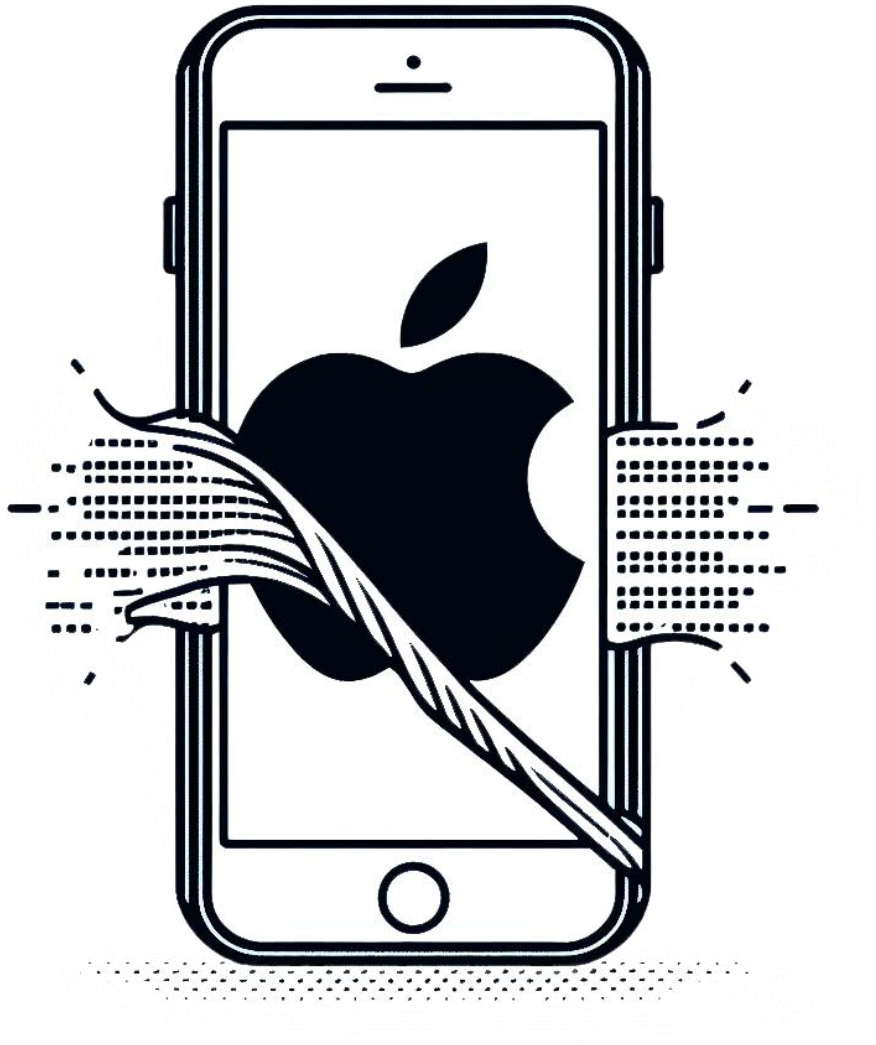


iObfuscate

Unraveling iOS Obfuscation Techniques

whoami

- Laurie Kirk
- Reverse Engineer at Microsoft
- Specialize in cross-platform malware with a focus on mobile malware
- Run YouTube channel @lauriewired
- Representing myself as an individual security researcher today (not representing Microsoft)

@lauriewired

Analysis Materials

- LaurieWired OBTS Github Repo
  - https://github.com/LaurieWired/ObjectiveByTheSea2023

Let’s make this...

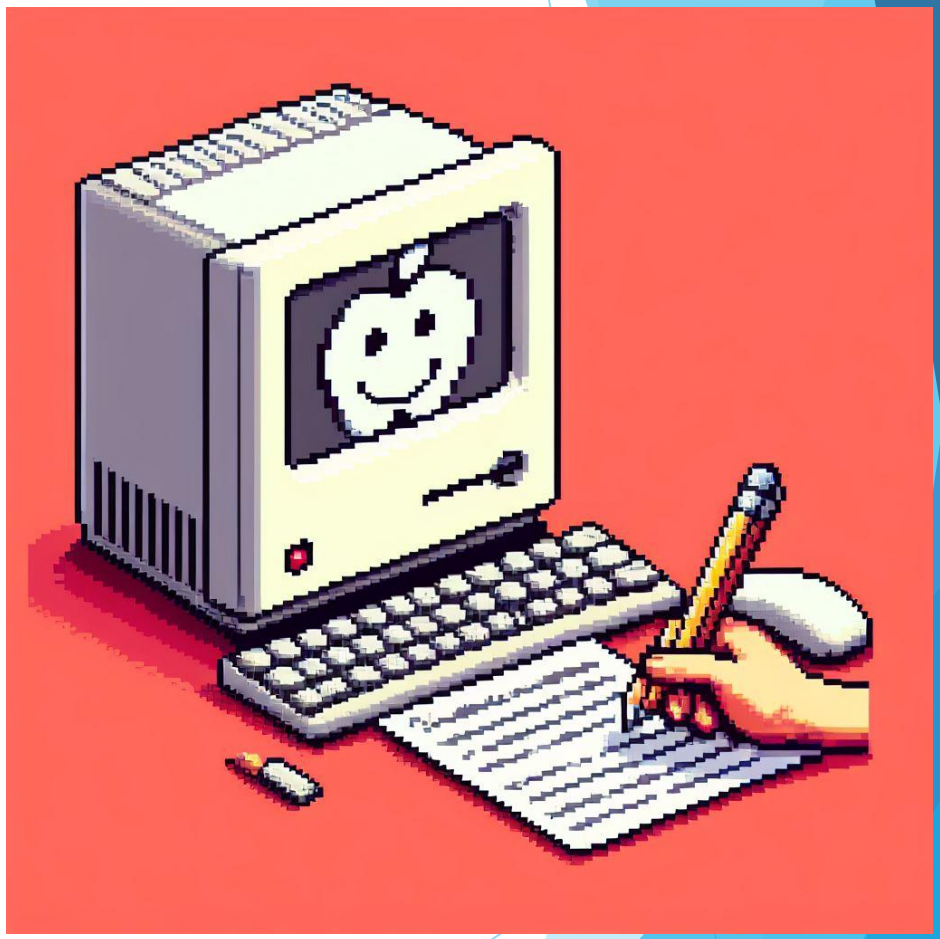
```
void _$s21ControlFlowFlattening11ContentViewVACycfC(void)
{
  _$s21ControlFlowFlattening03nonaB9FlattenedyyF();
  _$s21ControlFlowFlattening04withaB9FlattenedyyF();
  _$s21ControlFlowFlattening011withComplexaB9FlattenedyyF();
  return;
}
```

… look more like this

```
void ContentView.init(void)
{
  nonControlFlowFlattened();
  withControlFlowFlattened();
  withComplexControlFlowFlattened();
  return;
}
```

Agenda

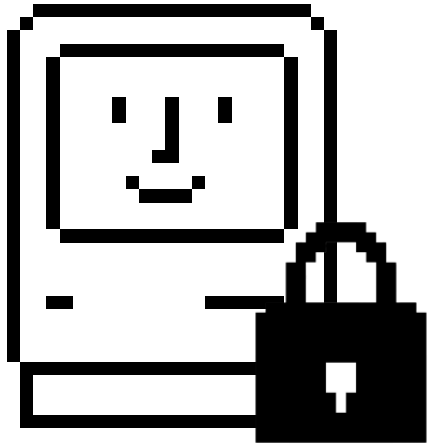
- Learn layered iOS application protections
- Reverse engineer obfuscated IPA files
- Provide new open-source iOS RE reference + deobfuscation scripts



What is obfuscation?

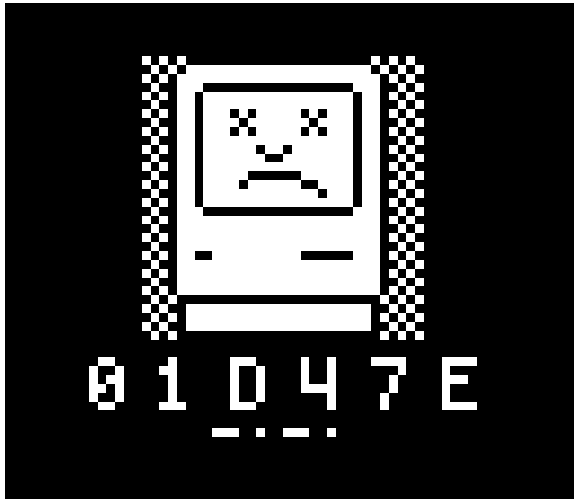
- Obfuscation obscures app data and functionality
- Common among all platforms
- Important for iOS applications
  - Contain full symbol data

Offensive and Defensive Obfuscation



Defensive

iOS developers protect their applications.



Offensive

Malware authors hide their malicious code.

Layers of IPA Protection

AppStore Encryption

Code Obfuscation

Runtime Protections

laurie

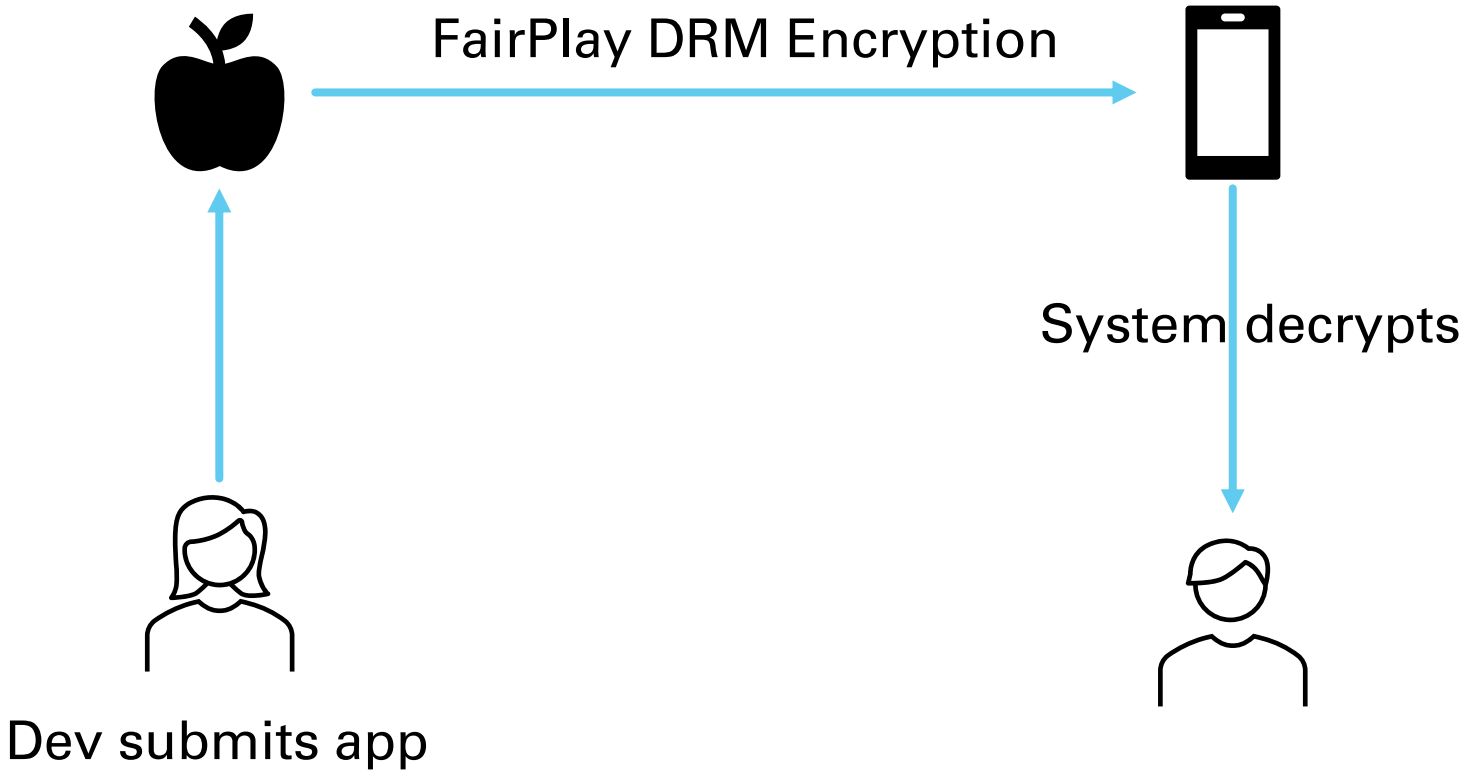
wired

Layer 01:
App Store Encryption

App Store Encryption

- Developer submits app
- Apple applies FairPlay encryption
- User downloads encrypted app
- iOS system decrypts app during installation

App Store Encryption



Pulling apps requires a jailbroken device.

To download a copy of the ipa file, use the `download` command.

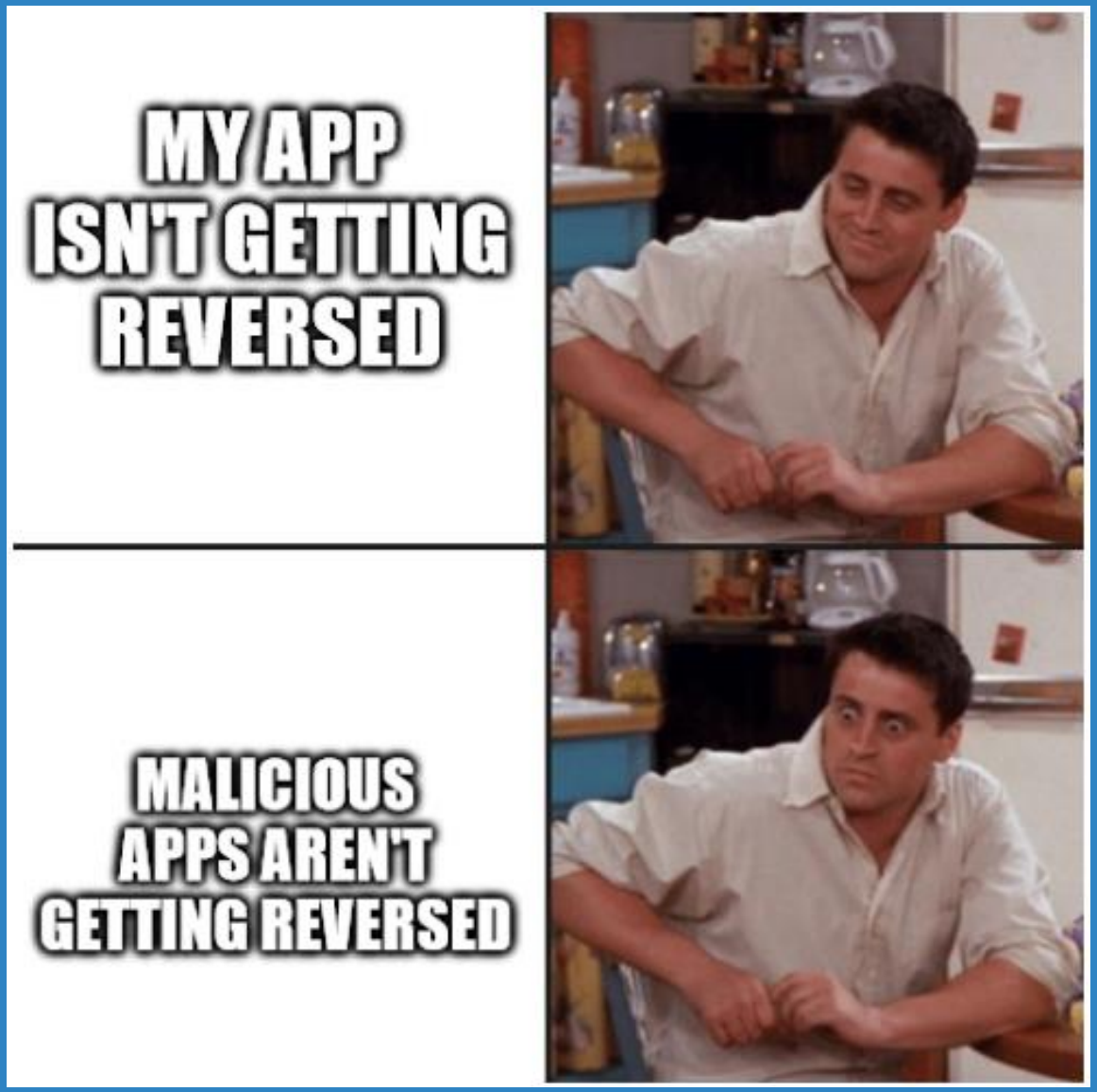
```
Download (encrypted) iOS app packages from the App Store

Usage:
  ipatool download [flags]
```

You might also burn your Apple ID.

Will my Apple ID get flagged for using this tool?

Maybe, but probably not. While this tool communicates with iTunes and the App Store directly, mimicking the behavior of iTunes running on macOS, I cannot guarantee its safety. I recommend using a throwaway Apple ID. Use this tool at your own risk.

Encryption effectively prevents reverse engineering.


MY APP
ISN'T GETTING
REVERSED
MALICIOUS
APPS AREN'T
GETTING REVERSED

laurie

wired

Layer 02:
Code Obfuscation

Code Protections

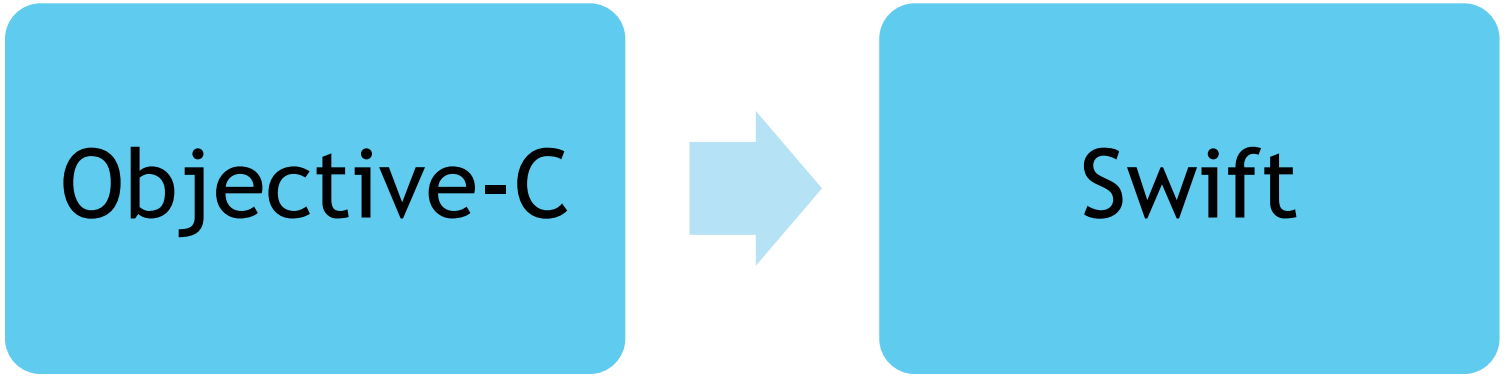
- Data encryption
- Identifier renaming
- Control-flow obfuscation
- Dead code

iOS code contains full symbol data.

Objective-C

```
void GraphViewController::viewDidLoad(undefined4 param_1)

{
  undefined4 uVar1;
  undefined4 uVar2;
  undefined4 uVar3;
  undefined4 local_24;
  class_t *local_20;

  local_20 = &objc::class_t::GraphViewController;
  local_24 = param_1;
  _objc_msgSendSuper2(&local_24,"viewDidLoad");
  uVar1 = _objc_msgSend(&_OBJC_CLASS_$_NSNumberFormatter,"new");
  _objc_msgSend(uVar1,"setNumberStyle:",1);
  _objc_msgSend(uVar1,"setMinimumFractionDigits:",0);
  _objc_msgSend(uVar1,"setMaximumFractionDigits:",0);
  uVar2 = _objc_msgSend(param_1,"graphView");
  _objc_msgSend(uVar2,"setYValuesFormatter:",uVar1);
  uVar2 = _objc_msgSend(&_OBJC_CLASS_$_NSDateFormatter,"new");
  _objc_msgSend(uVar2,"setTimeStyle:",0);
```

Apple moved to Swift



Swift has side benefits for obfuscation.

Swift Mangled Names

- Swift names are mangled by design
- Types, names, and parameters are jumbled together
- Ensures uniqueness

Mangled Swift Names

```
void _$s9TesHello20aB3AppV5$mainyyFZ(void) {
  long lVar1;

  lVar1 = _$s9TesHello20aB3AppVAC7SwiftUI0C0AAWl();
  _$s7SwiftUI3AppPAAE4mainyyFZ(&_$s9TesHello20aB3AppVN,lVar1);

  return;
}
```

Hands On:
Demangling Swift

Mangled names can be demangled.

Identifier Renaming

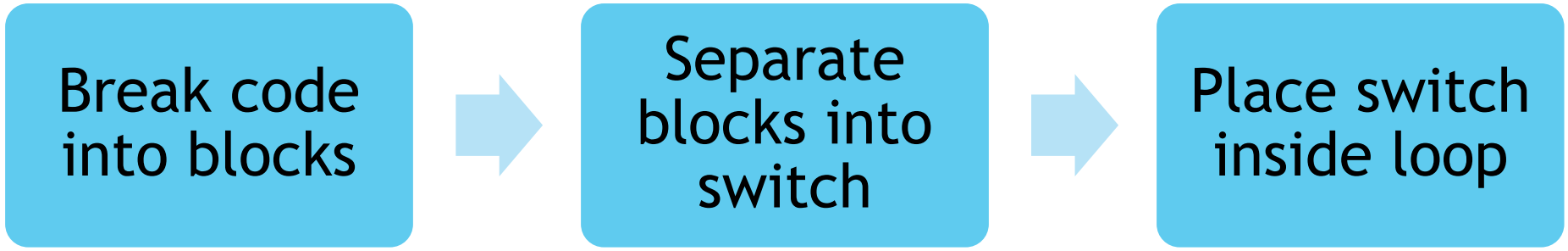
- Rename methods, classes, and variables
- Remove potential symbol data

Identifier Renaming

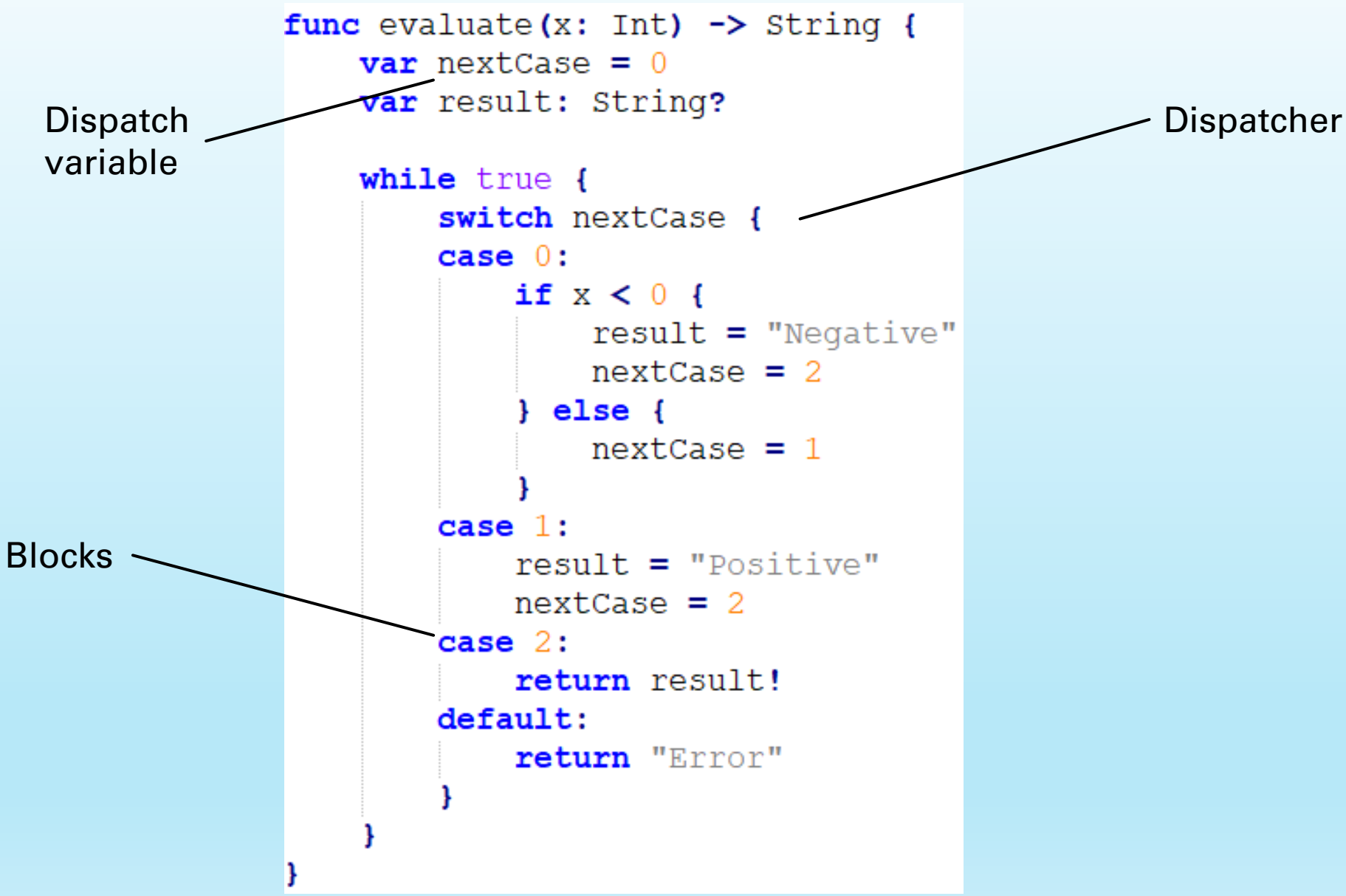
```
class O2aB3C_XyZb1: UIViewController {
    var x1Y2z3_AbC = 42

    override func n7M8L_o6P5Q() {
        super.n7M8L_o6P5Q()
        r4S5T_u3V2W()
        i9J0K(l0M9N: 1)
    }

    func i9J0K(l0M9N k9L8O: Int) {
        x1Y2z3_AbC += k9L8O
    }
}
```

Common obfuscation techniques also apply to iOS!

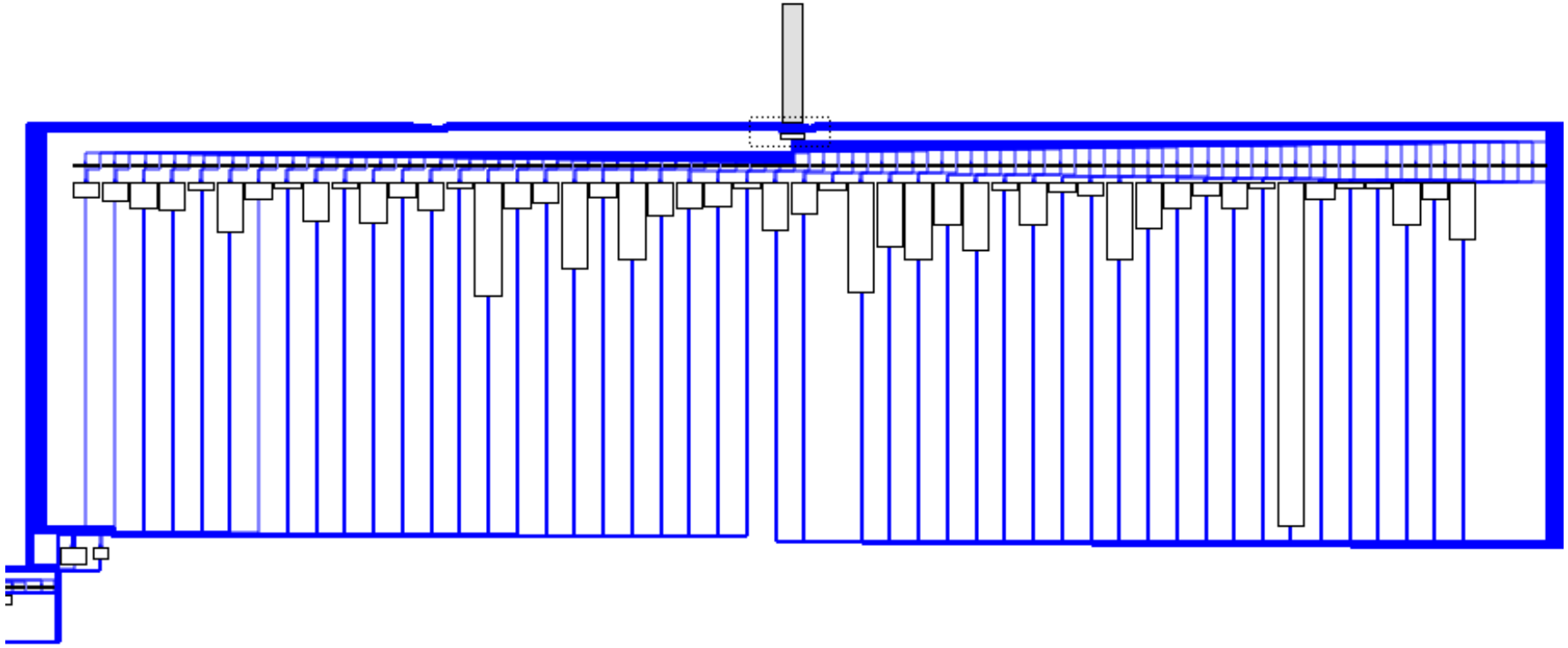
Control Flow Flattening



```
func evaluate(x: Int) -> String {
    if x < 0 {
        return "Negative"
    } else if x == 0 {
        return "Zero"
    } else {
        return "Positive"
    }
}
```


Dispatch variable
Dispatcher
Blocks
func evaluate(x: Int) -> String {
var nextCase = 0
var result: String?
while true {
switch nextCase {
case 0:
if x < 0 {
result = "Negative"
nextCase = 2
} else {
nextCase = 1
}
case 1:
result = "Positive"
nextCase = 2
case 2:
return result!
default:
return "Error"
}
}
}

Flattened Graph View



Dead Code

- Inserting unreferenced code
- Pads application binary
- Particularly effective in iOS
  - Asynchronous nature of events

laurie

wired

Layer 03:
Runtime Protections

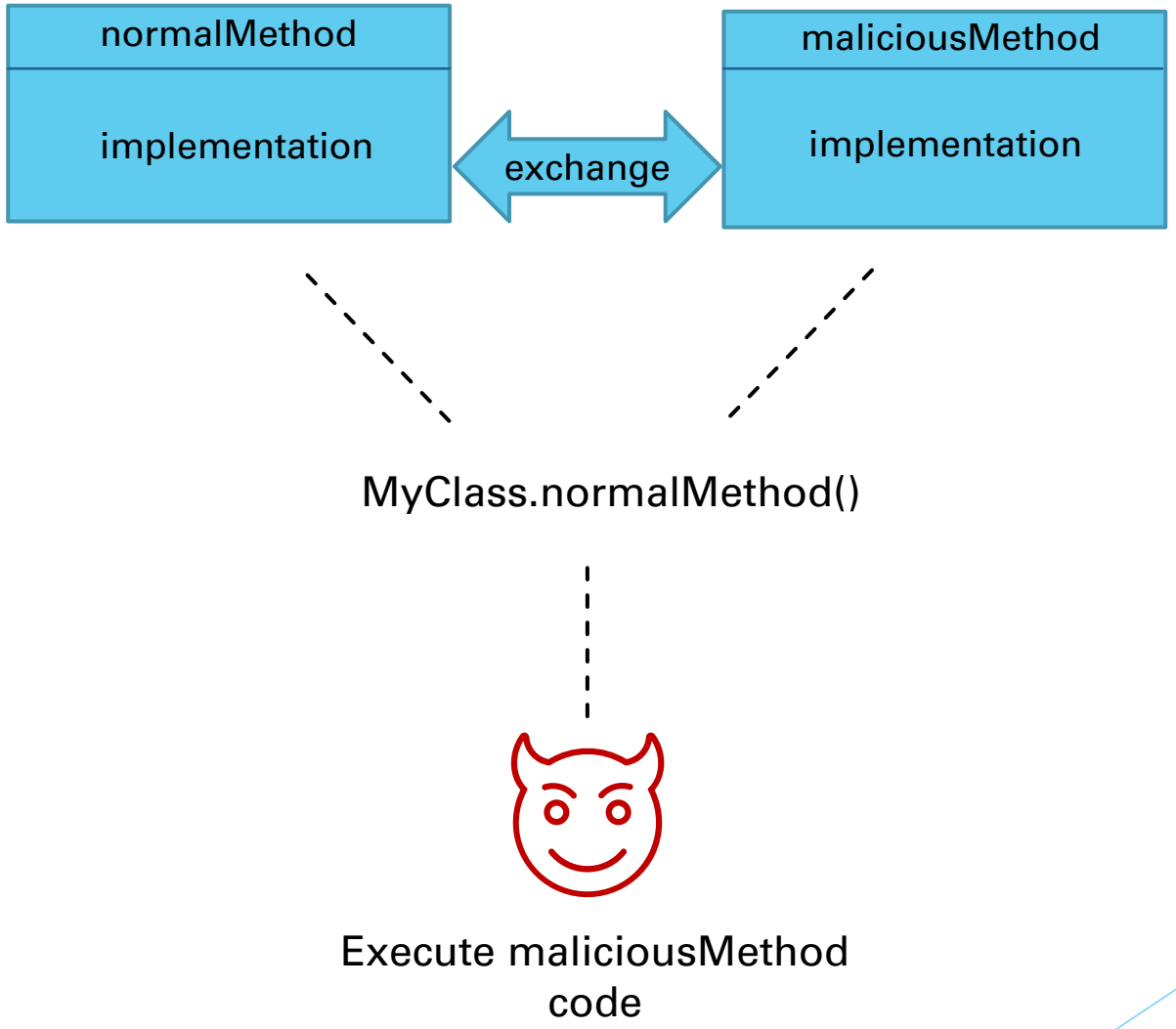
Dynamic Obfuscation Techniques

- Swizzling
- Anti-debug
- Anti-tampering
- Virtualization-based obfuscators

Swizzling

- Modifies method implementation at runtime
- Primarily used in Objective-C
- Possible in Swift as well

Swizzling Two Methods



Goontact Malware

- Spyware discovered in 2018
- Exfiltrates photos, messages, device info
- Phishing site convinces victim to sideload IPA

Hands On:
Examining Swizzling in Goontact


Pointers
Swizzling

Standard Anti-Debug / Anti-Tampering

- Prevent debugger from attaching
- Detect presence of Frida server
- Simulator check

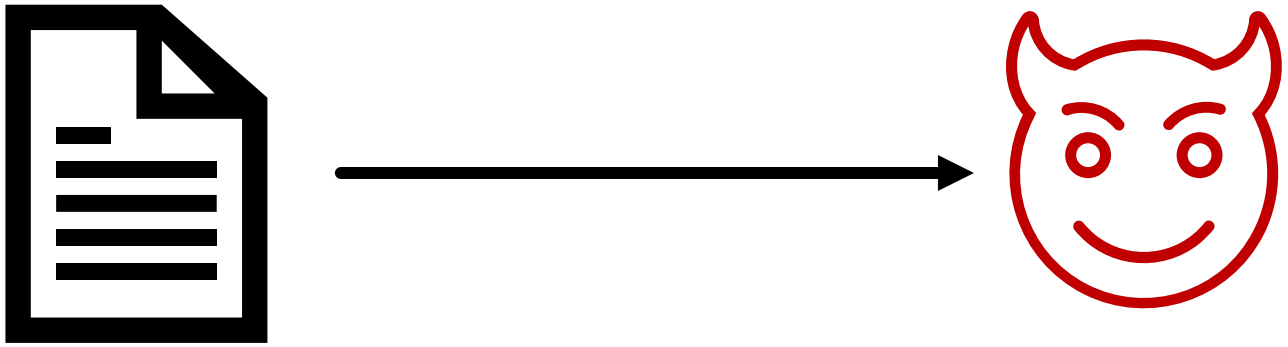
Jailbreak Detection

- Avoids execution on a jailbroken device
- Reverse engineers can pull app files
- Check device heuristics

Checking URL Schemes

```
private static func checkURLSchemes() -> CheckResult {
    let urlSchemes = [
        "undecimus://",
        "sileo://",
        "zbra://",
        "filza://",
        "activator://"
    ]
    return canOpenUrlFromList(urlSchemes: urlSchemes)
}
```

Checking File Permissions



Can I access this?

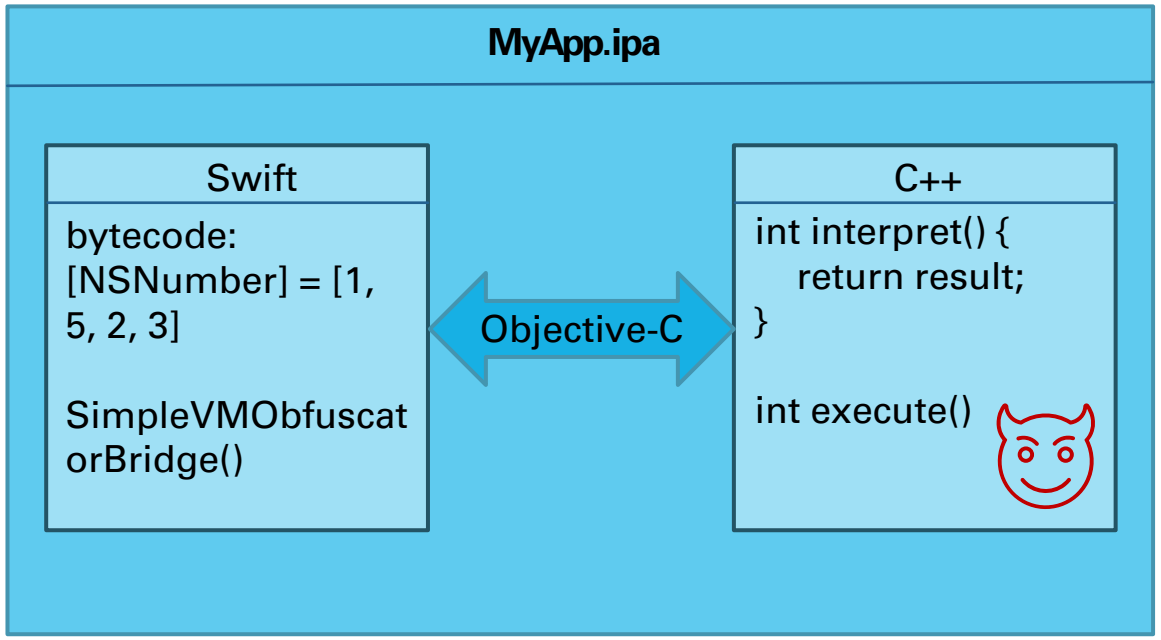
Guess I’m jailbroken

VM-Based Obfuscators

VM-Based Obfuscators

- Native instructions are replaced with virtual ones
- Obfuscator stores and hides secret virtual instructions
- Virtual instructions are translated to native at runtime

iOS VM-Based Obfuscation Architecture



```cpp
class SimpleVMObfuscator {
private:
    std::vector<int> bytecode;

    int interpret() {
        int result = 0;
        for (int i = 0; i < bytecode.size(); i++) {
            switch (bytecode[i]) {
                case 1:
                    result += bytecode[++i];
                    break;
                case 2:
                    result -= bytecode[++i];
                    break;
                default:
                    break;
            }
        }
        return result;
    }

public:
    SimpleVMObfuscator(std::vector<int> bc): bytecode(bc) {}

    int execute() {
        return interpret();
    }
};
```

Interpreter

1 represents addition

Virtual machine

Many popular apps employ
custom VM-based obfuscators.


VM
RESEARCHERS
Spyware

Real-World Examples

The iOS App Store can host real malware.

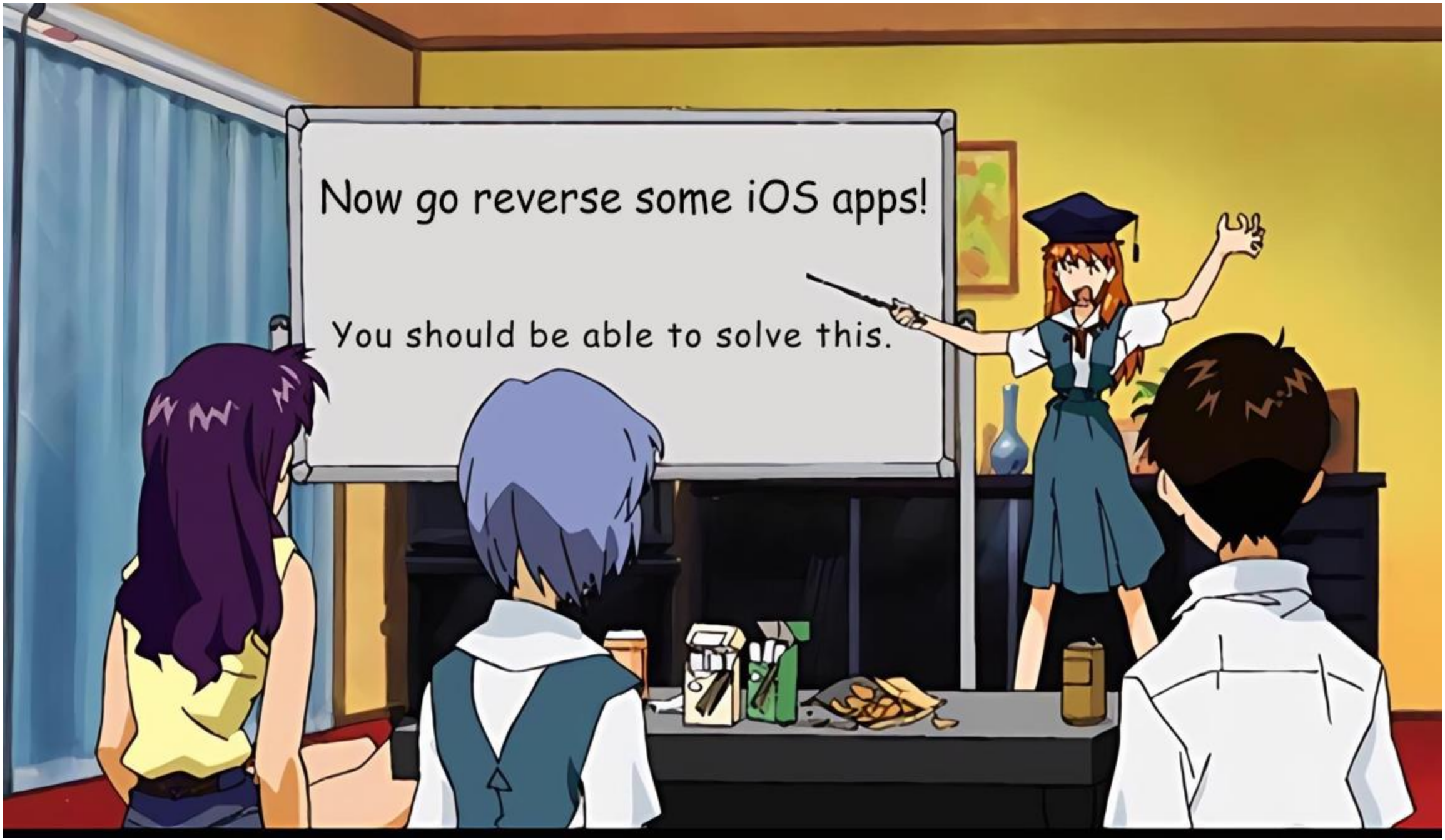
InstaAgent

- Instagram profile assistant
- Harvested Instagram credentials
  - Uploaded in cleartext to a C2 ☹
- Around 500,000 downloads

XcodeGhost

- Modified version of Xcode
- Inserted malware into legitimate iOS apps
- Apps were hosted in official App Store

Undiscovered?


Now go reverse some iOS apps!
You should be able to solve this.

Thank you!

iOS Reverse Engineering Repo

- LaurieWired iOS RE Github Repo
  - https://github.com/LaurieWired/iOS_Reverse_Engineering

Bonus Section

Static Obfuscators

- Obfuscator-LLVM
  - https://github.com/obfuscator-llvm/obfuscator/tree/llvm-4.0
- Sirius Obfuscator
  - Renames variables
  - https://github.com/Polidea/SiriusObfuscator
- https://github.com/pjebs/Obfuscator-iOS

Open-Source Runtime Protectors

- iOS Security Suite
  - https://github.com/securing/IOSSecuritySuite
- Runtime Application Self-Protection (RASP)
  - https://github.com/talsec/Free-RASP-iOS

Defensive Obfuscation

- Source code protection
- App size optimizations
- Performance enhancements
- Tamper defense
- Vulnerability hardening

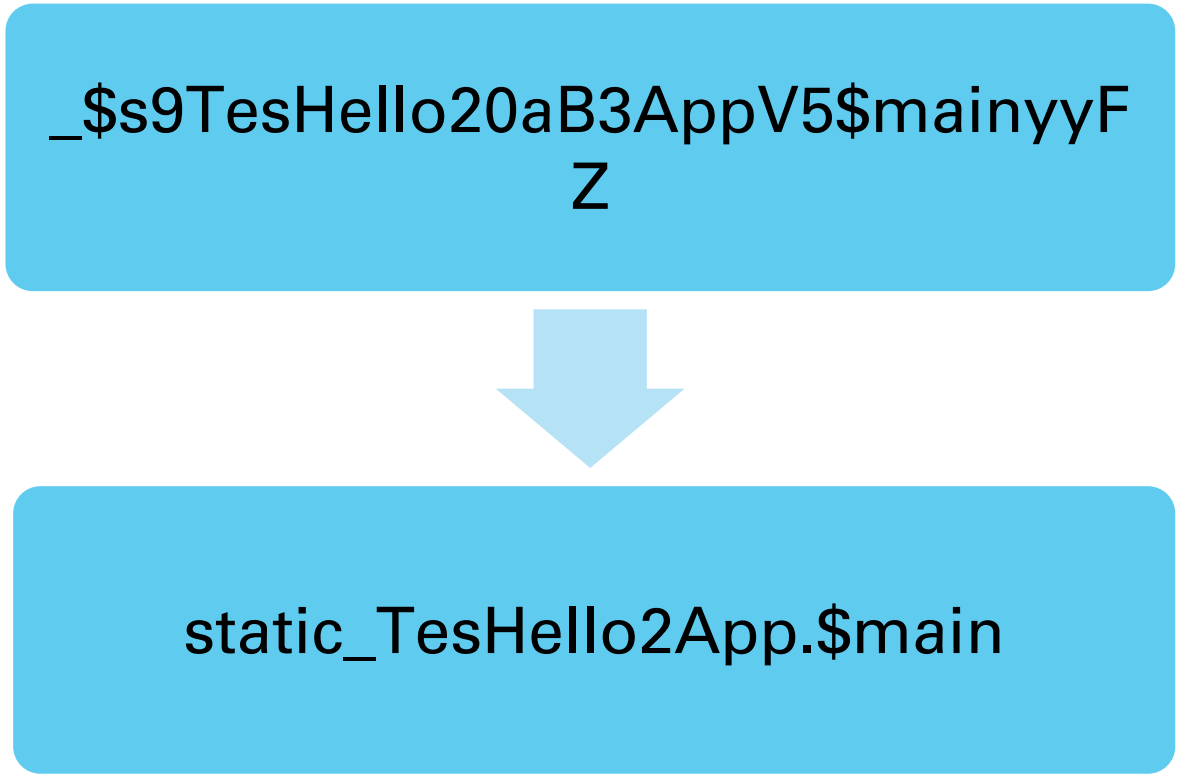
Offensive Obfuscation

- Prevent Reverse Engineering
- Evade antivirus detection
- Conceal malicious code
- Mask application origin

After Demangling

```
void static_TesHello2App.$main(void) {
  long lVar1;

  lVar1 = lazy_protocol_witness_table_accessor_for_type_TesHello2App();
  static_App.main(&type_metadata_for_TesHello2App,lVar1);

  return;
}
```

Demangling Swift Example



Obfuscator-LLVM

- Built on top of LLVM compiler
- Adds obfuscation during compilation process
- Obfuscates code for many platforms

iOS libraries also try to protect against swizzling.

Pros and Cons of VM-Based OBfuscation

Pros

- Thwarts static analysis
- Hides malicious code

Cons

- Challenging to implement
- Reduces performance