

目录

前言	1.1
iOS安全概览	1.2
iOS安全防护	1.3
iOS系统的安全设计	1.3.1
代码混淆	1.3.2
Obfuscator-LLVM	1.3.2.1
ios-class-guard	1.3.2.2
反调试	1.3.3
防止导出头文件	1.3.4
越狱检测	1.3.5
反爬	1.3.6
SSL证书绑定	1.3.6.1
数据存储加密	1.3.7
代码动态运行	1.3.8
设备指纹	1.3.9
附录	1.4
参考资料	1.4.1

iOS安全与防护

- 最新版本: v0.9
- 更新时间: 20221030

简介

介绍iOS安全与防护。包括iOS安全的概览，在安全领域所属的范畴；以及iOS的安全的具体防护手段，包括但不限于iOS系统本身的安全设计、代码混淆、反调试、防止逆向导出头文件、越狱检测、反爬、数据存储加密、代码动态执行、设备指纹等，反爬中涉及到SSL证书绑定。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

HonKit源码

- [crifan/ios_security_protect: iOS安全与防护](#)

如何使用此HonKit源码去生成发布为电子书

详见：[crifan/honkit_template: demo how to use crifan honkit template and demo](#)

在线浏览

- [iOS安全与防护 book.crifan.org](#)
- [iOS安全与防护 crifan.github.io](#)

离线下载阅读

- [iOS安全与防护 PDF](#)
- [iOS安全与防护 ePUB](#)
- [iOS安全与防护 Mobi](#)

版权和用途说明

此电子书教程的全部内容，如无特别说明，均为本人原创。其中部分内容参考自网络，均已备注了出处。如发现有侵权，请通过邮箱联系我 admin 艾特 crifan.com，我会尽快删除。谢谢合作。

各种技术类教程，仅作为学习和研究使用。请勿用于任何非法用途。如有非法用途，均与本人无关。

鸣谢

感谢我的老婆陈雪的包容理解和悉心照料，才使得我 crifan 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

其他

作者的其他电子书

本人 crifan 还写了其他 150+ 本电子书教程，感兴趣可移步至：

[crifan/crifan_ebook_readme: Crifan的电子书的使用说明](#)

关于作者

关于作者更多介绍，详见：

[关于CrifanLi李茂 – 在路上](#)

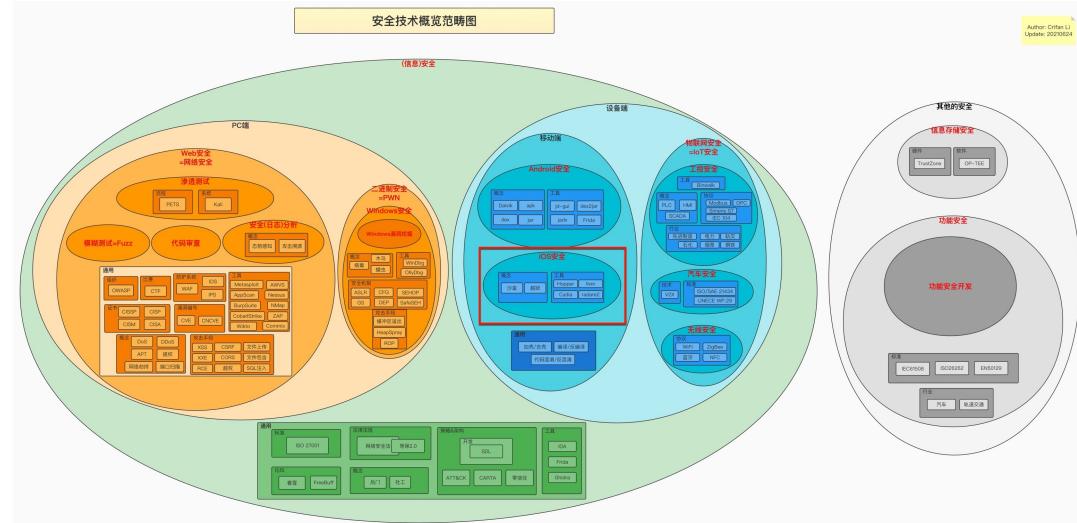
crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新： 2022-11-07 22:51:01

iOS安全概览

- iOS安全

- 所属范畴

- 从大的信息安全范畴，属于： 信息安全 中 设备端 中 移动端 中 iOS安全



- 详见：[信息安全概览 安全概览](#)
- iOS开发，从 安全 领域的 攻防 角度分：
 - 反方的 攻 : iOS逆向 = iOS破解 = iOS攻击
 - 正方的 防 : iOS安全 = iOS防护

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-21 14:41:20

iOS安全防护

iOS的逆向和破解，与之相对的正向的防护，叫：[iOS安全和防护](#)

- iOS安全防护的 手段 = 防护的角度 = 具体涉及内容
 - iOS系统
 - 本身已有的不少安全方面的设计和防护
 - 防止逆向后轻易看懂代码逻辑 = 代码角度： 代码混淆
 - 被反编译后，也只能看到乱码的函数
 - 部分防止被破解，被猜测到核心逻辑
 - 反调试：用技术手段实现不让app被调试
 - 增加了逆向后调试的可能性或难度
 - [反调试和反反调试 · iOS逆向开发：动态调试 \(crifan.org\)](#)
 - 防止被导出头文件
 - 把 ObjC 换 Swift，以增加被破解难度？
 - 越狱检测：检测设备是否已越狱
 - 如果已越狱，则不让运行或功能受限
 - [iOS逆向开发：越狱检测和反越狱检测 \(crifan.org\)](#)
 - 网络传输的数据的防护
 - 防抓包：从抓包角度，用技术手段，防止被抓包
 - SSL证书的 ssl pinning = 证书绑定
 - 甚至额外做本地证书校验
 - iOS端的数据存储：加密，提高安全
 - 隐藏核心代码逻辑
 - 把核心逻辑和代码，变成动态下载再运行
 - 设备指纹
 - 其他
 - 异常检测
 - 重签名检测
 - 动态库注入检测
 - 钩子检测
 - 扫描工具
 - [fortify](#)
 - 侧重于代码的安全漏洞
 - [coverity](#)
 - 侧重于代码质量

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook 最后更新：2022-10-30 21:21:37

iOS系统安全

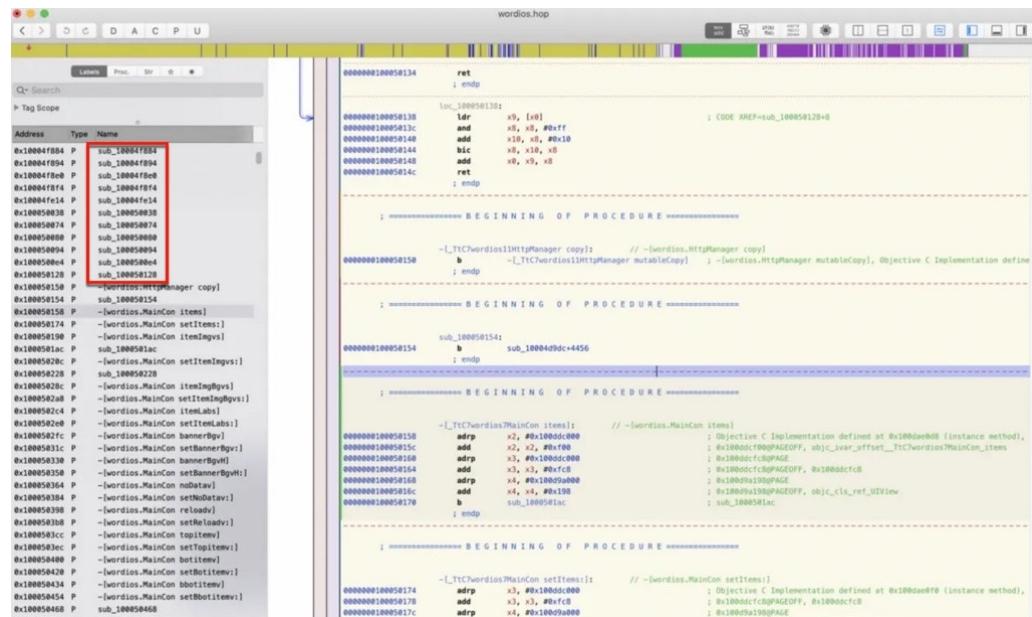
- iOS系统安全 = iOS操作系统级别的安全
 - 概述
 - 总体上说，iOS系统比Android系统更安全
 - 当然也更封闭
 - 当然，安全只是相对的，并没有绝对的安全
 - 高级黑客还是可以破解和黑你的iPhone的
 - 详解
 - 技术层面
 - iOS系统本身
 - 安全设计=安全机制
 - 可信启动链
 - 代码签名
 - 沙盒执行环境
 - 权限隔离和数据加密
 - 更严格的版本控制
 - 不能降级(安装低版本的iOS操作系统)
 - 该策略使得iOS设备一旦升级后，就只能停留在当前或者最新版本
 - 有效避免了操作系统版本碎片化问题，减少了已公开漏洞的影响范围
 - 严格掌控的应用市场
 - 杜绝向第三方应用开放高级数据访问权限，限制了iOS恶意应用的传播和能力

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-23 14:25:16

代码混淆

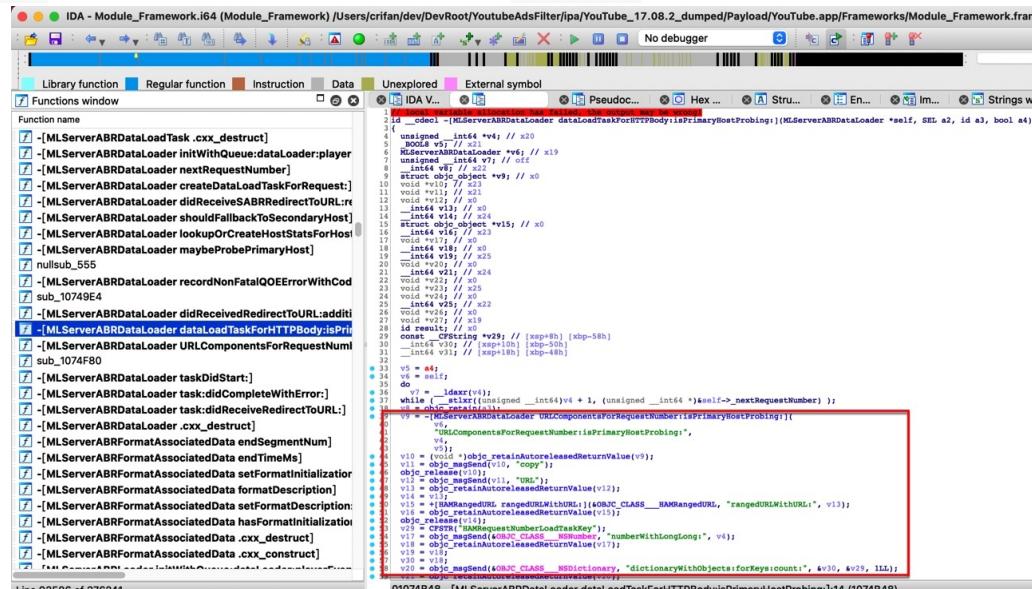
iOS安全从代码角度，可以去做： 代码混淆

- 代码混淆 目的
 - 更好地保护代码， 增加逆向破解难度 = 不被轻松地恶意分析破解
- 代码混淆 优缺点
 - 优点
 - 无须变动项目源码
 - 功能灵活可选， 根据需要自由组合
 - 缺点
 - 导致安装包体积增大
 - 混淆后的代码有会被编译优化掉的风险
 - 提交审核（AppStore）存在被拒的风险
- 代码混淆方式=子功能模块
 - obf=Obfuscator 类
 - bcf = Bogus Control Flow =虚假块=伪控制流
 - fla = cff = Control Flow Flattening =控制流展开=控制流平坦化
 - split =基本块分割
 - sub = Instructions Substitution =指令膨胀=指令替换
 - acd =anti class-dump=反class-dump
 - indibran =indiret branch=基于寄存器的相对跳转， 配合其他加固可以彻底破坏IDA/Hopper的伪代码(俗称F5) = 故意制造堆栈不平衡， 不能F5， 函数内利用寄存器跳转BR X12
 - strcry =字符串加密
 - funcwra =函数封装
 - 插入垃圾指令
- iOS代码混淆工具
 - Obfuscator-LLVM = ollvm
 - ios-class-guard = iOS Class Guard
 - Hikari
 - 基于ollvm
 - 主页
 - <https://github.com/HikariObfuscator/Hikari>
 - <https://github.com/HikariObfuscator/Hikari/wiki>
- iOS代码混淆后的效果
 - 导出头文件后， 函数名变乱码
 - 比如：
 - money 变成 xadsf32
 - showMoney 变成 AFAdsa123
 - iOS逆向后看到的代码中的函数， 都是无名的函数
 - 比如：
 - Hopper逆向app后， 有很多函数名都是： sub_xxx， 就表示， 该函数被混淆了



■ 对比

- YouTube：没有代码混淆 -> IDA伪代码中能看到明文的 `objc_msgSend` 的函数调用

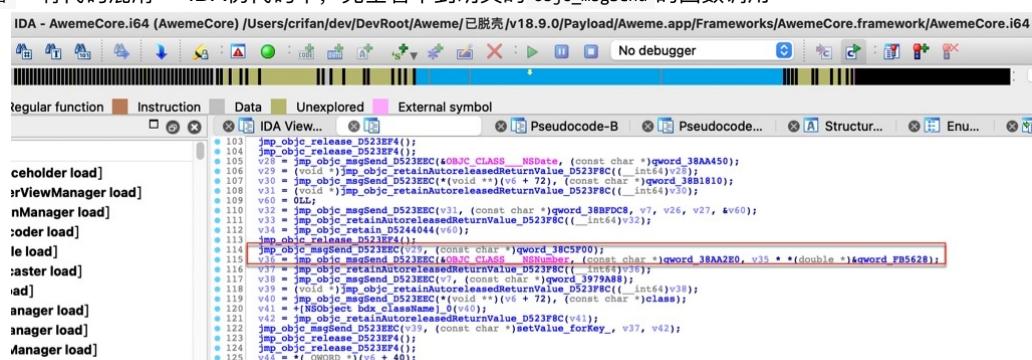


■ 都可以看到类似于：

- ```
■ objc_msgSend(&OBJC_CLASS_NSGestureRecognizer, "initWithLong:", 1)
```

#### ■ 这种近乎原始代码的效果

- 抖音：有代码混淆 -> IDA伪代码中，完全看不到明文的 objc\_msgSend 的函数调用



- 注：其中的 `jmp_objc_msgSend_xxxx(xxx, (char *)qword_xxx, xxx)` 这类的调用中的 `jmp_objc_msSend_xxxx` 是在逆向后，搞懂函数调用后，改名优化后的伪代码



# Obfuscator-LLVM

此处介绍，可以用于iOS代码混淆的工具：[Obfuscator-LLVM](#)

- `Obfuscator-LLVM = ollvm`
  - 是什么：基于LLVM的代码混淆工具
  - 谁开发的：瑞士伊夫尔东莱班的应用科学与艺术大学信息安全小组
  - 什么时候：2010年6月
  - 目的：增强软件代码安全
    - 基于LLVM的编译套件
    - 通过防篡改(tamper-proofing)和代码混淆(code obfuscation)
  - 支持语言
    - C, C++, Objective-C, Ada 和 Fortran
  - 支持架构
    - x86, x86-64, PowerPC, PowerPC-64, ARM, Thumb, SPARC, Alpha, CellSPU, MIPS, MSP430, SystemZ 和 XCore
  - 代码混淆方式
    - control flow flattening = 控制流扁平化 = 控制流平坦化
      - 语法：`-mllvm -fla`
    - instruction substitution = 指令替换
      - 语法：`-mllvm -sub`
    - bogus control flow = 控制流伪造 = 虚假控制流程
      - 语法：`-mllvm -bcf`
  - 资料
    - GitHub
      - obfuscator-llvm/obfuscator
        - <https://github.com/obfuscator-llvm/obfuscator>
      - 文档入口
        - Home · obfuscator-llvm/obfuscator Wiki
          - <https://github.com/obfuscator-llvm/obfuscator/wiki>
      - 快速上手
        - obfuscator/GettingStarted.rst at llvm-4.0 · obfuscator-llvm/obfuscator
          - <https://github.com/obfuscator-llvm/obfuscator/blob/llvm-4.0/docs/GettingStarted.rst>

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2022-10-21 15:35:09

## ios-class-guard

- `ios-class-guard` = iOS Class Guard
  - Polidea/ios-class-guard: Simple Objective-C obfuscator for Mach-O executables.
    - <https://github.com/Polidea/ios-class-guard>

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-23 14:43:23

# 反调试

TODO:

- 【整理】iOS反越狱相关：反调试 反反调试
- 【已解决】iOS中正向调用ptrace的PT\_DENY\_ATTACH防止调试
- 【已解决】iOS中的caddr\_t类型的定义
- 【已解决】debugserver启动iOS的app抖音报错：Segmentation fault 11
- 
- 【已解决】抖音反反调试：把二进制AwemeCore的svc 0x80指令替换成nop指令
- 【已解决】Mac中用IDA实现抖音二进制AwemeCore的svc 0x80替换成nop指令

iOS安全防护技术之一是

- 反调试
  - 防止你的iOS的app被别人逆向调试
  - 加了反调试后的效果
    - 现象：别人用调试工具去调试你的iOS的app，会报错退出，从而无法继续调试
    - 举例
      - debugserver 会报错： Segmentation fault 11
- 反调试实现手段
  - ptrace的PT\_DENY\_ATTACH
    - ptrace() = ptrace + PT\_DENY\_ATTACH
    - syscall() = syscall + ptrace + PT\_DENY\_ATTACH
    - inline asm=内联 (ARM) 汇编 = svc 0x80 + ptrace + PT\_DENY\_ATTACH
  - sysctl的getpid
  - 其他
    - SIGSTOP
    - task\_get\_exception\_ports
    - isatty
    - ioctl

## ptrace的PT\_DENY\_ATTACH

### ptrace + PT\_DENY\_ATTACH

- ptrace(PT\_DENY\_ATTACH, 0, 0, 0);

具体代码：

- main.m

```
#import <UIKit/UIKit.h>
#import "AppDelegate.h"
#import "CrifanLib.h"

int main(int argc, char * argv[]) {
 // anti-debug
 iOS_antiDebug_ptrace();

 NSString * appDelegateClassName;
 @autoreleasepool {
 // Setup code that might create autoreleased objects goes here.
 appDelegateClassName = NSStringFromClass([AppDelegate class]);
 }
}
```

```
 return UIApplicationMain(argc, argv, nil, appDelegateClassName);
}
```

- CrifanLib.h

```
void iOS_antiDebug_ptrace(void);
```

- CrifanLib.c

- <https://github.com/crifan/crifanLib/blob/master/c/crifanLib.c>

```
/*=====
iOS Anti-Debug
=====*/
typedef int (*func_ptrace)(int request, pid_t pid, caddr_t addr, int data);

#ifndef PT_DENY_ATTACH
#define PT_DENY_ATTACH 31
#endif // !defined(PT_DENY_ATTACH)

void iOS_antiDebug_ptrace(void) {
// ptrace(PT_DENY_ATTACH, 0, 0, 0);

// void* libHandle = dlopen(0, RTLD_GLOBAL | RTLD_NOW);
// libHandle void * 0xfffffffffffff
// if (NULL == libHandle) {
// char* errStr = dlerror();
// printf("Failed to open 0, error: %s", errStr);
// } else {
// func_ptrace ptrace_ptr = dlsym(libHandle, "ptrace");
// if (NULL != ptrace_ptr){
// ptrace_ptr(PT_DENY_ATTACH, 0, 0, 0);
// }
// dlclose(libHandle);
// }

func_ptrace ptrace_ptr = dlsym(RTLD_SELF, "ptrace");
// ptrace_ptr func_ptrace (libsystem_kernel.dylib'__ptrace) 0x000000018cee2df8
if (NULL != ptrace_ptr){
 // ptrace_ptr(PT_DENY_ATTACH, 0, 0, 0);
 ptrace_ptr(PT_DENY_ATTACH, 0, NULL, 0);
}
}
```

## syscall + ptrace + PT\_DENY\_ATTACH

```
syscall(SYS_ptrace, PT_DENY_ATTACH, 0, NULL, 0);
```

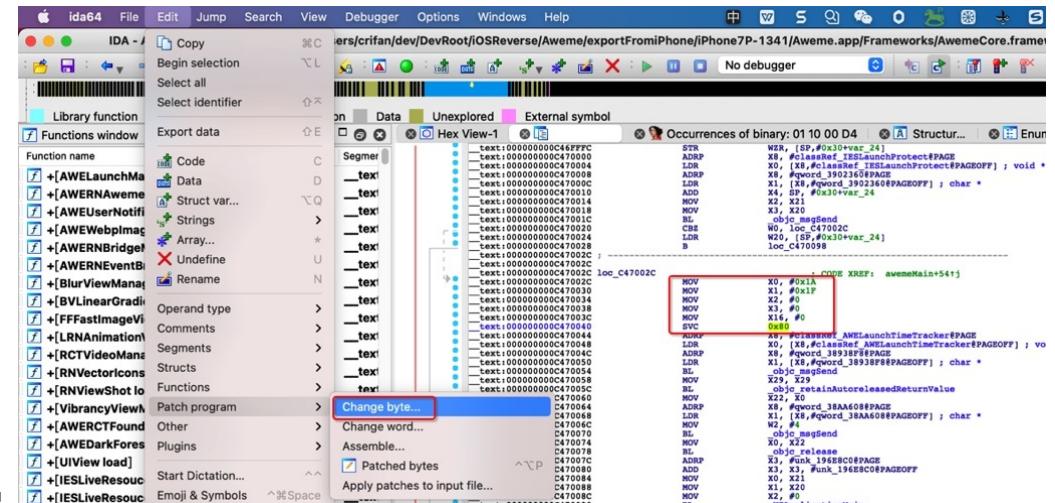
## svc 0x80 + ptrace + PT\_DENY\_ATTACH

- inline asm=内联 (ARM) 汇编 = svc 0x80 + ptrace + PT\_DENY\_ATTACH
  - 具体实现:

```
 mov x0, #26 // ptrace = 0x1A
 mov x1, #31 // PT_DENY_ATTACH = 0x1F
 mov x2, #0
 mov x3, #0
 mov x16, #0
 svc #0x80
```

- 效果: 编译后的iOS的app的二进制中, 加了反调试的汇编代码逻辑

- 举例
  - 抖音



## sysctl的getpid

实现代码：

```

int name[4]; //里面放节码。查询的信息
name[0] = CTL_KERN; //内核查询
name[1] = KERN_PROC; //查询进程
name[2] = KERN_PROC_PID; //传递的参数是进程的ID
name[3] = getpid(); //获取当前进程ID

size_t infoSize = sizeof(struct kinfo_proc);
struct kinfo_proc kernelInfoProc; //接受查询结果的结构体
memset(&kernelInfoProc, 0, infoSize);

sysctl(name, 4, &kernelInfoProc, &infoSize, NULL, 0);

```

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2022-10-30 18:35:41

## 防止导出头文件

iOS的ObjC的特性，使得有机会，在app运行期间，导出ObjC的类的头文件，得到众多的iOS的类的函数定义和具体属性。

而如果把iOS的 `objc` 代码，改为 `Swift` 代码，就可以避免被导出类的头文件，增加了破解难度。

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2022-10-21 14:51:23

## 越狱检测

iOS设备的另外一个安全防护手段是：

- 越狱检测
  - 检测当前iOS设备是否已越狱
    - 然后施行对应策略：不允许继续运行，或者功能受限，或者只是后台记录当前设备存在风险，供后续决策提供参考
  - 对应技术，详见：
    - [iOS逆向开发：越狱检测和反越狱检测 \(crifan.org\)](#)

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2022-10-30 21:19:55

# 反爬

iOS防护中，还有一部分的手段叫做：反爬

- 反爬
  - 目的：
    - 防止别人的爬虫爬取你的app的数据
    - 防止别人能用抓包工具抓包你的app，看到明文的数据
  - 手段
    - https
      - SSL pinning = 证书绑定
      - 证书校验

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2022-10-27 22:38:58

# SSL证书绑定

- 背景

- iOS的app，多数和服务器端通信，也是基于常见 HTTP / HTTPS 协议。
- 而iOS逆向中往往用，用抓包工具，比如 Charles 去抓包分析你的网络请求。
  - 其中因为抓包可以直接看到明文数据而多数已不用 HTTP 了。
  - 而改用加密的 HTTPS ，而抓包 HTTPS ，一般来说无法直接看到明文数据，只能看到加密后的乱码。
  - 但是采用了根证书信任等手段，往往也可以抓包到 HTTPS 的明文。
  - 而最新的手段一般是：采用 证书绑定 = SSL pinning ， app内部会对于SSL的证书和本地的证书做绑定和校验，使得抓包工具比如Charles的证书，无法通过验证，从而导致无法抓包到 HTTPS 的明文。
- 而iOS防护的话，不希望被抓包，被看到HTTPS的明文，所以往往也会去采用： 证书绑定 = SSL pinning
  - 而证书绑定中，更高级和更严格一点的手段是： 本地证书校验 ?
    - 比如 抖音 内部就做了证书校验
    - 注：想要逆向的话，可以通过写hook代码绕过证书校验，即可实现抓包 HTTPS 看到明文数据。

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2022-10-30 16:49:00

## 数据存储加密

iOS的app的本地端，一般都会涉及到存储很多数据，其中有些可能是敏感数据。

其中一种增加iOS的app的安全的做法是：把本地保存的数据加密。

- iOS端本地的数据库
  - 常用：SQLite
    - SQLite加密：增加破解难度
    - SQLCipher
      - <https://github.com/sqlcipher/sqlcipher>

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2022-10-30 21:22:02

## 代码动态执行

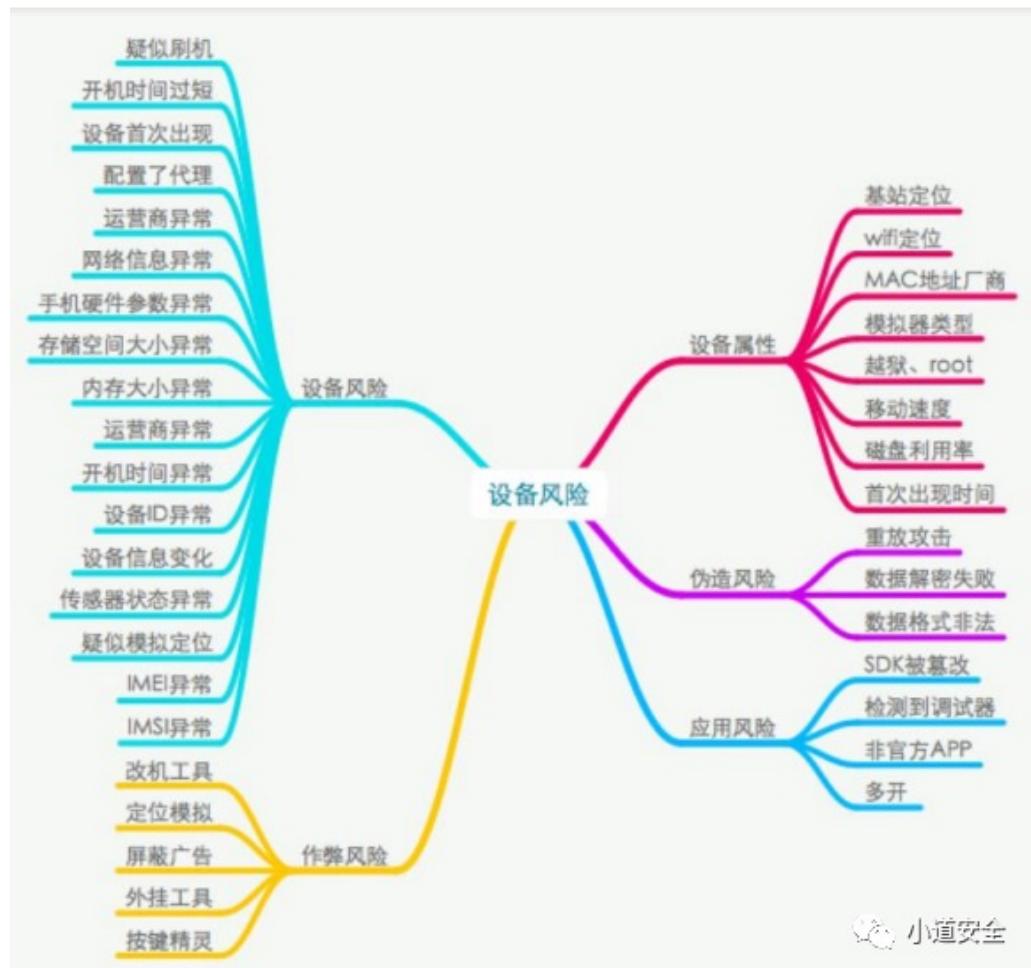
把iOS的app中，核心的代码，比如核心通信逻辑、加密算法、参数生成逻辑，放到动态代码，即从服务器端动态下载要运行的代码，然后在本地运行核心逻辑，生成要的值。

此种iOS安全防护手段，一般称为： 代码动态执行

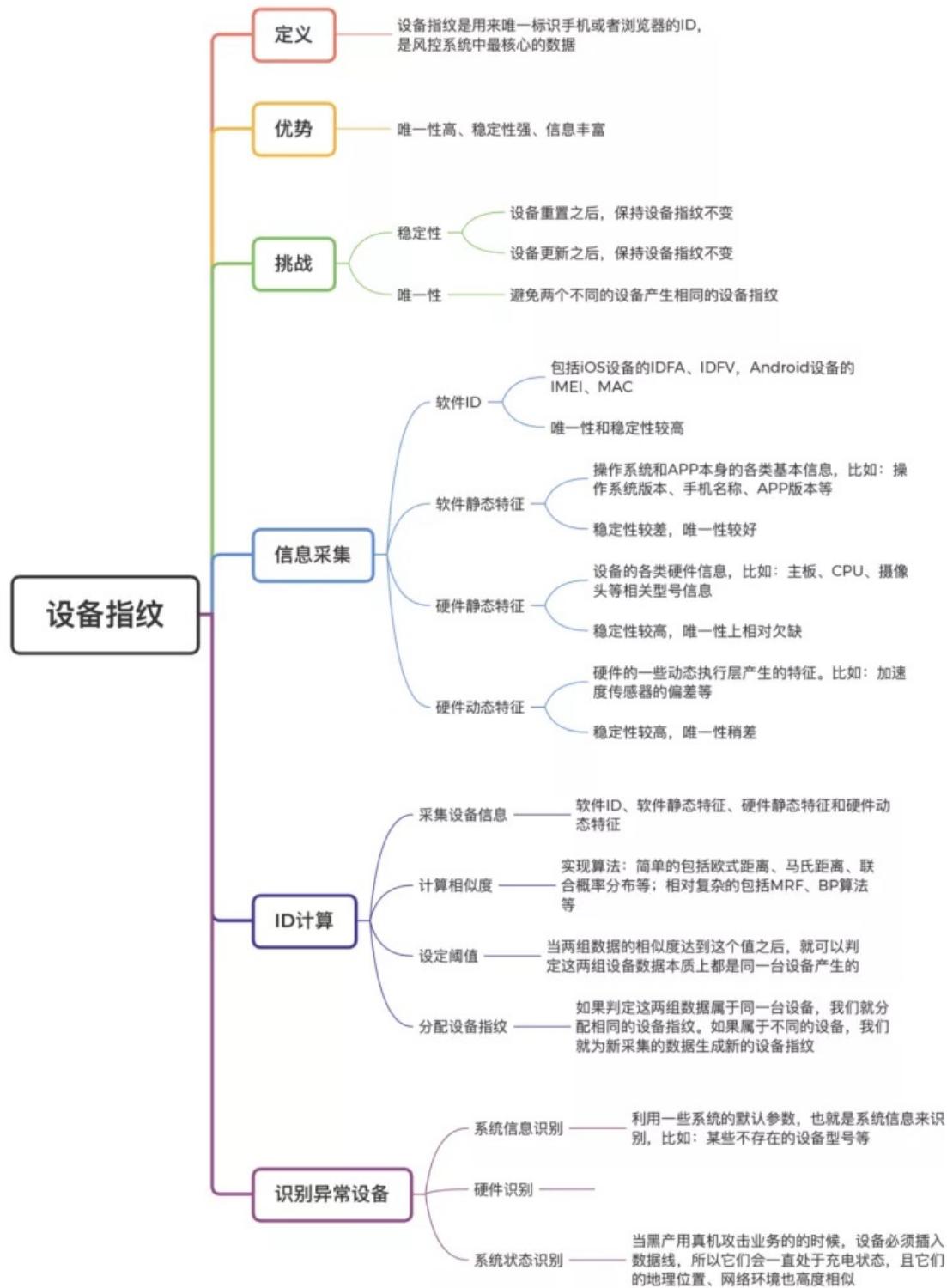
crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2022-10-21 14:53:37

# 设备指纹

- 设备指纹 = device fingerprint
- 背景
  - 各种非法设备和风险，希望能识别并检测出来
    - 模拟器、刷机改机=设备伪造、root越狱、劫持注入、自动注册、羊毛党等



- iOS设备
  - 唯一标识符发展历史
    - UDID
    - OpenUDID
    - MAC
    - IDFV
    - IDFA
  - 但都不够好-》可以被改机改掉，不能保证稳定和唯一
- 是什么：一个唯一的标识符，字符串
  - 设备指纹是指可以用于唯一标识出该设备的设备特征或者独特的设备标识
- 用途：设备信息唯一性、渠道流量检测、风险设备识别、通用风控策略
- 概述

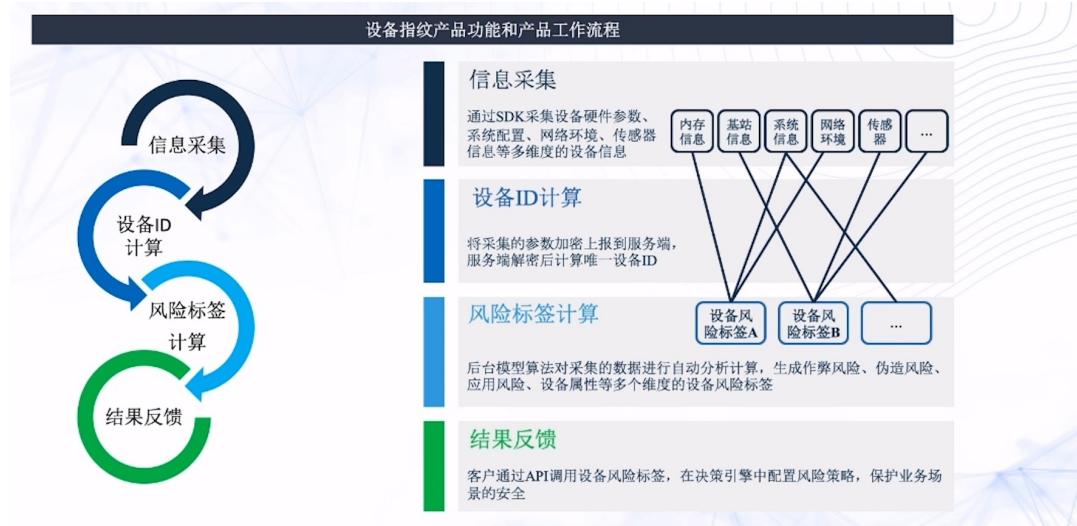




- 生成流程

- 文字
  - 采集信息
    - MAC
    - IMSI
    - IMEI
    - ICCID
    - BSSID
    - 等
  - 计算相似度
    - 常见算法
      - 简单的：欧式距离、马氏距离、联合概率分布等
      - 相对复杂的：MRF（马尔可夫随机场）、BP算法（置信度传播算法）等

- 图



## 附录

下面列出相关参考资料。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-21 11:52:29

## 参考资料

- [Class-dump导出头文件没有属性和方法名怎么处理 - 技能讨论 - 睿论坛](#)
- [iOS逆向攻防实战 - 掘金 \(juejin.cn\)](#)
- [\[原创\]某App去混淆-iOS安全-看雪论坛-安全社区|安全招聘|bbs.pediy.com](#)
- [iOS代码混淆工具 — Hikari（支持Xcode14以下全部版本混淆）-Apibug](#)
- [基于llvm的iOS代码混淆工具 -- Hikari - 简书](#)
- [ios手动代码混淆函数和变量名基本原理和注意事项教程\(含demo\)\\_小手琴师的博客-CSDN博客\\_xcode代码混淆](#)
- [iOS混淆--OLLLVM在iOS中的实践（逻辑混淆） - 简书](#)
- [SQLCipher之攻与防 - 腾讯云开发者社区-腾讯云](#)
- [使用 sqlcipher 加密解密 sqlite3 数据库 - 维唯为为](#)
- [独家食用指南系列|Android端SQLCipher的攻与防新编 - 掘金](#)
- [数据库开源框架之sqlcipher加密数据库51CTO博客开源数据库](#)
- [浅谈设备指纹技术和应用 - 网安 \(wangan.com\)](#)
- [以攻击者角度学习顶像风控设备指纹产品 - 我是小三 - 博客园 \(cnblogs.com\)](#)
- [设备指纹 - 顶象文档中心 \(dingxiang-inc.com\)](#)
- [Apple隐私政策趋严，设备指纹路在何方？\\_网易易盾 \(163.com\)](#)
- [iOS设备指纹的前世今生 - 知乎 \(zhihu.com\)](#)
- [29 | 设备指纹：面对各种虚拟设备，如何进行对抗？\\_wx60cc441e1698a的技术博客\\_51CTO博客](#)

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-27 23:06:08