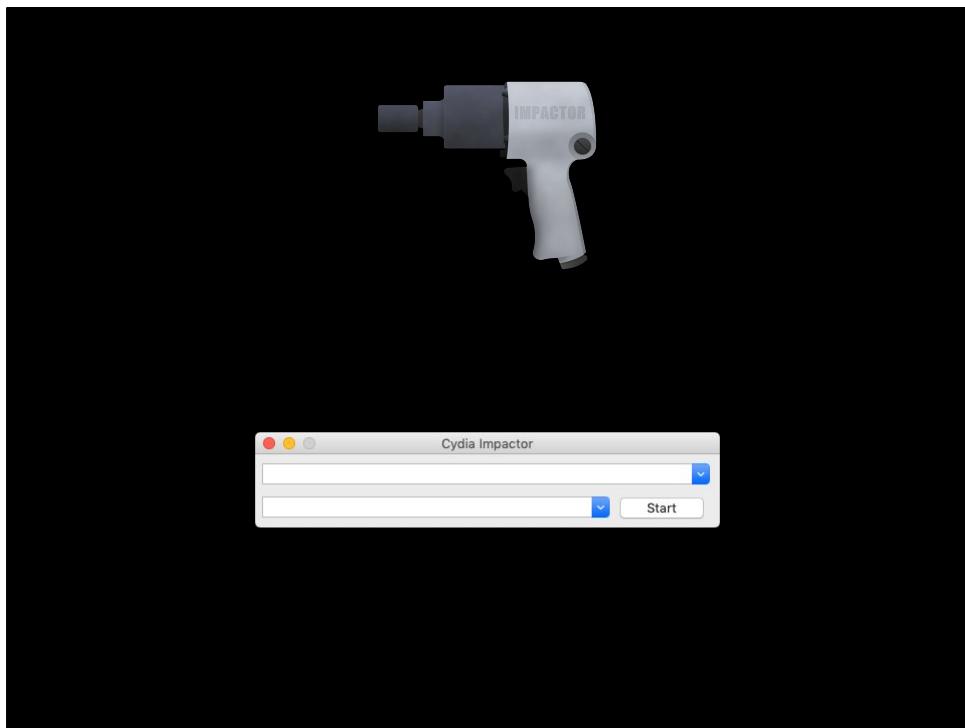
... Connor and Murphy use Hopper as a decompiler of choice. It has perfect support for iOS and macOS binaries, it matches their needs and is reasonably priced.



Let's dive into bits of CocoApp.

(demo)

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

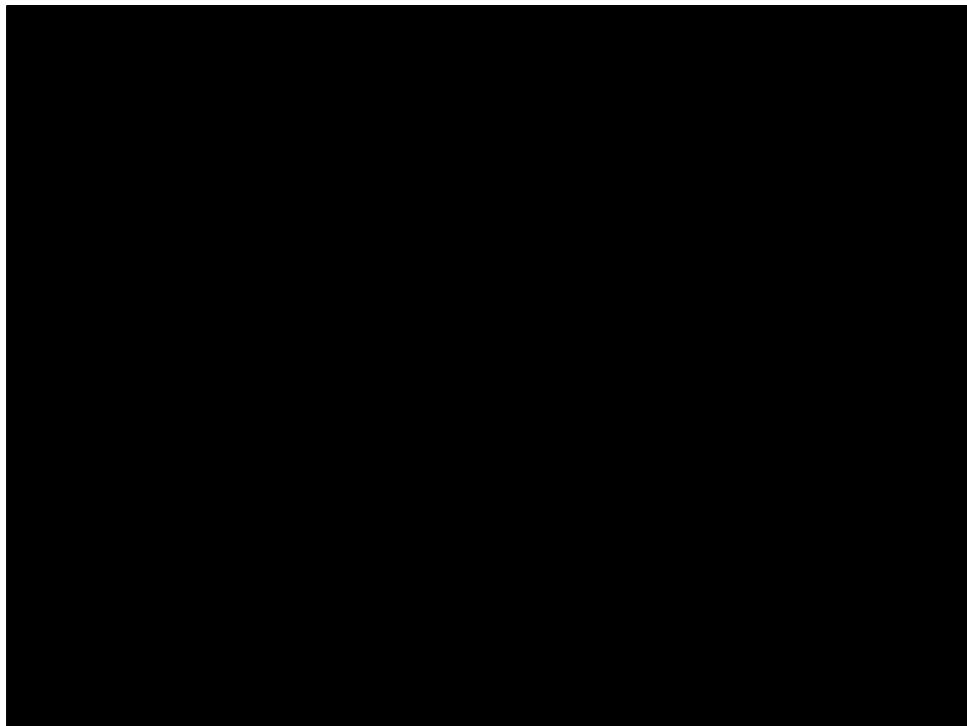


Such modified app can easily be installed using Cydia Impactor. Impactor doesn't need jailbroken device, Xcode or developer account in order to work, just a regular Apple ID.

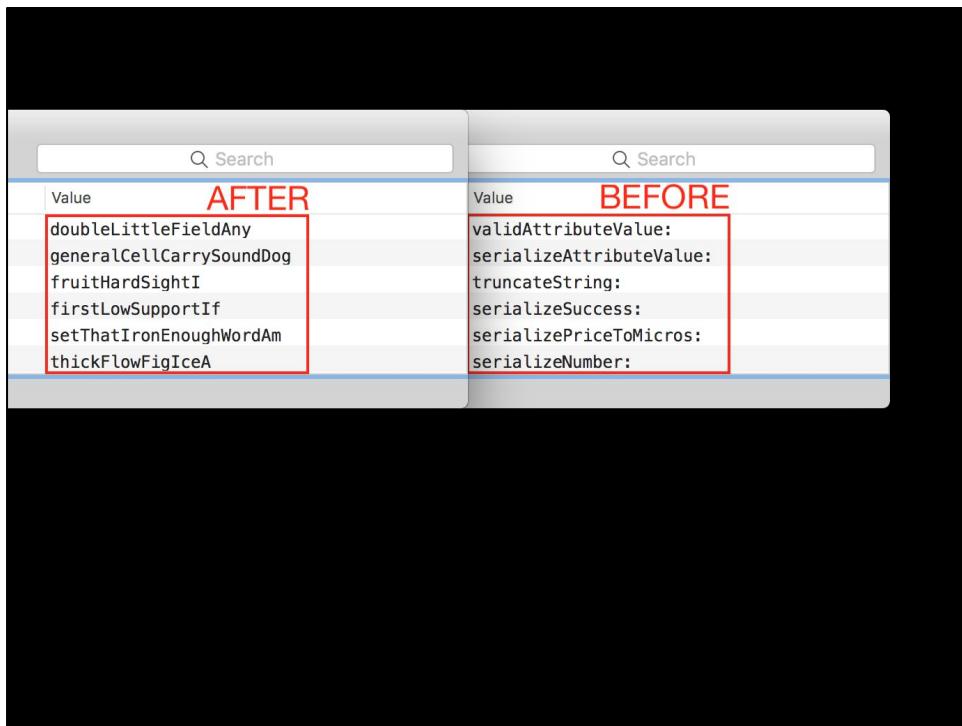
Now, imagine publishing this modified app on the Internet. This way...



... 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...

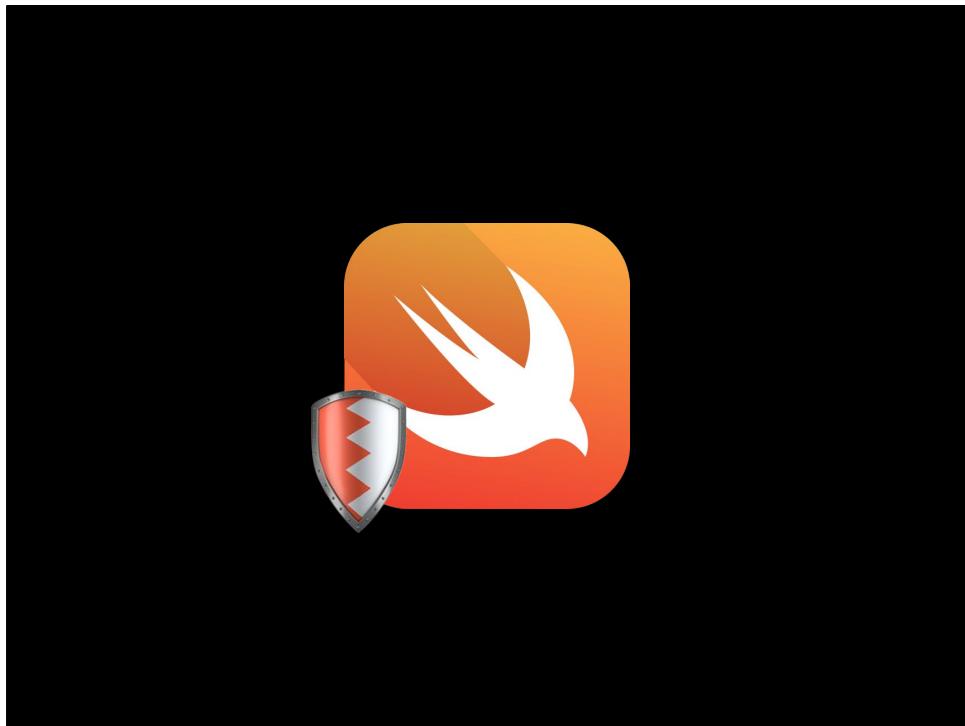
... with names that don't have much sense, but also don't impact the overall app behaviour.

This is not an ultimate protection, but it makes reading a large object-oriented assembly much harder.

[github.com/kam800/MachObfuscator](https://github.com/kam800/MachObfuscator)

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)

```
SWIFT_REFLECTION_METADATA_LEVEL = None;  
  
DEPLOYMENT_POSTPROCESSING = YES;
```

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.

## Takeaways

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.



## Be aware

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.

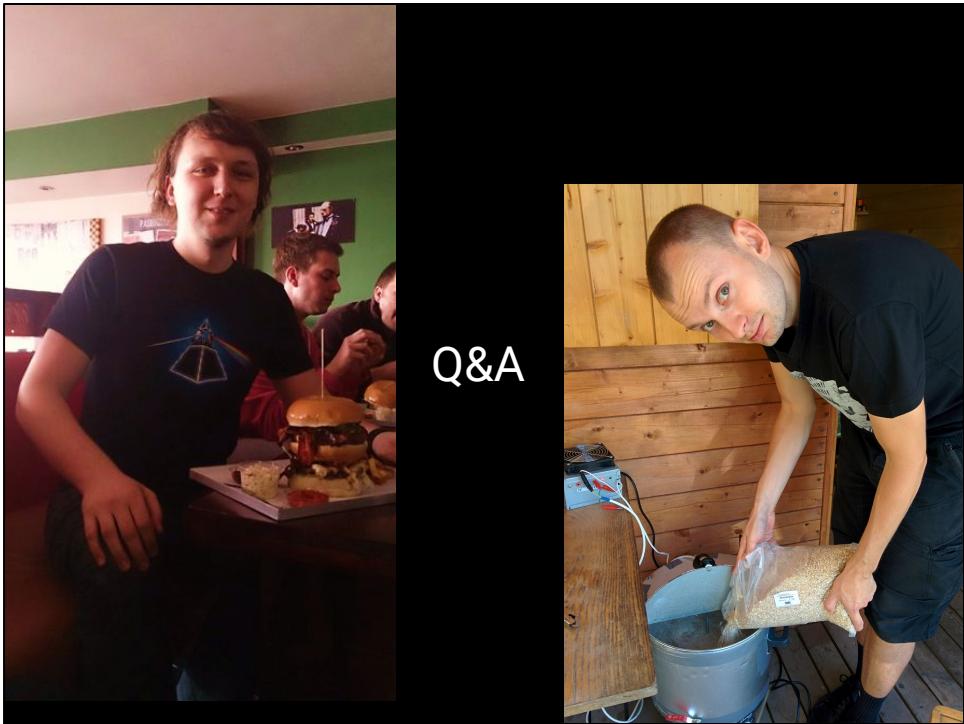
Most likely securing a cat pictures app isn't worth it. On the other hand – unsecured banking app could make a lot of other attacks easier.



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.



## Q&A

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.