

目录

前言	1.1
Frida概览	1.2
代码和架构	1.2.1
文档和资料	1.2.2
安装和升级	1.3
安装Frida	1.3.1
升级Frida	1.3.2
使用Frida	1.4
Frida典型使用逻辑	1.4.1
frida	1.4.2
调试目标方式	1.4.2.1
写js脚本	1.4.2.2
典型使用方式	1.4.2.3
iOS的ObjC	1.4.2.4
ObjC的参数	1.4.2.4.1
ObjC的变量类型	1.4.2.4.2
数据类型	1.4.2.5
NativePointer	1.4.2.5.1
Frida的Stalker	1.4.2.6
frida-trace	1.4.3
help语法	1.4.3.1
frida-tools	1.4.4
frida-ps	1.4.4.1
frida-ls	1.4.4.2
frida-ls-devices	1.4.4.3
其他相关	1.4.5
frida-server	1.4.5.1
Frida用法举例	1.5
经验心得	1.6
hook函数	1.6.1
frida	1.6.1.1
Interceptor	1.6.1.1.1
Stalker	1.6.1.1.1.1
frida-trace	1.6.1.2

常用iOS函数	1.6.1.3
js	1.6.2
console.log日志	1.6.2.1
自己编译frida-server	1.6.3
常见问题	1.7
通用	1.7.1
Process terminated	1.7.1.1
unable to find process with name	1.7.1.2
--no-pause和--pause	1.7.1.3
Bad access due to invalid address	1.7.1.4
ValueError file descriptor cannot be a negative integer	1.7.1.5
iOS	1.7.2
need Gadget to attach on jailed iOS	1.7.2.1
导致iPhone重启	1.7.2.2
Waiting for USB device to appear	1.7.2.3
unable to intercept function at	1.7.2.4
Android	1.7.3
java.lang.ClassNotFoundException	1.7.3.1
基于Frida的工具	1.8
Frida的用途	1.9
反调试	1.9.1
辅助反混淆	1.9.2
绕过参数加密	1.9.3
逆向app	1.9.4
附录	1.10
参考资料	1.10.1

逆向调试利器：Frida

- 最新版本： v4.0.0
- 更新时间： 20240721

简介

介绍支持Android、iOS等多个平台的通用逆向工具：Frida。先是Frida概览，包括Frida的代码和架构，以及相关文档和案例和资料。再介绍如何在PC端和移动端即iOS和安卓端的安装和升级Frida。然后是如何使用Frida，先介绍Frida的典型使用逻辑，然后再去介绍frida命令行工具，其中包括通用的逻辑，比如调试目标的方式，以及写js脚本；然后是典型的使用方式，以及此处iOS逆向涉及到的ObjC的内容，包括ObjC的参数和变量类型；接着是数据类型，包括NativePointer。接着是高级的Stalker。接着介绍frida-trace，以及frida-tools工具集合，包括frida-ps、frida-ls、frida-ls-devices等；接着介绍其他相关的内容，包括frida-server等。接着整理一些Frida开发期间的经验和心得，包括hook函数方面的，包括frida和frida-trace，frida中的Interceptor、Stalker、常用iOS函数等。其他还有工具类的函数、js以及其中的console.log，和自己编译frida-server，和其他常见问题和报错。接着整理基于Frida的工具。以及Frida的一些常见用途，包括反调试、辅助反混淆、绕过参数加密、逆向各种app等。最后贴出参考资料。

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/reverse_debug_frida: 逆向调试利器：Frida](#)

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

- [逆向调试利器：Frida book.crifan.org](#)
- [逆向调试利器：Frida crifan.github.io](#)

离线下载阅读

- [逆向调试利器：Frida PDF](#)
- [逆向调试利器：Frida ePUB](#)
- [逆向调试利器：Frida 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：

2024-07-21 16:41:57

Frida概览

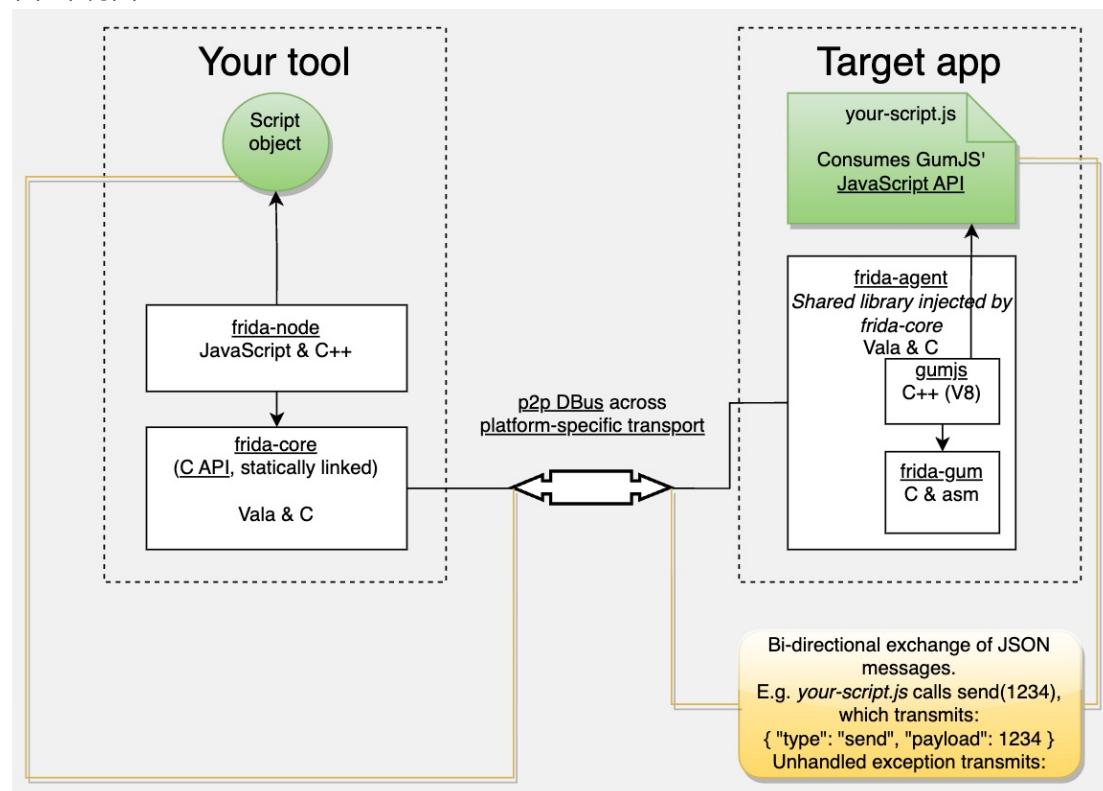
- Frida
 - 概述
 - 一款基于 `python` + `javascript` 的 `hook`框架
 - `Android`、`ios` 的 `app`逆向等领域中，最常用的工具之一
 - A world-class Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers
 - Inject JavaScript to observe and reprogram running programs on Windows, macOS, GNU/Linux, iOS, watchOS, tvOS, Android, FreeBSD, and QNX
 - 主要用法：iOS逆向期间，用 `frida+js`脚本 和 `frida-trace`、`Frida` 的 `Stalker` 等工具，去动态调试代码逻辑
 - 核心原理
 - 主要使用动态二进制插桩(DBI)技术
 - 将外部代码注入到现有的正在运行的二进制文件中，从而让它执行额外操作
 - 支持哪些额外操作
 - 访问进程内存
 - 在应用程序运行时覆盖函数
 - 从导入的类调用函数
 - 在堆上查找对象实例并使用
 - Hook、跟踪和拦截函数等
 - 注：调试器也能完成相应工作，不过非常麻烦，比如各种反调试
 - 功能和特点
 - Scriptable
 - Inject your own scripts into black box processes. Hook any function, spy on crypto APIs or trace private application code, no source code needed. Edit, hit save, and instantly see the results. All without compilation steps or program restarts.
 - Portable
 - 支持多种运行平台/系统：`Android` / `ios` / `Linux` / `Win` / `macos` 等
 - Works on Windows, macOS, GNU/Linux, iOS, Android, and QNX. Install the Node.js bindings from npm, grab a Python package from PyPI, or use Frida through its Swift bindings, .NET bindings, Qt/Qml bindings, or C API.
 - Free
 - Frida is and will always be free software (free as in freedom). We want to empower the next generation of developer tools, and help other free software developers achieve interoperability through reverse engineering.
 - Battle-tested
 - We are proud that NowSecure is using Frida to do fast, deep analysis of mobile apps at scale. Frida has a comprehensive test-suite and has gone through years of rigorous testing across a broad range of use-cases.
- 主页
 - 官网
 - <https://frida.re/>
 - 文档

- <https://frida.re/docs/home/>
- Github
 - <https://github.com/frida/frida>
- 作者: oleavr = Ole André Vadla Ravnås
 - Github
 - <https://github.com/oleavr>
 - 所属公司
 - NowSecure
 - <https://www.nowsecure.com/>

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:
2023-07-13 14:53:50

代码和架构

- Frida的架构
 - 总体架构
 - client=客户端
 - frida的各种工具
 - frida: frida主体工具本身，用的最多。
 - frida-tools: (Win/Mac等) PC端常用到的，除了frida之外的，frida-trace、frida-ps、frida-ls等命令行工具
 - frida-trace: 按顺序和带缩进的打印所有函数调用堆栈，极大地方便调试内部逻辑
 - server=服务端
 - frida-server
 - 原理图=架构图



- 其中可见很多关键的点=frida中的子项
 - frida-code
 - frida-node
 - frida-agent
 - frida-gum
 - GumJS
- 对应着源码中的各个子项目：详见下面内容。

Frida源码

- Frida源码

- 总入口: <https://github.com/frida/frida>
 - core : [frida-core](#)
 - Frida core library intended for static linking into bindings
 - gum : [frida-gum](#)
 - Cross-platform instrumentation and introspection library written in C
 - This library is consumed by frida-core through its `JavaScript` bindings, `GumJS`.
 - tools : [frida-tools](#)
 - Frida CLI tools
 - bindings
 - python -> [frida-python](#)
 - Node.js -> [frida-node](#)
 - Swift -> [frida-swift](#)
 - .NET -> [frida-clr](#)
 - GO -> [frida-go](#)
 - Rust -> [frida-rust](#)
 - QT/qml -> [frida-qml](#)
 - 其他
 - Frida支持多个移动端的互操作，所以有分别的内部的互操作相关的内容
 - iOS端的: [frida-objc-bridge](#)
 - Objective-C runtime interop from Frida
 - Android端的: [frida-java-bridge](#)
 - Java runtime interop from Frida
 - [Capstone](#)
 - 记得是：Frida内部用到了Capstone，但是有些额外的内容要微调，所以单独fork了Capstone源码，自己同步更新了

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:

2023-06-23 23:07:22

文档和资料

- 官网文档入口: [Frida Docs](#)
 - 入门指南=Getting Started
 - 快速入手: [Quick-start guide](#)
 - 安装: [Installation](#)
 - 操作模式: [Modes of Operation](#)
 - Gadget: [Gadget](#)
 - 内部原理: [Hacking](#)
 - Stalker: [Stalker](#)
 - 演讲文档资料: [Presentations](#)
 - 教程=Tutorials
 - 通用
 - 函数: [Functions](#)
 - 消息: [Messages](#)
 - 移动端
 - [iOS](#)
 - [Android](#)
 - 案例=Examples
 - PC端
 - [Windows](#)
 - [macOS](#)
 - [Linux](#)
 - 移动端
 - [iOS](#)
 - [Android](#)
 - 其他
 - [JavaScript](#)
 - 工具=Tools
 - [frida 命令行工具: Frida CLI](#)
 - [frida-ps](#)
 - [frida-trace](#)
 - [frida-discover](#)
 - [frida-ls-devices](#)
 - [frida-kill](#)
 - [gum-graft](#)
 - API文档=API Reference
 - [JavaScript API](#)
 - [C API](#)
 - [Swift API](#)
 - [Go API](#)
 - 其他细节=Miscellaneous
 - [最佳实践: Best Practices](#)
 - [排错: Troubleshooting](#)

- 自己编译Frida=参与Frida开发: [Building](#)
- 占用系统空间: [Footprint](#)
- 起源和发展=Meta
 - [GSoC Ideas 2015](#)
 - [GSoD Ideas 2023](#)
 - [History](#)
- 其他
 - 教程
 - [Frida basics - Frida HandBook \(learnfrida.info\)](#)
 - [Frida HandBook](#)

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:
2023-06-27 22:55:46

安装和升级

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：
2023-06-22 22:02:05

安装Frida

在能使用Frida之前，先要去安装Frida：

- PC端

- Mac

- 安装 `frida` => 使得有frida命令行工具可用

```
pip3 install frida
```

- 安装后查看版本

```
pip show frida
```

- 安装 `frida-tools` => 使得有frida-trace、frida-ps、frida-ls等命令行工具可用

```
pip3 install frida-tools
```

- 安装后查看版本

```
pip show frida-tools
```

- 额外说明

- 如果前面没有先单独安装 `frida`，则安装 `frida-tools` 时会自动安装所依赖的 `frida`

- 【可选】

- 安装Frida的 `gadget` => 使得后续使用frida-ios-dump时而不报错 need Gadget to attach on jailed iOS

- 下载gadget库文件

- 从[Frida的Github的release](https://github.com/frida/frida/releases)页面中，下载对应版本的 `frida-gadget` 的 `dylib`

- 举例

- <https://github.com/frida/frida/releases/download/16.0.8/frida-gadget-16.0.8-ios-universal.dylib.gz>

- 解压得到： `frida-gadget-16.0.8-ios-universal.dylib`

- 拷贝到对应位置： `~/.cache/frida/gadget-ios.dylib`

- 举例

```
cp frida-gadget-16.0.8-ios-universal.dylib /Users/crifan/.cache/frida/gadget-ios.dylib
```

- 安装： Frida的Node.js bindings -> 用得到，才需要安装， 默认不用安装

```
npm install frida
```

- 移动端

- (越狱) iPhone

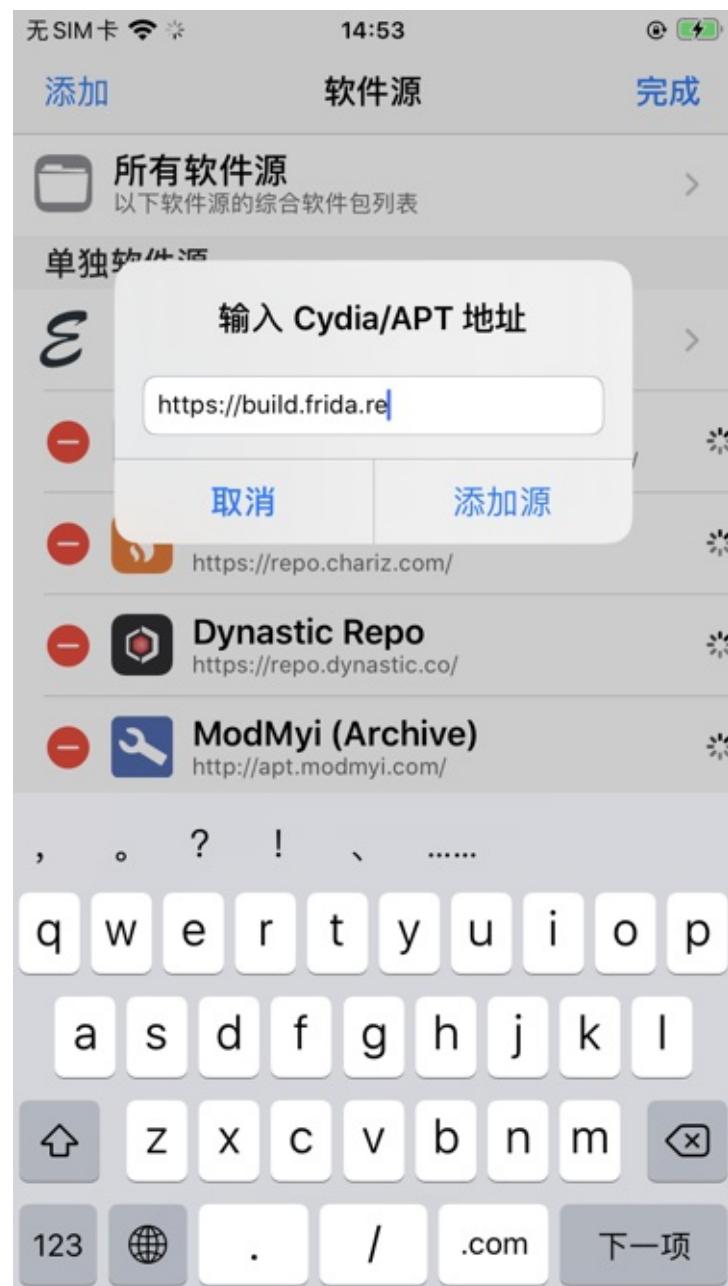
- 安装步骤

- 用 Sileo / Cydia , 添加软件源: <https://build.frida.re> , 搜索并安装 frida , 即可

- Sileo



- Cydia



- Frida安装后

- 确保iPhone端 frida-server 已经正常运行

```
iPhone8-150:~ root# ps -A | grep frida
2150 ??      0:00.02 /usr/sbin/frida-server
2194 ttys000  0:00.00 grep frida
```

- 插件详情页

- Sileo



 **Frida**
Frida Developers 更改

[详情](#) [更新日志](#)

Observe and reprogram running programs.

软件源

 [Frida](#) >

已安装的软件包

版本	16.0.8
显示软件包内容	>

re.frida.server (16.0.8)

 精选  新闻  软件源  软件包  搜索

■ Cydia



- 已安装的文件

- 列表

- /Library/LaunchDaemons/re.frida.server.plist
 - /usr/lib/frida/frida-agent.dylib
 - /usr/sbin/frida-server

- 图:

- Sileo

16:06



< 返回

已安装的文件

▼ /

▼ Library/

▼ LaunchDaemons/

re.frida.server.plist

▼ usr/

▼ lib/

▼ frida/

frida-agent.dylib

▼ sbin/

frida-server



精选



新闻



软件源



软件包



搜索

■ Cydia

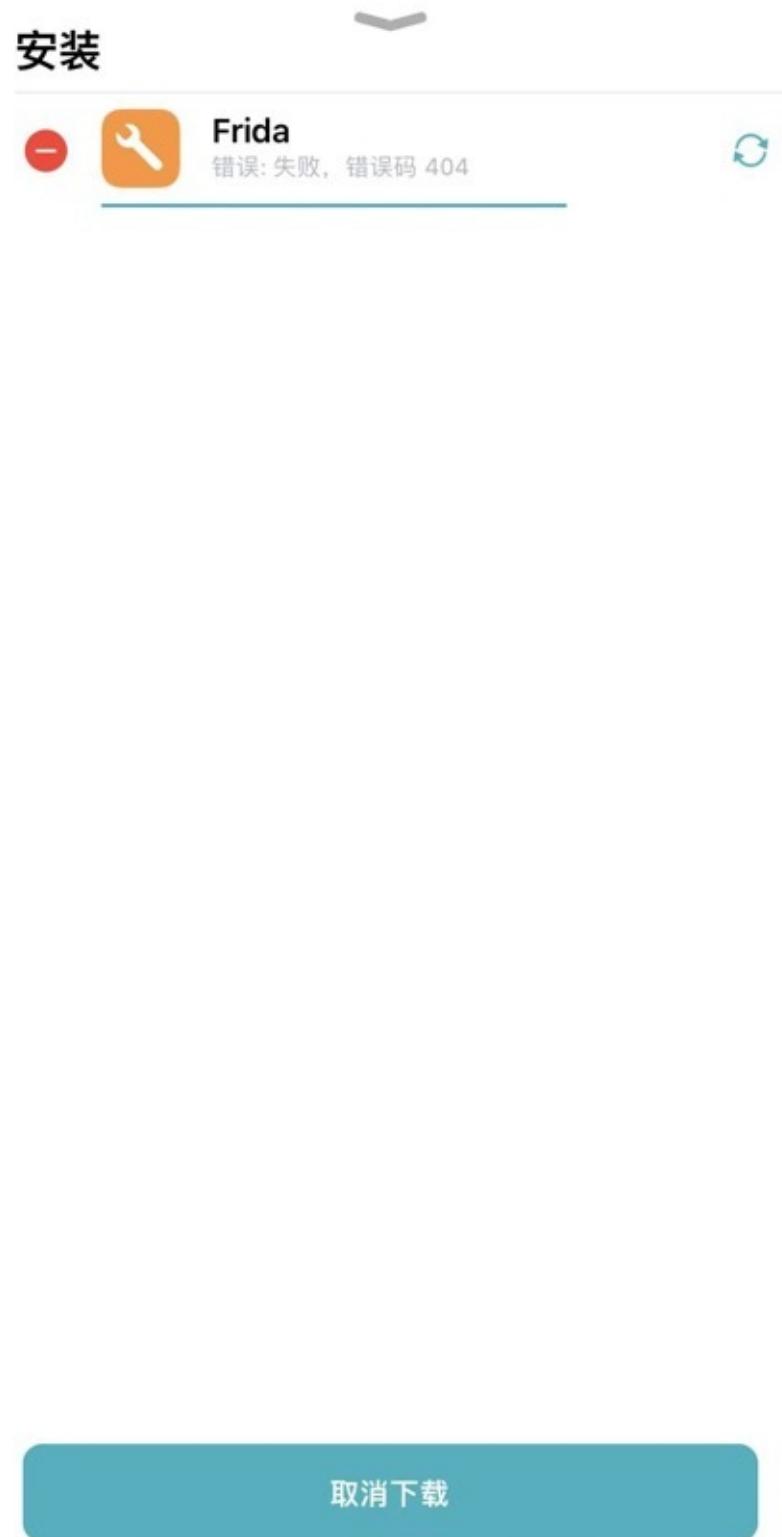


- 已root的安卓
 - 安装步骤
 - 概述
 - 安装 Magisk 插件: MagiskFrida
 - 详解
 - [初始化Frida开发环境 · Android逆向：动态调试 \(crifan.org\)](#)

常见问题

- XinaA15
 - 截至 XinaA15 v1.1.8 + Sileo Nightly v2.4 + Frida v16.0.11 : 在rootless越狱的 XinaA15 中, 无法通过 Sileo Nightly 正常安装和使用 Frida
 - 只要一使用frida工具（比如 frida-ps -u 等）就会导致iPhone重启
 - 且安装和卸载都会出现一些异常报错
 - 安装Frida

- /var/jb/var/ib/Library/LaunchDaemons/re.frida.server.plist service is disabled
- 且此处 Sileo Nightly v2.4 中看到的最新版 Frida v16.0.13，竟然还会出现无法安装：404错误



- 卸载Frida

- ```
/var/jb/var/jb/Library/LaunchDaemons/re.frida.server.plist: Could not
find specified service
```

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:

2023-08-15 22:23:28

## 升级Frida

### 什么时候才要升级Frida版本？

当遇到如下场景时，（可能）需要升级Frida：

- 当（两端的）Frida版本不匹配时
  - (iPhone等) 移动端中的Frida版本
    - 查看现有版本

```
frida-server --version
```

- (Mac等) PC端的Frida的版本
  - 查看现有版本

```
frida --version
```

- Frida有新版可用，想要升级到最新版
  - 举例
    - Sileo 中提示Frida有新版本 16.0.11

无SIM卡 上午 9:32

导出 软件包 愿望清单

搜索软件包

更新 全部更新

 **Frida**  
Frida Developers • 16.0.11  
Observe and reprogram running programs.

 **libapt-pkg6.0**  
Procursus Team • 2.5.6  
commandline package manager

 **libncursesw6**  
Procursus Team • 6.4  
shared libraries for terminal handling (wide chara...)

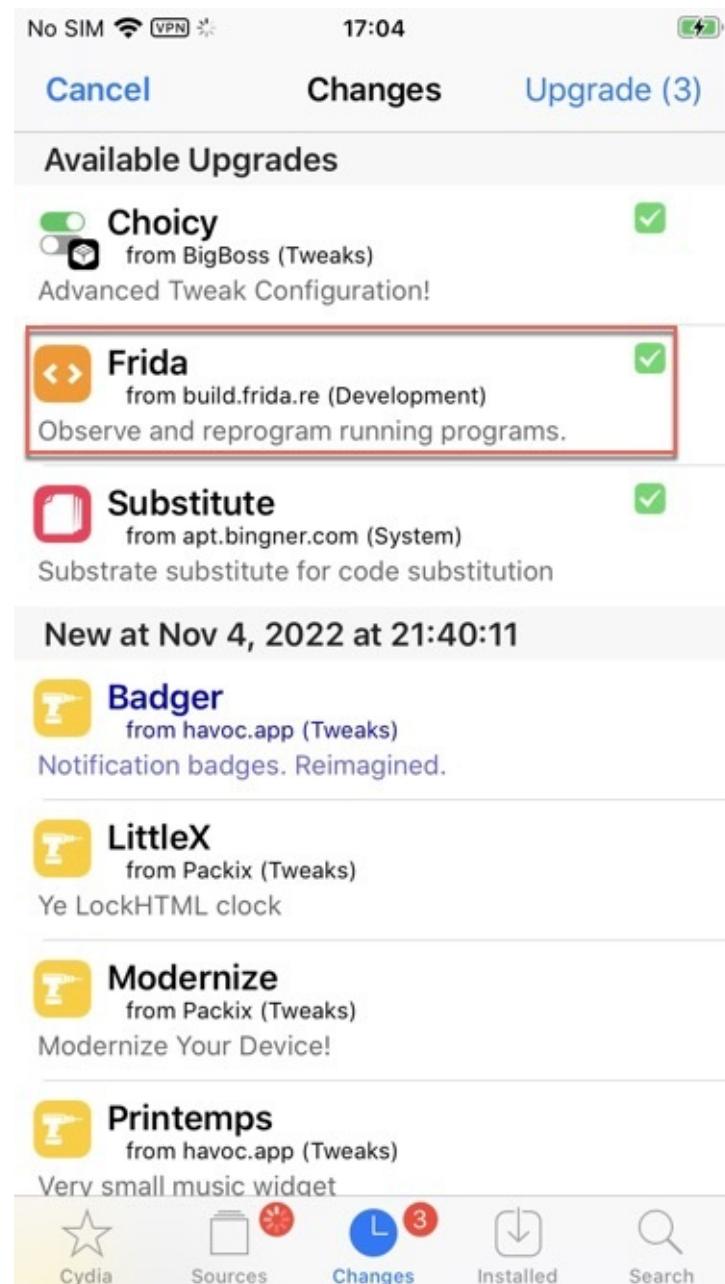
 **gzip**  
Procursus Team • 1.12  
GNU compression utilities

 **dpkg**  
Procursus Team • 1.21.20  
package maintainance tools from Debian

 **ncurses-term**  
Procursus Team • 6.4  
This package contains terminfo data files to sup...

精选 新闻 软件源 软件包 搜索

- Cydia 中提示有新版本Frida



## 如何升级Frida版本？

和[安装Frida](#)类似，也是分2部分：

- PC端
  - Mac
    - 用pip升级Frida

```
pip install --upgrade frida
```

- 如果需要，也去同时升级 frida-tools

```
pip install --upgrade frida-tools
```

- 移动端

- iPhone
  - 用 ( Cydia / Sileo 等) 包管理器, 升级 Frida 到最新版

## 如何安装指定版本的Frida?

TODO: 找之前如何指定frida版本, 当用pip安装时

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:  
2023-07-10 09:19:32

# 使用Frida

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：  
2023-06-22 22:08:11

## Frida典型使用逻辑

- 先搞懂大的概念
  - 从逻辑概念角度分
    - 核心工具: `frida` 命令行
    - 辅助工具: `frida-tools`
      - 包括 `frida-trace`、`frida-ps`、`frida-ls` 等工具
  - 从使用角度分
    - 最常用: `frida` 命令行
      - 分析调试逻辑的主力工具
    - 经常用: `frida-trace`
      - 分析函数调用堆栈关系
    - 偶尔用: `frida-tools` (包括 `frida-ps`、`frida-ls` 等)
      - 其他辅助用途, 比如查看进程PID等
- 去hook调试iPhone中的app或二进制 = Frida典型使用逻辑
  - 准备工作
    - 先要搞懂当前连接了哪些iPhone设备: `frida-ls-deviecs`
    - 再去搞懂要去hook的目标 (app或二进制) : `frida-ps` (或ssh连接的iPhone中的ps)
    - 辅助: 查看iPhone中有哪些相关文件: `frida-ls`
  - 真正去hook调试
    - 前期用=经常用: `frida-trace ->` 调试 (app页面操作过程中都) 调用了哪些ObjC的 (类和) 函数
    - 后期用=最常用: `frida ->` 写js去调试具体的ObjC的函数
      - 具体干什么
        - 查看
          - 传入的参数的类型和值
          - 函数调用堆栈
        - 微调=改动
          - (临时) 修改传入参数的值-> 研究程序逻辑变化
          - (临时) 修改返回值-> 研究程序逻辑变化
      - 常用到的内部功能
        - 主要是: `Interceptor` (的 `Interceptor.attach`) -> 去拦截=触发=hook对应的类的函数
        - 偶尔用: `ApiResolver` -> (模糊) 搜索有哪些类, 类有哪些函数等
        - 高级用法: `Stalker` -> 研究 (被混淆的) 代码实际的执行过程和逻辑等

## 用Frida辅助iOS逆向的思路

- iOS逆向开发心得=思路: 利用frida, 打印函数调用 -> 找到url、找到全部被调函数

对于iOS逆向的话, 有空试试 frida的hook。

就像之前别的某一个iOS技术人员沟通的方法: **Hook所有方法调用**

比如 估计是 hook `objc_msgSend` 这类函数?

都打印出来 就知道调用了哪些类的哪些函数了，甚至可以加上额外的统计调用次数之类的。

以及也可以把后面的几个参数的值，只要是字符串的也都打印出来。

另外去hook url调用HTTP的。

估计也是能轻易的搞清楚当前调用过哪些网络请求以及相关的什么参数。

估计就能找出来 `NSURLRequest` 之类的相关调用和url和其他参数。

总之有空还是去多试试frida的hook。

### 【后记1】

其实就是 所有类 的 所有方法

之前试过，效果还行。

不过要排除掉输出太多的，否则容易卡死崩溃。

### 【后记2】

有机会去试试：

- frida去hook：
  - `objc_msgSend`函数
    - 看看有多少objc的函数调用
    - 不过突然想到，貌似就是： `frida-trace` 加上 `-m *[* *]` 的意思？

### 【后记3】

Hook所有方法调用

看来应该是：

- 用Frida去hook：所有类的所有方法？
- 或者是：frida-trace去hook所有类的所有方法
- 又或者是：只hook `objc_msgSend`方法？这样也能打印出所有ObjC函数的调用

有空去试试

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：

2023-06-27 22:44:28

## frida

- frida
  - 是什么：命令行工具
    - frida 是 Frida 整个系统中，最基础，用的最多的，命令行工具
    - 官网称之为：frida-CLI
  - 文档
    - [frida-cli](#)

## 安装frida

- Mac中用Python安装frida

```
pip3 install frida
```

## help语法帮助

```
→ frida-trace frida --help
usage: frida [options] target

positional arguments:
 args extra arguments and/or target

options:
 -h, --help show this help message and exit
 -D ID, --device ID connect to device with the given ID
 -U, --usb connect to USB device
 -R, --remote connect to remote frida-server
 -H HOST, --host HOST connect to remote frida-server on HOST
 --certificate CERTIFICATE
 speak TLS with HOST, expecting CERTIFICATE
 --origin ORIGIN connect to remote server with "Origin" header set to ORIGIN
 --token TOKEN authenticate with HOST using TOKEN
 --keepalive-interval INTERVAL
 set keepalive interval in seconds, or 0 to disable (defaults to
-1 to auto-select based on transport)
 --p2p establish a peer-to-peer connection with target
 --stun-server ADDRESS
 set STUN server ADDRESS to use with --p2p
 --relay address,username,password,turn-{udp,tcp,tls}
 add relay to use with --p2p
 -f TARGET, --file TARGET
 spawn FILE
 -F, --attach-frontmost
 attach to frontmost application
 -n NAME, --attach-name NAME
 attach to NAME
 -N IDENTIFIER, --attach-identifier IDENTIFIER
 attach to IDENTIFIER
```

```

-p PID, --attach-pid PID
 attach to PID
-W PATTERN, --await PATTERN
 await spawn matching PATTERN
--stdio {inherit,pipe}
 stdio behavior when spawning (defaults to "inherit")
--aux option set aux option when spawning, such as "uid (int)42" (supported
types are: string, bool, int)
--realm {native,emulated}
 realm to attach in
--runtime {qjs,v8} script runtime to use
--debug enable the Node.js compatible script debugger
--squench-crash if enabled, will not dump crash report to console
-O FILE, --options-file FILE
 text file containing additional command line options
--version show program's version number and exit
-l SCRIPT, --load SCRIPT
 load SCRIPT
-P PARAMETERS_JSON, --parameters PARAMETERS_JSON
 parameters as JSON, same as Gadget
-C USER_CMODULE, --cmodule USER_CMODULE
 load CMODULE
--toolchain {any,internal,external}
 CModule toolchain to use when compiling from source code
-c CODESHARE_URI, --codeshare CODESHARE_URI
 load CODESHARE_URI
-e CODE, --eval CODE evaluate CODE
-q quiet mode (no prompt) and quit after -l and -e
-t TIMEOUT, --timeout TIMEOUT
 seconds to wait before terminating in quiet mode
--pause leave main thread paused after spawning program
-o LOGFILE, --output LOGFILE
 output to log file
--eternalize eternalize the script before exit
--exit-on-error exit with code 1 after encountering any exception in the SCRIPT
--kill-on-exit kill the spawned program when Frida exits
--auto-perform wrap entered code with Java.perform
--auto-reload Enable auto reload of provided scripts and c module (on by defa
ult, will be required in the future)
--no-auto-reload Disable auto reload of provided scripts and c module

```

## Frida调试目标的方式

### 背景知识

#### iOS的进程的类型

iOS的进程有2类：

- `app` = `Bundle` = 软件包
  - 举例
    - (iOS系统自带的) 设置`app`=包名是: `com.apple.Preferences`
- `Executable` = 可执行文件 = 二进制文件
  - 举例
    - (`AppleMediaServices.framework` 的) `amsaccounts`

#### frida和frida-trace是调试目标的方式的逻辑是一样的

```
frida --help
```

和：

```
frida-trace --help
```

都可以看到对应help语法：

```
-f TARGET, --file TARGET
 spawn FILE
-F, --attach-frontmost
 attach to frontmost application
-n NAME, --attach-name NAME
 attach to NAME
-N IDENTIFIER, --attach-identifier IDENTIFIER
 attach to IDENTIFIER
-p PID, --attach-pid PID
 attach to PID
-W PATTERN, --await PATTERN
 await spawn matching PATTERN
```

### frida / frida-trace 调试目标的方式

- frida / frida-trace 的调试目标方式 概述
  - 支持2种模式： Spawn 和 Attach
    - **Spawn模式**: 只有一种写法
      - `-f TARGET`
      - TARGET是 `app`包名 或 `Executable`二进制文件名

- **Attach模式**: 有多种写法=针对app或Executable有不同写法
  - 同时支持**app或Executable**的: `-p PID`
    - PID是app或Executable的进程ID
  - 针对**Executable**的: `-n NAME`
    - NAME是Executable的二进制文件名, 比如 `amsaccountsds`
  - 针对**app**的: `-N IDENTIFIER`
    - IDENTIFIER是app的包名, 比如 `com.apple.Preferences`
  - 特殊的: 针对当前手机中正在运行的 `frontmost` 最前台的: `-F`
    - 由于是, 当前最前台的正在运行的= 那只能是带页面显示的app, 且也无需再加额外参数指定app

## 详解

`frida / frida-trace` 的调试目标方式的详细解释:

- **Spawn模式**: 孵化出新进程
  - 典型使用场景: app/Executable还没运行, 想要从app/Executable刚开始启动就去hook调试
  - 语法
    - `-f TARGET , --file TARGET`
    - `spawn FILE`
  - 举例
 

```
frida -U -f com.apple.Preferences
```

- 说明
  - `TARGET` : 是 APP包名 或 Executable二进制文件名?
  - `spawn`方式启动后, 会立刻暂停运行
    - 需要继续运行, 需要手动输入: `%resume`
    - 如果不想要每次`spawn`启动后暂停, 则可以加上参数: `--no-pause`
      - 注: 旧版本才有`--no-pause`, 新版本已不支持`--no-pause`

- **Attach模式**: 挂载到一个已经正在运行的进程
  - 典型使用场景: app/Executable已经在运行了, 想要hook调试相关逻辑
  - 常见写法

- 针对于app的

- `frida -N com.apple.Preferences`
  - 语法
    - `-N IDENTIFIER, --attach-identifier IDENTIFIER`
    - `attach to IDENTIFIER`
  - 举例
 

```
frida -U -N com.apple.Preferences
```

- 说明
  - 如何查看到app的包名
    - Mac中用 `frida-ps`

```
frida-ps -Ua
```

```
frida-ps -Uai
```

- `frida -F`
  - 语法
    - `-F, --attach-frontmost`
      - attach to frontmost application
  - 举例
    - 当前iPhone中正在显示设置app的界面

```
frida -U -F
```

- 说明
  - F=Frontmost=Focused
  - 在操作之前，确保要调试的app，处在前台=正在运行=当前界面
- 针对于Executable的
  - `frida -n executableFilename`
    - 语法
      - `-n NAME, --attach-name NAME`
        - attach to NAME
    - 举例
      - `frida -U -n msaccounts`

- 针对于 app 或 Executable 的
  - `frida -p PID`
    - 语法
      - `-p PID, --attach-pid PID`
        - attach to PID
    - 举例
      - 12886是msaccounts的PID

```
frida -U -p 12886
```

- 19641是com.apple.Preferences的进程ID
  - `frida -U -p 19641`

- 说明
  - 如何获取PID
    - iPhone中通过ssh运行ps

```
ps -A | grep yourAppOrExecutableName
```

- Mac中用frida-ps

```
frida-ps -U
```

```
frida-ps -Ua
```

## 注意事项

- 如果调试方式对应参数使用有误，是无法正常调试的
  - 举例
    - 对于app, 用 `-n` (而不是 `-N`) , 会报错
      - 对于app: 系统内置的应用: `Preferences` (包名 `com.apple.Preferences`)
        - 虽然也有对应的二进制文件: `/Applications/Preferences.app/Preferences`
        - 但是是无法通过Executable的Attach参数去启动的
        - 即, 下面写法是无效的, 会报错的:

```
→ ~ frida-trace -U -n Preferences
Failed to spawn: unable to find process with name 'Preferences'
```

- 如果用 `-N` 出现异常情况，则可考虑换用 `-F`

- 举例
  - Frida调试设置app
    - 用 `-N` 包名 或 `-F` 不加包名
 

```
frida-trace -U -N com.apple.Preferences -O Preferences_accountLogin_hook.txt
```

```
frida-trace -U -F -O Preferences_accountLogin_hook.txt
```

- 就会导致: 设置app异常 -» 点击登录iPhone, 会出现弹框: 接入互联网以登录iPhone, 好像是: 此时设置app无法联网了
  - 而换用 `-F` 包名 去调试

```
frida-trace -U -F com.apple.Preferences -O Preferences_accountLogin_hook.txt
```

- 就一切正常, 设置app工作正常, 点击登录iPhone, 可以出现Apple ID登录页

## 写js脚本

frida 的利用方式，从使用角度来说，主要分2类：

- `frida-trace`: 无需js脚本，直接去hook对应的函数
- `frida ... -l xxx.js` : 最常见的用法，通过 `js` 脚本实现自己的功能
  - 即，写 `js` 脚本，再用 `frida` 去加载调用

## js脚本文件的类型 + 利用js脚本的方式

- 最早的，大家用的最多的，最常见的： `.js` 文件 = JavaScript 文件
  - 对应用法

```
frida ... -l xxx.js
```

- 举例

```
frida -U -p 8229 -l hookAccountLogin_singleClassSingleMethod.js
```

- 最新的，官网推荐的： `.ts` 文件 = TypeScript 文件
  - 对应用法

```
frida ... -l xxx.ts
```

## js脚本的来源

- 可以自己写
  - 比如 `iOS的ObjC` 中的各种例子
- 也可以，借用别人已有的
  - 比如 `Frida Codeshare`

## js脚本中的内容

主要取决于你的具体使用方法

根据使用场景分，主要有3种：

- `Interceptor` =拦截器
  - 作用：去实现hook函数，打印参数等调试操作
  - 效果：每次匹配到（对应函数），都会触发执行js代码
  - 举例：[Interceptor=hook函数](#)
- `ApiResolver` =解析器
  - 效果：只运行一次
    - 只有第一次（解析时）匹配到，才会执行，后续代码运行期间，不再去（匹配）执行
  - 作用

- 如果有些情况需要批量 Hook，比如对某一个类的所有方法进行 Hook，或者对某个模块的特定名称的函数进行 Hook，可能我们并不知道准确的名称是什么，只知道大概的关键字，这种情况怎么 Hook 呢？这时就得使用 API查找器（ApiResolver）先把感兴趣函数给找出来，得到地址之后就能 Hook 了
  - 举例：[ApiResolver](#)
- **Stalker** = 跟踪器
  - 效果：跟踪代码的实际运行的过程
    - 期间可以打印和查看对应的值，便于实现调试真正代码运行的逻辑
    - 详见：[Frida的Stalker](#)
  - 举例：[Stalker](#)

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：  
2023-06-27 23:00:47

## Frida的典型使用方式

在搞懂了[调试目标方式](#)和[写js脚本](#)后，再来介绍

- Frida的典型使用方式
  - 用哪个命令?
    - 主要是用: `frida`
      - 大都是都是 `frida` 然后加上 `-l xxx.js` 加上js脚本，实现自己特定的调试的目的
    - 偶尔使用: `frida-trace`
      - 偶尔直接用 `frida-trace`，加上要hook的类或函数，去追踪代码执行过程中，调用到哪些函数
  - 常见写法 = 怎么用?
    - 命令行中使用
      - `frida xxx`
      - `frida-trace xxx`
    - (其中 `xxx` 中) 常用参数有
      - 要调试的设备用: `-U`
        - `-U` : 调试的是USB连接的移动端的设备 (比如iPhone)
      - 要调试的目标app/二进制: `-p PID` / `-n executableName` / `-N com.app.identifier` / `-F` / `-f com.app.identifier`
        - 有多种写法，选择合适的一个即可
          - app或二进制均支持
            - 通过PID指定进程: `-p PID`
            - 仅支持二进制
              - 通过 `-n` 指定二进制可执行文件名: `-n executableName`
            - 仅支持app
              - 通过 `-N` 指定app的包名: `-N com.app.identifier`
              - 通过 `-F` 指定当前最顶层正在运行的app: `-F`
                - 有时候可以额外给 `-F` 再加上app的包名，效果更好: `-F com.app.identifier`
        - -» 所以常见写法举例
          - `frida`
          - 无需js: 直接动态交互式的调试
 

```
frida -U -f com.apple.Preferences
```
      - 需要js: 实现特定调试目的
        - 自己写js
          - 使用 `-p` 指定进程PID
 

```
frida -U -F -p 18533 -l ./hookAccountLogin_Interceptor.js
```
        - `-U` : 调试的设备是 (当前Mac用) USB连接的iPhone
          - `-F -p 18533` : 调试的app是

- -F : 表示是当前最前端显示的app
- -p 18533 : (额外) 加上PID, 指定app的进程ID
- -l ./hookAccountLogin\_Interceptor.js : 所要具体调试的内容, 放在js中
- 使用 -n 指定二进制文件名=进程名=ProcessName

```
frida -U -n akd -l ./fridaStalker_akdSymbol2575.js
```

- -n akd : 要调试的目标是, 进程名=二进制名是 akd 的进程
- 注: akd = AuthKit 的daemon进程
- 借助别人的js

```
frida --codeshare lichao890427/ios-utils -F com.apple.Preferences
```

#### ■ frida-trace

- 用frida-trace追踪Preferences设置中Apple账号登录过程调用了哪些函数

```
frida-trace -U -F com.apple.Preferences -m "*[AA* *]" -m "*[AK* *]" -m "*[AS* *]" -m "*[NSXPC* *]" -M "[ASDBundle copyWithZone:]" -M "[ASDInstallationEvent copyWithZone:]" -M "[NSXPCEncoder _encodeArrayOfObjects:forKey:]" -M "[NSXPCEncoder _encodeUnkeyedObject:]" -M "[NSXPCEncoder _replaceObject:]" -M "[NSXPCEncoder _checkObject:]" -M "[NSXPCEncoder _encodeObject:]" -M "[NSXPCCollection replaceItemAtIndex:object:]"
```

#### ■ 说明

- -U -F com.apple.Preferences : 调试目标设备和目标app是和frida写法一致
- 但是后续要trace的函数写法, 是frida-trace所特有的语法:
  - -m xxx : include=包含要调试的Objc的类
  - -M xxx : exclude=排除掉Objc的类, 不调试

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:

2023-06-23 23:15:37

# Frida中iOS的ObjC

Frida 支持hook调试 iOS 的 ObjC 的类和函数。

- Frida的ObjC
  - 官网文档
    - [Frida API - ObjC](#)
      - ObjC.available
      - ObjC.api
      - ObjC.classes
        - ObjC.Object
      - ObjC.protocols
        - ObjC.Protocol
      - ObjC.mainQueue
      - ObjC.schedule(queue, work)
      - new ObjC.Object(handle[, protocol])
      - new ObjC.Protocol(handle)
      - new ObjC.Block(target[, options])
      - ObjC.implement(method, fn)
      - ObjC.registerProxy(properties)
      - ObjC.registerClass(properties)
      - ObjC.registerProtocol(properties)
      - ObjC.bind(obj, data)
      - ObjC.unbind(obj)
      - ObjC.getBoundData(obj)
      - ObjC.enumerateLoadedClasses([options, ]callbacks)
      - ObjC.enumerateLoadedClassesSync([options])
      - ObjC.choose(specifier, callbacks)
      - ObjC.chooseSync(specifier)
      - ObjC.selector(name)
      - ObjC.selectorAsString(sel)

- 常用工具类
  - [iOS Utils](#)

下面继续介绍，关于Frida中ObjC的基本逻辑：

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：  
2023-06-23 23:29:20

## Frida中ObjC的参数

- 当触发到 Interceptor.attach 的 onEnter 函数后
  - 举例：相关的典型的代码

```
function hook_specific_method_of_class(className, funcName)
{
 var curClass = ObjC.classes[className];
 if (typeof(curClass) !== 'undefined') {
 var curMethod = curClass[funcName];
 if (typeof(curMethod) !== 'undefined') {
 Interceptor.attach(curMethod.implementation, {
 onEnter: function(args) {
 ...
 }
 });
 }
 }
}
```

- 传入Frida中 ios 的 objc 的参数 == ObjC的底层函数 objc\_msgSend 的参数
  - 注：objc\_msgSend 函数的定义

```
id objc_msgSend(id self, SEL _cmd, ...)
```

- 此时默认参数名叫做 args 的参数 == 是一个参数的数组列表，每个元素的类型是 NativePointer
- 此时对应 (Frida中 ios 的 objc ) 的参数逻辑是：
  - 第一个参数 = args[0] == objc\_msgSend 的 self (类型是 id )
  - 第二个参数 = args[1] == objc\_msgSend 的 \_cmd (类型是 SEL )
  - 后续才是：真正的函数的参数
    - 函数的真正的第1个参数 = args[2]
    - 函数的真正的第2个参数 = args[3]
    - ...

## Frida中获取ObjC的参数

### 第一个参数 args[0]

- 获取第一个参数 args[0]

```
// args[0] is self = id
const argSelf = args[0];
console.log("argSelf: ", argSelf);
```

- 典型log输出

```
argSelf: 0x105744c00
```

### 第二个参数 args[1]

获取第二个参数 args[1]：

```
// args[1] is selector
const argSel = args[1];
console.log("argSel: ", argSel);
```

- 典型log输出

```
argSel: 0x105744c00
```

## ObjC函数的真正参数

- 获取 函数的真正的第1个参数 = args[2]

```
// args[2] holds the first function argument
const args2 = args[2];
console.log("args2: ", args2);
```

- 典型log输出

```
args2: 0x105743850
```

- 获取 函数的其他更多参数 (如果有的话)

```
const args3 = args[3];
console.log("args3: ", args3);

const args4 = args[4];
console.log("args4: ", args4);
```

## 计算ObjC的函数的真正参数的个数 + 打印全部参数

TODO: 把这部分代码, 整理成 独立的工具类函数, 转移到: [工具类函数](#)

根据ObjC的基础知识, 目前得知: 可以通过 SEL = selector 字符串中的 := 冒号 的个数, 判断后续真正有几个参数

所以, 可以用如下代码, 循环的、批量的、挨个、打印所有参数:

```
/** Function that count occurrences of a substring in a string;
 * @param {String} string The string
 * @param {String} subString The sub string to search for
 * @param {Boolean} [allowOverlapping] Optional. (Default:false)
 *
 * @author Vitim.us https://gist.github.com/victornpb/7736865
 * @see Unit Test https://jsfiddle.net/Victornpb/5axuh96u/
 * @see https://stackoverflow.com/a/7924240/938822
 */
```

```

function occurrences(string, subString, allowOverlapping) {
 // console.log("string=" + string + ",subString=" + subString + ", allowOverlapping=" + allowOverlapping)
 string += "";
 subString += "";
 if (subString.length <= 0) return (string.length + 1);

 var n = 0,
 pos = 0,
 step = allowOverlapping ? 1 : subString.length;

 while (true) {
 pos = string.indexOf(subString, pos);
 // console.log("pos=" + pos)
 if (pos >= 0) {
 ++n;
 pos += step;
 } else break;
 }

 return n;
}

// ...

function hook_specific_method_of_class(className, funcName)
{
 var iosObjCallStr = toiosobjcCall(className, funcName)
 // var hook = ObjC.classes[className][funcName];
 // var hook = eval('ObjC.classes.' + className + '[' + funcName + ']');
 var curClass = ObjC.classes[className];
 if (typeof(curClass) !== 'undefined') {
 var curMethod = curClass[funcName];
 if (typeof(curMethod) !== 'undefined') {
 Interceptor.attach(curMethod.implementation, {
 onEnter: function(args) {
 console.log("===== [*] Detected call to: " + iosObjCallStr);
 }
 });

 const argSelStr = ObjC.selectorAsString(argSel);
 console.log("argSelStr: ", argSelStr);
 const argCount = occurrences(argSelStr, ":");

 // console.log("funcName=", funcName);
 // const argCount = occurrences(funcName, ":");
 // console.log("argCount: ", argCount);

 for (let curArgIdx = 0; curArgIdx < argCount; curArgIdx++) {
 const curArg = args[curArgIdx + 2];

 // const usePtr = false;
 const usePtr = true;
 if (usePtr) {
 // console.log("usePtr=", usePtr);
 const curArgPtr = ptr(curArg);
 }
 }
 }
 }
}

```

```

 console.log("----- [" + curArgIdx + "] curArgPtr=" + curArgPtr);
 if (curArgPtr.isNull()) {
 const curArgPtrObj = new ObjC.Object(curArgPtr);
 console.log("curArgPtrObj: ", curArgPtrObj);
 console.log("curArgPtrObj className: ", curArgPtrObj.$className);
 }
 } else {
 console.log("----- [" + curArgIdx + "] curArg=" + curArg);
 if (curArg && (curArg != 0x0)) {
 // console.log("curArg className: ", curArg.$className);
 const curArgObj = new ObjC.Object(curArg);
 console.log("curArgObj: ", curArgObj);
 console.log("curArgObj className: ", curArgObj.$className);
 }
 }
}

...

```

某次的输出日志的效果：

```

----- [*] Detected call to: +[NSURLRequest requestWithURL:cachePolicy:timeoutInte
rval:]
funcName= + requestWithURL:cachePolicy:timeoutInterval:
argSelStr: requestWithURL:cachePolicy:timeoutInterval:
argCount: 3
----- [0] curArgPtr 0x2831f2530
curArgPtrObj: https://setup.icloud.com/setup/signin/v2/login
curArgPtrObj className: NSURL
----- [1] curArgPtr 0x0
----- [2] curArgPtr 0x0

```

- 注意：
  - 当Frida调试期间，输出log日志太多，或者打印参数太多时，经常会导致app或进程崩溃而无法继续调试
    - 所以当遇到Frida调试导致的崩溃问题时，可以适当的减少log日志的输出，以缓解崩溃问题

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：  
2023-06-23 23:44:27

## Frida中ObjC的变量类型

### 通用逻辑

- 获取ObjC的某个类
  - 写法: `ObjC.classes.someClassName == ObjC.classes[someClassName]`
  - 举例

```
ObjC.classes.NSMutableData.data()
ObjC.classes.NSString.stringWithString_("Crifan Li")
```

### ObjC 的 Object

此处的第一个参数和后续的真正的函数的（第一个以及后续的其他）参数，往往是个（iOS中ObjC的）`Class` 或 `Instance`，在 Frida 中对应的叫做：

- ObjC 的 Object

### ObjC的Object的常见的操作

- 转换为ObjC的类

```
const argSelf = args[0]
const argSelfObj = new ObjC.Object(argSelf)

const args2 = args[2]
const args2Obj = new ObjC.Object(args2)
// const args2Obj = new ObjC.Object(ptr(args2))
```

以及紧接着的：

- 打印类的信息

```
console.log("argSelfObj: ", argSelfObj)
console.log("args2Obj: ", args2Obj)
```

- 典型log输出

```
argSelfObj: <NSXPCConnection: 0x105744c00> connection from pid 46847 on mach $
ervice named com.apple.ak.auth.xpc

args2Obj: <AKAppleIDAuthenticationService: 0x102d3e4d0>
```

- 以及，把 Object 转换成String字符串，再去打印

```
const args20bjStr = args20bj.toString();
console.log("args20bjStr: ", args20bjStr);
```

- 典型log输出

```
args20bjStr: AKAppleIDAuthenticationService: 0x102d3e4d0
```

- 注：此处从调试输出类的信息的角度来说，往往 `Object` 转换成 `String` 后再输出的字符串，和上述直接打印 `Object` 的效果是一样的

## ObjC 的 Object 的特殊属性

对于Frida中的ObjC的Object，有很多内置的特殊的属性：

- `$kind`
- `$super`
- `$superClass`
- `$class`
- `$className`
- `$moduleName`
- `$protocols`
- `$methods`
- `$ownMethods`
- `$ivars`

所以，一些常见操作是：

### `$className`

- 查看（当前 `Object` 对象的）类名 = 类的名字

```
const argSelfClassName = argSelfObj.$className;
console.log("argSelfClassName: ", argSelfClassName);

const args20bjClassName = args20bj.$className;
console.log("args20bjClassName: ", args20bjClassName);
```

- 典型log输出

```
argSelfClassName: NSXPConnection
args20bjClassName: AKAppleIDAuthenticationService
```

### `$kind`

- 查看（当前 `Object` 对象的）类型

```
const argSelfKind = argSelfObj.$kind;
console.log("argSelfKind:", argSelfKind);
```

- 典型log输出

```
argSelfKind: instance
```

## \$methods 和 \$ownMethods

- 注意
  - 此处的 \$methods 、 \$ownMethods 中的 method
    - method 的英文单词本意是： 方法 = 函数
    - 但实际上此处： method = 类的 property + 类的 method
      - 注：
        - property = 属性
        - method = 方法 = 函数
  - 举例： \$methods和\$ownMethods

## objc 的 Object 的自己类的属性

而对于本身 objc 的类的属性，也是支持访问的：

- 访问 objc 的自己类的属性的方式：把属性当做函数访问 -》 加上括号 ()
  - 语法： iosObjcObj.propertyName()
  - 举例
    - 获取NSXPConnection的服务名
      - 对于Objc的Object对象：
 

```
argSelfObj: <NSXPConnection: 0x105744c00> connection from pid 46847 on
mach service named com.apple.ak.auth.xpc
```
      - 去获取其serviceName属性的代码：
 

```
const connnServiceNameNSStr = argSelfObj.serviceName();
console.log("connnServiceNameNSStr: ", connnServiceNameNSStr);
```
      - 输出log
 

```
connnServiceNameNSStr: com.apple.ak.auth.xpc
```
    - 获取NSString的UTF8String
      - 代码：
 

```
const connnServiceNameJsStr = connnServiceNameNSStr.UTF8String();
console.log("connnServiceNameJsStr: ", connnServiceNameJsStr);
```
      - 输出log
 

```
connnServiceNameJsStr: com.apple.ak.auth.xpc
```

## objc 的 Object 的自己类的函数

Frida中，是可以调用Objc中类的函数的。

### 举例： +[NSString stringWithFormat:]

Objc 中 NSString 的原始函数：

```
+[NSString stringWithFormat:@"Hello World"]
```

根据规则：

- 把 冒号 = : 变成 下划线 = \_
- 给函数加上括号 ()
- 把参数放到括号 () 中

而变成：

```
const { NSString } = ObjC.classes;
NSString.stringWithString_("Hello World");
```

- 或另外一种写法：

```
ObjC.classes.NSString.stringWithString_("Hello World");
```

### 举例： -[NSString cStringUsingEncoding:]

- ObjC的函数： -[NSString cStringUsingEncoding:]  
◦ -> Frida中的调用：

```
const NSUTF8StringEncoding = 4;
const curNsStr = curNsStr.cStringUsingEncoding_(NSUTF8StringEncoding);
```

## ObjC 的 selector

Objc 中，有个稍微特殊一点的变量，叫做 SEL = selector

也就是前面获取到的，第二个参数，类型是 SEL

- SEL 的常见操作之一就是：转换成字符串，再去打印

```
const argSel = args[1];
const argSelStr = ObjC.selectorAsString(argSel);
console.log("argSelStr: ", argSelStr);
```

- 典型log输出

```
argSelStr: setExportedObject:
```

## ObjC 的 Protocol

## objc 的 Block

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:  
2023-06-27 23:10:39

# 数据类型

Frida中典型的数据类型有：

- `Int64`
  - <https://frida.re/docs/javascript-api/#int64>
- `UInt64`
  - <https://frida.re/docs/javascript-api/#uint64>
- `NativePointer`
- `ArrayBuffer`
  - <https://frida.re/docs/javascript-api/#arraybuffer>
- 其他相关
  - `Function`函数
    - `NativeFunction`
      - <https://frida.re/docs/javascript-api/#nativefunction>
    - `SystemFunction`
      - <https://frida.re/docs/javascript-api/#systemfunction>
  - `Callback`回调
    - `NativeCallback`
      - <https://frida.re/docs/javascript-api/#nativecallback>

## 常见缩写

和数据类型有关的，常见缩写有：

- `int64(v) == new Int64(v)`
- `uint64(v) == new UInt64(v)`
- `ptr(s) == new NativePointer(s)`
- `NULL == ptr("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：

2023-06-23 22:10:37

## NativePointer

- NativePointer
  - 是什么: Frida中一个最基本的变量类型
  - 常简称: ptr
  - 官网文档
    - [NativePointer - JavaScript API](#)

## NativePointer的函数=支持的操作

- NativePointer 的函数=支持的操作
  - new NativePointer(s) : 传入字符串, 生成 NativePointer
    - 大家常用=其简写为: ptr(s)
  - isNull() : 判断是否为空
  - add(rhs) , sub(rhs) , and(rhs) , or(rhs) , xor(rhs)
  - shr(n) , shl(n)
  - not()
  - sign([key, data])
  - strip([key])
  - blend(smallInteger)
  - equals(rhs)
  - compare(rhs)
  - toInt32()
  - toString([radix = 16])
  - toMatchPattern()
  - readPointer()
  - writePointer(ptr)
  - reads8() , readU8() , reads16() , readU16() , reads32() , readU32() ,
   
readShort() , readUShort() , readInt() , readUInt() , readFloat() , readDouble()
  - writeS8(value) , writeU8(value) , writeS16(value) , writeU16(value) ,
   
writeS32(value) , writeU32(value) , writeShort(value) , writeUShort(value) ,
   
writeInt(value) , writeUInt(value) , writeFloat(value) , writeDouble(value)
  - readS64() , readU64() , readLong() , readULong()
  - writeS64(value) , writeU64(value) , writeLong(value) , writeULong(value)
  - readByteArray(length)
  - writeByteArray(bytes)
  - readCString([size = -1]) , readUtf8String([size = -1]) , readUtf16String([length = -1]) , readAnsiString([size = -1])
  - writeUtf8String(str) , writeUtf16String(str) , writeAnsiString(str)

## NativePointer的典型用法

## 一些常见的写法

```
myOffsetPtr = myBaseAddr.add(ptr('0x76E'))
console.log(hexdump(ptr(0x7cbe43acd0)))
Memory.protect(ptr('0x1234'), 4096, 'rw-');
ptr('0x7fff870135c9')
```

## onEnter中的args

frida中最常涉及到 NativePointer 的地方就是： Interceptor.attach 的 onEnter 中的 args ，就是一个 NativePointer 的数组 = an array of NativePointer objects

[单个类的单个函数的代码举例](#)中的：

```
const curMethod = ObjC.classes.NSXPCConnection["- setExportedObject:"];
Interceptor.attach(curMethod.implementation, {
 onEnter: function(args) {
 const arg0 = args[0]
 const arg1 = args[1]
 const arg2 = args[2]
 console.log("args: Objc self id=" + arg0 + ", SEL=" + arg1 + ", objcRealArg1=" + arg2);
 }
});
```

中的 args ，就是NativePointer的数组，所以： arg0 、 arg1 、 arg2 ，都是 NativePointer = ptr 类型

而典型的后续的操作就是，去转换成对应的类型，去处理

比如：

- ObjC中，把传入的参数，转换成真正的ObjC的类

```
const argSelf = args[0];
const argSelfObj = new ObjC.Object(argSelf);
```

## 用add计算函数地址

[Stalker实例代码](#) 中的

```
var funcRelativeStartAddr = 0xa0460;
var functionSize = 0x24C8; // 9416 == 0x24C8
var funcRelativeEndAddr = funcRelativeStartAddr + functionSize;
const moduleName = "akd";
const moduleBaseAddress = Module.findBaseAddress(moduleName);
const funcRealStartAddr = moduleBaseAddress.add(funcRelativeStartAddr);
```

```
const funcRealEndAddr = funcRealStartAddr.add(functionSize);
```

中就是用到了 `Module.findBaseAddress` 的返回值，类型就是 `NativePointer`，然后：

- `const funcRealStartAddr = moduleBaseAddress.add(funcRelativeStartAddr);`
  - 通过 `add` 加上起始地址，计算出函数真正的起始地址
- `const funcRealEndAddr = funcRealStartAddr.add(functionSize);`
  - 通过 `add` 加上函数大小，计算出真正的结束地址

输出的相关log：

```
funcRelativeStartAddr=656480, functionSize=9416, funcRelativeEndAddr=665896
moduleName:akd, moduleBaseAddress=0x10015c000
funcRealStartAddr=0x1001fc460, funcRealEndAddr=0x1001fe928
```

注意：

- 如果是 `NativePointer` 类型的变量，直接相加 + 普通的数值，则结果，不是数值上的相加，而是：  
数值的字符串拼接
  - 对比
    - 上述代码

```
const funcRealEndAddr = funcRealStartAddr.add(functionSize);
```

- 结果：`funcRealEndAddr=0x1001fe928`
  - `= 0x1001fc460 + 0x24C8`

- 如果改为

```
const funcRealEndAddr = funcRealStartAddr + functionSize;
```

- 则结果是：`funcRealEndAddr=0x1001fc4609416`
  - `= 地址的字符串 0x1001fc460 拼接上 9416 (=16进制的 0x24C8 的10进制值)`

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：  
2023-06-23 23:17:59

# Frida的Stalker

- Frida 的 Stalker
  - 别称: frida-stalker
    - 有人翻译成: 潜行者
  - 作用: 汇编指令级别的hook
  - 用途: 追踪真实代码指令的执行过程
  - 官网文档
    - [Stalker介绍](#)
  - API接口
    - 接口形式
      - 普通用户常直接调用: JS的API
        - [Stalker - JavaScript API | Frida](#)
      - 内部接口: Gum接口
        - TypeScript type definitions
          - [DefinitelyTyped/index.d.ts](#)
    - 接口内容
      - 最核心接口
        - `Stalker.follow([threadId, options])`
      - 其他
        - `Stalker.exclude(range)`
        - `Stalker.unfollow([threadId])`
        - `Stalker.parse(events[, options])`
        - `Stalker.flush()`
        - `Stalker.garbageCollect()`
        - `Stalker.invalidate(address)`
        - `Stalker.invalidate(threadId, address)`
        - `Stalker.addCallProbe(address, callback[, data])`
        - `Stalker.removeCallProbe`
        - `Stalker.trustThreshold`
        - `Stalker.queueCapacity`
        - `Stalker.queueDrainInterval`
  - 用法
    - 概述
      - 核心逻辑
        - 在frida命令上, 和普通frida一样, 都是调用js
 

```
frida -U -n akd -l ./fridaStalker_akdSymbol2575.js
```
    - js内部逻辑
      - 最初要初始化: 计算出当前要hook的函数所属的模块和地址
      - 在普通的 `Interceptor.attach` 的 `onEnter` 中, 加上 `Stalker.follow`
        - 在 `transform` 中, 计算是否是原始函数的代码
          - 如果是, 再去: 实现特定的调试的目的

- 打印真实执行的指令的信息 `instruction.toString()`
- 打印当时的变量的值 `context`
- 等等
- 详解
  - 举例
    - [Stalker示例代码](#)

## 何时需要用到Frida的Stalker

如下场景：

- 你想要搞懂该函数内部的执行的逻辑，想要查看哪个函数，甚至是哪个代码块codeblock，被执行了
- 还比如你想要搞懂，当传入不同参数时，函数内部代码执行的流程路径，是否有何不同。

就可以去用：

- `Stalker.follow()`

## Frida的Stalker中transform的逻辑

Frida的Stalker中transform的逻辑：

除了官网文档：

<https://frida.re/docs/stalker/>

介绍了内部具体实现机制和过程之外：

对于，想要搞懂如何利用transform去调试代码来说：

需要明白的逻辑是：

此处代码：

```
Interceptor.attach(funcRealStartAddr, {
 onEnter: function(args) {
 ...
 var curTid = Process.getCurrentThreadId();
 console.log("curTid=", curTid);
 Stalker.follow(curTid, {
 events: {
 call: true, // CALL instructions: yes please
 ret: false, // RET instructions
 exec: false, // all instructions: not recommended as it's
 block: false, // block executed: coarse execution trace
 compile: false // block compiled: useful for coverage
 },
 // transform: (iterator: StalkerArm64Iterator) => {
 transform: function (iterator) {
 ...
 }
 })
 }
})
```

触发到的 `transform` 的 `iterator` 来说：

- 每次触发=每个iterator: 都 (对应着) 单个block= (basic) 代码块
- 此处代码块有2种
  - 非原始函数代码 == isAppCode=false
    - 对应着应该是Stalker内部实现原理说的, copy拷贝出的代码
    - 其中会额外加上很多逻辑, 用于实现Stalker的功能和逻辑
    - 估计就是这里说的这些内容
      - [原创] [sktrace: 基于 Frida Stalker 的 trace 工具-Android安全-看雪-安全社区|安全招聘|kanxue.com](#)
      - 每当执行到一个基本块, Stalker 都会做以下几件事:
        1. 对于方法调用, 保存 lr 等必要信息
        2. 重定位位置相关指令, 例如: ADR Xd, label
        3. 建立此块的索引, 如果此块在达到可信阈值后, 内容未曾变化, 下次将不再重新编译 (为了加快速度)
        4. 根据 transform 函数, 编译生成一个新的基本块 GumExecBlock , 保存到 GumSlab 。 void transform(GumStalkerIterator iterator, GumStalkerOutput output, gpointer user\_data) 可以控制读取, 改动, 写入指令。
        5. transform 过程中还可通过 void gum\_stalker\_iterator\_put\_callout (GumStalkerIterator self, GumStalkerCallout callout, gpointer data, GDestroyNotify data\_destroy) 来设置一个当此位置被执行到时的 callout。通过此 void callout(GumCpuContext cpu\_context, gpointer user\_data) 获取 cpu 信息。
        6. 执行一个基本快 GumExecBlock, 开始下一个基本快
      - 所以无需操作具体内部过程, 直接忽略即可
  - 原始函数代码 == isAppCode=true
    - 是实际运行的代码, 才是我们所要关注的代码
      - 才会真正的去处理: 比如判断是否是对应的 (某个偏移量的) 代码, 然后去打印查看调试寄存器的值等等

此处去通过计算是否是 原始函数代码isAppCode 决定是否处理

具体详见示例代码: [\\_\\_lldb\\_unnamed\\_symbol2575\\$\\$akd](#)

## Stalker.follow 中的 events 的属性含义

- Stalker.follow 中的 events 的属性含义
  - 概述
    - Stalker.follow中的events中某个属性是true, 含义是: 当出现对应指令, 则触发对应event事件
  - 详解
    - 属性
      - 对应的属性的含义是
        - call: call指令
          - Intel的: call指令
          - ARM的: BL类的指令
            - 普通的=arm64的: BL、BLR等
            - arm64e的, 带PAC的: BLRAA、BLRAAZ、BLRAB、BLRABZ等

- ret: ret指令
- exec: 所有指令
- block: (单个) block的(所有)指令
- compile: 特殊, (单个) block被编译时, 仅用于测试代码覆盖率?
- 除去特殊的compile参数, 其他几个参数, 按照范围大小去划分, 更容易理解:
  - exec: 所有代码的级别
  - block: 单个代码块的级别
  - 某些特殊指令的级别
    - call: 单独的call指令
    - ret: 单独的ret指令
- event事件
  - 会触发onReceive(events)函数
    - 其中可以events是二进制(的blob), 需要去用Stalker.parse()解析后才能看懂
- -» events和onReceive的作用
  - 暂时不完全懂, 只是知道, 可以设置参数, 决定call、ret等指令的触发时去打印, 其他用途暂时不清楚

## Stalker.follow() 内部实现原理

当用户调用 `Stalker.follow()` 时, 内部调用:

- 要么是: `gum_stalker_follow_me()` : 去跟踪当前的线程thread
  - 函数原型
 

```
GUM_API void gum_stalker_follow_me (GumStalker * self, GumStalkerTransformer * transformer, GumEventSink * sink);
```
  - 底层 JS 引擎: 是 QuickJS 或 v8
- 要么是: `gum_stalker_follow(thread_id)` : 去跟踪当前process进程中的其他某个线程thread

### gum\_stalker\_follow\_me 的内部原理

```
#ifdef __APPLE__

.globl _gum_stalker_follow_me

_gum_stalker_follow_me:
#else
.globl gum_stalker_follow_me

.type gum_stalker_follow_me, %function

gum_stalker_follow_me:
#endif
stp x29, x30, [sp, -16]

mov x29, sp

mov x3, x30
```

```
#ifdef __APPLE__

bl __gum_stalker_do_follow_me

#else
bl _gum_stalker_do_follow_me

#endif
ldp x29, x30, [sp], 16

br x0

```

-»

- 内部原理
  - LR=Link Register=X30=链接寄存器
    - AArch64架构中，根据LR去决定从哪里开始跟踪
    - 当遇到BR, BLR等跳转指令时，会去设置LR
      - LR被设置为，当前函数返回后，继续运行的地址
    - 由于只有一个LR，如果被调用函数调用了其他函数，此时LR的值就会被保存起来，比如保存到Stack栈上，后续当RET指令执行之前，会重新把LR从Stack栈中加载到寄存器中，最终返回到调用者
  - FP=Frame Pointer=X29=帧指针
    - FP始终指向Stack top栈的顶部，表示当前函数被调用时的栈的位置
    - 所以就可以通过固定的偏移量去访问到，所有通过Stack栈传入的参数和基于栈的局部变量
    - 且每个函数都有自己的FP，所以需要调用新函数时，保存之前的FP，返回之前函数时，恢复FP。
    - 在刚进入新函数后，备份FP后，就可以去设置：mov x29, sp，把SP给X29=FP了。

保持了原先传入 `x0-x2` 的3个参数，额外加上 `x3 = x30 = LR`，所以再去调用函数，就对应上参数了：

```
gpointer_gum_stalker_do_follow_me (GumStalker * self, GumStalkerTransformer * transformer, GumEventSink * sink, gpointer ret_addr)
```

## gum\_stalker\_follow 的内部原理

和 `gum_stalker_follow_me()` 类似，但有额外参数：`thread_id`

```
void
gum_stalker_follow (GumStalker * self,
 GumThreadId thread_id,
 GumStalkerTransformer * transformer,
 GumEventSink * sink)
{
 if (thread_id == gum_process_get_current_thread_id ())
 {
 gum_stalker_follow_me (self, transformer, sink);
 }
 else
 {
```

```

 GumInfectContext ctx;

 ctx.stalker = self;
 ctx.transformer = transformer;
 ctx.sink = sink;

 gum_process_modify_thread (thread_id, gum_stalker_infect, &ctx);
}
}

```

其中： `gum_process_modify_thread()`， 不属于 `stalker`， 但属于 `Gum`

回调callback会去修改： `GumCpuContext`

## GumCpuContext

`GumCpuContext`的定义：

```

typedef GumArm64CpuContext GumCpuContext;

struct __GumArm64CpuContext
{
 quint64 pc;
 quint64 sp;

 quint64 x[29];
 quint64 fp;
 quint64 lr;
 quint8 q[128];
};

```

相关：

```

static void
gum_stalker_infect (GumThreadId thread_id,
 GumCpuContext * cpu_context,
 gpointer user_data)

```

- `gum_process_modify_thread()` 内部实现
  - Linux/Android: ptrace
    - GDB也用的这个：挂载到进程上，读写寄存器

Stalker每次只处理一个代码块block

内部机制：

新申请一块内存，写入给原始代码中加了调试代码后的代码

加的指令，用于生成事件、提供其他Stalker所支持的功能。

以及根据情况去relocate重定位指令代码。

比如对于下面代码，要重定位：

- ADR Address of label at a PC-relative offset.
- `ADR Rd, label`
- `Rd` Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.
- `label` Is the program label whose address is to be calculated. It is an offset from the address of this instruction, in the range ±1MB.

底层通过Gum的Relocator

[frida-gum/gumarm64relocator.c at main · frida/frida-gum · GitHub](#)

现在，回想一下我们说过潜行者一次工作一个块。那么我们如何检测下一个块呢？我们还记得每个块也以分支指令结尾，如果我们修改这个分支以分支回 Stalker 引擎，但确保我们存储分支打算结束的目的地，我们可以检测下一个块并在那里重定向执行。这个相同的简单过程可以一个接一个地继续。

Stalker=潜行者

现在，这个过程可能有点慢，因此我们可以应用一些优化。首先，如果我们多次执行相同的代码块（例如循环，或者可能只是一个多次调用的函数），我们不必重新检测它。我们可以重新执行相同的检测代码。出于这个原因，我们保留了一个哈希表，其中包含我们之前遇到的所有块以及我们放置块的检测副本的位置。

其次，当遇到呼叫指令时，在发出检测的呼叫后，我们随后会发出一个着陆板，我们可以返回该着陆板而无需重新进入 Stalker。Stalker 使用记录真实返回地址 (`real_address`) 和此着陆垫 (`code_address`) 的 `GumExecFrame` 结构构建了一个侧堆栈。当一个函数返回时，我们发出代码，该代码将根据 `real_address` 检查侧堆栈中的返回地址，如果匹配，它可以简单地返回到 `code_address`，而无需重新进入运行时。这个着陆板最初将包含进入 Stalker 引擎以检测下一个块的代码，但稍后可以将其反向修补以直接分支到该块。这意味着可以处理整个返回序列，而无需输入和离开 Stalker。

如果返回地址与存储的 `GumExecFrame real_address` 不匹配，或者我们在侧堆栈中的空间不足，我们只需从头开始重新构建一个新的。我们需要在应用程序代码执行时保留 `LR` 的值，以便应用程序不能使用它来检测 Stalker 的存在（反调试），或者如果它将其用于除简单返回之外的任何其他目的（例如，在代码部分中引用内联数据）。此外，我们希望 Stalker 能够随时取消关注，所以我们不想不得不返回我们的堆栈来更正我们在此过程中修改的 `LR` 值。

最后，虽然我们总是用对 Stalker 的调用来替换分支以检测下一个块，但根据 `Stalker.trustThreshold` 的配置，我们可能会对此类检测代码进行反向修补，以将调用替换为下一个检测块的直接分支。确定性分支（例如，目的地是固定的，分支不是有条件的）很简单，我们可以将分支替换为下一个块。但是我们也可以处理条件分支，如果我们检测两个代码块（一个是分支，一个是不是）。然后，我们可以将原始条件分支替换为一个条件分支，该条件分支将控制流定向到获取分支时遇到的块的检测版本，然后是另一个检测块的无条件分支。我们还可以部分处理目标不是静态的分支。假设我们的分支是这样的：

`BR X0` 这种指令在调用函数指针或类方法时很常见。虽然 `X0` 的值可以更改，但通常它实际上总是相同的。在这种情况下，我们可以将最终的分支指令替换为代码，该代码将 `X0` 的值与我们的已知函数进行比较，如果它将分支与代码的检测副本的地址匹配。然后，如果不匹配，则可以将无条件分支返回到 Stalker 引擎。因此，如果函数指针 `say` 的值发生了变化，那么代码仍然有效，无论我们最终到达哪里，我们都将重新输入 Stalker 和乐器。但是，如果如我们预期的那样它保持不变，那么我们可以完全绕过 Stalker 引擎并直接进入仪器化功能。



## frida-trace

- frida-trace
  - 官网文档
    - [frida-trace](#)
  - 官网代码
    - [frida-tools/frida\\_tools/tracer.py](#)
  - 核心参数解释
    - 调试目标方式
      - frida-trace 去hook调试的目标的写法，是和 frida 是一样的
        - 详见：[frida调试目标方式](#)
    - 通用逻辑
      - `-m`、`-M` 等参数，支持通配符
        - `*`：表示任意个数的字符 =所有的（类或方法名）
        - 举例
 

```
-m "-[NSURL *]" // 匹配NSURL类的所有实例方法
-m "+[NSURL *]" // 匹配NSURL类的所有类方法
-m "**[NSURL *]" // 匹配NSURL类的所有方法
-m "**[*URL *]" // 匹配以URL结尾类的所有方法
-m "**[URL* *]" // 匹配以URL开头类的所有方法
-m "**[*URL* *]" // 匹配包含URL的类的所有方法
-m "**[*URL* *login*]" // 匹配包含URL的类的带login的所有方法
```
      - 注意
        - 不支持：`?`
          - 如果用了 `?`，则会报错： Error: invalid query; format is: -
 

```
[NS*Number foo:bar:], +[Foo foo*] or *[Bar baz]
```
      - 参数
        - `-m`
          - 语法：`-m OBJC_METHOD, --include-objc-method OBJC_METHOD`
            - include OBJC\_METHOD
          - 含义：（要去hook的）包含的ObjC的（类和）方法
        - `-M`
          - 语法：`-M OBJC_METHOD, --exclude-objc-method OBJC_METHOD`
            - exclude OBJC\_METHOD
          - 含义：（不要hook的）排除掉的ObjC的（类和）方法
        - `-O`
          - 语法：`-O FILE, --options-file FILE`
            - text file containing additional command line options
          - 含义：用Option文件，包含额外的命令行参数
            - 典型用法是，觉得 `-m`、`-M` 等要去hook和要去排除掉的类和方法，太多了，就转去放到Option文件中
      - 举例
        - 概述

- hook调试 Preferences : -m 、 -M 等参数直接放到命令行中

- hook单个的类: AAUISignInViewController

```
frida-trace -U -F com.apple.Preferences -m "*[AAUISignInViewController *]"
```

- (用 \* 匹配) 去hook多个类, 且排除特定 (实际上会输出很多次但对调试逻辑没帮助) 的函数

```
frida-trace -U -F com.apple.Preferences -m "*[AA* *]" -M "-[ASDBundle copyWithZone:]" -M "-[* copyWithZone:]" -M "-[AAUILabel *]"
```

- 通用参数含义说明

- -U -F com.apple.Preferences : 调试目标设备和目标app是和frida写法一致
  - -U : 调试设备是 (Mac中通过) USB连接的iPhone
  - -F com.apple.Preferences : 调试的app是当前iPhone中, 最前端所显示的app = 设置app = 包名是 com.apple.Preferences
- 但是后续要trace的函数写法, 是frida-trace所特有的语法
  - -m xxx : include=包含要调试的Objc的类
  - -M xxx : exclude=排除掉Objc的类, 不调试

- hook调试 akd : 觉得 -m 、 -M 等参数太多, 则放到 -o 的Option文件中

```
frida-trace -U -n akd -O akdObjcMethods.txt
```

- Option文件 akdObjcMethods.txt

```
-m "*[AA* *]"
-m "*[AK* *]"
-m "*[AS* *]"
-m "*[NSXPC* *]"
-M "-[NSXPC*coder *]"
-M "-[NSXPConnection replacementObjectForEncoder:object:]"
```

- 一些用法

```
frida-trace -U -i "CCCryptorCreate*" Twitter
frida-trace --decorate -i "recv*" -i "send*" Safari
frida-trace -m "-[NSView drawRect:]" Safari
frida-trace -U -f com.toyopagroup.picaboo -I "libcommonCrypto*"
frida-trace -U -f com.google.android.youtube --runtime=v8 -j '*!*certificate*/isu'
frida-trace -U -i "Java_*" com.samsung.faceservice
frida-trace -p 1372 -i "msvcrt.dll!*mem*"
frida-trace -p 1372 -i "*open*" -x "msvcrt.dll!*open*"
frida-trace -p 1372 -a "libjpeg.so!0x4793c"
```

- 详见

- [frida-trace的iOS的ObjC的举例](#)



## frida-trace的help语法

```

→ ~ frida-trace --help
usage: frida-trace [options] target

positional arguments:
 args extra arguments and/or target

options:
 -h, --help show this help message and exit
 -D ID, --device ID connect to device with the given ID
 -U, --usb connect to USB device
 -R, --remote connect to remote frida-server
 -H HOST, --host HOST connect to remote frida-server on HOST
 --certificate CERTIFICATE
 speak TLS with HOST, expecting CERTIFICATE
 --origin ORIGIN connect to remote server with "Origin" header set to ORIGIN
 --token TOKEN authenticate with HOST using TOKEN
 --keepalive-interval INTERVAL
 set keepalive interval in seconds, or 0 to disable (defaults to
 -1 to auto-select based on transport)
 --p2p establish a peer-to-peer connection with target
 --stun-server ADDRESS
 set STUN server ADDRESS to use with --p2p
 --relay address,username,password,turn-{udp,tcp,tls}
 add relay to use with --p2p
 -f TARGET, --file TARGET
 spawn FILE
 -F, --attach-frontmost
 attach to frontmost application
 -n NAME, --attach-name NAME
 attach to NAME
 -N IDENTIFIER, --attach-identifier IDENTIFIER
 attach to IDENTIFIER
 -p PID, --attach-pid PID
 attach to PID
 -W PATTERN, --await PATTERN
 await spawn matching PATTERN
 --stdio {inherit,pipe}
 stdio behavior when spawning (defaults to "inherit")
 --aux option set aux option when spawning, such as "uid (int)42" (supported
types are: string, bool, int)
 --realm {native,emulated}
 realm to attach in
 --runtime {qjs,v8} script runtime to use
 --debug enable the Node.js compatible script debugger
 --squench-crash if enabled, will not dump crash report to console
 -O FILE, --options-file FILE
 text file containing additional command line options
 --version show program's version number and exit
 -I MODULE, --include-module MODULE
 include MODULE
 -X MODULE, --exclude-module MODULE

```

```
 exclude MODULE
-i FUNCTION, --include FUNCTION
 include [MODULE!]FUNCTION
-x FUNCTION, --exclude FUNCTION
 exclude [MODULE!]FUNCTION
-a MODULE!OFFSET, --add MODULE!OFFSET
 add MODULE!OFFSET
-T INCLUDE_IMPORTS, --include-imports INCLUDE_IMPORTS
 include program's imports
-t MODULE, --include-module-imports MODULE
 include MODULE imports
-m_OBJC_METHOD, --include-objc-method_OBJC_METHOD
 include_OBJC_METHOD
-M_OBJC_METHOD, --exclude-objc-method_OBJC_METHOD
 exclude_OBJC_METHOD
-j JAVA_METHOD, --include-java-method JAVA_METHOD
 include JAVA_METHOD
-J JAVA_METHOD, --exclude-java-method JAVA_METHOD
 exclude JAVA_METHOD
-s DEBUG_SYMBOL, --include-debug-symbol DEBUG_SYMBOL
 include_DEBUG_SYMBOL
-q, --quiet do not format output messages
-d, --decorate add module name to generated onEnter log statement
-S PATH, --init-session PATH
 path to JavaScript file used to initialize the session
-P PARAMETERS_JSON, --parameters PARAMETERS_JSON
 parameters as JSON, exposed as a global named 'parameters'
-o OUTPUT, --output OUTPUT
 dump messages to file
```

## frida-tools

`frida-tool` 是Frida的一套工具的集合，包括：

- 相对常用的
  - `frida-trace`
  - `frida-ps`
  - `frida-ls`
- 不太常用的
  - `frida-apk`
  - `frida-compile`
  - `frida-create`
  - `frida-discover`
  - `frida-join`
  - `frida-ls-devices`
  - `frida-kill`
  - `frida-pull`
  - `frida-push`
  - `frida-rm`

对应源码是：[frida-tools](#)

## 安装frida-tools

Mac 中用 Python 的 pip3 去安装：

```
pip3 install frida-tools
```

## 其他工具

- [gum-graft](#)

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：  
2023-06-23 23:20:40

## frida-ps

- `frida-ps`
  - 是 [frida-tools](#) 中一个常用的工具
  - 作用：显示系统进程列表
    - 类似于 `ps`
    - 支持显示当前（Mac等）PC 端的进程和（iPhone 等）移动端中的进程
  - 官网文档
    - [frida-ps](#)

## 用法

- 显示当前 PC 端进程：不加参数

```
frida-ps
```

- 显示 USB 连接的（iPhone 等）移动端的（默认显示）所有进程，进程：`-U`

```
frida-ps -U
```

- 显示 USB 连接的（iPhone 等）移动端的正在运行的应用程序的进程：`-Ua` = USB + only (running) applications

```
frida-ps -Ua
```

- 显示 USB 连接的（iPhone 等）移动端的应用程序的已安装的全部（正在运行和没有运行）的应用程序的进程：`-Uai` = USB + only (running) applications + (not running but) installed

```
frida-ps -Uai
```

另外：

- 可以加 `grep` 去搜索特定 app
  - 举例

```
→ frida-ps -Ua | grep -i preferences
46847 设置
com.apple.Preferences
```

## 举例

### frida-ps

```
→ ~ frida-ps
PID Name

```

```

1205 weatherd
639 wifianalyticsd
612 wifip2pd
913 wifivelocityd
11153 wine64-preloader
...
11120 wineserver
633 writeconfig
17438 xpcroleaccountd
...
94172 zsh

```

**frida-ps -U**

```

crifan@licrifandeMacBook-Pro: ~/dev/dev_tool/reverse_security/iOS/palera1n$ frida-ps -U
PID Name

262 App Store
328 Filza
260 GTA Car Tracker
261 Safari浏览器
291 Shadowrocket
350 Sileo
265 TrollStore
316 palera1n
234 微信
315 截屏
312 搜索
264 播客
263 设置
100 ACCHWComponentAuthService
339 ASPCarryLog
230 AppPredictionIntentsHelperService
252 AppSSODaemon
81 AppleCredentialManagerDaemon
140 BlueTool
219 CAResultingService
335 CMFSyncAgent
276 CacheDeleteAppContainerCaches
285 CacheDeleteExtension
307 CategoriesService

```

```

237 CategoriesService
214 CategoriesService
187 CategoriesService
119 CloudKeychainProxy
91 CommCenter
185 CommCenterMobileHelper
188 ContextService
322 EnforcementService
...

```

**frida-ps -Ua**

```

→ ~ frida-ps -Ua
 PID Name Identifier
5 -----
27692 Sileo org.coolstar.SileoStore
23650 微信 com.tencent.xin
11941 日历 com.apple.mobilecal
19641 设置 com.apple.Preferences

```

**frida-ps -Uai**

```

→ ~ frida-ps -Uai
 PID Name Identifier
5 -----
27692 Sileo org.coolstar.SileoStore
23650 微信 com.tencent.xin
11941 日历 com.apple.mobilecal
19641 设置 com.apple.Preferences
- App Store com.apple.AppStore
- Apple Store com.apple.store.Jolly
- CocoaTop ru.domo.cocoatop64
- FaceTime通话 com.apple.facetime
- Filza com.tigisoftware.Filza
- GTA Car Tracker com.icraze.gtatracker
- Safari浏览器 com.apple.mobilesafari
- Shadowrocket com.liuguangming.Shadowrocket
- ShowSystemInfo com.crifan.ShowSystemInfo
- Substitute com.ex.substitute.settings
- TrollStore com.opa334.TrollStore
- Watch com.apple.Bridge
- iTunes Store com.apple.MobileStore
- palera1n com.llsc12.palera1nLoader
- 信息 com.apple.MobileSMS
- 健康 com.apple.Health
- 反馈 com.apple.appleseed.FeedbackAssistant
- 图书 com.apple.iBooks
- 地图 com.appleMaps
- 备忘录 com.apple.mobilenotes
- 天气 com.apple.weather

```

|         |                             |
|---------|-----------------------------|
| - 家庭    | com.apple.Home              |
| - 快捷指令  | com.apple.shortcuts         |
| - 指南针   | com.apple.compass           |
| - 提示    | com.apple.tips              |
| - 提醒事项  | com.apple.reminders         |
| - 播客    | com.apple.podcasts          |
| - 放大器   | com.apple.Magnifier         |
| - 文件    | com.apple.DocumentsApp      |
| - 时钟    | com.apple.mobletimer        |
| - 查找    | com.apple.findmy            |
| - 测距仪   | com.apple.measure           |
| - 照片    | com.apple.mobileslideshow   |
| - 爱思全能版 | com.ownbook.notes           |
| - 电话    | com.apple.mobilephone       |
| - 相机    | com.apple.camera            |
| - 翻译    | com.apple.Translate         |
| - 股市    | com.apple.stocks            |
| - 视频    | com.apple.tv                |
| - 计算器   | com.apple.calculator        |
| - 语音备忘录 | com.apple.VoiceMemos        |
| - 通讯录   | com.apple.MobileAddressBook |
| - 邮件    | com.apple.mobilemail        |
| - 钱包    | com.apple.Passbook          |
| - 音乐    | com.apple.Music             |

## 语法help

```
→ ~ frida-ps --help
usage: frida-ps [options]

options:
 -h, --help show this help message and exit
 -D ID, --device ID connect to device with the given ID
 -U, --usb connect to USB device
 -R, --remote connect to remote frida-server
 -H HOST, --host HOST connect to remote frida-server on HOST
 --certificate CERTIFICATE
 speak TLS with HOST, expecting CERTIFICATE
 --origin ORIGIN connect to remote server with "Origin" header set to ORIGIN
 --token TOKEN authenticate with HOST using TOKEN
 --keepalive-interval INTERVAL
 set keepalive interval in seconds, or 0 to disable (defaults to
-1 to auto-
 select based on transport)
 --p2p establish a peer-to-peer connection with target
 --stun-server ADDRESS
 set STUN server ADDRESS to use with --p2p
 --relay address,username,password,turn-{udp,tcp,tls}
 add relay to use with --p2p
 -O FILE, --options-file FILE
 text file containing additional command line options
 --version show program's version number and exit
```

```
-a, --applications list only applications
-i, --installed include all installed applications
-j, --json output results as JSON
```

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:  
2023-06-23 23:19:37

## frida-ls

- `frida-ls`
  - 是 `frida-tools` 中一个工具
  - 作用：显示系统中的文件列表
    - 类似于 `ls`
    - 支持：当前（Mac等）PC 端的文件和（iPhone 等）移动端中的文件

## 用法

- 显示当前 PC 端的文件：不加参数

```
frida-ls
```

- 显示 USB 连接的（iPhone 等）移动端中的所有文件：`-U == USB`

```
frida-ls -U
```

## 举例

### frida-ls

```
→ ~ frida-ls
drwxr-xr-x 20 root wheel 640 Thu Feb 9 09:39:53 2023 .
drwxr-xr-x 20 root wheel 640 Thu Feb 9 09:39:53 2023 ..
lrwxr-xr-x 1 root admin 36 Thu Feb 9 09:39:53 2023 .VolumeIcon.icns -> System/Volumes/.VolumeIcon.icns
----- 1 root admin 0 Thu Feb 9 09:39:53 2023 .file
drwxr-xr-x 2 root wheel 64 Thu Feb 9 09:39:53 2023 .vol
drwxrwxr-x 45 root admin 1440 Wed Jun 14 14:45:54 2023 Applications
drwxr-xr-x 66 root wheel 2112 Fri May 12 19:14:14 2023 Library
drwxr-xr-x 10 root wheel 320 Thu Feb 9 09:39:53 2023 System
drwxr-xr-x 5 root admin 160 Sat Mar 18 14:26:26 2023 Users
drwxr-xr-x 3 root wheel 96 Mon Jun 5 02:34:08 2023 Volumes
drwxr-xr-x 39 root wheel 1248 Thu Feb 9 09:39:53 2023 bin
drwxrwxr-x 2 root admin 64 Mon Oct 24 06:20:41 2022 cores
dr-xr-xr-x 4 root wheel 5127 Sat Mar 18 14:26:04 2023 dev
lrwxr-xr-x 1 root wheel 11 Thu Feb 9 09:39:53 2023 etc -> private/etc
lrwxr-xr-x 1 root wheel 25 Sat Mar 18 14:27:10 2023 home -> /System/Volumes/Data/home
drwxr-xr-x 7 root wheel 224 Mon May 29 03:57:47 2023 opt
drwxr-xr-x 6 root wheel 192 Sat Mar 18 14:27:03 2023 private
drwxr-xr-x 64 root wheel 2048 Thu Feb 9 09:39:53 2023 sbin
lrwxr-xr-x 1 root wheel 11 Thu Feb 9 09:39:53 2023 tmp -> private/tmp
drwxr-xr-x 11 root wheel 352 Thu Feb 9 09:39:53 2023 usr
lrwxr-xr-x 1 root wheel 11 Thu Feb 9 09:39:53 2023 var -> private/var
```

### frida-ls -U

- 注：Mac中用USB连接的是，iOS 15.0的 iPhone8，已用 palera1n 实现 rootful 越狱

```
→ ~ frida-ls -U
drwxr-xr-x 26 root wheel 832 Fri May 5 07:03:51 2023 .
drwxr-xr-x 26 root wheel 832 Fri May 5 07:03:51 2023 ..
-rw-r--r-- 1 mobile staff 6148 Mon Feb 6 03:41:36 2023 .DS_Store
drwx----- 2 root wheel 64 Thu Sep 16 05:39:21 2021 .ba
----- 1 root admin 0 Thu Sep 16 05:39:21 2021 .file
drwx----- 2 root wheel 64 Fri May 5 06:58:46 2023 .fsevents
drwx----- 2 root wheel 64 Thu Sep 16 05:39:21 2021 .mb
-rw-r--r-- 1 mobile staff 0 Sun Nov 20 01:58:22 2022 .palecursus_strapped
-rw-r--r-- 1 root staff 0 Fri May 5 07:03:50 2023 .procursus_strapped
drwxrwxr-x 121 root admin 3872 Wed Jun 14 13:34:36 2023 Applications
drwxr-xr-x 7 root wheel 306 Thu Sep 16 05:39:46 2021 Developer
drwxrwxr-x 26 root admin 832 Thu May 18 15:05:28 2023 Library
drwxr-xr-x 5 root wheel 160 Thu Sep 16 05:39:21 2021 System
lrwxr-xr-x 1 root wheel 12 Fri May 5 07:03:50 2023 User -> //var/mobile
drwxr-xr-x 39 root wheel 1248 Fri May 5 07:40:50 2023 bin
drwxr-xr-x 2 root wheel 64 Fri May 5 07:03:50 2023 boot
drwxrwxr-x 4 root admin 440 Wed Jun 14 06:35:22 2023 cores
dr-xr-xr-x 4 root wheel 1492 Wed Jun 14 06:35:21 2023 dev
lrwxr-xr-x 1 root wheel 11 Thu Sep 16 05:39:21 2021 etc -> private/etc
drwxr-xr-x 2 root wheel 64 Fri May 5 07:03:50 2023 lib
drwxr-xr-x 2 root wheel 64 Fri May 5 07:03:50 2023 mnt
drwxr-xr-x 7 root wheel 224 Thu Sep 16 05:39:21 2021 private
drwxr-xr-x 24 root wheel 768 Fri May 5 07:39:45 2023 sbin
lrwxr-xr-x 1 root wheel 15 Thu Sep 16 05:39:21 2021 tmp -> private/var/tmp
drwxr-xr-x 10 root wheel 320 Fri May 5 07:03:49 2023 usr
lrwxr-xr-x 1 root admin 11 Thu Sep 16 05:39:21 2021 var -> private/var
```

## 语法help

```
→ ~ frida-ls --help
usage: frida-ls [options] [FILE]...

positional arguments:
 files files to list information about

options:
 -h, --help show this help message and exit
 -D ID, --device ID connect to device with the given ID
 -U, --usb connect to USB device
 -R, --remote connect to remote frida-server
 -H HOST, --host HOST connect to remote frida-server on HOST
 --certificate CERTIFICATE
 speak TLS with HOST, expecting CERTIFICATE
 --origin ORIGIN connect to remote server with "Origin" header set to ORIGIN
 --token TOKEN authenticate with HOST using TOKEN
 --keepalive-interval INTERVAL
 set keepalive interval in seconds, or 0 to disable (defaults to
-1 to auto-
 --p2p select based on transport)
 establish a peer-to-peer connection with target
```

```
--stun-server ADDRESS
 set STUN server ADDRESS to use with --p2p
--relay address,username,password,turn-{udp,tcp,tls}
 add relay to use with --p2p
-O FILE, --options-file FILE
 text file containing additional command line options
--version
 show program's version number and exit
```

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:  
2023-06-22 22:01:30

## frida-ls-devices

- 官网文档: [frida-ls-devices](#)

### 举例

```
→ frida-ls-devices
Id Type Name OS

local local crifanlideMacBook-Pro.local macOS 13
.2.1
abdc0dd961c3cb96f5c4afe109de4eb48b88433a usb iPhone8_150 iPhone 0
S 15.0
socket remote Local Socket
```

### 语法help

```
→ frida-ls-devices --help
usage: frida-ls-devices [options]

options:
-h, --help show this help message and exit
-O FILE, --options-file FILE
 text file containing additional command line options
--version show program's version number and exit
```

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:  
2023-06-22 23:19:24

## 其他相关

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：  
2023-06-23 08:10:11

## frida-server

越狱手机 iPhone 安装 Frida 后，就可以通过 Frida 插件的 已安装文件：

16:06



&lt; 返回

## 已安装的文件

▼ /

▼ Library/

▼ LaunchDaemons/

re.frida.server.plist

▼ usr/

▼ lib/

▼ frida/

frida-agent.dylib

▼ sbin/

frida-server



精选



新闻



软件源



软件包



搜索

看到安装了2个核心文件：

- /usr/lib/frida/frida-agent.dylib
- /usr/sbin/frida-server

- 其中 frida-server 就是 Frida 的server端

## frida-server 语法help

```
iPhone11-151:~ root# frida-server --help
Usage:
 frida [OPTION?]

Help Options:
 -h, --help Show help options

Application Options:
 --version Output version information and exit
 -l, --listen ADDRESS Listen on ADDRESS
 --certificate CERTIFICATE Enable TLS using CERTIFICATE
 --origin ORIGIN Only accept requests with ?origin? header matching ORIGIN (by default any origin will be accepted)
 --token=TOKEN Require authentication using TOKEN
 --asset-root ROOT Serve static files inside ROOT (by default no files are served)
 -d, --directory DIRECTORY Store binaries in DIRECTORY
 -D, --daemonize Detach and become a daemon
 --policy-softener system internal Select policy softener
 -P, --disable-preload Disable preload optimization
 -C, --ignore-crashes Disable native crash reporter integration
 -v, --verbose Be verbose
```

# Frida用法举例

- 概述
  - Frida例子
    - 官网
      - [Frida CodeShare](#)
    - 其他
      - [oleavr/frida-agent-example: Example Frida agent written in TypeScript](#)
      - [frida-presentations](#)
      - [Frida在iOS平台进行OC函数hook的常用方法 | 8Biiit's Blog](#)
      - [personal\\_script/Frida\\_script at master · lich4/personal\\_script · GitHub](#)
      - [frida-ios-hook/frida-scripts](#)
      - [Oxdea/frida-scripts](#)
      - [misc/frida-read-process-memory.py at master · poxyran/misc · GitHub](#)
      - [frida-snippets/README.md at master · iddoeldor/frida-snippets · GitHub](#)
- 详解
  - [Frida逆向实例和工具函数](#)

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:  
2024-03-19 09:44:38

## 经验心得

此处整理frida中的一些经验和心得。

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:  
2023-06-23 22:04:58

# hook函数

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：  
2023-06-24 23:42:49

# frida

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：  
2023-06-24 21:57:15

# Interceptor

## 心得

### 通过函数地址去hook拦截函数

- 已知：函数的（二进制内偏移量）地址
- 想要：去hook拦截函数
- 举例说明
  - akd中函数 `__lldb_unnamed_symbol12575$$akd` 的二进制内偏移量是：`0xa0460`
    - 思路：动态计算加上模块后的真实函数地址，再去hook
    - 核心代码：

```
console.log("dynamicDebug/frida/scripts/fridaStalker_akdSymbol12575.js");
// var akdSymbol12575_functionAddress = 0x1000a0460;
var akdSymbol12575_functionAddress = 0xa0460;
// arm64 akd: __lldb_unnamed_symbol12575$$akd
const moduleName = "akd";
const moduleBaseAddress = Module.findBaseAddress(moduleName);
console.log("moduleName=", moduleName, "moduleBaseAddress=", moduleBaseAddress);
const functionRealAddress = moduleBaseAddress.add(akdSymbol12575_functionAddress);
console.log("functionRealAddress=", functionRealAddress);
Interceptor.attach(functionRealAddress, {
 onEnter: function(args) {
 var arg0 = args[0]
 var arg1 = args[1]
 var arg2 = args[2]
 console.log("arg0=" + arg0 + ", arg1=" + arg1 + ", arg2=" + arg2);
 var curTid = Process.getCurrentThreadId();
 console.log("curTid=", curTid);
 ...
});
```



#### ■ 输出

```
dynamicDebug/frida/scripts/fridaStalker_akdSymbol12575.js
moduleName= akd moduleBaseAddress= 0x102b40000
functionRealAddress= 0x102be0460
arg0 0xfffffffffffffe, arg1 0x16d346838, arg2 0x16d346838
curTid= 35847
...
```

- 完整代码和输出，详见：[\\_\\_lldb\\_unnamed\\_symbol12575\\$\\$akd](#)

### onLeave中如何获取到self类本身实例变量

- 思路：`onEnter` 时，把`self`变量赋值保存给 `this.xxx`，`onLeave` 时去读取即可

- 代码

```

Interceptor.attach(curMethod.implementation, {
 onEnter: function(args) {
 this.argSelfObj = argSelfObj
 ...
 },
 onLeave: function (retval) {
 var retValObj = ObjC.Object(retval);
 var retValObjStr = retValObj.toString();
 console.log("onLeave: retValObj=" + retValObj + ", retValObjstr=" + retValObjStr)

 const curSelfObj = this.argSelfObj
 console.log("curSelfObj=" + curSelfObj)
 }
})

```

- 输出

```

=====
-[NSXPConnection setExportedObject:] =====
argSelfObj: <ACTrackedXPConnection: 0x154907c50> connection to service named com.apple.accountsds.accountmanager
onLeave: retValObj nil, retValObjStr nil
curSelfObj=<ACTrackedXPConnection: 0x154907c50> connection to service named com.apple.accountsds.accountmanager

```

## 问题

### TypeError: cannot read property 'implementation' of undefined

- 问题：

写frida的hook代码调试ObjC的函数

```

function hook_specific_method_of_class(className, funcName)
{
 var hook = ObjC.classes[className][funcName];
 Interceptor.attach(hook.implementation, {
 onEnter: function(args) {
 console.log("[*] Detected call to: " + className + " -> " + funcName);
 }
 });
}

//Your class name and function name here
hook_specific_method_of_class("AAUISignInViewController", "_nextButtonSelected:")

```

始终报错：

```
TypeError: cannot read property 'implementation' of undefined
```

- 原因：此处iOS的ObjC的函数名 `_nextButtonSelected:` 写法有误
- 解决办法：改为正确的写法：`- _nextButtonSelected:`
  - 注：此处的ObjC的类的函数名的写法：
    - 加上类型是 `class` 的还是 `instance`，所对应的 加号 `= +` 还是 减号 `= -`，以及中间带上空格 `=`
  - 相关代码改动：
 

```
hook_specific_method_of_class("AAUISignInViewController", "- _nextButtonSelected:")
```
  - 完整代码详见：[单个类的单个函数](#)

## frida去hook函数时确保已执行函数但不触发不输出日志log

- 问题：之前用了 `ApiResolver` 去写了hook代码，后续调试的函数确保被触发了，但是却没输出我们希望的日志
- 原因：此处 `ApiResolver` 只是search查找，函数是否存在，只运行一次，后续的确，本身就不会再次触发，所以后续代码被触发时，就不会打印日志
- 解决办法：真正的要实现，hook函数，待函数被运行后触发日志打印，应该改用：`Interceptor`
  - 注：
    - `ApiResolver` 的完整代码和解释，详见：[ApiResolver](#)
    - `Interceptor` 的完整示例代码，详见：[Interceptor=hook函数](#)

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：  
2023-06-26 23:16:00

## Stalker

### 实际内存地址和Stalker输出地址不匹配不一致

某次Stalker调试的log是：

```
===== dynamicDebug/frida/scripts/fridaStalker_akdSymbol12575.js =====
funcRelativeStartAddr=656480, funcSize=9416, funcRelativeEndAddr=665896
moduleName:akd, moduleBaseAddress=0x10054c000
funcRealStartAddr=0x1005ec460, funcRealEndAddr=0x1005ec4609416
[iPhone::PID::13098]-> ----- arg0=0xfffffffffffffe, arg1=0x16fe26838, arg2=0x16fe268
38, arg3=0xfffffffffffffe
curTid=26635

+++ into iterator=
[0x1071dbcd8] b #0x1071dbce8
+++ into iterator=
[0x1071dbce8] str wZR, [x19, #0x90]
[0x1071dbcec] ldr x0, [x19, #0xa0]
[0x1071dbcfc0] str xZR, [x19, #0xa0]
[0x1071dbcfc4] cbz x0, #0x1071dbcfc
+++ into iterator=
[0x1071dbcfc8] bl #0x1070dfef8
+++ into iterator=
[0x1071dbcfc] ldr x0, [x19, #0x98]
[0x1071dbd00] str xZR, [x19, #0x98]
[0x1071dbd04] cbz x0, #0x1071dbd14
...
```

其中指令地址看起来不匹配=不一致：

- 指令地址：0x1071dbcd8
- 函数二进制内偏移量：0xa0460
  - 此处akd模块基地址：0x10054c000
    - 所以函数的实际真实起始地址：0x1005ec460

后来才明白是：

Frida的Stalker中，看到2个函数地址不匹配：

- 函数实际内存起始地址：0x1005ec460
  - 0x1005exxxx之类的地址
- Stalker中运行期间的函数地址：0x1071dbcd8
  - 0x1071dxxxx之类的地址

的原因：

Stalker内部的hook指令的原理就是：把当前代码拷贝一份，加上hook代码及其他逻辑

而此处的：拷贝一份，到别处，别的一段内存地址空间，此处就是指的是：

- 0x1071dbc8
  - 0x1071dxxxx 之类的地址

所以，原始函数代码和被Stalker去hook的代码，两个地址是不同的，是可以理解的。

## Frida的Stalker的transform中Instruction的属性

Frida的Stalker的transform中Instruction指令，有哪些属性，参考官网：

<https://frida.re/docs/javascript-api/#instruction>

得知有如下属性：

- address
- next
- size
- mnemonic
- opStr
- operands
- regsAccessed
- regsRead
- regsWritten
- groups
- `toString()`
- `toJSON()`

注：但是没有（其实希望也有的）`bytes = opcode` 属性。

而这些属性的来源是：`Capstone`

- Capstone
  - Java接口
    - [capstone/Capstone.java at next · capstone-engine/capstone · GitHub](#)
  - Python接口
    - [capstone/init.py at next · capstone-engine/capstone · GitHub](#)

## 举例

如之前示例代码：

[\\_\\_lldb\\_unnamed\\_symbol2575\\$\\$akd](#)

中的，但是注释掉的代码：

(注：当时没加 `regsAccessed` 和 `toJSON()` )

```
// console.log("instruction: address=" + instruction.address
// + ",next=" + instruction.next()
// + ",size=" + instruction.size
// + ",mnemonic=" + instruction.mnemonic
// + ",opStr=" + instruction.opStr
```

```

 // + ",operands=" + JSON.stringify(instruction.operands)
 // + ",regsAccessed=" + JSON.stringify(instruction.regsAccessed)
 // + ",regsRead=" + JSON.stringify(instruction.regsRead)
 // + ",regsWritten=" + JSON.stringify(instruction.regsWritten)
 // + ",groups=" + JSON.stringify(instruction.groups)
 // + ",toString()=" + instruction.toString()
 // + ", toJSON()=" + instruction.toJSON()
 //);

```

取消注释后，可以输出log：

```

instruction: address 0x10f4ecef4,next 0x4,size 4,mnemonic ldr,opStr=x0, #0x10f4ecf78,operands=[{"type":"reg","value":"x0","access":"w"}, {"type":"imm","value":4551790456,"access":"r"}],regsRead=[],regsWritten=[],groups=[],toString()=ldr x0, #0x10f4ecf78 [0x10f4ecef4] ldr x0, #0x10f4ecf78

instruction: address 0x10f4ecef8,next 0x4,size 4,mnemonic bl,opStr=#0x1091a500c,operands=[{"type":"imm","value":4447686668,"access":"r"}],regsRead=[],regsWritten=["lr"],groups=["call","jump","branch_relative"],toString()=bl #0x1091a500c [0x10f4ecef8] bl #0x1091a500c

```

## Frida的Stalker中去判断是否是函数的代码指令的逻辑

- 需求：Frida的Stalker调试期间，想要知道当前所执行的代码，是否属于：真正的要hook的函数的真实代码
- 解决办法

经过调试用如下代码：

```

console.log("===== Frida Stalker hook for arm64 __lldb_unnamed_symbol12575$$akd =====");

// arm64 akd: __lldb_unnamed_symbol12575$$akd
// var funcRelativeStartAddr = 0x1000a0460;
var funcRelativeStartAddr = 0xa0460;
var functionSize = 0x24C8; // 9416 == 0x24C8
var funcRelativeEndAddr = funcRelativeStartAddr + functionSize;
console.log("funcRelativeStartAddr=" + funcRelativeStartAddr + ", functionSize=" + functionSize + ", funcRelativeEndAddr=" + funcRelativeEndAddr);
const moduleName = "akd";
const moduleBaseAddress = Module.findBaseAddress(moduleName);
console.log("moduleName=" + moduleName + ", moduleBaseAddress=" + moduleBaseAddress);
// console.log("moduleName=%s, moduleBaseAddress=%p", moduleName, moduleBaseAddress);
const funcRealStartAddr = moduleBaseAddress.add(funcRelativeStartAddr);
// var funcRealEndAddr = funcRealStartAddr + functionSize;
const funcRealEndAddr = funcRealStartAddr.add(functionSize);
console.log("funcRealStartAddr=" + funcRealStartAddr + ", funcRealEndAddr=" + funcRealEndAddr);
var curTid = null;
Interceptor.attach(funcRealStartAddr, {
 onEnter: function(args) {
 var arg0 = args[0]
 var arg1 = args[1]
 var arg2 = args[2]

```

```

 var arg3 = args[3]
 console.log("----- arg0=" + arg0 + ", arg1=" + arg1 + ", arg2=" + arg2 + ", arg
3=" + arg3);
 var curTid = Process.getCurrentThreadId();
 console.log("curTid=", curTid);
 Stalker.follow(curTid, {
 events: {
 call: true, // CALL instructions: yes please
 ret: false, // RET instructions
 exec: false, // all instructions: not recommended as it's
 block: false, // block executed: coarse execution trace
 compile: false // block compiled: useful for coverage
 },
 // transform: (iterator: StalkerArm64Iterator) => {
 transform: function (iterator) {
 var instruction = iterator.next();
 const startAddress = instruction.address;
 console.log("+++ into iterator: startAddress=" + startAddress);
 // const isAppCode = startAddress.compare(funcRealStartAddr) >= 0 && st
artAddress.compare(funcRealEndAddr) === -1;
 // const isAppCode = (startAddress.compare(funcRealStartAddr) >= 0) &&
(startAddress.compare(funcRealEndAddr) < 0);
 const gt_realStartAddr = startAddress.compare(funcRealStartAddr) >= 0
 const lt_realEndAddr = startAddress.compare(funcRealEndAddr) < 0
 const isAppCode = gt_realStartAddr && lt_realEndAddr
 console.log("isAppCode=" + isAppCode + ", gt_realStartAddr=" + gt_reals
tAddr + ", lt_realEndAddr=" + lt_realEndAddr);
 do {
 if (isAppCode) {
 var curRealAddr = instruction.address;
 var curOffset = curRealAddr.sub(funcRealStartAddr)
 var curOffsetInt = curOffset.toInt32()
 var instructionStr = instruction.toString()
 console.log("\t" + curRealAddr + " <+" + curOffsetInt + ">: " +
instructionStr);
 }
 iterator.keep();
 } while ((instruction = iterator.next()) !== null);
 }
 })
 },
 onLeave: function(retval) {
 console.log("retval:", new ObjC.Object(retval))
 if (curTid != null) {
 Stalker.unfollow(curTid);
 console.log("Stalker.unfollow curTid=", curTid)
 }
 }
});

```

可以输出：

```

===== Frida Stalker hook for arm64 __lldb_unnamed_symbol12575$$akd =====
funcRelativeStartAddr=656480, funcSize=9416, funcRelativeEndAddr=665896

```

```
moduleName akd, moduleBaseAddress 0x10237c000
funcRealStartAddr=0x10241c460, funcRealEndAddr=0x10241e928
[iPhone::akd]-> ----- arg0 0xfffffffffffffff, arg1=0x16dcae838, arg2=0x16dcae838, arg3
 0xfffffffffffffff
curTid= 15635
+++ into iterator: startAddress=0x1089dbcd8
isAppCode false, gt_realStartAddr=true, lt_realEndAddr=false
+++ into iterator: startAddress=0x1089dbce8
...
isAppCode true, gt_realStartAddr=true, lt_realEndAddr=true, arg3=0xfffffffffffffff
 0x10241c470 +18 : stp x22, x21, [sp, #0xc0]
 0x10241c474 +20 : stp x20, x19, [sp, #0xd0]
 0x10241c478 +24 : stp x29, x30, [sp, #0xe0]
 0x10241c47c +28 : add x29, sp, #0xe0
...
```

算是实现了，通过计算，最后用 `isAppCode` 这个变量去判断是否是当前代码

上述代码的所属的完整的示例代码，详见：

[\\_\\_lldb\\_unnamed\\_symbol2575\\$\\$akd](#)

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：  
2023-06-25 23:54:22

## frida-trace

### hook指定的多个二进制库文件

想要去用frida-trace追踪的iOS的ObjC的库=Framework=二进制：

- AppleAccount
- AppleAccountUI
- Accounts
- AccountsDaemon
- AuthKitUI
- AuthKit
- UserManagement

最后是用：

- 去include 多个 module

```
frida-trace -U -F com.apple.Preferences -I "AppleAccount*" -I "UserManagement" -I "Accounts*" -I "AuthKit*"
```

可以运行，但是结果：

- 只找到少数几个函数，不是我们要的
  - -》这么多库，一共才有107个函数
  - -》其中AppleAccount只有7个函数，其中就有上面的\_AALogSystem仅仅是日志的函数，而不是我们要的：很多其他账号登录相关的函数
  - -》正常情况下，应该会有非常多的函数才对

注：

也可以换另外写法：

```
frida-trace -U -F com.apple.Preferences -i "AppleAccount\!" -i "AppleAccountUI\!" -i "Accounts\!" -i "AccountsDaemon\!" -i "AuthKit\!" -i "AuthKitUI\!" -i "UserManagement\!"
```

不过结果都一样：hook的函数都很少，不是我们想要的。

## 输出日志到文件中

- 解决办法：加上 -o 参数，去指定输出log到文件
  - 命令

```
frida-trace -U -F com.apple.Preferences -o traceAccount.log -m "*[AA* *]" -m "*[AK* *]" -m "*[AS* *]" -m "*[NSXPC* *]" -M "-[ASDBundle copyWithZone:]" -M "-[ASDInstallationEvent copyWithZone:]" -M "-[NSXPCEncoder _encodeArrayOfObjects:forKey:]" -M "-[NSXPCEncoder _encodeUnkeyedObject:]" -M "-[NSXPCEncoder _replaceObject:]" -M "-[NSXPCEncoder _checkObject:]" -M "-[NSXPCEncoder _encodeObject:]" -M "-[NSXPCCollection replacementObjectForEncoder:object:]"
```



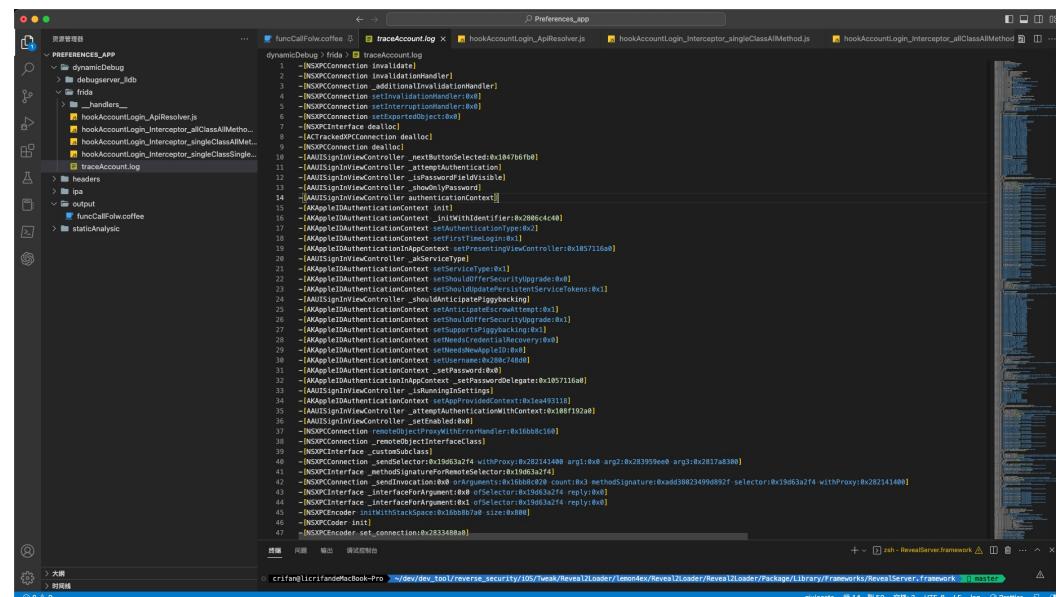
## ○ 缺点

- 之前=输出日志到终端：带颜色和缩进 -> 利于查看函数调用关系

```
 53% 31 11 GB 26 kB↑ 3/15, 17:03 .cDebug(frida (-zsh)) X3

27645 ms +[AMURLProtocol canInitWithRequest:0x280069170] 0 14 kB
27645 ms +[AMURLProtocol canonicalRequestForRequest:0x280069170]
27645 ms -[AMURLProtocol initWithRequest:0x28006a20 cachedResponse:@0 client:0x280d44510]
27646 ms -[AMURLProtocol startLoading]
27646 ms | +[AMNetoServer sharedServer]
27646 ms | | +[AMNetoServer _initWithRequestAndResponseWithCompletion:0x16c031a40]
27646 ms | | | -[AMNetoServer _configurationCacheInvalidatingIfNecessary]
27646 ms | | | | -[AMNetoServer _configurationLock configurationCacheInvalidatingIfNecessary]
27646 ms | | | | -[AMNetoServer _configurationLock configurationCreationDate]
27646 ms | | | | -[AMNetoServer _configurationCache configuration]
27646 ms | | | | -[AMNetoServerConfigurationCache response]
27646 ms | | | | -[AMURLConfiguration urlForRequest:0x280e0606]
27646 ms | | | | -[AMURLProtocol _urlLoading]
27647 ms /* TID 0x2c82c */
27647 ms -[AMURLSession _nativeSessionTask:0x185f6c0 task:0x172745a30 urlRequest:0x172745a30 completionHandler:0x16c2ee650]
27647 ms | +[AMAppleIDSession generateAppleIDHeadersForSessionTask:0x172745a30 withCompletion:0x16c2ee530]
27647 ms | | -[AMAppleIDSession _nativeNetSettlesController]
27647 ms | | | -[AMAppleIDSession _nativeNetSettlesController init]
27647 ms | | | | -[AMAppleIDSession _nativeNetSettlesController _fetchNetSettlesAndProvissionIfNecessary:0x1 withCompletion:0x16c2ee428]
27647 ms | | | | | -[AMAppleIDSession _nativeNetSettlesController fetchNetSettlesAndProvissionIfNecessary:0x1 withCompletion:0x16c2ee428]
27647 ms | | | | | | -[AMAppleIDSession _nativeNetSettlesController _anissetteServiceConnection]
27647 ms | | | | | | | -[AMAppleIDSession _nativeNetSettlesController _INEXPConnection init]
27647 ms | | | | | | | | -[AMAppleIDSession _nativeNetSettlesController _XPCInterface]
27647 ms | | | | | | | | -[AMAppleIDSession _nativeNetSettlesController _setNettedObjectInterface:0x28063ff60]
27647 ms | | | | | | | | -[AMAppleIDSession _nativeNetSettlesController _NSURLConnection setNettedObjectInterface:0x28063ff60]
27647 ms | | | | | | | | -[AMAppleIDSession _nativeNetSettlesController _INEXPConnection setNettedObjectInterface:0x28063ff60]
27647 ms | | | | | | | | -[AMAppleIDSession _nativeNetSettlesController _INEXPConnection resume]
27647 ms | | | | | | | | -[AMAppleIDSession _nativeNetSettlesController _INSPCInterfaceConnection remoteObjectProxyWithExportHandler:0x16c2ee240]
27647 ms | | | | | | | | -[AMAppleIDSession _nativeNetSettlesController _INSPCInterfaceConnection _INEXPInterfaceCustomSubclass]
27647 ms | | | | | | | | -[AMSPCInterface _methodSignatureForRemoteObject:0x16c2ee1a0]
27647 ms | | | | | | | | -[AMSPCInterface _remoteObject:0x16c2ee1a0 selector:0x19d1le17a withProxy:0x28216ad0]
27647 ms | | | | | | | | -[AMSPCInterface _interfaceForArgument:0x16c2ed4f0 selector:0x16c2ee1a0]
27647 ms | | | | | | | | -[AMSPCInterface _initWithStackSpace:0x16c2ed4f0 size:0x800]
27647 ms | | | | | | | | -[AMSPCDecoder _decodeAndInvokeWithEvent:0x283d56760 sequence:0x1 replyInfo:0x2817b3c0]
27647 ms | | | | | | | | -[AMSPCDecoder init]
27647 ms | | | | | | | | | -[INSPCDecoder init]
27647 ms | | | | | | | | | | -[INSPCDecoder _setConnection:0x28332a20]
27647 ms | | | | | | | | | | -[INSPCDecoder _setDecoder:0x28332a20]
27647 ms | | | | | | | | | | -[INSPCDecoder _encodeInvocation:0x16c2ef9f0 isReply:0x0 into:0x28261bb0]
27647 ms | | | | | | | | | | -[INSPCDecoder _decodeXPCObject:0x283d56760]
27647 ms | | | | | | | | | | -[INSPCDecoder _dealloc]
27647 ms | | | | | | | | | | -[INSPCInterface _hasProxiesInReplyBlockArgumentsOfSelector:0x19d1le17a]
27647 ms /* TID 0x2c827 */
27654 ms [NSXPConnection _decodedAndInvokeWithEvent:0x283d56760 sequence:0x1 replyInfo:0x2817b3c0]
27654 ms | -[INSPCDecoder init]
27654 ms | | -[INSPCDecoder init]
27654 ms | | | -[INSPCDecoder _setConnection:0x28332a20]
27655 ms | | | -[INSPCDecoder _decodeXPCObject:0x283d56760 forSelector:0x19d1le17a interface:0x28063ff60]
27655 ms | | | -[INSPCDecoder _decodeXPCObject:0x283d56760 allowingInpMessageSend:0x0 invocation:0x16dbaa00 outArguments:0x0 outArgumentsMaxCount:0x0 outMethodSignature:0x0 isReply:0x1 replySelector:0x19d1le17a]
27655 ms | | | -[INSPCDecoder _startReadingForXPCObject:0x283d56760]
27655 ms | | | | -[INSPCDecoder _selectForAllTowed:0x19d1le17a isReply:0x1 methodSignature:0x16dbaa3f0 allowedClasses:0x16dbaa3f0]
```

- 现在=输出日志到文件：丢失了颜色，更主要是缩进也丢失了 -> 非常不利于查看函数调用关系



想要保留所有的frida-trace的带缩进的日志

如上所述，想要保留所有的frida-trace的日志，但如果用输出到日志文件，却又丢失缩进（和颜色）

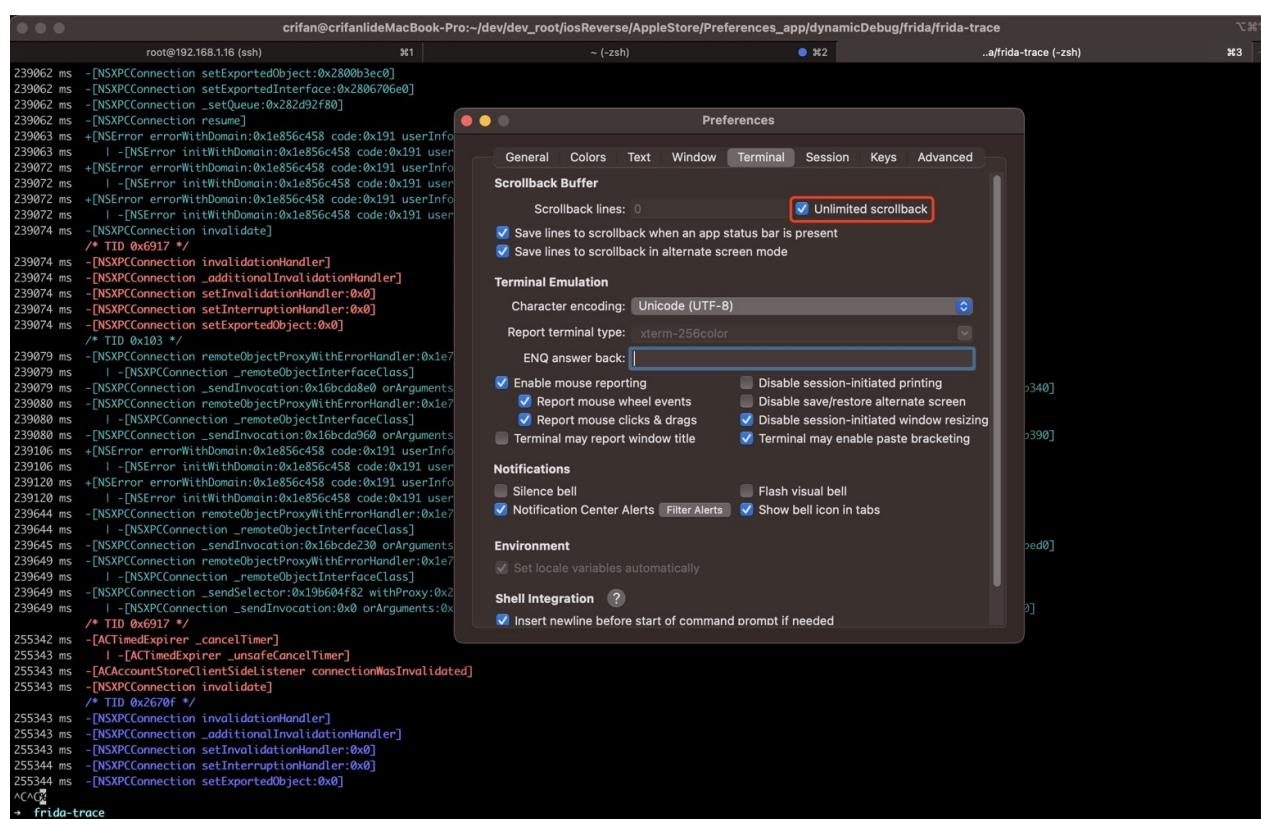
另外有个规避办法：

修改当前终端的最大显示行数 -> 就可以避免输出日志行数太多，之前内容被冲掉，看不到的问题了

比如：

- Mac 中 iTerm2 中，右键当前tab页顶部-》右键-》 Edit Session -> Terminal -> Scrollback Buffer -> Scrollback line : 改为足够大的数值. 比如 5000

甚至如果log日志特别长，那么可以考虑：设置为无限行数都保留 == Unlimited scrollback



这样就完全不用担心日志行数太多，前面的日志被冲掉，看不到的问题了。

# 通过修改特定函数的hook的js去打印参数值

对于之前的需求：frida-trace时，打印其中特定的某个ObjC函数的参数

之前不知道如何解决，后来参考[这里](#)，突然想到：

倒是可以借助其所说的，对于frida-trace自动为每个类的函数，所生成的js文件：

- 位置：`__handlers__/{ClassName}/{FunctionName}.js`

去修改js代码，加上打印对应的 `args` 的代码，即可打印对应参数值了。

## 举例

### -[AAAccountManager addAccount:]

frida-trace为函数 `-[AAAccountManager addAccount:]` 自动生成的：

- js文件

- `/Users/crifan/dev/dev_root/iosReverse/AppleStore/AuthKit_akd/dynamicDebug/frida/scripts/__handlers__/AAAccountManager/addAccount_.js`

```

fridaStalker_akdSymbol2575.js hook_NSXPConnection_setReportsObject.js hook_NSXPConnection_printAllObjects.js addAccount_.js
dynamicDebug > /dev/scripts > __handlers__ > AAAccountManager > addAccount_.js
1 /*
2 * Auto-generated by Frida. Please modify to match the signature of -[AAAccountManager addAccount:].
3 *
4 * This stub is currently auto-generated from manpages when available.
5 *
6 * For full API reference, see: https://frida.re/docs/javascript-api/
7 */
8
9 /**
10 * Called synchronously when about to call -[AAAccountManager addAccount:].
11 *
12 * @this {object} - Object allowing you to store state for use in onLeave.
13 * @param {function} log - Call this function with a string to be presented to the user.
14 * @param {array} args - Function arguments represented as an array of NativePointer objects.
15 *
16 * For example use args[0].readasString() if the first argument is a pointer to a C string encoded as UTF-8.
17 *
18 * It is also possible to modify arguments by assigning a NativePointer object to an element of this array.
19 * @param {object} state - Object allowing you to keep state across function calls.
20 *
21 * See onEnter for details.
22 */
23 onEnter(log, args, state) {
24 }
25
26 /**
27 * Called synchronously when about to return from -[AAAccountManager addAccount:].
28 *
29 * See onEnter for details.
30 */
31 accounts();
32
33 /**
34 * @this {object} - Object allowing you to access state stored in onLeave.
35 * @param {function} log - Call this function with a string to be presented to the user.
36 * @param {NativePointer} retval - Return value represented as a NativePointer object.
37 * @param {object} state - Object allowing you to keep state across function calls.
38
39 onLeave(log, retval, state) {
40 }
41
42 dealloc();
43
44 primaryAccount();
45 reloadAccount();
46 removeAccount();
47 saveAllAccounts();
48 sharedManager();
49 updateAccount();
50
51 AAAccountRecoveryManagementViewModel();
52 AAAddEmailURLRequest();
53 AAAppleIDSettingsRequest();
54 AAAppleTVRequest();
55 AAATestSigner();
56
57 大纲
58 时间线
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1
```

```

 * @param {function} log - Call this function with a string to be presented to the
 user.
 * @param {array} args - Function arguments represented as an array of NativePoint
 er objects.
 * For example use args[0].readUtf8String() if the first argument is a pointer to
 a C string encoded as UTF-8.
 * It is also possible to modify arguments by assigning a NativePointer object to
 an element of this array.
 * @param {object} state - Object allowing you to keep state across function calls

 * Only one JavaScript function will execute at a time, so do not worry about race
 -conditions.
 * However, do not use this to store function arguments across onEnter/onLeave, bu
 t instead
 * use "this" which is an object for keeping state local to an invocation.
 */
onEnter(log, args, state) {
 log(`-[AAAccountManager addAccount:${args[2]}]`);
}

/**
 * Called synchronously when about to return from -[AAAccountManager addAccount:
].
*
* See onEnter for details.
*
* @this {object} - Object allowing you to access state stored in onEnter.
* @param {function} log - Call this function with a string to be presented to the
 user.
* @param {NativePointer} retval - Return value represented as a NativePointer obj
 ect.
* @param {object} state - Object allowing you to keep state across function calls

*/
onLeave(log, retval, state) {
}
}

```

可以拷贝把其中的：

```
log(`-[AAAccountManager addAccount:${args[2]}]`);
```

改为：

```
log(`-[AAAccountManager addAccount:${new ObjC.Object(args[2])}]`);
```

应该就可以：打印出**ObjC对象的信息了，而不仅仅是：ptr = NativePointer 的指针的字符串而已。**

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：  
2023-06-27 23:22:30



# 常用iOS函数

Frida中逆向iOS的app期间，常会涉及到，一些相对通用的，常用的函数，整理如下：

- 网络相关

- 请求=request

- NSURLRequest

```
+[NSURLRequest requestWithURL:]
+[NSURLRequest requestWithURL:cachePolicy:timeoutInterval:]
-[NSURLRequest requestWithURL:cachePolicy:timeoutInterval:]
-[NSURLRequest initWithURL:]
-[NSURLRequest initWithURL:cachePolicy:timeoutInterval:]
```

- NSMutableURLRequest

```
-[NSMutableURLRequest initWithURL:]
-[NSMutableURLRequest setHTTPBody:]
```

- NSURL

```
+[NSURL URLWithString:]
+[NSURL URLWithString:relativeToURL:]
-[NSURL initWithScheme:host:path:]
-[NSURL initWithString:]
-[NSURL initWithString:relativeToURL:]
```

- NSURLConnection

```
+[NSURLConnection sendSynchronousRequest:returningResponse:error:]
+[NSURLConnection sendAsynchronousRequest:queue:completionHandler:]
+[NSURLConnection connectionWithRequest:delegate:]
-[NSURLConnection initWithRequest:delegate:]
-[NSURLConnection start]
-[NSURLConnection cancel]
-[NSURLConnection connection:didReceiveResponse:]
-[NSURLConnection connection:didReceiveData:]
-[NSURLConnection connectionDidFinishLoading:]
-[NSURLConnection connection:didFailWithError:]
```

- 响应=response

- NSHTTPURLResponse

```
-[NSHTTPURLResponse initWithURL:statusCode:HTTPVersion:headerFields:]
-[NSHTTPURLResponse allHeaderFields]
-[NSHTTPURLResponse statusCode]
```

- NSURLResponse

```
-[NSURLResponse initWithURL:MIMEType:expectedContentLength:textEncodingName:]
```



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：  
2023-10-24 17:48:36

# js

此处整理Frida中js的一些经验心得。

## 常见错误

### RangeError: invalid array index

- 错误： Frida的hook脚本js中

```
Interceptor.attach(functionRealAddress, {
 onEnter: function(args) {
 console.log(args);
```

- 会报错： RangeError: invalid array index

- 原因： 无法知道准确的 args 的数组的大小，无法直接打印，所以报错

- 解决办法：不去直接打印，而改为去获取对应的前几个参数（前提：已知参数个数），再去打印：

```
Interceptor.attach(functionRealAddress, {
 onEnter: function(args) {
 var arg0 = args[0]
 var arg1 = args[1]
 var arg2 = args[2]
 console.log("arg0=" + arg0 + ", arg1=" + arg1 + ", arg2=" + arg2);
```

- 进一步优化：甚至是，计算出此处的ObjC的参数的个数，循环批量打印所有参数

- 具体详见：[ObjC的参数](#)中的 计算ObjC的函数的真正参数的个数 + 打印全部参数

## Frida对于js支持的不够好

Frida对于js支持的不够好，不够完美：

- js报错时，代码错误行数指示有误
- console.log的参数格式化不支持
- putCallout中传递有名的独立函数会导致崩溃，改为匿名函数即可

下面详细解释：

### js报错时，代码错误行数指示有误

- 问题：

Frida的 js 代码文件中，参考别人的 ts 的代码，写了：

```
transform: (iterator: StalkerArm64Iterator) => {
 ...
 var arm64Context = ctx as Arm64CpuContext;
```

结果语法报错

```
Error: could not parse 'xxxxf/ridaStalker_akdSymbol2575.js' line 1: expecting ',',
at anonymous (/frida/repl-2.js:1)
```

The screenshot shows a macOS desktop environment with several windows open. On the left, there's a Finder sidebar showing a project structure for 'AUTHKIT\_AKD'. The main area has a terminal window and a code editor.

**Terminal Content:**

```
fridaStalker_akdSymbol2575.js - AuthKit_akd
fridaStalker_akdSymbol2575.js 3 ✘ commands.coffee └ const skdSymbol2575__functionAddress = 0x Untitled-1 ●
dynamicDebug > frida > scripts > JS fridaStalker_akdSymbol2575.js > ⌂ onLeave
 1 const akdSymbol2575_functionAddress = 0x0000001000a0460 // arm64 akd: __lldb_unnamed_symbol2575$akd
 2 const moduleBaseAddress = Module.moduleBaseAddress(moduleName);
 3 console.log(`moduleName=${moduleName}, moduleBaseName=${moduleBaseAddress}`);
 4 const functionRealAddress = moduleBaseAddress.add(akdSymbol2575_functionAddress);
 5 console.log(`functionRealAddress=${functionRealAddress}`);
 6 Interceptor.attach(functionRealAddress, {
 7 onEnter: function(args) {
 8 console.log(args);
 9 var curtid = Process.getCurrentThreadId();
 10 console.log(curtid);
 11 Stalker.follow(curtid, {
 12 events: {
 13 call: true, // CALL instructions: yes please
 14 ret: false, // RET instructions
 15 exec: false, // all instructions: not recommended as it's
 16 block: false, // block executed: coarse execution trace
 17 compile: false // block compiled: useful for coverage
 18 },
 19 transform: (iterator: StalkerArm4Iterator) => {
 20 var instruction = iterator.next();
 21 const startAddress = instruction.address;
 22 var isAppCode = startAddress.compare(baseAddr.add(addr)) >= 0 && startAddress.compare(base
 23 do {
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1078
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1088
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1097
 1098
 1099
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1158
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1178
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1188
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1197
 1198
 1199
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1208
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1218
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1278
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1297
 1298
 1299
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1308
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1328
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1338
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1358
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1378
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1398
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1408
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1418
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1428
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1478
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1488
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1498
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1508
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1518
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1528
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1538
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1558
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1568
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1578
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1588
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1598
 1598
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1608
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1618
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1628
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1638
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1658
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1668
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1678
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1688
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1698
 1698
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1708
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1718
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1728
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1738
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1748
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1758
 1759
 1760
 1761
 1762
 1763
 1764
 1765
 1766
 1767
 1768
 1768
 1769
 1770
 1771
 1772
 1773
 1774
 1775
 1776
 1777
 1778
 1778
 1779
 1780
 1781
 1782
 1783
 1784
 1785
 1786
 1787
 1788
 1788
 1789
 1790
 1791
 1792
 1793
 1794
 1795
 1796
 1797
 1798
 1798
 1799
 1800
 1801
 1802
 1803
 1804
 1805
 1806
 1807
 1808
 1808
 1809
 1810
 1811
 1812
 1813
 1814
 1815
 1816
 1817
 1818
 1818
 1819
 1820
 1821
 1822
 1823
 1824
 1825
 1826
 1827
 1828
 1828
 1829
 1830
 1831
 1832
 1833
 1834
 1835
 1836
 1837
 1838
 1838
 1839
 1840
 1841
 1842
 1843
 1844
 1845
 1846
 1847
 1848
 1848
 1849
 1850
 1851
 1852
 1853
 1854
 1855
 1856
 1857
 1858
 1858
 1859
 1860
 1861
 1862
 1863
 1864
 1865
 1866
 1867
 1868
 1868
 1869
 1870
 1871
 1872
 1873
 1874
 1875
 1876
 1877
 1878
 1878
 1879
 1880
 1881
 1882
 1883
 1884
 1885
 1886
 1887
 1888
 1888
 1889
 1890
 1891
 1892
 1893
 1894
 1895
 1896
 1897
 1898
 1898
 1899
 1900
 1901
 1902
 1903
 1904
 1905
 1906
 1907
 1908
 1908
 1909
 1910
 1911
 1912
 1913
 1914
 1915
 1916
 1917
 1918
 1918
 1919
 1920
 1921
 1922
 1923
 1924
 1925
 1926
 1927
 1928
 1928
 1929
 1930
 1931
 1932
 1933
 1934
 1935
 1936
 1937
 1938
 1938
 1939
 1940
 1941
 1942
 1943
 1944
 1945
 1946
 1947
 1948
 1948
 1949
 1950
 1951
 1952
 1953
 1954
 1955
 1956
 1957
 1958
 1958
 1959
 1960
 1961
 1962
 1963
 1964
 1965
 1966
 1967
 1968
 1968
 1969
 1970
 1971
 1972
 1973
 1974
 1975
 1976
 1977
 1978
 1978
 1979
 1980
 1981
 1982
 1983
 1984
 1985
 1986
 1987
 1988
 1988
 1989
 1990
 1991
 1992
 1993
 1994
 1995
 1996
 1997
 1998
 1999
 1999
 2000
 2
```

- 解决过程
    - Frida中对js代码报错，始终是第一行代码有问题
    - 后来是，花了很长时间和精力，才定位到真正问题：
  - 原因：js代码中，后面的某2行中，先后出现的2个变量，没有定义
    - 2个变量是：`StalkerArm64Iterator` 和 `Arm64CpuContext`
    - 导致的整体js代码无法运行而报错
  - 解决办法：注释掉2个未定义的变量（`StalkerArm64Iterator` 和 `Arm64CpuContext`）
  - 心得：
    - 总之说明，js代码的解析，是Frida整个框架去基于js引擎（v8或其他？）去解析的
    - 所以最终输出的报错信息，不够友好，导致容易让人误判错误原因，从而增加解决问题的难度

`putCallout`中传递有名的独立函数会导致崩溃

- 问题概述
    - 给putCallout传递函数：
      - 传递有名的独立的函数：会导致崩溃
      - 传递匿名的函数：就可以正常运行
  - 详解

Frida的Stalker的transform中去写putCallout代码

注：完整代码详见：[lldb unnamed symbol2575\\$\\$akd](#)

如果把putCallout写成是传入一个独立的js函数：

```
transform: function (iterator) {
 ...
 if (curOffsetInt == 8516) {
 iterator.putCallout(needDebug);
 }
}

...
function needDebug (context) {
 console.log("into needDebug: context=" + context);
}
}
```

->就会导致此处出现崩溃：

```
...
Process terminated

Thank you for using Frida
Fatal Python error: _enter_buffered_busy: could not acquire lock for <_io.BufferedReader name='<stdin>'> at interpreter shutdown, possibly due to daemon threads
Python runtime state: finalizing (tstate 0x000000010308e0c8)

Current thread 0x00000001f4ab4140 (most recent call first):
 no Python frame
```

而如果改为：js匿名名函数

```
transform: function (iterator) {
 ...
 if (curOffsetInt == 8516) {
 // iterator.putCallout(needDebug);
 iterator.putCallout((context) => {
 console.log("into needDebug: context=" + context);
 });
 }
}
```

就不会崩溃

很是诡异。暂时不清楚具体原因。

- 心得：Frida中对于js的支持，还是不够完善
  - 容易遇到一些诡异的bug

## console.log的参数格式化不支持

详见后续章节：[console.log日志](#)

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：

2023-06-26 22:19:15

## console.log

### 问题

#### console.log不支持参数的格式化

- 问题
  - Frida中console.log打印参数，去格式化参数：

```
console.log("moduleName=%s, moduleBaseAddress=%p", moduleName, moduleBaseAddress
)
```

输出：

```
moduleName %s, moduleBaseAddress %p akd 0x10015c000
```

- 结论：Frida中js的console.log，不支持参数格式化 == %s、%d、%o、%p 等格式化参数无效
- 规避办法

- 用逗号 = , 或加号 = + 去打印参数

```
console.log("moduleName=", moduleName, "moduleBaseAddress=", moduleBaseAddress
)

console.log("moduleName=" + moduleName + "moduleBaseAddress=" + moduleBaseAddress)
```

### 心得

## 用 JSON.stringify 打印对象

既然console.log中无法直接（用 %o 去格式化）打印对象，那么可以考虑用 JSON.stringify 去转化为JSON再去打印：

举例：

```
console.log("instruction: address=" + instruction.address
+ ",next=" + instruction.next
+ ",size=" + instruction.size
+ ",mnemonic=" + instruction.mnemonic
+ ",opStr=" + instruction.opStr
+ ",operands=" + JSON.stringify(instruction.operands)
+ ",regsRead=" + JSON.stringify(instruction.regsRead)
+ ",regsWritten=" + JSON.stringify(instruction.regsWritten)
+ ",groups=" + JSON.stringify(instruction.groups)
+ ",toString()=" + instruction.toString())
```

);

输出效果：

```

instruction: address 0x1091dbcd8, next 0x4, size 4, mnemonic b, opStr #0x1091dbce8, operands=[{"type": "imm", "value": "4447911144", "access": "r"}], regsRead=[], regsWritten=[], groups=[["jump", "branch_relative"]], toString()=b #0x1091dbce8
[0x1091dbcd8] b #0x1091dbce8
+++ into iterator=
instruction: address 0x1091dbce8, next 0x4, size 4, mnemonic str, opStr wZR, [x19, #0x90], operands=[{"type": "reg", "value": "wZR", "access": "r"}, {"type": "mem", "value": {"base": "x19", "disp": 144}, "access": "rw"}], regsRead=[], regsWritten=[], groups=[], toString()=str wZR, [x19, #0x90]
[0x1091dbce8] str wZR, [x19, #0x90]

```

## 让console.log输出日志到文件

- 需求：希望Frida中console.log打印的日志，不是输出到当前终端，而是输出到文件中
  - 目的：方便后续随时查看，方便调试
- 解决办法：

在运行Frida带js的命令行最后加上：

- > xxx.log 2>&1
  - 参数说明
    - > xxx.log：把普通的= stdout 日志信息，都输出到 log文件
    - 2>&1：把特殊的 stderr =错误日志信息，输出到终端
- 举例

```
frida -U -p 13098 -l ./fridaStalker_akdSymbol12575.js > stalker_akd1575.log 2 &1
```

## Frida的Stalker中优化hook到指令时的log日志输出

- 需求：Frida的Stalker中，希望输出的log日志，尽量也模拟之前Xcode的汇编代码的形式
- 核心代码：

```

var curRealAddr = instruction.address;
var curOffsetHexPtr = curRealAddr.sub(funcRealStartAddr)
var curOffsetInt = curOffsetHexPtr.toInt32()
var instructionStr = instruction.toString()
console.log("\t" + curRealAddr + " <+" + curOffsetInt + ">: " + instructionStr);

```

- 输出效果

```

+++ into iterator: startAddress=0x104b48470
0x104b48470 +16 : stp x22, x21, [sp, #0xc0]
0x104b48474 +20 : stp x20, x19, [sp, #0xd0]
0x104b48478 +24 : stp x29, x30, [sp, #0xe0]
0x104b4847c +28 : add x29, sp, #0xe0
0x104b48480 +32 : nop

```

```
0x104b48484 +36 : ldr x8, #0x104b9c7d8
```

- 注：完整代码和输出，详见：[\\_\\_lldb\\_unnamed\\_symbol2575\\$\\$akd](#)

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：  
2023-06-26 22:46:48

## 自己编译frida-server

iPhone11 中，已用 XinaA15 进行了 rootless越狱，然后去用 frida：

```
crifan@licrifandeMacBook-Pro ~ $ frida -U -f com.apple.store.Jolly -l /Users/crifan/dev
/dev_root/iosReverse/AppleStore/dynamicDebug/frida/hookNSFileManager.js

 / _ \ | Frida 16.0.8 - A world-class dynamic instrumentation toolkit
 | () |
 | >_ |
 /_/_ | Commands:
 + + + + help -> Displays the help system
 + + + + object? -> Display information about 'object'
 + + + + exit/quit -> Exit
 + + + + More info at https://frida.re/docs/home/
 + + + + Connected to iPhone (id 00008030-00011C49366B802E)
Failed to attach: missing gProcessInfo
```

结果报错： Failed to attach missing gProcessInfo

之后就是尝试解决此问题，最终涉及到：自己去编译frida-server的过程。

此处记录相关内容和心得，供参考。

## 研究gProcessInfo的来源

### 之前frida-ios-dump也遇到类似问题

而之前就见过此处的 missing gProcessInfo，找到之前的：

- 原因是：属于偶尔的bug
- 解决办法：多试几次

-》此处：继续尝试多次，始终无法规避，始终报错。

## 研究frida中是否存在导入外部变量gProcessInfo

从报错信息 Failed to attach: missing gProcessInfo 中推测：

gProcessInfo是（iOS的app启动阶段涉及到的）dyld中的变量

怀疑是类似于：

- frida import 变量：gProcessInfo
- dyld export 变量：gProcessInfo

这种机制

所以去研究看看：

frida中，是否有import的变量，叫做gProcessInfo

后来确认，missing gProcessInfo 来自iPhone端的 frida-server

去导出 frida-server

```
scp root@192.168.2.12:/var/sbin/frida-server frida-server
```

然后继续静态分析：

```
rabin2 -i frida-server > fridaServer_rabin2_i_imports.txt
rabin2 -E frida-server > fridaServer_rabin2_E_exports.txt
```

没找到 gProcessInfo

另外找到：

- dyld/dyldMain.cpp

```
namespace dyld4 {
...

#if TARGET_OS_OSX
static void* getProcessInfo()
{
 return gProcessInfo;
}
```

而根据：

【记录】dyld相关资料：启动过程

知道了：

- dyld
  - 之前是：dyld2
  - 后来是：dyld3
  - 此处是：dyld4

即：dyld 有3个版本， dyld2 、 dyld3 、 dyld4

最后确认：

不是Frida（的frida-server）引用了外部的变量：

dyld 源码中的 gProcessInfo

而是：

frida源码中有gProcessInfo

即：

frida-core/src/fruity/injector.vala 中就有对应代码：

```
ensure_libsystem_initialized_for_dyld_v4_and_above
 ...
 throw new Error.UNSUPPORTED ("Missing gProcessInfo");
```

## Frida源码中找到了：gProcessInfo

missing gProcessInfo 相关完整的代码：

```
frida-core > src / fruity > injector.vala
819 private async void ensure_libsystem_initialized (Cancellable? cancellable) throws GLib.Error {
820 ...
821 if (!libsystem_initialized)
822 ...
823 else
824 yield restore_main_thread_state (cancellable);
825
826 uint64? libdyld_initialize = dyld_symbols["__ZN5dyld44APIS19__libdyld_initializeEPKNS_16LibSystemHelpersE"];
827 if (libdyld_initialize != null)
828 yield ensure_libsystem_initialized_for_dyld_v4_and_above (libdyld_initialize, cancellable);
829 else
830 yield ensure_libsystem_initialized_for_dyld_v3_and_below (cancellable);
831
832 yield save_main_thread_state (cancellable);
833
834 yield libdb.write_bool (dyld.fields.libsystem_initialized, true, cancellable);
835 libsystem_initialized = true;
836 }
837
838 private async void ensure_libsystem_initialized_for_dyld_v4_and_above (uint64 libdyld_initialize,
839 Cancellable? cancellable) throws GLib.Error {
840 uint64? process_info_ptr = dyld_symbols["_gProcessInfo"];
841 if (process_info_ptr == null)
842 throw new Error.UNSUPPORTED ("Missing gProcessInfo");
843 }
```

- frida-core/src/fruity/injector.vala

```
private async void ensure_libsystem_initialized_for_dyld_v4_and_above (uint64 libdyld_initialize,
 Cancellable? cancellable) throws GLib.Error {
 uint64? process_info_ptr = dyld_symbols["_gProcessInfo"];
 if (process_info_ptr == null)
 throw new Error.UNSUPPORTED ("Missing gProcessInfo");
 ...
}
```

其他地方也有：

- frida-core/src/fruity/helpers/symbol-fetcher.c

```
size_t
frida_fetch_dyld_symbols (char * output_buffer, const void * dyld_load_address)
{
 ...
 for (i = dyld.dysymtab->nlocalsym; i <= dyld.dysymtab->nlocalsym; i++)
 {
 const struct nlist_64 * sym = &symbols[i];
 const char * name = strings + sym->n_un.n_strx;

 if (frida_str_contains (name, "libdyld_initialize") ||
 frida_str_contains (name, "restartWithDyldInCache") ||
 frida_str_equals (name, "_gProcessInfo") ||
 frida_str_contains (name, "launchWithClosure") ||
 frida_str_contains (name, "initializeMainExecutable") ||
 frida_str_contains (name, "registerThreadHelpers") ||
 frida_str_has_prefix (name, "_dlopen") ||
 frida_str_has_prefix (name, "_strcmp") ||
 frida_str_contains (name, "doModInitFunctions") ||
```

```

 frida_str_contains (name, "doGetDOFSections"))
 {
 if (n != 0)
 frida_append_char (&cursor, '\n');

 frida_append_uint64 (&cursor, (uint64_t) (dyld.base + sym->n_value));
 frida_append_char (&cursor, '\t');
 frida_append_string (&cursor, name);

 n++;
 }
}

```

- frida-core/src/darwin/frida-helper-backend-glue.m

```

modern_entry_address = gum_darwin_module_resolve_symbol_address (dyld, "__ZN5dyld44APIs
19_dyld_initializeEPKNS_16LibSystemHelpersE");
instance->dyld_flavor = (modern_entry_address != 0) ? FRIDA_DYLD_V4_PLUS : FRIDA_DYLD_V
3_MINUS;
if (instance->dyld_flavor == FRIDA_DYLD_V4_PLUS)
{
 instance->modern_entry_address = modern_entry_address;
 legacy_entry_address = 0;

 instance->info_ptr_address = gum_darwin_module_resolve_symbol_address (dyld, "_gProce
ssInfo");
 if (instance->info_ptr_address == 0)
 goto dyld_probe_failed;
}

...
instance->dlopen_address = gum_darwin_module_resolve_symbol_address (dyld, "_dlopen");

if (instance->dlopen_address == 0)
 instance->dlopen_address = gum_darwin_module_resolve_symbol_address (dyld, "_dlopen
_internal");
instance->register_helpers_address = gum_darwin_module_resolve_symbol_address (dyld,
"__ZL21registerThreadHelpersPKN4dyld16LibSystemHelpersE");
instance->dlerror_clear_address = gum_darwin_module_resolve_symbol_address (dyld, "__
ZL12dlerrorClearv");
instance->info_address = gum_darwin_module_resolve_symbol_address (dyld, "__ZN4dyld12
gProcessInfoE");
instance->helpers_ptr_address = gum_darwin_module_resolve_symbol_address (dyld, "__ZN
4dyld17gLibSystemHelpersE");
instance->do_modinit_strcmp_checks = frida_find_modinit_strcmp_checks (task, dyld);
...

```

心得：

其中有很多这种：

编译器编译后的固定的函数名：

- \_dlopen

- `_dlopen_internal`
- `__ZL21registerThreadHelpersPKN4dyld16LibSystemHelpersE`
- `__ZL12dlerrorClearv`
- `__ZN4dyld12gProcessInfoE`
  - 其中包含: `gProcessInfo`
- `__ZN4dyld17gLibSystemHelpersE`

## 研究Frida中Missing gProcessInfo出错的逻辑和原因

经过后续了解: 【记录】 dyld源码中的gProcessInfo

应该把:

- 只判断是否存在: `_gProcessInfo`

改为:

- 同时判断多种情况 (先后顺序是)
  - `_gProcessInfo`
    - 对应原始代码中: `gProcessInfo`
  - `__ZN5dyld412gProcessInfoE`
    - 对应原始代码中: `dyld4::gProcessInfo`
  - `__ZN4dyld12gProcessInfoE`
    - 对应原始代码中: `dyld::gProcessInfo` 应该就可以了。

## 研究二进制/usr/lib/dyld中是否包含或导出变量\_gProcessInfo

对于iPhone中的dyld:

```
iPhone11-151:~ root# ls -lh /usr/lib/dyld
-rwxr-xr-x 1 root wheel 630K Oct 15 2021 /usr/lib/dyld*
```

用:

```
scp root@192.168.2.12:/usr/lib/dyld dyld
```

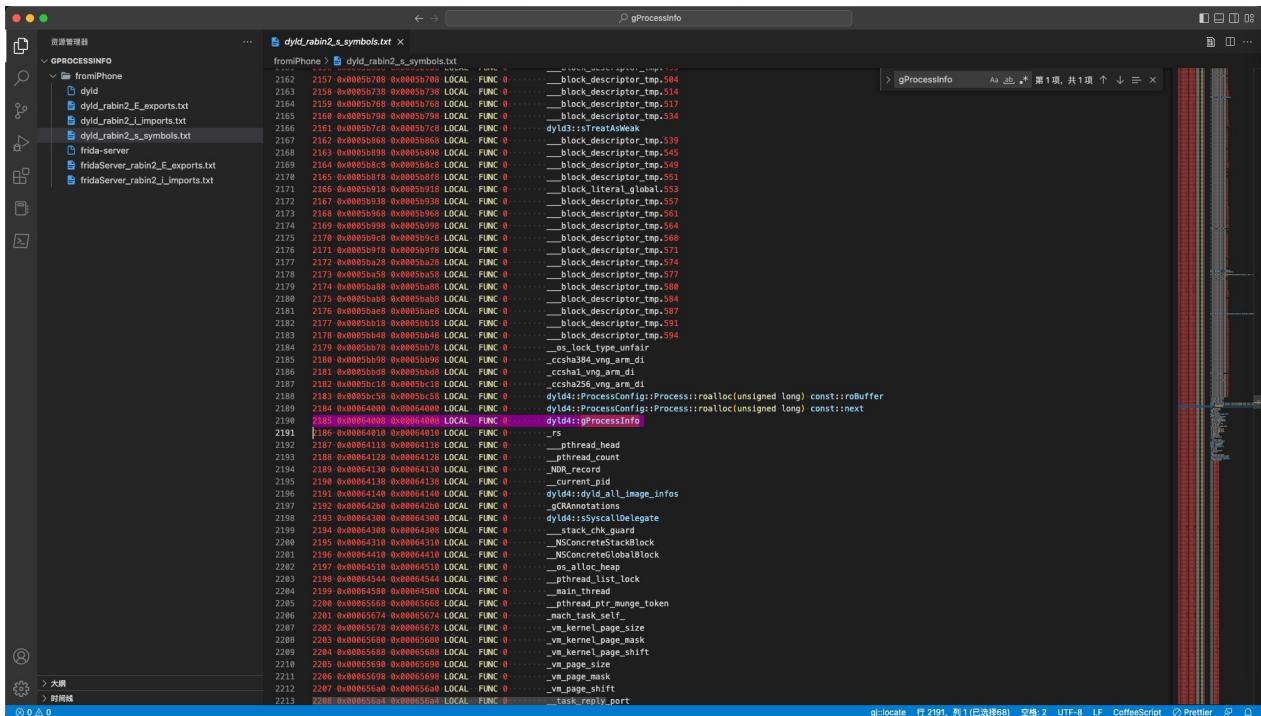
导出后, 再从dyld中导出符号:

```
rabin2 -s dyld > dyld_rabin2_s_symbols.txt
```

发现是有的:

- `dyld_rabin2_s_symbols.txt`

```
2185 0x00064008 0x00064008 LOCAL FUNC 0 dyld4::gProcessInfo
```



另外，突然注意到：

- fromiPhone/dyld\_rabin2\_s\_symbols.txt

```

891 0x0002e500 0x0002e500 LOCAL FUNC 0 dyld4::APIs::_dyld_shared_cache_optimized()
892 0x0002e57c 0x0002e57c LOCAL FUNC 0 dyld4::APIs::_dyld_register_for_image_loads(void (*)(mach_header const*, char const*, bool))
893 0x0002e680 0x0002e680 LOCAL FUNC 0 ____ZN5dyld44APIs30_dyld_register_for_image_loadsEPFvPK11mach_headerPKcbE_block_invoke
894 0x0002e76c 0x0002e76c LOCAL FUNC 0 ____ZN5dyld44APIs30_dyld_register_for_image_loadsEPFvPK11mach_headerPKcbE_block_invoke_2
895 0x0002e7b4 0x0002e7b4 LOCAL FUNC 0 ____ZN5dyld44APIs35_dyld_register_for_bulk_image_loadsEPFvjPPK11mach_headerPPKcE_block_invoke_2
896 0x0002e7c8 0x0002e7c8 LOCAL FUNC 0 dyld4::APIs::_dyld_shared_cache_file_path()
897 0x0002e810 0x0002e810 LOCAL FUNC 0 dyld4::APIs::_dyld_has_inserted_or_interposing_libraries()
898 0x0002e874 0x0002e874 LOCAL FUNC 0 dyld4::APIs::_dyld_shared_cache_find_iterate_text(unsigned char const*, char const**, void (block_pointer)(dyld_shared_cache_dylib_text_info const*))
899 0x0002ea8c 0x0002ea8c LOCAL FUNC 0 dyld4::findCacheInDirAndMap(dyld4::RuntimeState , unsigned char const*, char const*, unsigned long&)
900 0x0002eb5c 0x0002eb5c LOCAL FUNC 0 ____ZN5dyld44APIs35dyld_shared_cache_find_iterate_textEPKhPPKcU13block_pointerFvPK33dyld_shared_cache_dylib_text_infoE_block_invoke.173
901 0x0002ebc0 0x0002ebc0 LOCAL FUNC 0 dyld4::APIs::_dyld_shared_cache_iterate_text(unsigned char const*, void (block_pointer)(dyld_shared_cache_dylib_text_info const*))
902 0x0002ec60 0x0002ec60 LOCAL FUNC 0 dyld4::APIs::_dyld_fork_child()

```

即：

此处symbol中，也是有一些：

- `__ZN5dyld44APIs30_dyld_register_for_image_loadsEPFvPK11mach_headerPKcbE_block_invoker`

这种编译后的函数名的值的

同时，也有，编译前的，普通的函数名：

- `dyld4::APIs::_dyld_register_for_image_loads(void ()(mach_header const, char const*, bool))`

-》所以突然想到：

估计是，编译前的，普通函数名，是：

此处rabin2，自动帮忙翻译的（因为其懂得编译和反编译函数名 symbol的内在逻辑？）

-》所以去找找：

是否有机会，让rabin2，只输出：

编译后的symbol名字？

这样就能找到，确认：

```
dyld4::gProcessInfo
```

是不是：

```
__ZN5dyld44gProcessInfo
```

了

去找找看：

rabin2的其他参数，能输出原始的symbol的？

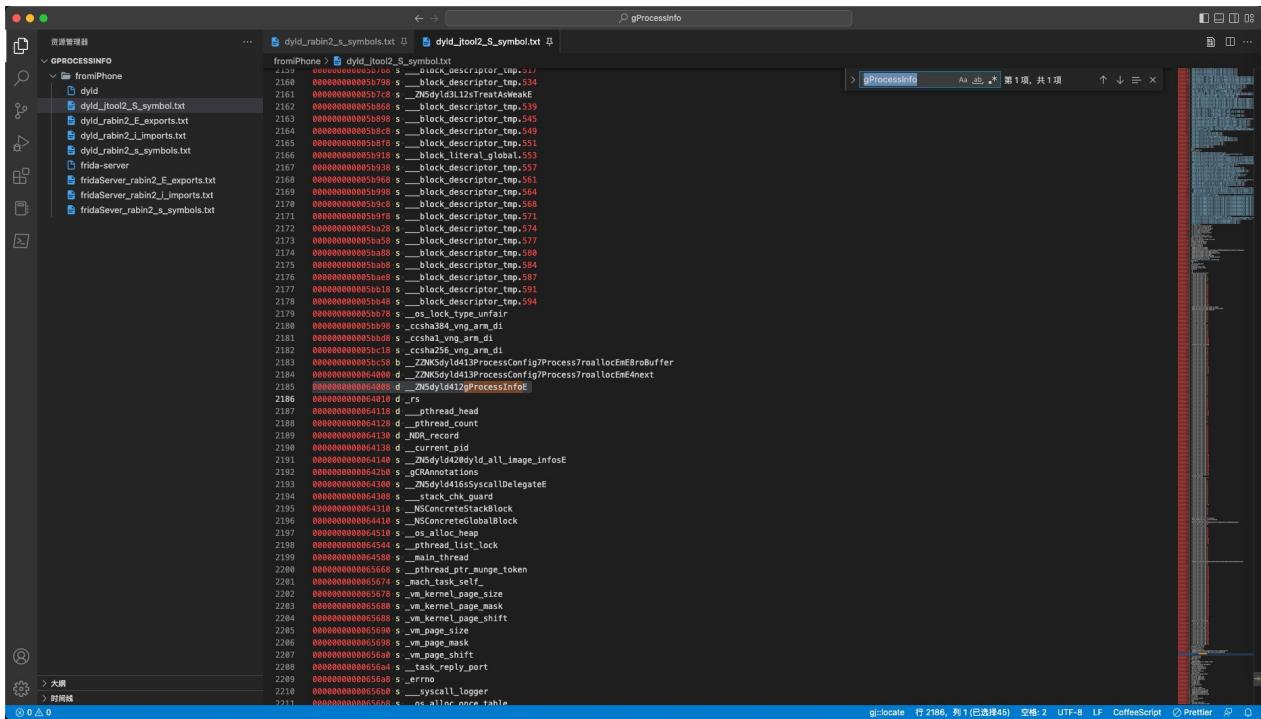
对了，或许也可以用另外的工具： `jtool2`

```
jtool2 -S dyld > dyld_jtool2_S_symbol.txt
```

果然是我们希望的，原始的，编译后的，没有被解析的：`gProcessInfo`

- `dyld_jtool2_S_symbol.txt`

```
00000000000064008 d __ZN5dyld412gProcessInfoE
```



所以就是：

- `dyld4::gProcessInfo`
  - 编译生成: `__ZN5dyld412gProcessInfoE`
  - 不是我以为的: `__ZN5dyld44gProcessInfo`

再去dyld源码中，多搜搜：

```
namespace dyld4
```

看看是否有其他新发现

```
namespace dyld4
```

- `/Users/crifan/dev/dev_src/ios_reverse/AppleOpenSource/dyld/dyld-dyld-1042.1/dyld/DebuggerSupport.h`

```
namespace dyld4 {
 using lsl::Allocator;
 void addImagesToAllImages(RuntimeState* state, uint32_t infoCount, const dyld_image_info info[], uint32_t initialImageCount);
 void removeImageFromAllImages(const mach_header loadAddress);
}

extern "C" void lldb_image_notifier(enum dyld_image_mode mode, uint32_t infoCount, const dyld_image_info info[]);

extern dyld_all_image_infos* gProcessInfo;
```

->也还是: `gProcessInfo`

->不是放在 `namespace dyld4` 中的

- /Users/crifan/dev/dev\_src/ios\_reverse/AppleOpenSource/dyld/dyld-dyld-1042.1/dyld/DyldAPIs.cpp

```
// internal libc.a variable that needs to be reset during fork()
extern mach_port_t mach_task_self_;

using dyld3::MachOFile;
using dyld3::MachOLoaded;

extern const dyld3::MachOLoaded __dso_handle;

...

namespace dyld4 {
...
}
```

-》自己当前是dyld4的namespace，但是也会引用外部变量：

- 其中也有：

- 没有namespace的：
  - extern mach\_port\_t mach\_task\_self\_;
- 和另外的namespace的： dyld3
  - extern const dyld3::MachOLoaded \_\_dso\_handle;

然后去：

【未解决】rabin2输出C++的未解析的原始的编译后的函数名mangle name

所以去：

【未解决】C++代码中函数变量编译生成符号symbol的规则

期间去：

【已解决】从C++的编译后的符号symbol得到原始的变量函数名

另外，看看此处的dyld版本：

```
iPhone11-151:~ root# /usr/lib/dyld --version
-sh: /usr/lib/dyld: cannot execute binary file: Exec format error
```

无法查看。

【已解决】iOS 13.3的iPhone7中/usr/lib/dyld的版本和gProcessInfo相关信息

## 也去研究：dyld源码中的gProcessInfo

至此找到：

- gProcessInfo
  - 变量来源：
    - dyld/dyld-dyld-1042.1/dyld/DebuggerSupport.cpp

```
struct dyld_all_image_infos* gProcessInfo = &dyld_all_image_infos;
```

以及：

- struct dyld\_all\_image\_infos
  - 定义
    - libdyld/dyld\_process\_info\_internal.h
    - struct dyld\_all\_image\_infos\_32
    - struct dyld\_all\_image\_infos\_64
    - include/mach-o/dyld\_images.h
    - struct \_\_attribute\_\_((aligned(16))) dyld\_all\_image\_infos

具体定义详见：

(1) include/mach-o/dyld\_images.h

```
// Must be aligned to support atomic updates
// Note sim cannot assume alignment until all host dylds are new enough
#if TARGET_OS_SIMULATOR
struct dyld_all_image_infos
#else
struct __attribute__((aligned(16))) dyld_all_image_infos
#endif
{
 uint32_t version, /* 1 in Mac OS X 10.4 and 10.5 */
 uint32_t infoArrayCount;
#if defined(__cplusplus) && (BUILDING_LIBDYLD || BUILDING_DYLD)
 std::atomic<const struct dyld_image_info*> infoArray;
#else
 const struct dyld_image_info* infoArray;
#endif
 dyld_image_notifier notification;
 bool processDetachedFromSharedRegion;
 /* the following fields are only in version 2 (Mac OS X 10.6, iPhoneOS 2.0) and later */
 bool libSystemInitialized;
 const struct mach_header* dyldImageLoadAddress;
 /* the following field is only in version 3 (Mac OS X 10.6, iPhoneOS 3.0) and later */
 void* jitInfo;
 /* the following fields are only in version 5 (Mac OS X 10.6, iPhoneOS 3.0) and later */
 const char* dyldVersion;
 const char* errorMessage;
 uintptr_t terminationFlags;
 /* the following field is only in version 6 (Mac OS X 10.6, iPhoneOS 3.1) and later */
 void* coreSymbolicationShmPage;
 /* the following field is only in version 7 (Mac OS X 10.6, iPhoneOS 3.1) and later */
 uintptr_t systemOrderFlag;
 /* the following field is only in version 8 (Mac OS X 10.7, iPhoneOS 3.1) and later */
 uintptr_t uuidArrayCount;
```

```

const struct dyld_uuid_info* uuidArray; /* only images not in dyld shared
cache */

/* the following field is only in version 9 (Mac OS X 10.7, iOS 4.0) and later */
struct dyld_all_image_infos* dyldAllImageInfosAddress;

/* the following field is only in version 10 (Mac OS X 10.7, iOS 4.2) and later */
uintptr_t initialImageCount;

/* the following field is only in version 11 (Mac OS X 10.7, iOS 4.2) and later */
uintptr_t errorKind;
const char* errorClientOfDylibPath;
const char* errorTargetDylibPath;
const char* errorSymbol;

/* the following field is only in version 12 (Mac OS X 10.7, iOS 4.3) and later */
uintptr_t sharedCacheSlide;

/* the following field is only in version 13 (Mac OS X 10.9, iOS 7.0) and later */
uint8_t sharedCacheUUID[16];

/* the following field is only in version 15 (macOS 10.12, iOS 10.0) and later */
uintptr_t sharedCacheBaseAddress;

#if defined(__cplusplus) && (BUILDING_LIBDYLD || BUILDING_DYLD)
 // We want this to be atomic in libdyld so that we can see updates when we map it shared
 std::atomic<uint64_t> infoArrayChangeTimestamp;
#else
 uint64_t infoArrayChangeTimestamp;
#endif
 const char* dyldPath;
 mach_port_t notifyPorts[DYLD_MAX_PROCESS_INFO_NOTIFY_COUNT];
#endif __LP64__
 uintptr_t reserved[11 (DYLD_MAX_PROCESS_INFO_NOTIFY_COUNT/2)];
};

#ifndef __LP64__
 uintptr_t reserved[9 DYLD_MAX_PROCESS_INFO_NOTIFY_COUNT];
#endif

// The following fields were added in version 18 (previously they were reserved padding fields)
 uint64_t sharedCacheFSID;
 uint64_t sharedCacheFSObjID;

/* the following field is only in version 16 (macOS 10.13, iOS 11.0) and later */
 uintptr_t compact_dyld_image_info_addr;
 size_t compact_dyld_image_info_size;
 uint32_t platform; // FIXME: really a dyld_platform_t, but those aren't exposed here.

/* the following field is only in version 17 (macOS 10.16) and later */
 uint32_t aotInfoCount;
 const struct dyld_aot_image_info aotInfoArray;
 uint64_t aotInfoArrayChangeTimestamp;
 uintptr_t aotSharedCacheBaseAddress;
 uint8_t aotSharedCacheUUID[16];
};


```

(2) 还有个分32和64的：

- struct dyld\_all\_image\_infos\_32

- struct dyld\_all\_image\_infos\_64
- >
- libdyld/dyld\_process\_info\_internal.h

```

struct dyld_all_image_infos_32 {
 uint32_t version;
 uint32_t infoArrayCount;
 std::atomic<uint32_t> infoArray;
 notification;
 bool processDetachedFromSharedRegion;
 bool libSystemInitialized;
 dyldImageLoadAddress;
 jitInfo;
 dyldVersion;
 errorMessage;
 terminationFlags;
 coreSymbolicationShmPage;
 systemOrderFlag;
 uint32_t uuidArrayCount;
 uint32_t uuidArray;
 dyldAllImageInfosAddress;
 initialImageCount;
 errorKind;
 errorClientOfDylibPath;
 errorTargetDylibPath;
 errorSymbol;
 sharedCacheSlide;
 sharedCacheUUID;
 sharedCacheBaseAddress;
 infoArrayChangeTimestamp;
 dyldPath;
 notifyMachPorts[8];
 reserved;
 sharedCacheFSID;
 sharedCacheFSObjID;
 compact_dyld_image_info_addr;
 compact_dyld_image_info_size;
 uint32_t platform;
 // the aot fields below will not be set in the 32 bit case
 uint32_t aotInfoCount;
 std::atomic<uint64_t> aotInfoArray;
 uint64_t aotInfoArrayChangeTimestamp;
 uint64_t aotSharedCacheBaseAddress;
 std::array<uint8_t, 16> aotSharedCacheUUID[16];
};

struct dyld_all_image_infos_64 {
 uint32_t version;
 uint32_t infoArrayCount;
 std::atomic<uint64_t> infoArray;
 uint64_t notification;
 bool processDetachedFromSharedRegion;
 bool libSystemInitialized;

```

```

 uint32_t paddingToMakeTheSizeCorrectOn32bitAndDoesntAffect64b; // NO
T PART OF DYLD_ALL_IMAGE_INFOS!
 uint64_t dyldImageLoadAddress;
 uint64_t jitInfo;
 uint64_t dyldVersion;
 errorMessage;
 terminationFlags;
 coreSymbolicationShmPage;
 systemOrderFlag;
 uint64_t uuidArrayCount;
 uint64_t uuidArray;
 uint64_t dyldAllImageInfosAddress;
 uint64_t initialImageCount;
 errorKind;
 errorClientOfDylibPath;
 errorTargetDylibPath;
 errorSymbol;
 sharedCacheSlide;
 std::array<uint8_t, 16> sharedCacheUUID;
 uint64_t sharedCacheBaseAddress;
 std::atomic<uint64_t> infoArrayChangeTimestamp;
 uint64_t dyldPath;
 uint32_t notifyMachPorts[8];
 uint64_t reserved[7];
 uint64_t sharedCacheFSID;
 uint64_t sharedCacheFSObjID;
 compact_dyld_image_info_addr;
 compact_dyld_image_info_size;
 uint32_t platform;
 uint32_t aotInfoCount;
 std::atomic<uint64_t> aotInfoArray;
 uint64_t aotInfoArrayChangeTimestamp;
 uint64_t aotSharedCacheBaseAddress;
 std::array<uint8_t, 16> aotSharedCacheUUID[16];
};


```

然后：

【未解决】 dyld-932.4中gProcessInfo编译后symbol却是\_\_ZN5dyld412gProcessInfoE

## 自己编译arm64e版的Frida

安装依赖库：

```
pip install colorama prompt-toolkit pygments
```

设置Python用新版 3.10.6

```
local再去设置为3.10.6的版本：
crifan@licrifandeMacBook-Pro ~$/dev/dev_src/ios_reverse/frida$ pyenv versions
system
3.5.2
3.6.6
```

```

3.7.3
* 3.9.4 (set by /Users/crifan/.pyenv/version)
 3.10.6
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida$ pyenv local 3.10.6
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida$ python --version
Python 3.10.6

```

clone frida的代码：

```
git clone --recurse-submodules https://github.com/frida/frida.git
```

先make看看有哪些编译选项：

```

crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida$ \ main$ make
make[1]: Entering directory '/Users/crifan/dev/dev_src/ios_reverse/frida'
Usage: make TARGET [VARIABLE value]

Where TARGET specifies one or more of:

/* gum */
gum-macos Build for macOS
gum-ios Build for iOS
gum-watchos Build for watchOS
gum-tvos Build for tvOS
gum-android-x86 Build for Android/x86
gum-android-x86_64 Build for Android/x86-64
gum-android-arm Build for Android/arm
gum-android-arm64 Build for Android/arm64
check-gum-macos Run tests for macOS

/* core */
core-macos Build for macOS
core-ios Build for iOS
core-watchos Build for watchOS
core-tvos Build for tvOS
core-android-x86 Build for Android/x86
core-android-x86_64 Build for Android/x86-64
core-android-arm Build for Android/arm
core-android-arm64 Build for Android/arm64
check-core-macos Run tests for macOS

/* python */
python-macos Build Python bindings for macOS
check-python-macos Test Python bindings for macOS

/* node */
node-macos Build Node.js bindings for macOS
check-node-macos Test Node.js bindings for macOS

/* tools */
tools-macos Build CLI tools for macOS
check-tools-macos Test CLI tools for macOS

```

```

And optionally also VARIABLE values:
 PYTHON Absolute path of Python interpreter including version suffix
 ix
 NODE Absolute path of Node.js binary

For example:
$ make python-macos PYTHON /usr/local/bin/python3.6
$ make node-macos NODE /usr/local/bin/node

make[1]: Leaving directory '/Users/crifan/dev/dev_src/ios_reverse/frida/frida'

```

此处要去编译：ios 的，所以看起来是：

```
core-ios Build for iOS
```

所以最后去：

```
make core-ios
```

期间解决了证书问题：

- 【已解决】Mac中编译frida-core报错：FAILED /usr/bin/codesign IOS\_CERTID not set

继续：

```

x crifan@licrifandeMacBook-Pro ~/dev/dev_src/ios_reverse/frida/frida` \ main` export
MACOS_CERTID frida-cert
crifan@licrifandeMacBook-Pro ~/dev/dev_src/ios_reverse/frida/frida` \ main` export I
OS_CERTID frida-cert
crifan@licrifandeMacBook-Pro ~/dev/dev_src/ios_reverse/frida/frida` \ main` export W
ATCHOS_CERTID frida-cert
crifan@licrifandeMacBook-Pro ~/dev/dev_src/ios_reverse/frida/frida` \ main` export T
VOS_CERTID frida-cert
crifan@licrifandeMacBook-Pro ~/dev/dev_src/ios_reverse/frida/frida` \ main` make cor
e-ios
make[1]: Entering directory '/Users/crifan/dev/dev_src/ios_reverse/frida/frida'
build/frida-env-ios-arm64.rc; \
builddir build/tmp-ios-arm64/frida-core; \
if [! -f $builddir/build.ninja]; then \
 meson_args="--native-file build/frida-macos-x86_64.txt"; if [ios-arm64 != \
macos-x86_64]; then meson_args="$meson_args --cross-file build/frida-ios-arm64.txt"; fi \
; python3 /Users/crifan/dev/dev_src/ios_reverse/frida/frida/releng/meson/meson.py setup \
$meson_args \
 --prefix /usr \
 --default-library static -Doptimization=s -Db_ndebug=true --strip -Dconnectivity \
enabled -Dmapper auto \
 -Dassets_installed \
 frida-core $builddir || exit 1; \
fi \
&& python3 /Users/crifan/dev/dev_src/ios_reverse/frida/frida/releng/meson/meson.py \
compile -C $builddir \
 && DESTDIR="/Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64" \
python3 /Users/crifan/dev/dev_src/ios_reverse/frida/frida/releng/meson/meson.py install \
-C $builddir
INFO: autodetecting backend as ninja

```

```

INFO: calculating backend command to run: /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/toolchain-macos-x86_64/bin/ninja -C /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/tmp-ios-arm64/frida-core
ninja: Entering directory '/Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/tmp-ios-arm64/frida-core'
[85/85] Generating lib/gadget/frida-gadget with a custom command
ninja: Entering directory '/Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/tmp-ios-arm64/frida-core'
ninja: no work to do.
Installing lib/base/libfrida-base-1.0.a to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib
Installing lib/base/frida-base.h to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/include/frida-1.0
Installing lib/base/frida-base-1.0.vapi to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/share/vala/vapi
Installing lib/payload/libfrida-payload-1.0.a to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib
Installing lib/payload/frida-payload.h to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/include/frida-1.0
Installing lib/payload/frida-payload-1.0.vapi to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/share/vala/vapi
Installing lib/agent/frida-agent.dylib to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib/frida
Installing lib/gadget/frida-gadget.dylib to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib/frida
Installing src/frida-helper to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib/frida
Installing src/api/frida-core.h to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/include/frida-1.0
Installing src/api/frida-core-1.0.vapi to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/share/vala/vapi
Installing src/api/frida-core-1.0.deps to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/share/vala/vapi
Installing src/api/libfrida-core-1.0.a to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib
Installing server/frida-server to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/bin
Installing portal/frida-portal to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/bin
Installing inject/frida-inject to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/bin
Installing /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/tmp-ios-arm64/frida-core/meson-private/frida-base-1.0.pc to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib/pkgconfig
Installing /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/tmp-ios-arm64/frida-core/meson-private/frida-payload-1.0.pc to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib/pkgconfig
Installing /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/tmp-ios-arm64/frida-core/meson-private/frida-core-1.0.pc to /Users/crifan/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/lib/pkgconfig
make[1]: Leaving directory '/Users/crifan/dev/dev_src/ios_reverse/frida/frida'

```

即可：编译完成。

我们要找到的：frida-server，貌似是：

- Installing server/frida-server to /Users/crifan/dev/dev\_src/ios\_reverse/frida/frida/build/frida-ios-arm64/usr/bin

去看看build目录

```
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/\ main" cd build
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/build"\ main" ll
total 56
-rw-r--r-- 1 crifan staff 190B 1 16 11:45 frida-env-ios-arm64.rc
-rw-r--r-- 1 crifan staff 193B 1 16 11:45 frida-env-macos-x86_64.rc
drwxr-xr-x 3 crifan staff 96B 1 16 11:47 frida-ios-arm64
-rw xr-xr-x 1 crifan staff 430B 1 16 11:45 frida-ios-arm64-pkg-config
-rw-r--r-- 1 crifan staff 2.9K 1 16 11:45 frida-ios-arm64.txt
-rw xr-xr-x 1 crifan staff 436B 1 16 11:45 frida-macos-x86_64-pkg-config
-rw-r--r-- 1 crifan staff 2.9K 1 16 11:45 frida-macos-x86_64.txt
-rw-r--r-- 1 crifan staff 217B 1 16 11:44 frida-version.h
drwxr-xr-x 8 crifan staff 256B 1 16 11:45 sdk-ios-arm64
drwxr-xr-x 8 crifan staff 256B 1 16 11:44 sdk-macos-x86_64
drwxr-xr-x 4 crifan staff 128B 1 16 11:47 tmp-ios-arm64
drwxr-xr-x 8 crifan staff 256B 1 16 11:44 toolchain-macos-x86_64
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/build"\ main" cd
frida-ios-arm64
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64"\ main" ll
total 0
drwxr-xr-x 6 crifan staff 192B 1 16 11:47 usr
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64"\ main" cd usr
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr"\ main" ll
total 0
drwxr-xr-x 6 crifan staff 192B 1 16 15:14 bin
drwxr-xr-x 3 crifan staff 96B 1 16 11:47 include
drwxr-xr-x 13 crifan staff 416B 1 16 15:14 lib
drwxr-xr-x 3 crifan staff 96B 1 16 11:47 share
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr"\ main" cd bin
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/bin"\ main" ll
total 40752
-rw xr-xr-x 1 crifan staff 6.2M 1 16 15:13 frida-inject
-rw xr-xr-x 1 crifan staff 5.2M 1 16 15:13 frida-portal
-rw xr-xr-x 1 crifan staff 6.4M 1 16 15:13 frida-server
-rw xr-xr-x 1 crifan staff 2.1M 1 16 11:47 gum-graft
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_src/ios_reverse/frida/frida/build/frida-ios-arm64/usr/bin"\ main" file ./frida-server
./frida-server: Mach-O 64-bit executable arm64
```

的确是编译成功了

回去看看之前frida-server的大小和file输出信息

```
crifan@licrifandeMacBook-Pro ~ ~/dev/dev_root/iosReverse/AppleStore/debug/gProcessInfo/i
```

```

Phone11_151「 11
total 55248
-rw-r-xr-x@ 1 crifan staff 629K 1 13 10:06 dyld
-rw-r--r-- 1 crifan staff 2.5M 1 13 22:14 dyld.id0
-rw-r--r-- 1 crifan staff 1.6M 1 13 22:13 dyld.id1
-rw-r--r-- 1 crifan staff 40K 1 13 22:13 dyld.nam
-rw-r--r-- 1 crifan staff 1.3K 1 13 22:13 dyld.til
-rw-r--r-- 1 crifan staff 173K 1 13 10:44 dyld_jtool2_S_symbol.txt
-rw-r--r-- 1 crifan staff 277B 1 13 10:07 dyld_rabin2_E_exports.txt
-rw-r--r-- 1 crifan staff 125B 1 13 10:07 dyld_rabin2_i_imports.txt
-rw-r--r-- 1 crifan staff 249K 1 13 21:33 dyld_rabin2_s_r_symbols.txt
-rw-r--r-- 1 crifan staff 381K 1 13 10:13 dyld_rabin2_s_symbols.txt
-rwxr-xr-x 1 crifan staff 20M 1 12 17:55 frida-server
-rw-r--r-- 1 crifan staff 169B 1 12 17:58 fridaServer_rabin2_E_exports.txt
-rw-r--r-- 1 crifan staff 17K 1 12 17:58 fridaServer_rabin2_i_imports.txt
-rw-r--r-- 1 crifan staff 1.9M 1 13 10:18 fridaSever_rabin2_s_symbols.txt
crifan@licrifandeMacBook-Pro「 ~/dev/dev_root/iosReverse/AppleStore/debug/gProcessInfo/
iPhone11_151「 file ./frida-server
./frida-server: Mach-O universal binary with 3 architectures: [arm64:Mach-O 64-bit executable arm64] [arm64e] [arm64]
./frida-server (for architecture arm64): Mach-O 64-bit executable arm64
./frida-server (for architecture arm64e): Mach-O 64-bit executable arm64e
./frida-server (for architecture arm64e): Mach-O 64-bit executable arm64e

```

-》此处frida-server很大：20M

不过明显是：FAT格式，包含多个架构：

- arm64
- arm64e

那看起来，貌似有个问题：

此处，从iPhone导出的真正在用的 frida-server，支持： arm64e

而此处自己编译出来的，只支持 arm64，不支持 arm64e

而记得：此处的 iPhone 中的架构都是： arm64e 的？

感觉需要：

- 确认 iOS 15.1 的 iPhone11 中，此处 arm 的架构是： arm64e 还是 arm64
  - 确定其中的 frida-server 是否需要支持 arm64e
- 如果需要支持 arm64e，再去看： frida 编译 core-ios 时，如何指定或加上 arm64e 的支持

先去：

【基本解决】iOS 15.1的iPhone11中frida-server所用架构是arm64e还是arm64

-》 arm64 的二进制，是能放到 arm64e 的 A13 的 iPhone11 中运行的。

那就继续看看：

【未解决】自己编译出的arm64的frida-server能否在iPhone11正常运行

其他过程详见：

- 【未解决】用frida源码自己编译出frida的iOS的包含frida-server的deb安装包
- 【未解决】自己编译出包含arm64和arm64e的FAT格式的frida-server二进制
- 【未解决】自己修改编译frida-core源码以尝试解决Frida的Missing gProcessInfo问题
- 【未解决】Frida中如何编译出iOS的arm64e的frida-server二进制
- 【未解决】自己编译Frida的frida-core代码生成可用二进制frida-server

## 找arm64e版的Frida

### 从Frida源码和build中找

- 【未解决】找Frida中iOS的arm6e4：从Frida源码和build中找

### 从Frida的github中找

- 【未解决】找Frida中iOS的arm6e4：从Frida的github中找

### 自己给make加echo打印日志调试

- 【未解决】找Frida中iOS的arm6e4：自己给make加echo打印日志调试

### 从github的ci的workflow中找arm64e

- 【未解决】找Frida中iOS的arm6e4：从github的ci的workflow中找arm64e

### 从编译日志中的Downloading ios-arm64入手

- 【未解决】找Frida中iOS的arm6e4：从编译日志中的Downloading ios-arm64入手

### make时如何传入arm64e的arch参数

- 【未解决】找Frida中iOS的arm6e4：make时如何传入arm64e的arch参数

### 从make编译时的log日志入手

- 【未解决】找Frida中iOS的arm6e4：从make编译时的log日志入手

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：  
2023-06-27 22:18:57

## 常见问题

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：  
2024-07-21 15:57:12

## 通用

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：  
2024-07-21 16:00:50

# Process terminated

关于 Process terminated = 进程结束 = 崩溃退出，目前遇到多种现象和可能原因：

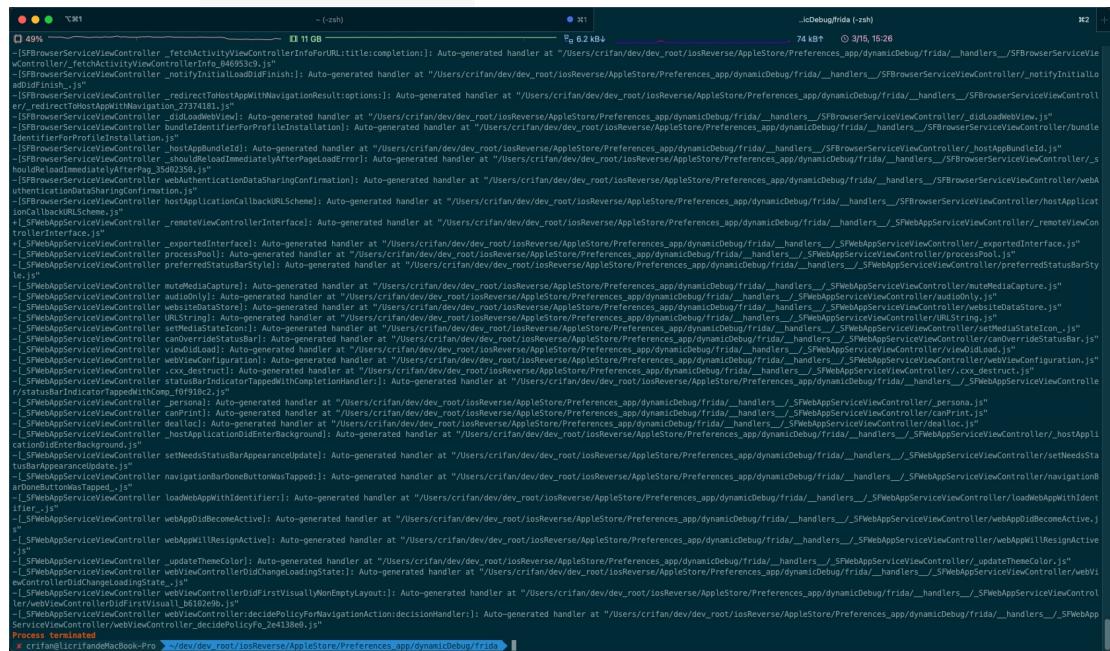
## 由于hook函数太多导致崩溃

- 现象一：用frida-trace去hook太多的类Obj的函数

```
frida-trace -U -F com.apple.Preferences -m "*[AA* *]" -m "*[AK* *]" -m "*[AS* *]" -m
"*[NS* *]" -M "-[ASDBundle copyWithZone:]" -M "-[ASDInstallationEvent copyWithZone
:]"
```



- 导致崩溃退出报错： Process terminated



- 原因： frida-trace 去hook的函数太多了，估计是，加了 -m "\*[NS\* \*]" 后导致崩溃
  - 注：iOS的ObjC的内部的多数，甚至是大多数，都是 NS 开头的，导致匹配到太多的类和函数，系统处理不过来了，导致frida崩溃，同时导致被调试的app崩溃。
    - 注： NS = NextStep，是iOS系统前身的苹果收购的NextStep公司名字
  - 解决办法：减少hook的范围=缩小匹配范围，比如此处改为： -m "\*[NSXPCC\* \*]"，暂时只关注我们要调试的 NSXPCCConnection 的相关内容，基本上可以：避免崩溃

## frida的 new ObjC.Object 方面的bug

- 现象：

- 概述 frida 调试时，由于加了 new ObjC.Object(someArg) 的 ptr 转换成 ObjC的对象，结果就会时不时的遇到 Process terminated，而崩溃停止退出调试
- 详解=具体现象

代码：

```

...
function hook_class_method(class_name, method_name)
{
 var hook = ObjC.classes[class_name][method_name];
 Interceptor.attach(hook.implementation, {
 onEnter: function(args) {
 console.log("===== [*] Detected call to: " + class_name + " -> " + method_name);
 //objc的函数，第0个参数是id，第1个参数是SEL，真正的参数从args[2]开始
 const argId = args[0];
 // console.log("argId: ", argId);

 const argSel = args[1];
 // console.log("argSel: ", argSel);

 const argSelStr = ObjC.selectorAsString(argSel);
 console.log("argSelStr: ", argSelStr);

 const argCount = occurrences(argSelStr, ":");
 console.log("argCount: ", argCount);

 for (let curArgIdx = 0; curArgIdx < argCount; curArgIdx++) {
 const curArg = args[curArgIdx + 2];
 // console.log("[%d] curArg: ", curArgIdx, curArg);
 // console.log("[%d]=", curArgIdx, " ,curArg=", curArg);
 // console.log("[%d]="+ curArgIdx + " ,curArg=" + curArg);
 console.log("----- [" + curArgIdx + "] curArg=" + curArg);
 if (curArg && (curArg != 0x0)) {
 console.log("curArg className: ", curArg.$className);
 const curArgObj = new ObjC.Object(curArg);
 console.log("curArgObj: ", curArgObj);
 console.log("curArgObj className: ", curArgObj.$className);
 }
 }
 }
 });
}
...

```

可以hook输出部分log日志，但是很快，时不时的，就崩溃退出了：

```

x crifan@licrifandeMacBook-Pro ~ ~/dev/dev_root/iosReverse/AppleStore/Preferences_app/dynamicDebug/frida$ frida -U -l hookAccountLogin_NSURL.js -F

| _ | Frida 16.0.10 - A world-class dynamic instrumentation toolkit
| (|
> - | Commands:
/_/_ | help -> Displays the help system
+ + + | object? -> Display information about 'object'
+ + + | exit/quit -> Exit
+ + + |
+ + + | More info at https://frida.re/docs/home/
+ + + |

```

```
 + + + + Connected to iPhone (id abdc0dd961c3cb96f5c4afe109de4eb48b88433a)
[*] Started: Hook all methods of a specific class
[+] Class Name: NSURL
 [*] Omit hooking + allocWithZone:
 [*] Omit hooking - _cfTypeID
 [*] Omit hooking - retain
 [*] Omit hooking - release
 [*] Omit hooking - copyWithZone:
[*] Completed: Hook all methods of a specific class
[iPhone::设置]-> ===== [*] Detected call to: NSURL -> - scheme
argSelStr: scheme
argCount: 0
===== [*] Detected call to: NSURL -> - _cfurl
argSelStr: _cfurl
argCount: 0
===== [*] Detected call to: NSURL -> - scheme
argSelStr: scheme
argCount: 0
===== [*] Detected call to: NSURL -> - _cfurl
argSelStr: _cfurl
argCount: 0
===== [*] Detected call to: NSURL -> + fileURLWithPath:isDirectory:
argSelStr: fileURLWithPath:isDirectory:
argCount: 2
----- [0] curArg 0x282336580
curArg className: undefined
curArgObj: /var/mobile/Library/Caches/com.apple.AppleAccount
curArgObj className: NSPathStore2
----- [1] curArg 0x0
===== [*] Detected call to: NSURL -> - initFileURLWithPath:isDirectory:
argSelStr: initFileURLWithPath:isDirectory:
argCount: 2
----- [0] curArg 0x282336580
curArg className: undefined
curArgObj: /var/mobile/Library/Caches/com.apple.AppleAccount
curArgObj className: NSPathStore2
----- [1] curArg 0x0
===== [*] Detected call to: NSURL -> - setResourceValues:error:
argSelStr: setResourceValues:error:
argCount: 2
Process terminated
[iPhone::设置]->
```

Thank you for using Frida!

```
Thank you for using Frida!
x crifan@lirifandeMacBook-Pro ~ /dev/dev_root/iosReverse/AppleStore/Preferences_app/dynamicDebug/frida frida -U -l hookAccountLogin NSURL.js -F
/ _| Frida 16.0.10 - A world-class dynamic instrumentation toolkit
| _|| Commands:
/_/ |_ help -> Displays the help system
.... object? -> Display information about 'object'
.... exit/quit -> Exit
.... More info at https://frida.re/docs/home/
.... Connected to iPhone (id=abdc0dd961c3cb96f5c4afe109de4eb48b88433a)
[*] Started: Hook all methods of a specific class
[+] Class Name: NSURL
 [*] Omit hooking + allocWithZone:
 [*] Omit hooking - _cfTypeID
 [*] Omit hooking - retain
 [*] Omit hooking - release
 [*] Omit hooking - copyWithZone:
[*] Completed: Hook all methods of a specific class
[iPhone::设置]-> ===== [*] Detected call to: NSURL -> - scheme
argSelStr: scheme
argCount: 0
===== [*] Detected call to: NSURL -> - _cfurl
argSelStr: _cfurl
argCount: 0
===== [*] Detected call to: NSURL -> - scheme
argSelStr: scheme
argCount: 0
===== [*] Detected call to: NSURL -> - _cfurl
argSelStr: _cfurl
argCount: 0
===== [*] Detected call to: NSURL -> + fileURLWithPath:isDirectory:
argSelStr: fileURLWithPath:isDirectory:
argCount: 2
----- [0] curArg=0x282336580
curArg className: undefined
curArgObj: /var/mobile/Library/Caches/com.apple.AppleAccount
curArgObj className: NSPathStore2
----- [1] curArg=0x0
===== [*] Detected call to: NSURL -> - initFileURLWithPath:isDirectory:
argSelStr: initFileURLWithPath:isDirectory:
argCount: 2
----- [0] curArg=0x282336580
curArg className: undefined
curArgObj: /var/mobile/Library/Caches/com.apple.AppleAccount
curArgObj className: NSPathStore2
----- [1] curArg=0x0
===== [*] Detected call to: NSURL -> - setResourceValues:error:
argSelStr: setResourceValues:error:
argCount: 2
Process terminated
[iPhone::设置]->

Thank you for using Frida!
x crifan@lirifandeMacBook-Pro ~ /dev/dev_root/iosReverse/AppleStore/Preferences_app/dynamicDebug/frida
```

- 原因：暂不清楚
    - 可能原因：frida的objc的object转换方面的bug，暂时无法解决
      - 详见：
        - 【未解决】frida中hook函数打印参数值时最后app崩溃frida输出Process terminated
        - 【未解决】frida中hook调试iOS的ObjC的函数参数时始终出现崩溃Process terminated

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:

2024-07-21 15:57:08

## Failed to spawn: unable to find process with name

- 报错: Failed to spawn: unable to find process with name 'Preferences'

◦

- 原因: frida命令用的是

```
frida -U -l ./hookAccountLogin.js -n Preferences
```

- 其中 -n 是加二进制名称, 此处 Preferences 是app, 所以属于参数使用错误, 调试目标语法搞错了

- 解决办法:

- 搞懂frida的调试目标方式, 改为别的方式即可

- 方式1: 用 -N app\_package\_id

```
frida -U -l ./hookAccountLogin.js -N com.apple.Preferences
```

- 方式2: 换 -p PID

```
frida -U -l ./hookAccountLogin.js -p 18031
```

- 其中是用iPhone中ssh中通过ps查看到

```
~ ps -A | grep Preferences
...
18031 ?? 0:02.43 /Applications/Preferences.app/Preferences
```

- 得知 Preferences 的 PID 是 18031

- 方式3: 用 -F

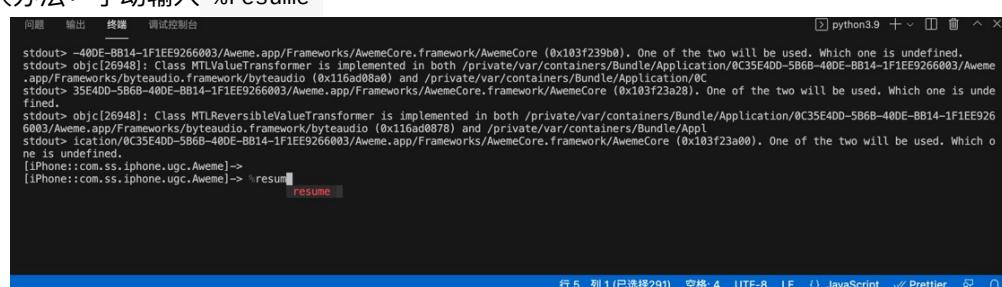
```
frida -U -l ./hookAccountLogin.js -F
```

- 注: 确保 Preferences =系统的设置app, 处于最前台在运行才能用 -F

## --no-pause和--pause

关于frida启动后，被调试的目标，是否暂停运行的问题：

- 背景
    - frida启动调试后，被调试的目标（app或进程），是否已经暂停运行
      - 旧版frida：自动暂停运行
        - 支持参数：`--no-pause`
      - 新版frida：（逻辑已经变成了）不暂停运行 = 已经继续运行了
        - 支持参数：`--pause`
  - 所以
    - 旧版frida
      - 常会遇到一个问题：每次frida（以Attach后Spawn去）启动调试后，程序自动暂停运行



- 所以就希望：frida调试开始后，自动继续运行，不要每次都输入 %resume 才继续运行

- 解决办法：加 `--no-pause` 参数
    - 参数含义：`--no-pause` automatically start main thread after startup
    - 举例

```
frida -U --no-pause -f com.ss.iphone.uac.Aweme -l frida/dvldimage.js
```

- ### ◦ 新版frida

- 用新版frida，加了参数 `--no-pause`，报错不支持此参数： `frida: error: unrecognized arguments: --no-pause`
    - 解决办法：不要加任何参数
    - 举例

```
frida -U -N com.apple.Preferences -l hookAccountLogin_singleClassAllMethod.is
```

- 8 -

- 如果需要启动后
    - 自动继续运行
      - 则：无需加任何参数
        - 因为新版frida已经变成这个逻辑了
    - 自动暂停运行
      - 再去加新版才支持的参数： --pause
        - 参数含义： --pause leave main program



# Process crashed: Bad access due to invalid address

- 问题:

```
Called: -[WAResponseBuilder clientLogRequestURLWithCurrentScreen:previousScreen:
actionTaken:]
...
argCount: 3
----- [0] curArgPtr=0xc
Process crashed: Bad access due to invalid address
```

```
curlArgPtr[0]: className= _NSCFString, value= SEDEF7B8-C68A-4324-90X0-A1536E9864CF
onLeave: retValObjStr=>WAURQueryItem: 0x28214345e0
onLeave: retValObjStr=>WAURQueryItem: 0x28214345e0
----- Called: +[WAURQueryItem queryItemWithHomeName:value:] -----
argCount: 2
----- [0] curArgPtr=0x10236f550
curArgPtr[0]: className= _NSCFString, value= expid
----- [1] curArgPtr=0x2823e2680
curArgPtr[0]: className= _NSCFString, value= gRnURURqUu2I9.1gRddg=
----- Called: +[WAURQueryItem initWithHomeName:value:] -----
argCount: 2
----- [0] curArgPtr=0x10236f550
curArgPtr[0]: className= _NSCFString, value= expid
----- [1] curArgPtr=0x2823e2680
curArgPtr[0]: className= _NSCFString, value= gRnURURqUu2I9.1gRddg=
----- [0] curArgPtr=>WAURQueryItem: 0x2821434640
onLeave: retValObjStr=>WAURQueryItem: 0x2821434640
onLeave: retValObjStr=>WAURQueryItem: 0x2821434640
----- [0] curArgPtr=>WAURQueryItem: 0x282313ef6b
----- Called: +[WAURQueryItemBuilder clientSideRequestRunInCurrentScreen:previousScreen:actionToken:] -----
----- printFunctionCallStack:symbol -----
0 WhatsApp mem::0x10469f4c file::0x10236f64 [WARegistrationManager performLogoffForSession:currentScreen:previousScreen:actionToken:withCompletion:]
1 WhatsApp mem::0x10469f51 file::0x10236f51a [WARegistrationLogger logWithSession:currentScreen:]
2 WhatsApp mem::0x10469f51a file::0x10236f51a [WARegistrationLogger logCurrentSessionWithSession:]
3 WhatsApp mem::0x10469f51b file::0x10236f51b [WARegistrationLogger logClientWithNextState:]
4 WhatsApp mem::0x10469f2c4 file::0x1022c2e294 [WAVerificationRegistrationController moveToState:]
5 WhatsApp mem::0x103a076d4 file::0x10270f60a [WARegistrationPhoneInputViewModel handleFailedAccountDefensWithIVhViewController:session:reason:response:]
6 WhatsApp mem::0x103a07734 file::0x10170f734 [WARegistrationPhoneInputViewModel handleEnteredPhoneWithViewController:session:eligibleForCodeFromOldWa:eligibleForCodeFromEmail:]
7 WhatsApp mem::0x10469f77c file::0x1022d7734 [moveC function?]
8 UIKitCore mem::0x1cd3d0e0 file::0x10236f51b [UIPresentationController invalidateHandlersForAction:]
9 UIKitCore mem::0x1cd3d0e82 file::0x10236f51b [UIPresentationController dismissAnimated:triggeringAction:triggeredBy:popoverDimmingView:dismissCompletion:]_block_invoke.458
10 UIKitCore mem::0x1cd3d1f80 file::0x183e1cf80 [UIPresentationController transitionDidFinish:]
11 UIKitCore mem::0x1cd3d176c file::0x183e2176c _56-[UIPresentationController runTransitionForCurrentState]_block_invoke.503
12 UIKitCore mem::0x1cd3d1568 file::0x183f25568 [UIViewControllerAnimatedTransitionContext completeTransition:]
13 UIKitCore mem::0x1cd3d195d8 file::0x18349d169 [UIViewControllerAnimatedTransitionContext didEndBlockAnimation:finished:context:]
14 UIKitCore mem::0x1cd3d195c70 file::0x18349d169 [UIViewControllerAnimatedTransitionContext sendDelegateNotification:@0x5178:finished:]
15 UIKitCore mem::0x1cd3d191b78 file::0x18349d169 [UIViewControllerAnimatedTransitionContext animationDidStop:finished:]

argCount: 3
----- [0] curArgPtr=>c
Process crashed: Bad access due to invalid address

```

- 原因
    - 表面原因：访问了非法地址： 0xc
    - 深层次原因：对于值明显异常的地址，没有做过滤，没有排除掉
  - 解决办法：加上过滤，排除掉，地址值明显异常的地址
  - 具体步骤：

代码改为：

```
// check pointer is valid or not
// example
// 0x103e79560 => true
// 0xc => false
function isValidPointer(curPtr){
 let MinValidPointer = 0x10000
 var isValid = curPtr > MinValidPointer
 return isValid
```

```
if (isValidPointer(curArg)) {
 ...
}
```

即可避免访问非法地址指针，避免崩溃。

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：  
2024-07-21 16:04:45

## ValueError: file descriptor cannot be a negative integer

- 问题: `frida-ps`、`frida-ls-devices` 等`frida-tools`工具运行时报错: `ValueError: file descriptor cannot be a negative integer (-42)`
- 原因: 当前 12.0.3 的 `frida-tools` 有bug
- 解决办法: 升级到最新版 `frida-tools`
- 具体步骤:

```
pip install --upgrade frida_tools
```

- 注: 查看当前`frida-tools`的版本:

```
pip show frida_tools
```

# iOS

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：  
2024-07-21 16:02:29

## Failed to attach: need Gadget to attach on jailed iOS

- 问题：Mac中用frida启动iOS版抖音

```
frida -U -f com.ss.iphone.ugc.Aweme
```

- 报错

```
Failed to attach: need Gadget to attach on jailed iOS; its default location is:
/Users/crifan/.cache/frida/gadget-ios.dylib
```

- 原因：缺少对应的Frida的 gadget 库文件
- 解决办法
  - 概述：下载对应版本的 gadget 库文件，放到对应位置（此处提示的 /Users/crifan/.cache/frida/gadget-ios.dylib ）即可
  - 详见：[安装Frida](#)中的 安装Frida的gadget

## 导致iPhone重启

之前遇到过：

XinaA15的rootless越狱的iPhone11中，用最新版 v16.0.10 的 Frida，但是却会导致iPhone重启（从而丢失XinaA15的越狱，需要再去恢复越狱）

但是：没有解决方案。

目前的结论是：

(XinaA15等) rootless越狱后，Frida 的使用，基本上是个大问题，有时候（某个旧版本）能用，有时候（此处新版 v16.0.10 ）却又无法正常使用。

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：  
2024-07-21 15:49:46

## Waiting for USB device to appear

- 问题：Mac中运行frida去调试app

```
frida -U -f com.apple.store.Jolly
```

- 但是报错

```
Waiting for USB device to appear...
```

- 原因：（连接iPhone到Mac的）USB数据线没插好
- 解决办法：重新拔插USB数据线，确保USB连接正常
  - 注：
    - 如何确认iPhone是否已插好
      - 方式1：通过爱思助手可以确认
        - 已插好：能看到iPhone详情
        - 没插好：看不到iPhone设备
      - 方式2：用frida的工具[frida-ls-devices](#)

## Error unable to intercept function at please file a bug

- 现象：Frida的hook的js代码中，函数地址写的是：

```
var akdSymbol2575_functionAddress = 0x1000a0460;
```

- 导致报错：

```
moduleName= akd moduleBaseAddress= 0x102b40000
functionRealAddress)= 0x202be0460
Error: unable to intercept function at 0x202be0460; please file a bug
at value (frida/runtime/core.js:367)
```

- 原因：函数地址写错了 -» 找不到函数地址 -» 所以报错
- 解决办法：确保函数地址是正确的
- 具体做法：代码改为：

```
var akdSymbol2575_functionAddress = 0xa0460;
```

- 说明

- 0x1000a0460 是加了 VM 虚拟地址后的 akd二进制内函数的地址
- 0xa0460 : 是不带VM的，真正的函数内的偏移地址

# Android

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：  
2024-07-21 16:02:35

## Error java.lang.ClassNotFoundException Didn't find class on path DexPathList zip file

### 情况1：类名错了

- 现象

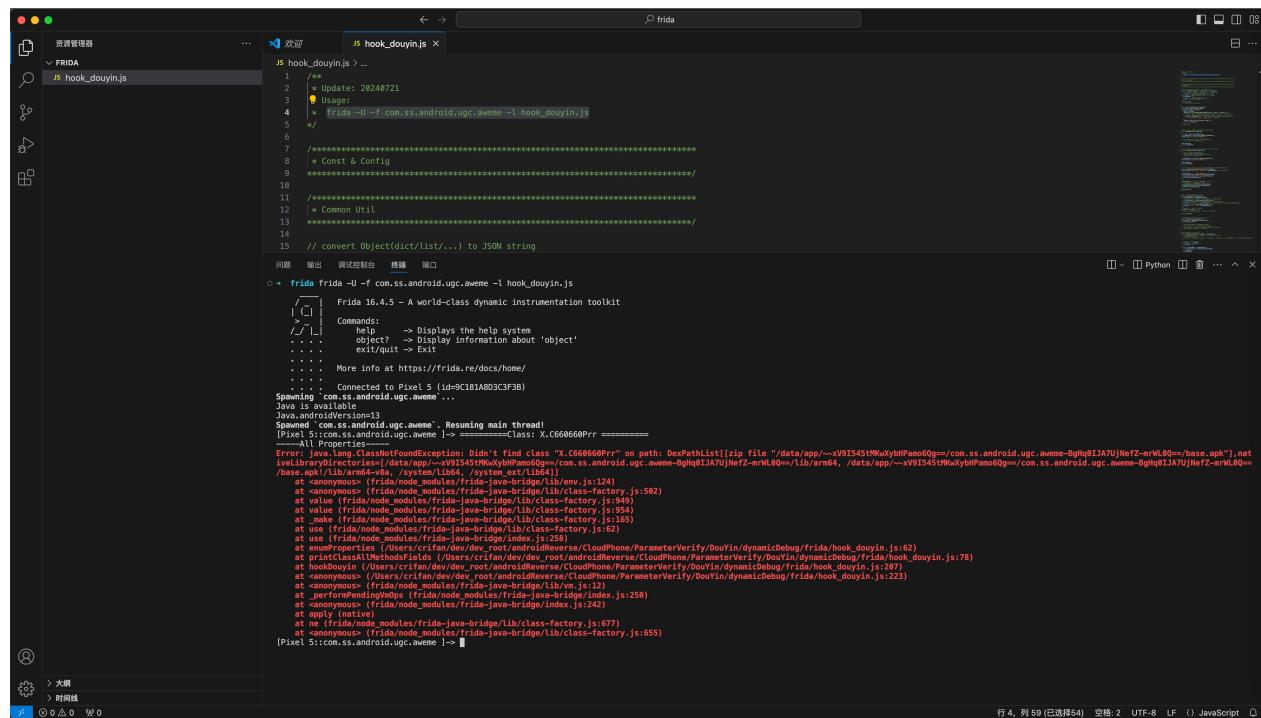
Frida去hook抖音代码：

```
var C660660PrrClassName = "X.C660660Prr"
printClassAllMethodsFields(C660660PrrClassName)

var C660660PrrCls = Java.use(C660660PrrClassName)
console.log("C660660PrrCls=" + C660660PrrCls)
```

报错：

```
[Pixel 5::com.ss.android.ugc.aweme] - ======Class: X.C660660Prr ======
----All Properties-----
Error: java.lang.ClassNotFoundException: Didn't find class "X.C660660Prr" on path: DexPathList[[zip file "/data/app/~~xV9I545tMKwXybHPamo6Qg==/com.ss.android.ugc.aweme-BgHq0IJ
JA7UjNefZ-mrWLQ==/base.apk"],nativeLibraryDirectories [/data/app/~~xV9I545tMKwXybHPamo
6Qg=/com.ss.android.ugc.aweme-BgHq0IJ
JA7UjNefZ-mrWLQ=/lib/arm64, /data/app/~~xV9I545t
MKwXybHPamo6Qg=/com.ss.android.ugc.aweme-BgHq0IJ
JA7UjNefZ-mrWLQ=/base.apk /lib/arm64-
v8a, /system/lib64, /system_ext/lib64]]
 at anonymous (frida/node_modules/frida-java-bridge/lib/env.js:124)
 at anonymous (frida/node_modules/frida-java-bridge/lib/class-factory.js:502)
 at value (frida/node_modules/frida-java-bridge/lib/class-factory.js:949)
 at value (frida/node_modules/frida-java-bridge/lib/class-factory.js:954)
 at _make (frida/node_modules/frida-java-bridge/lib/class-factory.js:165)
 at use (frida/node_modules/frida-java-bridge/lib/class-factory.js:62)
 at use (frida/node_modules/frida-java-bridge/index.js:258)
 at enumProperties (/Users/crifan/dev/dev_root/androidReverse/xxx/ParameterVerify/Do
uYin/dynamicDebug/frida/hook_douyin.js:62)
 at printClassAllMethodsFields (/Users/crifan/dev/dev_root/androidReverse/xxx/Parame
terVerify/DouYin/dynamicDebug/frida/hook_douyin.js:78)
 at hookDouyin (/Users/crifan/dev/dev_root/androidReverse/xxx/ParameterVerify/DouYin
/dynamicDebug/frida/hook_douyin.js:207)
 at anonymous (/Users/crifan/dev/dev_root/androidReverse/xxx/ParameterVerify/DouYi
n/dynamicDebug/frida/hook_douyin.js:223)
 at anonymous (frida/node_modules/frida-java-bridge/lib/vm.js:12)
 at _performPendingVmOps (frida/node_modules/frida-java-bridge/index.js:250)
 at anonymous (frida/node_modules/frida-java-bridge/index.js:242)
 at apply (native)
 at ne (frida/node_modules/frida-java-bridge/lib/class-factory.js:677)
 at anonymous (frida/node_modules/frida-java-bridge/lib/class-factory.js:655)
```



即：找不到类名

- 原因：类名写错了
- 根本原因：

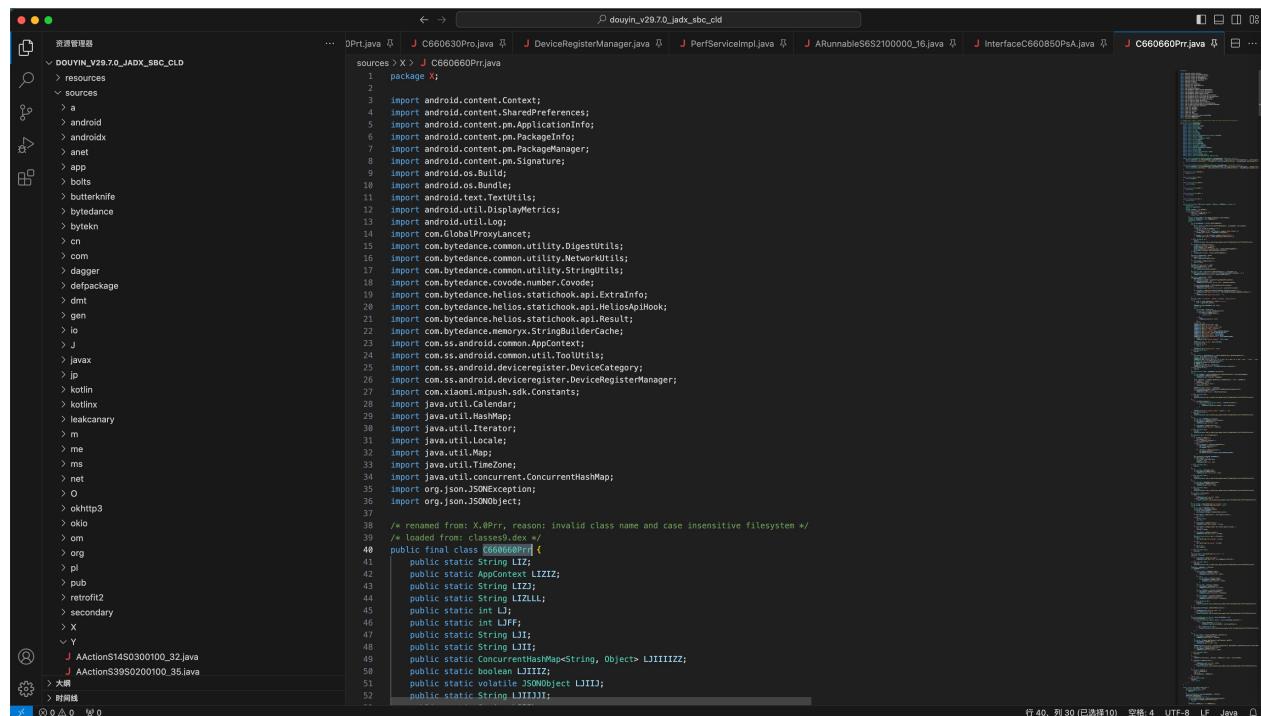
之前Jadx反编译出的源码是：

`sources/X/C660660Prr.java`

以为对应的类名是：`x.C660660Prr`

但是实际上是， jadx反编译中的注释

```
/* renamed from: X.0Prr, reason: invalid class name and case insensitive filesystem */
/* loaded from: classes9.dex */
public final class C660660Prr {
```



中的： X.0Prr

- 解决办法：把名字改为真正的Java的类名： X.0Prr
- 具体步骤

代码改为：

```
var C660660PrrClassName = "X.0Prr"
printClassAllMethodsFields(C660660PrrClassName)

var C660660PrrCls = Java.use(C660660PrrClassName)
console.log("C660660PrrCls=" + C660660PrrCls)
```

## 情况2：类不在当前hook的二进制中

- 背景

安卓逆向期间，去根据之前Logcat日志：

```
2023-08-31 10:26:23.911 6481-6481 AppButtonsPrefCtl com.android.settings
D Stopping package com.wallpaper.hd.funny
```

找到了安卓内部的类： com.android.server.pm.Settings 中的函数： createNewSetting ， 输出了上述的log

- 现象

去Frida中用js代码：

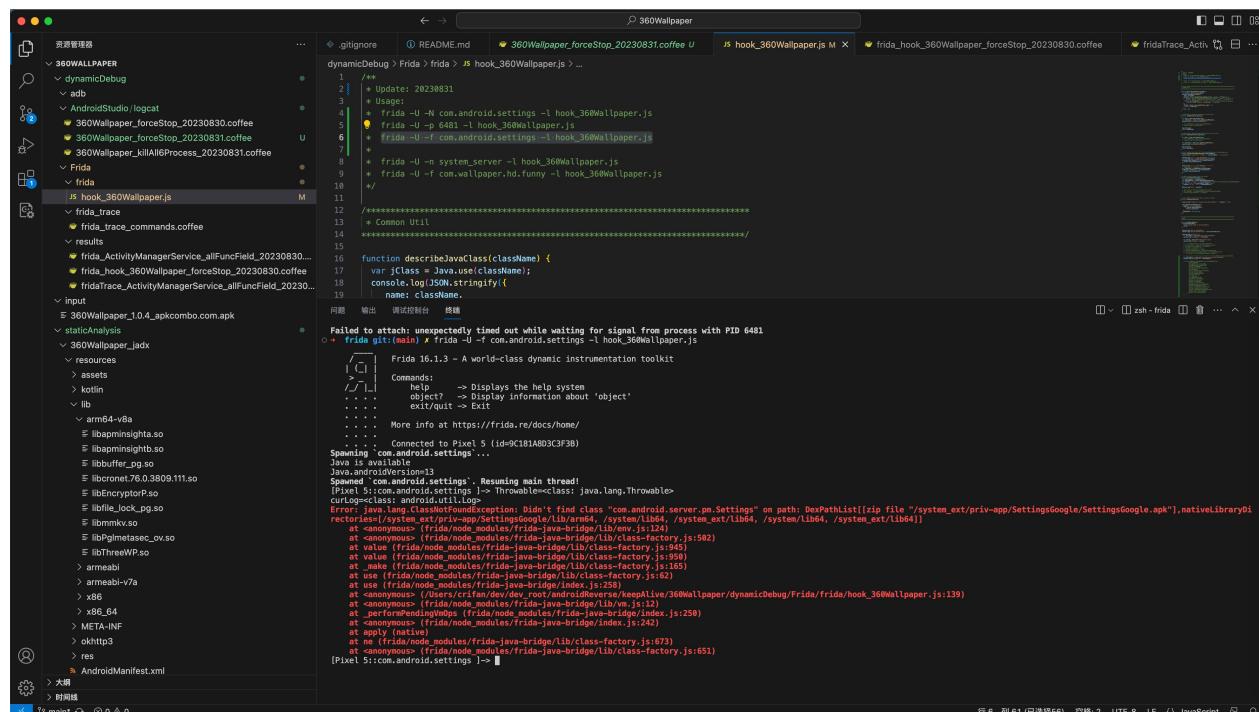
```
var SettingsCls = Java.use("com.android.server.pm.Settings")
```

去hook安卓内部的类: com.android.server.pm.Settings

```
frida -U -f com.android.settings -l hook_360Wallpaper.js
```

报错:

```
Error: java.lang.ClassNotFoundException: Didn't find class "com.android.server.pm.Settings" on path: DexPathList[[zip file "/system_ext/priv-app/SettingsGoogle/SettingsGoogle.apk"],nativeLibraryDirectories [/system_ext/priv-app/SettingsGoogle/lib/arm64, /system/lib64, /system_ext/lib64, /system/lib64, /system_ext/lib64]]
at anonymous (frida/node_modules/frida-java-bridge/lib/env.js:124)
at anonymous (frida/node_modules/frida-java-bridge/lib/class-factory.js:502)
at value (frida/node_modules/frida-java-bridge/lib/class-factory.js:945)
at value (frida/node_modules/frida-java-bridge/lib/class-factory.js:950)
at _make (frida/node_modules/frida-java-bridge/lib/class-factory.js:165)
at use (frida/node_modules/frida-java-bridge/lib/class-factory.js:62)
at use (frida/node_modules/frida-java-bridge/index.js:258)
at anonymous (/Users/crifan/dev/dev_root/androidReverse/keepAlive/360Wallpaper/dynamicDebug/Frida/frida/hook_360Wallpaper.js:139)
at anonymous (frida/node_modules/frida-java-bridge/lib/vm.js:12)
at _performPendingVmOps (frida/node_modules/frida-java-bridge/index.js:250)
at anonymous (frida/node_modules/frida-java-bridge/index.js:242)
at apply (native)
at ne (frida/node_modules/frida-java-bridge/lib/class-factory.js:673)
at anonymous (frida/node_modules/frida-java-bridge/lib/class-factory.js:651)
```



## • 原因

- 此处安卓系统的app: 设置, 包名: com.android.settings
- 好像没包含, 此处输出的Stopping package的日志的代码:
- 类 com.android.server.pm.Settings 的函数 createNewSetting
- 因为是另外的二进制程序 system\_server 才包含此函数

- 解决办法

经过实测，换 system\_server 去hook，即可找到。

- 具体步骤

```
frida -U -n system_server -l hook_360Wallpaper.js
```

即可找到该类，正常输出：

```
SettingsCls<-- class: com.android.server.pm.Settings
```

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：  
2024-07-21 16:22:12

## 基于Frida的工具

- 基于Frida的工具
  - 脱壳/砸壳工具
    - frida-ios-dump
      - <https://github.com/AloneMonkey/frida-ios-dump>
      - pull decrypted ipa from jailbreak device
    - frida-unpack
      - <https://github.com/dstmath/frida-unpack>
      - 基于Frida的脱壳工具
  - 调试工具
    - frick
      - <https://github.com/iGio90/frick>
      - frick - aka the first debugger built on top of frida
      - frick: 基于 (Unicorn、capstone和) frida的调试工具
  - frida-ios-hook
    - <https://github.com/noobpk/frida-ios-hook>
      - A tool that helps you easily trace classes, functions, and modify the return values of methods on iOS platform
  - Fridump
    - <https://github.com/Nightbringer21/fridump>
      - A universal memory dumper using Frida
  - for Android
    - frida\_dump
      - [https://github.com/lasting-yang/frida\\_dump](https://github.com/lasting-yang/frida_dump)
      - frida dump dex, frida dump so
  - frida-cycript
    - <https://github.com/nowsecure/frida-cycript>
      - Cycript fork powered by Frida
  - r2frida
    - <https://github.com/nowsecure/r2frida>
      - Radare2 and Frida better together
  - frida-trace
    - <https://github.com/nowsecure/frida-trace>
      - Trace APIs declaratively through Frida

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:  
2023-06-24 21:31:58

## Frida的用途

可以用 Frida 实现各种逆向的用途。

其中 [Frida Codeshare](#) 就有很多例子，感兴趣的自己去挖掘挖掘。

下面整理出部分相关内容。

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：  
2023-06-27 23:00:31

## 反调试

据说，可以用frida脚本（临时）去规避掉反调试的 ptrace：

- [使用Frida绕过iOS反调试 | La0s](#)
- [一例简单的frida反调试绕过 - 网安](#)

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：  
2023-06-27 22:28:26

## 辅助反混淆

据说可以借助于 Frida 的 Stalker , 追踪提示真正指令执行过程后, 然后: 用于辅助反混淆。

关于Stalker的使用案例, 详见: [Stalker](#)

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:  
2023-06-27 22:36:43

## 绕过参数加密

- 通过frida的hook去实现绕过参数加密
  - 用frida绕过apk中加密参数?

部分参考资料：

- 利用frida快速解决APP中某tong加密 | 大熊の笔记
- frida跑加密算法和绕过抓包验证\_不想写程序(〒^\_^(〒)的博客-CSDN博客
- 【保姆级教学】某金融app FRIDA hook加解密算法+jsrpc=乱杀 - T00ls.Com
- 某金融app的加解密hook+rpc+绕过SSLPinning抓包 - T00ls.Com

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:

2023-06-27 22:29:51

# 逆向app

## 抖音

- 从app启动最开始就开始hook调试

- 交互式调试

```
frida -U -f com.ss.iphone.ugc.Aweme
```

- 用js脚本

- 调试 dyld

```
frida -U -f com.ss.iphone.ugc.Aweme -l frida/dyldImage.js
```

- 其他

- 【未解决】Mac中Frida启动抖音app进程并调试和hook函数
- 【未解决】用Frida的frida-trace去hook函数iOS版抖音
- 【未解决】frida调试抖音app去hook函数: \_dyld\_get\_image\_name
- 【未解决】frida去hook函数\_dyld\_get\_image\_name时打印参数为字符串
- 【未解决】用Frida动态调试iOS版抖音app
- 【未解决】Mac中用Frida调试iOS版抖音
- 【已解决】用frida启动hook调试iOS抖音app
- 【未解决】尝试Frida的stalker能否修复抖音AwemeCore中函数名常量字符串

## Apple账号 ~= AppleStore

- Apple账号 ~= AppleStore = Preferences + (AuthKit 的daemon) akd + AppleAccount + AppleAccountUI + AppleMediaServices + libMobileGestalt + 其他
  - 【记录】iOS逆向Apple账号：用frida和frida-trace去hook打印更多账号相关函数调用
  - 【无法解决】iOS逆向Apple账号：用frida的ssl bypass脚本尝试解决Charles抓包代理报错
  - 【未解决】iOS逆向Apple账号：用Frida去监控NSURL去调试Apple账号登录过程
  - 【未解决】iOS逆向Apple账号：用Frida去调试NSURL核心网络请求函数调用
  - 【未解决】iOS逆向Apple账号：分析研究frida抓包到的Apple账号登录过程和网络相关的内容

## 迅雷

- 迅雷

- 【记录】用frida动态调试重新打包后的安卓迅雷apk
- 【未解决】Mac中搭建Frida的动态调试安卓apk的开发环境



## 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：  
2023-06-22 22:04:13

## 参考资料

- 【整理】 Frida相关资料
- 【已解决】 frida-core代码对应着frida中是哪个文件
- 【未解决】 Mac中安装frida
- 【记录】 iOS的iPhone中安装Frida
- 【记录】 Sileo中安装tweak插件： Frida
- 【已解决】 Mac M2 Max中安装Frida环境
- 【已解决】 Mac中升级Frida到最新版本16.0.11
- 【未解决】 Mac中安装frida
- 【已解决】 Mac中升级Frida到新版本16.0.10
- 【记录】 palera1n普通越狱后安装插件： Frida
- 【规避解决】 palera1n的rootless越狱iPhone中初始化frida开发环境
- 【记录】 下载Frida的动态库： FridaGadget.dylib
- 【记录】 iPhone8中的升级frida到最新版16.0.11
- 【记录】 iOS的iPhone中安装Frida
- 【已解决】 确认palera1n的rootful普通越狱中Frida是否可用
- 【已解决】 frida-ios-dump砸壳ipa报错： need Gadget to attach on jailed iOS
- 【已解决】 frida启动时到底有哪几种attach和spwan启动方式
- 【整理】 再次研究确认Frida调试目标的几种方式
- 【已解决】 iOS逆向： Frida调试设置出现页面弹框： 接入互联网以登录iPhone
- 【未解决】 XinaA15越狱的rootless的iPhone11中frida出错： 安装和卸载时的路径异常/var/jb/var/jb/
- 【记录】 iPhone11重新安装XinaA15后重新安装Frida
- 【已解决】 XinaA15越狱的iPhone11中安装Frida
- 【记录】 XinaA15恢复越狱环境后重新安装Frida
- 【记录】 iPhone8中用Sileo安装Frida
- 【记录】 iPhone11中Sileo中安装最新版16.0.11的Frida
- 【记录】 Sileo中安装tweak插件： Frida
- 【记录】 iPhone11中用Sileo重新安装最新版16.0.10的Frida
- 【已解决】 iPhone中Cydia中安装升级最新版的16.0.2的Frida
- 【未解决】 Frida从16.0.8到升级到新版本16.0.10
- 【已解决】 越狱后的iPhone11中卸载和重新安装Frida
- 【已确认】 iPhone11中重新确认新版XinaA和新版Frida是否可以正常使用
- 【已解决】 Frida启动js脚本报错： Error could not parse line 1 expecting ,
- 【基本解决】 Frida的Stalker.follow中events的含义和用法
- 【已解决】 优化Frida调用脚本时无需每次查看PID直接传入进程名
- 【已解决】 iOS逆向： frida调试hook类NSXPConnection的setExportedObject
- 【已解决】 Frida中Stalker如何在Stalker.follow中写代码追踪指令代码运行
- 【已解决】 用frida去hook打印一个类的所有的方法
- 【已解决】 frida去hook函数时确保已执行函数但不触发不输出日志log
- 【已解决】 iOS逆向Apple账号： 用frida打印com.apple.ak.auth.xpc的NSXPConnection的其他属性值
- 【记录】 iOS逆向akd： frida-trace追踪出iPhone8中iOS15.0中账号验证的函数调用过程结果
- 【已解决】 Mac中如何使用frida的Stalker的js脚本
- 【已解决】 Frida调试时如何指定进程

- 【已解决】 Frida中如何通过frida启动被测app程序iOS版抖音
- 【已解决】 用frida去hook报错： Failed to spawn unable to find process with name
- 【已解决】 frida调试进程直接运行不要每次都输入%resume才运行
- 【未解决】 iOS逆向Apple账号： 用frida去hook打印所有iOS的ObjC函数调用
- 【已解决】 Frida中js代码中两个地址指针数值相加
- 【已解决】 frida中ptr的含义
- 【已解决】 js中把ObjC函数调用写法转换成Frida中ObjC函数调用写法
- 【已解决】 frida打印iOS的ObjC函数的参数的值
- 【已解决】 frida动态判断iOS的ObjC函数的参数个数并打印参数值
- 【已解决】 frida去hook单个类的单个函数
- 【已解决】 用frida去hook打印一个类的所有的方法
- 【已解决】 frida去hook监控ObjC函数时把类名和函数名整合成Objc的语法
- 【已解决】 Frida中hook打印iOS中的类和函数
- 【已解决】 iOS逆向： Frida的hook中onLeave中如何获取到self类本身实例变量
- 【已解决】 frida中ObjC.Object的语法和含义
- 【未解决】 frida中打印ObjC参数类型NSPathStore2的值
- 【已解决】 frida的hook中如何把NSString转换为C的string并打印
- 【未解决】 frida去hook时判断函数参数的类型是否是NSString
- 【已解决】 frida中打印操作Objc的参数SEL导致程序崩溃
- 【已解决】 Mac中Frida去hook函数： 用frida还是frida-trace
- 【未解决】 Mac中frida-trace报错： Failed to spawn unable to find process with name
- 【已解决】 iOS逆向akd： 用frida-trace追踪账号验证的过程
- 【已解决】 frida-trace中把更多参数放到参数文件中
- 【已解决】 frida-trace去hook导致app崩溃最后输出Process terminated
- 【已解决】 frida-trace输出log日志到文件中
- 【记录】 iOS逆向Apple账号： 分析frida-trace去hook到的下一步按钮内部的函数调用逻辑
- 【记录】 iOS逆向akd： frida-trace追踪出iPhone8中iOS15.0中账号验证的函数调用过程结果
- 【记录】 iOS逆向Apple账号： frida-trace去hook逻辑加上NSString和NSError等通用的类
- 【已解决】 iOS逆向调试： 用Frida-trace实时追踪相关ObjC函数调用
- 【未解决】 iOS逆向Apple账号： 用frida-trace去hook监控NSURL网络请求
- 【未解决】 iOS逆向Apple账号： 用Frida去hook网络请求NSURL相关函数调用
- 【记录】 iOS逆向Apple账号： frida-trace追踪多个ObjC函数的调用过程
- 【记录】 iOS逆向Apple账号： 优化frida-trace的函数hook范围以定位账号登录过程核心逻辑
- 【未解决】 iOS逆向Apple账号： 用frida-trace调试signin/v2/login的函数调用过程
- 【记录】 frida-trace调试追踪到的输入密码下一步的账号登录过程中的ObjC函数调用过程
- 【记录】 Mac中测试Frida各个命令的效果
- 【已解决】 Mac中frida报错： ValueError file descriptor cannot be a negative integer -42
- 【记录】 frida的frida-ps用法
- 【整理】 frida心得： frida-ps用法详解
- 【未解决】 iPhone11中如何停止frida-server
- 【未解决】 iPhone中用launchctl去启动和停止frida-server服务进程
- 【未解决】 frida导致iPhone重启： 分析panic-full的ips是否和frida-server崩溃有关系
- 【基本解决】 iOS 15.1的iPhone11中frida-server所用架构是arm64e还是arm64
- 【未解决】 Frida的stalker trace
- 【基本解决】 Frida的Stalker.follow中events的含义和用法
- 【已解决】 Frida去hook函数报错： Error unable to intercept function at please file a bug

- 【无法解决】 Frida的Stalker中transform中的instruction是否可以获取到bytes即opcode
- 【已解决】 优化Frida的Stalker的代码追踪逻辑
- 【已解决】 Frida的Stalker中函数地址和指令地址匹配不上
- 【已解决】 搞懂Frida的Stalker.follow的transform的调试指令运行的逻辑
- 【已解决】 Frida的Stalker.follow中transform中的指令instruction
- 【记录】 分析Frida的Stalker对于\_\_lldb\_unnamed\_symbol2575代码追踪的输出结果
- 【记录】 学习Frida官网文档Stalker搞懂基本用法和逻辑
- 【已解决】 Frida的Stalker中去判断是否是函数的代码指令的逻辑
- 【整理】 Frida中对于js代码的支持不够完美
- 【规避解决】 frida中console.log打印时参数格式化无效
- 【未解决】 Frida中js的console.log日志打印格式化参数
- 【已解决】 frida的脚本中console.log打印args报错： RangeError invalid array index
- 【已解决】 给Frida的Stalker中输出log日志到文件
- 【已解决】 Frida的Stalker中优化hook到指令时的log日志输出
- 【已解决】 js中console.log如何打印对象[object Object]
- 【已解决】 Frida启动js脚本报错： Error could not parse line 1 expecting ,
- 【已解决】 Frida的Stalker中调试报错： Fatal Python error \_enter\_buffered\_busy: could not acquire lock
- 【未解决】 frida报错： Failed to spawn: the connection is closed
- 【已解决】 frida启动抖音app报错： Failed to attach need Gadget to attach on jailed iOS
- 【已解决】 frida运行报错： Waiting for USB device to appear
- 【未解决】 Mac中用Frida但报错Failed to enumerate processes the connection is closed且导致iPhone重启
- 【未解决】 frida-server运行报错： Failed to spawn this feature requires an iOS Developer Disk Image to be mounted
- 【已解决】 frida去hook函数报错： TypeError cannot read property implementation of undefined
- 【未解决】 frida中hook函数打印参数值时最后app崩溃frida输出Process terminated
- 【未解决】 frida去hook监控iOS的ObjC函数时经常会崩溃Process terminated
- 【未解决】 研究frida崩溃Process terminated： 通过Preferences的崩溃日志找原因
- 【记录】 Frida调试导致iPhone8重启而丢失palera1n越狱
- 【未解决】 Mac中Frida报错且导致iPhone重启： 重新修复XinaA15越狱环境
- 【未解决】 frida导致iPhone重启： 从崩溃日志ips文件分析去找可能原因
- 【未解决】 Mac中用Frida但报错Failed to enumerate processes the connection is closed且导致iPhone重启
- 【未解决】 Frida调试Apple Store报错： Failed to attach missing gProcessInfo
- 【已解决】 frida-ios-dump给iOS 15.1的iPhone中app砸壳报错： missing gProcessInfo
- 【已解决】 研究frida中是否存在导入外部变量gProcessInfo
- 【未解决】 研究Frida中Missing gProcessInfo出错的逻辑和原因
- 【已解决】 研究二进制/usr/lib/dyld中是否包含或导出变量\_gProcessInfo
- 【记录】 dyld源码中的gProcessInfo
- 【已解决】 Frida源码中找到了： gProcessInfo
- 【未解决】 找Frida中iOS的arm6e4： 从Frida源码和build中找
- 【未解决】 找Frida中iOS的arm6e4： 从Frida的github中找
- 【未解决】 找Frida中iOS的arm6e4： 自己给make加echo打印日志调试
- 【未解决】 找Frida中iOS的arm6e4： 从github的ci的workflow中找arm64e
- 【未解决】 找Frida中iOS的arm6e4： 从编译日志中的Downloading ios-arm64入手

- 【未解决】找Frida中iOS的arm6e4: make时如何传入arm64e的arch参数
- 【未解决】找Frida中iOS的arm6e4: 从make编译时的log日志入手
- 【基本解决】iOS 15.1的iPhone11中frida-server所用架构是arm64e还是arm64
- 【未解决】自己编译出的arm64的frida-server能否在iPhone11正常运行
- 【未解决】用frida源码自己编译出frida的iOS的包含frida-server的deb安装包
- 【未解决】自己编译出包含arm64和arm64e的FAT格式的frida-server二进制
- 【未解决】自己修改编译frida-core源码以尝试解决Frida的Missing gProcessInfo问题
- 【未解决】Frida中如何编译出iOS的arm64e的frida-server二进制
- 【未解决】自己编译Frida的frida-core代码生成可用二进制frida-server
- 【已解决】iOS逆向: 如何写Frida的Stalker代码去监控函数\_\_lldb\_unnamed\_symbol2575\$\$akd的指令运行
- 【未解决】iOS逆向: 如何反代码混淆反混淆去混淆
- 【未解决】Mac中Frida启动抖音app进程并调试和hook函数
- 【未解决】用Frida的frida-trace去hook函数iOS版抖音
- 【未解决】frida调试抖音app去hook函数: \_dyld\_get\_image\_name
- 【未解决】frida去hook函数\_dyld\_get\_image\_name时打印参数为字符串
- 【未解决】用Frida动态调试iOS版抖音app
- 【未解决】Mac中用Frida调试iOS版抖音
- 【已解决】用frida启动hook调试iOS抖音app
- 【未解决】尝试Frida的stalker能否修复抖音AwemeCore中函数名常量字符串
- 【记录】iOS逆向Apple账号: 用frida和frida-trace去hook打印更多账号相关函数调用
- 【无法解决】iOS逆向Apple账号: 用frida的ssl bypass脚本尝试解决Charles抓包代理报错
- 【未解决】iOS逆向Apple账号: 用Frida去监控NSURL去调试Apple账号登录过程
- 【未解决】iOS逆向Apple账号: 用Frida去调试NSURL核心网络请求函数调用
- 【未解决】iOS逆向Apple账号: 分析研究frida抓包到的Apple账号登录过程和网络相关的内容
- 【记录】用frida动态调试重新打包后的安卓迅雷apk
- 【未解决】Mac中搭建Frida的动态调试安卓apk的开发环境
- 【已解决】iPhone和Mac中升级Frida到最新版16.1.1
- 【已解决】iOS逆向WhatsApp: Frida的js的函数堆栈打印优化: 给个别特定函数加到排除列表
- 【已解决】iOS逆向WhatsApp: 用Frida的带函数堆栈调用的js去hook注册过程
- 【已解决】js中判断字符串是否在列表中
- 【已解决】Frida中js打印日志log优化: 自动生成中间字符串加上左右对称的单行log日志
- 【已解决】Frida打印iOS函数调用堆栈: 优化同一函数只输出一次
- 【已解决】iOS逆向WhatsApp: 加了代理后Frida去hook却始终崩溃
- 【已解决】Frida去hook抖音报错: Error java.lang.ClassNotFoundException Didn't find class on path DexPathList zip file
- 【已解决】Frida去hook安卓类找不到: Error java.lang.ClassNotFoundException Didn't find class com.android.server.pm.Settings
- 
- frida-ios-dump · iOS逆向开发: 砸壳ipa (crifan.org)
- 
- 某金融app的加解密hook+rpc+绕过SSLPinning抓包 - T00ls.Com
- 【保姆级教学】某金融app FRIDA hook加解密算法+jsrpc=乱杀 - T00ls.Com
- 利用frida快速解决APP中某tong加密 | 大熊の笔记
- frida跑加密算法和绕过抓包验证\_不想写程序(╥^\_^╥)的博客-CSDN博客
- 使用Frida绕过iOS反调试 | La0s

- [一例简单的frida反调试绕过 - 网安](#)
- [【iOS逆向】某营业厅算法分析\\_小陈\\_InfoQ写作社区](#)
- [\[原创\] sktrace：基于 Frida Stalker 的 trace 工具-Android安全-看雪-安全社区|安全招聘|kanxue.com](#)
- [oleavr \(Ole André Vadla Ravnås\)](#)
- [Frida 12.3 Debuts New Crash Reporting Feature - NowSecure](#)
- [【iOS逆向与安全】frida-trace入门 - 移动端小陈 - 博客园](#)
- [Frida HandBook](#)
- [regex - How do you access the matched groups in a JavaScript regular expression? - Stack Overflow](#)
- [NativePointer - JavaScript API](#)
- [frida-ios-hook/hook-all-methods-of-specific-class.js at master · noobpk/frida-ios-hook · GitHub](#)
- [mikeash.com: objc\\_msgSend's New Prototype](#)
- [Frida basics - Frida HandBook](#)
- [iOS security overview & reverse engineering tools / Habr](#)
- [objc\\_msgSend | Apple Developer Documentation](#)
- [Frida CodeShare](#)
- [Project: iOS Utils](#)
- [Frida CodeShare BROWSE CODE](#)
- [Project: iOS SSL Bypass](#)
- [personal\\_script/Frida\\_script at master · lich4/personal\\_script · GitHub](#)
- [Frida在iOS平台进行OC函数hook的常用方法 | 8Biiit's Blog](#)
- [frida-ios-hook/frida-scripts](#)
- [0xdeaf/frida-scripts](#)
- [Beginning Frida: by example. Frida, <https://frida.re/>, is one of... | by Román Ramírez | Medium](#)
- [Hacking | Frida • A world-class dynamic instrumentation toolkit](#)
- [Frida 1.6.3 Released | Frida • A world-class dynamic instrumentation toolkit](#)
- [Frida 1.6.2 Released | Frida • A world-class dynamic instrumentation toolkit](#)
- [frida/frida: Clone this repo to build Frida](#)
- [frida-core](#)
- [frida-gum](#)
- [frida-tools](#)
- [frida-python](#)
- [frida-node](#)
- [frida-swift](#)
- [frida-clr](#)
- [frida-go](#)
- [frida-rust](#)
- [frida-qml](#)
- [frida-objc-bridge](#)
- [frida-java-bridge](#)
- [Capstone](#)
- [Frida Docs](#)
- [Quick-start guide](#)
- [Installation](#)
- [Modes of Operation](#)
- [Gadget](#)
- [Hacking](#)

- [Stalker](#)
- [Presentations](#)
- [Functions](#)
- [Messages](#)
- [iOS](#)
- [Android](#)
- [Windows](#)
- [macOS](#)
- [Linux](#)
- [iOS](#)
- [Android](#)
- [JavaScript](#)
- [Frida CLI](#)
- [frida-ps](#)
- [frida-trace](#)
- [frida-discover](#)
- [frida-ls-devices](#)
- [frida-kill](#)
- [gum-graft](#)
- [JavaScript API](#)
- [C API](#)
- [Swift API](#)
- [Go API](#)
- [Best Practices](#)
- [Troubleshooting](#)
- [Building](#)
- [Footprint](#)
- [GSoC Ideas 2015](#)
- [GSoD Ideas 2023](#)
- [History](#)
- [frida-presentations](#)
- [【Frida 实战】API查找器和拦截器的组合使用 – exchen's blog](#)
- [Stalker的API](#)
- [DefinitelyTyped/index.d.ts](#)
- [Tampering and Reverse Engineering on iOS - OWASP MASTG \(gitbook.io\)](#)
- 

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：  
2024-07-21 16:14:18