目录

iOS逆向开发：越狱检测和反越狱检测

- 最新版本：`v1.0`
- 更新时间：`20221110`

简介

介绍iOS逆向期间涉及到的iOS的越狱检测和反越狱检测方面的内容。包括常用的越狱检测手段，比如URL Scheme、文件方面的：文件属性、文件打开和文件写入；其中文件打开又包括C函数和iOS函数，其中C函数又包括syscall的C函数和svc 0x80内联汇编；以及文件写入包括C函数和iOS函数；以及其他检测手段：环境变量、是否可调试、system、沙箱完整性校验、越狱相关进程、dyld动态库，包括dylib的dladdr、dlopen、dlsym和_dyld开头的系列函数、ObjC运行时、app本身、已安装app、SSH相关、getsectiondata等；以及上述各种方法的反越狱检测的具体实现，以及其他一些额外的反越狱检测手段，比如内核级反越狱；接着整理越狱检测和反越狱检测的通用的内容，比如app的启动过程、越狱路径相关，尤其是越狱路径列表。以及其他一些心得。且均已贴出相关的越狱检测和反越狱检测的代码段和独立的代码仓库。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

HonKit源码

- crifan/ios_re_jb_detection: iOS逆向开发：越狱检测和反越狱检测

如何使用此HonKit源码去生成发布为电子书

详见：crifan/honkit_template: demo how to use crifan honkit template and demo

在线浏览

- iOS逆向开发：越狱检测和反越狱检测 book.crifan.org
- iOS逆向开发：越狱检测和反越狱检测 crifan.github.io

离线下载阅读

- iOS逆向开发：越狱检测和反越狱检测 PDF
- iOS逆向开发：越狱检测和反越狱检测 ePub
- iOS逆向开发：越狱检测和反越狱检测 Mobi

版权和用途说明

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

iOS越狱检测概览

TODO：

- 【未解决】越狱iOS如何实现反越狱检测
- 【整理】越狱检测和反越狱检测相关手段及进度

---

iOS逆向的所涉及的内容和领域，其中就有：

- `防` 的 `iOS越狱检测`：检测iOS设备（iPhone等）是否越狱
  - 如果发现越狱，则轻则提示和警告，重则禁用部分功能，甚至完全不可用
- `攻` 的 `iOS反越狱检测`
  - 想办法绕过越狱检测，从而对应目的，比如实现技术上的攻防研究、甚至是刷机改机等手段去赚黑灰产的钱。

iOS越狱检测

TODO：

- 【已解决】Mac中如何判断iPhone手机中iOS系统已越狱
- 【已解决】iOS中被测app实现越狱检测

现已公开发布源码：

crifan/iOSJailbreakDetection: iOS的ObjC的app，实现iOS越狱检测

本教程后续章节中贴出的iOS越狱检测的代码片段，均摘录自其中。

URL Scheme

TODO：

- 【已解决】iOS的canOpenURL报错：failed for URL OSStatus error -10814
- 【已解决】iOS越狱检测：cydia://开头的URL scheme
- 【已解决】iOS中canOpenURL测试普通可以打开的app的url scheme
- 【已解决】iOS的canOpenURL报错：This app is not allowed to query for scheme weixin
-

```
- (IBAction)detectCydiaBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"Clicked detect cydia://");
    BOOL canOpen = FALSE;

//    NSString *fakeCydiaStr = @"cydia://package/com.fake.packagename";
//    NSString *fakeCydiaStr = @"CYDIA://package/com.fake.packagename";
//    NSString *fakeCydiaStr = @"Cydia://package/xxx";
//    NSString *openPrefAbout = @"Prefs:root=General&path=About";
//    NSString *openPrefAbout = @"prefs:root=General&path=About";

    NSString *curToOpenStr = NULL;

//    curToOpenStr = @"weixin://";
    curToOpenStr = @"cydia://";

    NSURL *curToOpenUrl = [NSURL URLWithString:curToOpenStr];
    canOpen = [[UIApplication sharedApplication] canOpenURL:curToOpenUrl];
    NSString *canOpenStr = canOpen ? @"可以打开" : @"无法打开";
    NSString *conclusionStr = canOpen ? @"可能是越狱手机" : @"很可能不是越狱手机";
    NSString *resultStr = [NSString stringWithFormat:@"%@: %@\n-> %@", canOpenStr, curToOpenUrl,
 conclusionStr];
    NSLog(@"resultStr=%@", resultStr);
    _detectResultTv.text = resultStr;
}
```

文件

iOS越狱检测手段中，常见的一种方式，是基于文件的检测：

- 基于 `文件属性` 的
  - 举例
    - `/etc/fstab` 的大小
- 基于 `文件打开` 的
  - 逻辑：能否打开某个（越狱后才会存在的）文件
    - 如果能打开，说明该文件存在，说明是越狱手机
  - 函数类型
    - 基于 `C` 语言的（系列）函数
      - `open` 、 `stat` 、 `access` 等
        - 引申：同样的函数，可以换成更高级的调用方式
          - 通过 `syscall` ，传递不同的 `number` 编号，实现对应函数的调用
            - 更高级的：把 `syscall` 的调用，改为通过svc 0x80的内联汇编，增加被hook拦截的难度
    - 基于 `iOS` 语言的
      - `NSFileManager` 、 `NSURL` 等
- 基于 `文件写入` 的
  - 举例
    - 向 `/private` 中尝试能否写入

文件属性

TODO：

- 【未解决】iOS反越狱检测：/etc/fstab大小是否异常

文件打开

TODO：

- 【记录】研究越狱iPhone中有哪些lib库
- 【记录】整理和对比不同iPhone的dyld输出的动态库
- 【已解决】越狱iOS中/bin/sh和/bin/bash以及/etc/alternatives/sh关系

C函数

TODO:

- 【未解决】iOS越狱检测之打开文件方式

open系列

TODO:

【未解决】iOS越狱检测之打开文件：open系列函数

open

TODO:

- 【已解决】iOS越狱检测：iOS的app中用open打开文件
- 【基本解决】iOS越狱检测之打开文件：正向调用fopen

```
        isUseFd = TRUE;
        retFd = open(filePathStr, O_RDONLY);
```

opendir

TODO:

【已解决】iOS越狱检测之打开文件之底层C函数：opendir

```
    } else if (FUNC_OPENDIR == funcType) {
        DIR* retDir = opendir(filePathStr);
        if (NULL != retDir){
            NSLog(@"opendir OK: filePathStr=%s -> retDir=%p", filePathStr, retDir);
            NSLog(@"\tDIR: __dd_fd=%d,__dd_loc=%ld,__dd_size=%ld,__dd_buf=%s,__dd_len=%d,__dd_s
eek=%ld,__padding=%ld,__dd_flags=%d",
                retDir->__dd_fd, retDir->__dd_loc, retDir->__dd_size, retDir->__dd_buf, retDir->
__dd_len, retDir->__dd_seek, retDir->__padding, retDir->__dd_flags);
            isOpenOk = TRUE;
        } else {
            NSLog(@"opendir fail for filePathStr=%s", filePathStr);
            isOpenOk = FALSE;
        }

        NSLog(@"opendir filePathStr=%s -> retDir=%p -> isOpenOk=%s", filePathStr, retDir, boolT
oStr(isOpenOk));
```

__opendir2

```
    } else if (FUNC___OPENDIR2 == funcType) {
        DIR* retDir = __opendir2(filePathStr, DTF_HIDEW DTF_NODUP);
        if (NULL != retDir){
            NSLog(@"__opendir2 OK: filePathStr=%s -> retDir=%p", filePathStr, retDir);
            NSLog(@"\tDIR: __dd_fd=%d,__dd_loc=%ld,__dd_size=%ld,__dd_buf=%s,__dd_len=%d,__dd_s
eek=%ld,__padding=%ld,__dd_flags=%d",
                retDir->__dd_fd, retDir->__dd_loc, retDir->__dd_size, retDir->__dd_buf, retDir->
__dd_len, retDir->__dd_seek, retDir->__padding, retDir->__dd_flags);
            isOpenOk = TRUE;
        } else {
            NSLog(@"__opendir2 fail for filePathStr=%s", filePathStr);
            isOpenOk = FALSE;
        }

        NSLog(@"__opendir2 filePathStr=%s -> retDir=%p -> isOpenOk=%s", filePathStr, retDir, bo
olToStr(isOpenOk));
    }
```

access系列

access

TODO:

- 【已解决】iOS越狱检测之打开文件：access

```
    } else if (FUNC_ACCESS == funcType) {
        int retValue = access(filePathStr, F_OK);
        NSLog(@"access %s -> %d", filePathStr, retValue);

        if (retValue != ACCESS_OK){
            isOpenOk = FALSE;
        } else {
            isOpenOk = TRUE;
        }
```

faccessat

TODO:

- 【已解决】iOS越狱检测之打开文件：faccessat
- 【已解决】iOS中越狱检测之打开文件：faccessat正向检测

```
    } else if (FUNC_FACCESSAT == funcType) {
        int curDirFd = 0;
        int retValue = ACCESS_FAILED;

//        // 1. test relative path
////        const char* curDir = "/private/var/mobile/Library/Filza/";
////        const char* curFile = "scripts/README.url";
```

```
//
////        const char* curDir = "/private/var/mobile/Library/";
////        const char* curFile = "Filza/scripts/README.url";
////        const char* curDir = "/private/./var/../var/mobile/Library/./";
////        const char* curFile = "Filza/./scripts/../scripts/README.url";
//        const char* curDir = "/usr/lib";
//        const char* curFile = "libsubstrate.dylib";
//
//        curDirFd = open(curDir, O_RDONLY);
//        NSLog(@"curDir=%s -> curDirFd=%d", curDir, curDirFd);
//
////        // for debug: get file path from fd
////        char filePath[PATH_MAX];
////        int fcntlRet = fcntl(curDirFd, F_GETPATH, filePath);
////        const int FCNTL_FAILED = -1;
////        if (fcntlRet != FCNTL_FAILED){
////            NSLog(@"fcntl OK: curDirFd=%d -> filePath=%s", curDirFd, filePath);
////        } else {
////            NSLog(@"fcntl fail for curDirFd=%d", curDirFd);
////        }
//
//        retValue = faccessat(curDirFd, curFile, F_OK, AT_EACCESS);
//        NSLog(@"faccessat curDir=%s,curFile=%s -> %d", curDir, curFile, retValue);

        // 2. test input path
        const int FAKE_FD = 0;
        curDirFd = FAKE_FD;
        retValue = faccessat(curDirFd, filePathStr, F_OK, AT_EACCESS);
        NSLog(@"faccessat curDirFd=%d, filePathStr=%s -> %d", curDirFd, filePathStr, retValue);

        if (retValue != ACCESS_FAILED){
            isOpenOk = TRUE;
        } else {
            isOpenOk = FALSE;
        }
```

stat系列

TODO：

- 【未解决】iOS越狱检测之打开文件之stat系列函数

stat

TODO：

- 【已解决】iOS越狱检测之打开文件：stat函数
- 【已解决】iOS用stat打开和检测文件是否存在检测是否越狱

```
    if (FUNC_STAT == funcType){
        isUseStatInfo = TRUE;
        openResult = stat(filePathStr, &stat_info);
```

lstat

TODO：

- 【已解决】iOS越狱检测之打开文件：lstat正向越狱检测
- 【已解决】lstat检测普通文件但却通过S_IFLNK误判出是软链接

```
    } else if (FUNC_LSTAT == funcType) {
        isOpenOk = FALSE;
        bool isLink = FALSE;

        struct stat statInfo;
        int lstatRet = lstat(filePathStr, &statInfo);
        if (STAT_OK == lstatRet){
//            isLink = statInfo.st_mode & S_IFLNK;
            isLink = S_ISLNK(statInfo.st_mode);
            if (isLink) {
                isOpenOk = TRUE;
            }
        }

        NSLog(@"lstat filePathStr=%s -> isLink=%s -> isOpenOk=%s", filePathStr, boolToStr(isLink
), boolToStr(isOpenOk));
```

fstat

```
    } else if (FUNC_FSTAT == funcType) {
        isOpenOk = FALSE;
        int tmpFd = open(filePathStr, O_RDONLY);

        if (tmpFd > 0){
            isOpenOk = TRUE;

            struct stat statInfo;
            memset(&statInfo, 0, sizeof(struct stat));
            int fstatRet = fstat(tmpFd, &statInfo);
            if (STAT_OK == fstatRet) {
                isOpenOk = TRUE;
            } else {
                isOpenOk = FALSE;
            }
        } else {
            // when fd < 0, normally is -1, means open file failed
            isOpenOk = FALSE;
            NSLog(@"open() failed for %@", filePath);
        }
```

fstatfs

```
    } else if (FUNC_FSTATFS == funcType) {
        isOpenOk = FALSE;
```

```
        int tmpFd = open(filePathStr, O_RDONLY);

        if (tmpFd > 0){
            isOpenOk = TRUE;

            struct statfs statfsInfo;
            memset(&statfsInfo, 0, sizeof(struct statfs));
            int fstatfsRet = fstatfs(tmpFd, &statfsInfo);
            if (STATFS_OK == fstatfsRet) {
                isOpenOk = TRUE;
            } else {
                isOpenOk = FALSE;
            }
        } else {
            // when fd < 0, normally is -1, means open file failed
            isOpenOk = FALSE;
            NSLog(@"open() failed for %@", filePath);
        }
```

realpath

```
    } else if (FUNC_REALPATH == funcType) {
        char parsedRealPath[PATH_MAX];
        char *resolvedPtr = realpath(filePathStr, parsedRealPath);
        if (NULL != resolvedPtr){
            NSLog(@"realpath OK: filePathStr=%s -> parsedRealPath=%s", filePathStr, parsedRealP
ath);
            isOpenOk = TRUE;
        } else {
            NSLog(@"realpath fail for filePathStr=%s", filePathStr);
            isOpenOk = FALSE;
        }
        NSLog(@"realpath filePathStr=%s -> isOpenOk=%s", filePathStr, boolToStr(isOpenOk));
```

syscall

TODO：

- 【已解决】iOS的app中如何实现syscall函数调用
- 【已解决】iOS中用syscall调用stat64实现越狱文件检测
- 【整理】syscall内核系统调用和svc 0x80相关基础知识
  - 【整理】iOS中syscall的系统调用编号number的定义
  - 【整理】iOS中的带函数原型和说明的系统调用system call

SYS_fork

```
NSString * parseForkResult(int forkRetPid){
    NSString *forkResultStr = NULL;
    if (forkRetPid < 0){
        forkResultStr = @"无法fork->旧版iOS:非越狱, 新版iOS:无法判断";

        // log print erro info
        NSLog(@"errno=%d\n", errno);
        char *errMsg = strerror(errno);
        NSLog(@"errMsg=%s\n", errMsg);
    } else{
        forkResultStr = @"可以fork -> 旧版iOS: 越狱手机";
    }

    return forkResultStr;
}

- (IBAction)syscallForkBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"syscall(fork) check");
    int retPid = syscall(SYS_fork);

    NSString *forkResultStr = parseForkResult(retPid);
    NSLog(@"syscall(fork) return retPid=%d, forkResultStr=%@", retPid, forkResultStr);
    _detectResultTv.text = [NSString stringWithFormat:@"%@ -> %@", @"syscall(fork)", forkResult
Str];
}
```

SYS_stat

```
    } else if (FUNC_SYSCALL_STAT == funcType) {
        isUseStatInfo = TRUE;
        //Note: for open normal file, return 0 is OK, but st_mode is abnormal !
        openResult = syscall(SYS_stat, filePathStr, &stat_info);
```

SYS_stat64

```
    } else if (FUNC_SYSCALL_STAT64 == funcType){
        isUseStatInfo = TRUE;
        openResult = syscall(SYS_stat64, filePathStr, &stat_info);
```

SYS_lstat

```
    } else if (FUNC_SYSCALL_LSTAT == funcType){
        isUseStatInfo = TRUE;
        openResult = syscall(SYS_lstat, filePathStr, &stat_info);
```

SYS_fstat

```
    } else if (FUNC_SYSCALL_FSTAT == funcType){
        isUseStatInfo = TRUE;

        int curFd = open(filePathStr, O_RDONLY);
        if (curFd > 0){
            openResult = syscall(SYS_fstat, curFd, &stat_info);
        } else {
            isOpenOk = FALSE;
        }
```

SYS_fstatat

```
    } else if (FUNC_SYSCALL_FSTATAT == funcType){
        // NOTE: syscall(SYS_fstatat) not work until 20220316 -> awalys return -1

        // int fstatat(int fd, const char* pathname, struct stat* buf, int flags);
        isUseStatInfo = TRUE;

//        int curFd = open(filePathStr, O_RDONLY);
//        if (curFd > 0){
//            openResult = syscall(SYS_fstatat, curFd, filePathStr, &stat_info, F_DUPFD);
//        } else {
//            isOpenOk = FALSE;
//        }

//        int notUsedDirfd = -1;
//        openResult = syscall(SYS_fstatat, notUsedDirfd, filePathStr, &stat_info, F_DUPFD);
//        openResult = syscall(SYS_fstatat, notUsedDirfd, filePathStr, &stat_info, 0);
        openResult = syscall(SYS_fstatat, AT_FDCWD, filePathStr, &stat_info, 0);
```

SYS_statfs

```
    } else if (FUNC_SYSCALL_STATFS == funcType){
        isUseStatInfo = TRUE;
        // int statfs(const char *path, struct statfs *buf);
        openResult = syscall(SYS_statfs, filePathStr, &stat_info);
```

SYS_fstatfs

```
    } else if (FUNC_SYSCALL_FSTATFS == funcType){
        isUseStatInfo = TRUE;
        // int fstatfs(int fd, struct statfs *buf);
        int curFd = open(filePathStr, O_RDONLY);
        if (curFd > 0){
            openResult = syscall(SYS_fstatfs, curFd, &stat_info);
        } else {
            isOpenOk = FALSE;
        }
    }
```

SYS_open

```
    } else if (FUNC_SYSCALL_OPEN == funcType){
        isUseFd = TRUE;
//        retFd = syscall(SYS_open, filePathStr, O_RDONLY);
        retFd = syscall(SYS_open, filePathStr, O_RDONLY, MODE_NONE);
    }
```

SYS_access

```
    } else if (FUNC_SYSCALL_ACCESS == funcType) {
        int retValue = syscall(SYS_access, filePathStr, F_OK);
        NSLog(@"SYS_access %s -> %d", filePathStr, retValue);
        if (retValue != ACCESS_OK){
            isOpenOk = FALSE;
        } else {
            isOpenOk = TRUE;
        }
    }
```

SYS_faccessat

```
    } else if (FUNC_SYSCALL_FACCESSAT == funcType) {
        // NOTE: syscall(SYS_faccessat) not work until 20220317 -> awalys return -1

        int curDirFd = 0;
        int retValue = ACCESS_FAILED;

        const int FAKE_FD = 0;
```

```
        curDirFd = FAKE_FD;
//        retValue = syscall(SYS_faccessat, curDirFd, filePathStr, F_OK, AT_EACCESS);
        retValue = syscall(SYS_faccessat, curDirFd, filePathStr, F_OK, 0);
        if (retValue != ACCESS_FAILED){
            isOpenOk = TRUE;
        } else {
            isOpenOk = FALSE;
        }
    }
```

svc 0x80内联汇编

TODO：

- 【整理】syscall内核系统调用和svc 0x80相关基础知识
- 【已解决】iOS正向越狱检测：app中实现svc 0x80实现系统调用
- 【已解决】iOS中优化asm汇编代码新增syscall的number参数
- 【整理】iOS中syscall的系统调用编号number的定义

```
//---------- svc 0x80 define ----------

//#define asm_set_syscall_number(SYSCALL_NUMBER) "mov x16, #SYSCALL_NUMBER\n"
//
//#define asm_svc_0x80_stat64() \
//    "mov x0, %[pathname_p]\n" \
//    "mov x1, %[stat_info_p]\n" \
//    asm_set_syscall_number(SYS_stat64) \
//    "svc #0x80\n" \
//    "mov %[ret_p], x0\n"

//                    "mov x16, #338\n" \

//__attribute__((always_inline)) long svc_0x80_stat_stat64(int syscall_number, const char * pat
hname, struct stat * stat_info) {
//        long ret = 0;
//        long long_syscall_number = syscall_number;
//        __asm__ volatile(
//             "mov x0, %[pathname_p]\n"
//             "mov x1, %[stat_info_p]\n"
//             "mov x16, %[long_syscall_number_p]\n"
//             "svc #0x80\n"
//             "mov %[ret_p], x0\n"
//            : [ret_p]"=r"(ret)
//            : [long_syscall_number_p]"r"(long_syscall_number), [pathname_p]"r"(pathname), [st
at_info_p]"r"(stat_info)
//             : "x0", "x1", "x16"
//        );
//        return ret == 0 ? ret : -1;
//}

__attribute__((always_inline)) int svc_0x80_stat_stat64(int syscall_number, const char * pathna
me, struct stat * stat_info) {
    register const char * x0_pathname asm ("x0") = pathname; // first arg
    register struct stat * x1_stat_info asm ("x1") = stat_info;  // second arg
    register int x16_syscall_number asm ("x16") = syscall_number; // special syscall number sto
re to x16
    register int x4_ret asm("x4") = OPEN_FAILED; // store result
    __asm__ volatile(
         "svc #0x80\n"
         "mov x4, x0\n"
```

```
        : "=r"(x4_ret)
        : "r"(x0_pathname), "r"(x1_stat_info), "r"(x16_syscall_number)
//         : "x0", "x1", "x4", "x16"
    );
    return x4_ret;
}

//__attribute__((always_inline)) int svc_0x80_open(const char * pathname, int flags, mode_t mod
e) {
__attribute__((always_inline)) int svc_0x80_open(const char * pathname, int flags) {
    register const char * x0_pathname asm ("x0") = pathname; // first arg
    register int x1_flags asm ("x1") = flags;  // second arg
//    register unsigned int x2_mode asm ("x2") = (unsigned int)mode;  // third arg
    register int x16_syscall_number asm ("x16") = SYS_open; // special syscall number store to
x16
    register int x4_ret asm("x4") = OPEN_FD_INVALID; // store result
    __asm__ volatile(
//         "mov x16, #5\n" // SYS_open
        "svc #0x80\n"
        "mov x4, x0\n"
        : "=r"(x4_ret)
        : "r"(x0_pathname), "r"(x1_flags), "r"(x16_syscall_number)
// : "r"(x0_pathname), "r"(x1_flags), "r"(x2_mode), "r"(x16_syscall_number)
//         : "x16"
//         : "x0", "x1", "x5", "x16"
    );
    return x4_ret;
}

//---------- svc 0x80 call ----------

...
    } else if (FUNC_SVC_0X80_STAT == funcType) {
        isUseStatInfo = TRUE;
        //Note: for open normal file, return 0 is OK, but st_mode is abnormal !
        openResult = svc_0x80_stat_stat64(SYS_stat, filePathStr, &stat_info);
    } else if (FUNC_SVC_0X80_STAT64 == funcType) {
        isUseStatInfo = TRUE;
        openResult = svc_0x80_stat_stat64(SYS_stat64, filePathStr, &stat_info);
...
    } else if (FUNC_SVC_0X80_OPEN == funcType) {
        isUseFd = TRUE;
//         retFd = svc_0x80_open(filePathStr, O_RDONLY, MODE_NONE);
        retFd = svc_0x80_open(filePathStr, O_RDONLY);
```

iOS函数

TODO：

【已解决】iOS中NSURL的checkResourceIsReachableAndReturnError底层调用lstat返回文件结果异常

NSFileManager

```
    } else if (FUNC_NSFILEMANAGER == funcType) {
        NSFileManager *defaultManager = [NSFileManager defaultManager];
        NSString *filePathNsStr = [NSString stringWithFormat:@"%s", filePathStr];

        BOOL isExisted = [defaultManager fileExistsAtPath: filePathNsStr];
        NSLog(@"isExisted=%s", boolToStr(isExisted));

        BOOL isDir = FALSE;
        BOOL isExistedWithDir = [defaultManager fileExistsAtPath:filePathNsStr isDirectory: &isDir];
        NSLog(@"isExistedWithDir=%s, isDir=%s", boolToStr(isExistedWithDir), boolToStr(isDir));

        isOpenOk = isExisted || isExistedWithDir;

        NSString *curResultStr = @"";

        if(isExisted){
            curResultStr = [NSString stringWithFormat:@"%@ 是否是目录: %@", @"路径存在", isDir ? @"是":@"否"];
        } else{
            curResultStr = [NSString stringWithFormat:@"%@", @"路径不存在"];
        }

        NSLog(@"fileExistsAtPath %@ -> %@", filePathNsStr, curResultStr);
```

NSURL

```
    } else if (FUNC_NSURL == funcType) {
        NSString *fileStr = [NSString stringWithUTF8String:filePathStr];
        NSString *fileWithFilePrefix = [NSString stringWithFormat:@"file://%@", fileStr];
        NSURL *fileUrl = [NSURL URLWithString:fileWithFilePrefix];
        NSError *error = NULL;
        BOOL isReachable = [fileUrl checkResourceIsReachableAndReturnError:&error];
        NSLog(@"isReachable=%s, error=%@", boolToStr(isReachable), (error != NULL) ? error : @"");

        // for debug
        if (isReachable){
            NSLog(@"fileStr=%@", fileStr);
        }
```

```
        isOpenOk = isReachable;
        NSLog(@"NSURL checkResourceIsReachableAndReturnError %@ -> %s", fileStr, boolToStr(isRe
achable));
```

文件写入

TODO：

- 【已解决】iOS越狱检测：尝试向/private写入文件
- 【未解决】iOS越狱检测之是否能写入文件到特定目录

C函数

iOS函数

TODO：

- 【未解决】iOS越狱检测之打开文件之上层iOS层函数
- 【已解决】iOS越狱检测之iOS层函数：尝试向/private写入文件
- 【无法解决】越狱iOS中用writeToFile去写入/private报错：NSCocoaErrorDomain Code 513 You don’t have permission to save the file in the folder private

NSFileManager

TODO：

- 【未解决】iOS越狱检测之打开文件之iOS层函数之NSFileManager

```
- (IBAction)writeFileBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"write file check");

    //    NSStringEncoding strEncoding = NSStringEncodingConversionAllowLossy;
    NSStringEncoding strEncoding = NSUTF8StringEncoding;
    BOOL isAtomicWriteFile = YES;
    BOOL isUseAuxiliaryFile = NO;
    NSDataWritingOptions writeOption = NSDataWritingAtomic;

    NSString *testFile = @"/private/testWriteToFile.txt";
    // for debug
//    NSString *testFile = @"/private/var/mobile/Containers/Data/Application/EEFACEA4-2ADB-4D25-9DB4-B5D643EA8943/Documents/bd.turing/";
//    NSString *testFile = @"/private/var/mobile/Containers/Data/Application/EEFACEA4-2ADB-4D25-9DB4-B5D643EA8943/Documents/test_douyin_write.txt";
    NSString *withPrefixTestFile = [NSString stringWithFormat:@"file://%@", testFile];
    NSURL *testFileUrl = [NSURL URLWithString:withPrefixTestFile];
    NSString *testStr = @"just some test string for test write file";
    NSData *testData = [testStr dataUsingEncoding:strEncoding];

    id objects[] = { @"demo string", @123, @45.67 };
    NSUInteger count = sizeof(objects) / sizeof(id);
    NSArray *testArr = [NSArray arrayWithObjects:objects count:count];

    NSDictionary *testDict = @{
        @"intValue": @123,
        @"floatValue": @45.678,
        @"strValue": @"test write file",
    };

    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSError *error = NULL;
    BOOL isWriteOk = FALSE;
    BOOL isFinalWriteOk = FALSE;
```

```
    // 1. NSString
//    // (1) [NSString writeToFile:atomically:]
//    isWriteOk = [testStr writeToFile:testFile atomically:isAtomicWriteFile];
//    NSLog(@"isWriteOk=%s", boolToStr(isWriteOk));
//    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (2) [NSString writeToFile:atomically:encoding:error:]
    [testStr writeToFile testFile atomically isAtomicWriteFile encoding strEncoding error &error
];
//    [testStr writeToFile:testFile atomically:isAtomicWriteFile encoding:strEncoding error:NUL
L];
    NSLog(@"isWriteOk=%s, error=%@", boolToStr(isWriteOk), error);
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (3) [NSString writeToURL:atomically:]
    isWriteOk = [testStr writeToURL testFileUrl atomically isAtomicWriteFile];
    NSLog(@"isWriteOk=%s", boolToStr(isWriteOk));
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (4) [NSString writeToURL:atomically:encoding:error:]
    isWriteOk = [testFile writeToURL testFileUrl atomically isAtomicWriteFile encoding strEncod
ing error &error];
    NSLog(@"isWriteOk=%s, error=%@", boolToStr(isWriteOk), error);
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // 2. NSData
    // (1) [NSData writeToFile:atomically:]
    isWriteOk = [testData writeToURL testFileUrl atomically isAtomicWriteFile];
    NSLog(@"isWriteOk=%s", boolToStr(isWriteOk));
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (2) [NSData writeToFile:options:error:]
    isWriteOk = [testData writeToFile testFile options writeOption error &error];
    NSLog(@"isWriteOk=%s, error=%@", boolToStr(isWriteOk), error);
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (3) [NSData writeToURL:atomically:]
    isWriteOk = [testData writeToURL testFileUrl atomically isAtomicWriteFile];
    NSLog(@"isWriteOk=%s", boolToStr(isWriteOk));
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (4) [NSData writeToURL:options:error:]
    isWriteOk = [testData writeToURL testFileUrl options writeOption error &error];
    NSLog(@"isWriteOk=%s, error=%@", boolToStr(isWriteOk), error);
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // 3. NSArray
    // (1) [NSArray writeToFile:atomically:]
    isWriteOk = [testArr writeToFile testFile atomically isUseAuxiliaryFile];
    NSLog(@"isWriteOk=%s", boolToStr(isWriteOk));
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (2) [NSArray writeToFile:atomically:]
    isWriteOk = [testArr writeToURL testFileUrl atomically isAtomicWriteFile];
    NSLog(@"isWriteOk=%s", boolToStr(isWriteOk));
```

```
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (3) [NSArray writeToFile:error:]
    isWriteOk = [testArr writeToURL:testFileUrl error:&error];
    NSLog(@"isWriteOk=%s, error=%@", boolToStr(isWriteOk), error);
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // 4. NSDictionary
    // (1) [NSDictionary writeToFile:atomically:]
    isWriteOk = [testDict writeToFile:testFile atomically:isUseAuxiliaryFile];
    NSLog(@"isWriteOk=%s", boolToStr(isWriteOk));
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (2) [NSDictionary writeToURL:error:]
    isWriteOk = [testDict writeToURL:testFileUrl error:&error];
    NSLog(@"isWriteOk=%s, error=%@", boolToStr(isWriteOk), error);
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

    // (3) [NSDictionary writeToURL:atomically:]
    isWriteOk = [testDict writeToURL:testFileUrl atomically:isAtomicWriteFile];
    NSLog(@"isWriteOk=%s", boolToStr(isWriteOk));
    isFinalWriteOk = isWriteOk || isFinalWriteOk;

//    // for debug: test removeItemAtPath
//    isWriteOk = TRUE;

    NSLog(@"isFinalWriteOk=%s", boolToStr(isFinalWriteOk));
    if (isFinalWriteOk)
    {
        NSLog(@"Ok to write file %@", testFile);

        BOOL isDeleteOk = FALSE;

        isDeleteOk = [fileManager removeItemAtPath:testFile error:&error];
        NSLog(@"isDeleteOk=%s, *error=%@", boolToStr(isDeleteOk), error);
//        if(error == nil){
//            isDeleteOk = TRUE;
//        }

        isDeleteOk = [fileManager removeItemAtURL:testFileUrl error:&error];
        NSLog(@"isDeleteOk=%s, *error=%@", boolToStr(isDeleteOk), error);
//        if(error == nil){
//            isDeleteOk = TRUE;
//        }

        if (isDeleteOk){
            NSLog(@"Ok to delete file %@", testFile);
        } else {
            NSLog(@"Fail to delete file %@", testFile);
        }
    } else{
        NSLog(@"Fail to write file %@", testFile);
    }
    NSString* finalResult = @"";
    if (isFinalWriteOk){
        finalResult = @"可以写入 -> 越狱手机";
```

```
    } else {
        finalResult = @"无法写入 -> 很可能是非越狱手机";
    }
    _detectResultTv.text = finalResult;
}
```

环境变量

TODO

- 【已解决】iOS越狱测试：getenv获取其他DYLD的环境变量值
- 【未解决】越狱iPhone中getenv获取DYLD_INSERT_LIBRARIES返回为空
- 【已解决】iOS反越狱检测：dyld的getenv获取环境变量DYLD_INSERT_LIBRARIES

```
- (IBAction)getenvDyInsLibBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"getenv(DYLD_INSERT_LIBRARIES) check");

    char* dyldPrintEnv = getenv("DYLD_PRINT_ENV");
    NSLog(@"dyldPrintEnv=%s", dyldPrintEnv);

    char* insertLibs = getenv("DYLD_INSERT_LIBRARIES");
    NSLog(@"insertLibs=%s", insertLibs);

    const char* dyldEnvList[] = {
        "DYLD_FRAMEWORK_PATH",
        "DYLD_FALLBACK_FRAMEWORK_PATH",
        "DYLD_VERSIONED_FRAMEWORK_PATH",
        "DYLD_LIBRARY_PATH",
        "DYLD_FALLBACK_LIBRARY_PATH",
        "DYLD_VERSIONED_LIBRARY_PATH",
        "DYLD_ROOT_PATH",
        "DYLD_SHARED_REGION",
        "DYLD_INSERT_LIBRARIES",
        "DYLD_FORCE_FLAT_NAMESPACE",
        "DYLD_IMAGE_SUFFIX",
        "DYLD_PRINT_OPTS",
        "DYLD_PRINT_ENV",
        "DYLD_PRINT_LIBRARIES",
        "DYLD_PRINT_LIBRARIES_POST_LAUNCH",
        "DYLD_BIND_AT_LAUNCH",
        "DYLD_NO_FIX_PREBINDING",
        "DYLD_DISABLE_DOFS",
        "DYLD_PRINT_APIS",
        "DYLD_PRINT_BINDINGS",
        "DYLD_PRINT_INITIALIZERS",
        "DYLD_PRINT_REBASINGS",
        "DYLD_PRINT_SEGMENTS",
        "DYLD_PRINT_STATISTICS",
        "DYLD_PRINT_DOFS",
        "DYLD_PRINT_RPATHS",
        "DYLD_SHARED_CACHE_DIR",
        "DYLD_SHARED_CACHE_DONT_VALIDATE",
    };
    const int dyldEnvListLen = sizeof(dyldEnvList)/sizeof(const char *);

    for(int curIdx = 0; curIdx < dyldEnvListLen; curIdx++){
```

```
        const char* curDyldEnv = dyldEnvList[curIdx];
        char* curEnvRet = getenv(curDyldEnv);
        NSLog(@"dyld: [%d] %s -> %s", curIdx, curDyldEnv, curEnvRet);
    }

    NSString* insertLibResultStr = @"";

    if (NULL != insertLibs){
        insertLibResultStr = [NSString stringWithFormat: @"检测出DYLD_INSERT_LIBRARIES -> 越狱手
机; DYLD_INSERT_LIBRARIES=%s", insertLibs];
    } else{
        insertLibResultStr = @"未检测出DYLD_INSERT_LIBRARIES -> 非越狱手机";
    }
    NSLog(@"dyld: insertLibResultStr=%@", insertLibResultStr);

    _detectResultTv.text = insertLibResultStr;
}
```

是否可调试

```
- (IBAction)isDebugableBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"is debugable check");

//    /* tmp to debug getuid */
//    uid_t curUid = getuid();
//    NSLog(@"curUid=%d", curUid);

    int SYSCTL_OK = 0;
    NSString* resultStr = @"";
    BOOL isDebugable = FALSE;

    // Initialize mib, which tells sysctl the info we want, in this case
    // we're looking for information about a specific process ID.
    int name[4];            //里面放字节码。查询的信息
    name[0] = CTL_KERN;     //内核查询
    name[1] = KERN_PROC;    //查询进程
    name[2] = KERN_PROC_PID; //传递的參数是进程的ID
//    name[3] = getpid();      //获取当前进程ID

    int pidToCheck = -1;

    int currentPID = getpid();
    NSLog(@"currentPID=%d", currentPID);
    pidToCheck = currentPID;

//    //for debug
//    int parentPID = getppid();
//    NSLog(@"parentPID=%d", parentPID);
//    pidToCheck = parentPID;

    NSLog(@"pidToCheck=%d", pidToCheck);
    name[3] = pidToCheck;

    // [3]    int    13679

//    size_t infoSize = sizeof(kernelInfoProc);  // 结构体大小
    size_t infoSize = sizeof(struct kinfo_proc);
    struct kinfo_proc kernelInfoProc;  //接受查询结果的结构体
    // Initialize the flags so that, if sysctl fails for some bizarre reason, we get a predicta
ble result.
//    kernelInfoProc.kp_proc.p_flag = 0;
    memset(&kernelInfoProc, 0, infoSize);

    // infoSize    size_t    648
//    int sysctlRet = sysctl(name, 4, &kernelInfoProc, &infoSize, 0, 0);
    int sysctlRet = sysctl(name, 4, &kernelInfoProc, &infoSize, NULL, 0);
    NSLog(@"sysctlRet=%d", sysctlRet);
    if(sysctlRet == SYSCTL_OK){
        int pFlag = kernelInfoProc.kp_proc.p_flag;
        NSLog(@"pFlag=0x%x", pFlag);
```

```
        isDebugable = ((pFlag & P_TRACED) != 0);
        NSLog(@"isDebugable=%s", boolToStr(isDebugable));
        if (isDebugable) {
            resultStr = @"可被调试 -> 越狱手机";
        } else {
            resultStr = @"不可被调试 -> 非越狱手机";
        }
    } else {
        NSLog(@"errno=%d\n", errno);
        char *errMsg = strerror(errno);
        NSLog(@"errMsg=%s\n", errMsg);
        resultStr = [NSString stringWithFormat:@"检测失败: %s", errMsg];
    }
    NSLog(@"resultStr=%@\n", resultStr);
    _detectResultTv.text = resultStr;
}
```

system

TODO：

- 【已解决】iOS越狱检测：system(NULL)
- 【已解决】iOS代码XCode中报错：system is unavailable not available on iOS
- 【已解决】iOS中传入fork调用system的返回值的含义和逻辑

```
- (IBAction)systemBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"system() check");

//    int systemRet = system(NULL);
    const char* command = NULL;
//    command = "ls -lh";
//    command = "fork";
    int systemRet = iOS_system(command);

    const int SYSTEM_RET_SHELL_EXEC_CMD_FAIL = 32512; // == 0x7F00 -> bit 15-8 is 0x7F = 127
    const int SYSTEM_RET_FORK_FAIL = -1;

    NSString* conclusionStr = @"未知结果";
    if (NULL == command){
//        if (0 == systemRet){
        if (systemRet > 0){
            conclusionStr = @"sh存在 -> 越狱手机";
        } else {
            conclusionStr = @"sh不存在 -> 非越狱手机";
        }
    } else {
        if (SYSTEM_RET_SHELL_EXEC_CMD_FAIL == systemRet){
            conclusionStr = @"shell执行命令失败 -> 可能是非越狱手机";
        } else if (SYSTEM_RET_FORK_FAIL == systemRet){
            conclusionStr = @"fork或waitpid失败 -> 可能是非越狱手机";
        } else {
            conclusionStr = [NSString stringWithFormat:@"shell退出状态值为%d -> 无法判断", syste
mRet];
        }
    }
    NSString* systemResultStr = [NSString stringWithFormat:@"system(%s)返回: %d -> %@", command
, systemRet, conclusionStr];
    NSLog(@"%@", systemResultStr);
    _detectResultTv.text = systemResultStr;
}
```

沙箱完整性校验

TODO：

- 【无需解决】已越狱iOS中fork()失败返回-1
- 【已解决】iOS中系统调用fork返回值的含义和逻辑

```
- (IBAction)forkBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"Fork() check");
    // SandBox Integrity Check
    int retPid = fork(); //返回值：子进程返回0，父进程中返回子进程ID，出错则返回-1
    NSString *forkResultStr = parseForkResult(retPid);
    NSLog(@"fork() return retPid=%d, forkResultStr=%@", retPid, forkResultStr);
    _detectResultTv.text = [NSString stringWithFormat:@"%@ -> %@", @"fork()", forkResultStr];
}
```

越狱相关进程

TODO：

- 【未解决】iOS越狱检测：检测是否有越狱相关进程

```
- (IBAction)processCheckBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"process check");
    NSString *resultStr = @"TODO";

    NSArray *processes = [CrifanLibiOS runningProcesses];
    NSLog(@"processes=%@", processes);

    if (NULL == processes) {
        resultStr = @"此检测手段已失效：sysctl(CTL_KERN, KERN_PROC, KERN_PROC_ALL)";
    }

    // proc_listpids(type, typeinfo, buffer, buffersize)
    // type = PROC_ALL_PIDS, typeinfo = 0 (use proc_listallpids)
    // type = PROC_PGRP_ONLY, typeinfo = process group id (use proc_listpgrppids)
    // type = PROC_TTY_ONLY, typeinfo = tty
    // type = PROC_UID_ONLY, typeinfo = uid
    // type = PROC_RUID_ONLY, typeinfo = ruid
    // type = PROC_PPID_ONLY, typeinfo = ppid (use proc_listchildpids)
    // Call with buffer = NULL to return number of pids.
//    int numberOfProcesses = proc_listpids(PROC_ALL_PIDS, 0, NULL, 0);
//    NSLog(@"numberOfProcesses=%d", numberOfProcesses);

    NSLog(@"resultStr=%@", resultStr);
    _detectResultTv.text = resultStr;
}
```

dyld动态库

TODO：

- 抖音
  - 【未解决】iOS中如何检测抖音app当前进程加载或注入了哪些dylib动态库

_dyld系列

TODO：

- 【已解决】iOS越狱检测：辅助用_dyld_get_image_header解析动态库文件信息
- 【已解决】iOS越狱检测：优化dyld的动态库文件和其他越狱文件列表
- 【已解决】iOS越狱检测：用_dyld_image_count() 和 _dyld_get_image_name()检测越狱相关动态库
- 【已解决】iOS正向越狱检测：_dyld_register_func_for_add_image及相关

- `_dyld` 系列 = `_dyld` 开头的一系列函数
  - 最基础也最常用的：`_dyld_image_count` + `_dyld_get_image_name`
  - 很少用到的：`_dyld_get_image_vmaddr_slide`
  - 更高级的：`_dyld_register_func_for_add_image` 和 `_dyld_register_func_for_remove_image`

`_dyld_image_count` + `_dyld_get_image_name`

```
- (void) dbgPrintLibInfo: (int)curImgIdx{
    // debug slide
    intptr_t curSlide = _dyld_get_image_vmaddr_slide(curImgIdx);
    NSLog(@"[%d] curSlide=0x%lx", curImgIdx, curSlide);

    // debug header info
    const struct mach_header* libHeader = _dyld_get_image_header(curImgIdx);
    if (NULL != libHeader){
        int magic = libHeader->magic;
        int cputype = libHeader->cputype;
        int cpusubtype = libHeader->cpusubtype;
        int filetype = libHeader->filetype;
        int ncmds = libHeader->ncmds;
        int sizeofcmds = libHeader->sizeofcmds;
        int flags = libHeader->flags;

        NSLog(@"[%d] magic=0x%x,cputype=0x%x,cpusubtype=0x%x,filetype=%d,ncmds=%d,sizeofcmds=%d
,flags=0x%x",
              curImgIdx,
              magic, cputype, cpusubtype, filetype, ncmds, sizeofcmds, flags);
        // 2021-12-17 09:37:46.814810+0800 ShowSysInfo[11192:1067220] [0] magic=0xfeedfacf,cput
ype=0x100000c,cpusubtype=0x0,filetype=2,ncmds=23,sizeofcmds=3072,flags=0x200085
    } else {
        NSLog(@"[%d] mach_header is NULL", curImgIdx);
    }
}

- (IBAction)dyldImgCntNameBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"_dyld_image_count and _dyld_get_image_name check");

//    //for debug
//    int testImgIdx = 282; // hooked:279 ~ real: 284
```

```
//    [self dbgPrintLibInfo: testImgIdx];

    uint32_t imageCount = _dyld_image_count();
    NSLog(@"dyld: imageCount=%d", imageCount);

    NSMutableArray *loadedDylibList = [NSMutableArray array];

    NSMutableArray *jbDylibList = [NSMutableArray array];

    for (uint32_t i = 0 ; i < imageCount; ++i) {
        const char* curImageName = _dyld_get_image_name(i);

        // for debug
//        bool isNeedDebug = (0 == i) || (1 == i) || (2 == i) || (275 == i);
        bool isNeedDebug = (277 == i) || (278 == i);

        if (NULL != curImageName){
            NSString *curImageNameStr = [[NSString alloc]initWithUTF8String:curImageName];
            NSLog(@"[%d] %@", i, curImageNameStr);

            [loadedDylibList addObject:curImageNameStr];

    //        if([JbPathList.jbDylibList containsObject:curImageNameStr]){
    //        if([JbPathList isJbDylib: curImageNameStr]){

            if(isJailbreakDylib(curImageName)){
                [jbDylibList addObject: curImageNameStr];

                // for debug
                isNeedDebug = true;
            }
        } else {
            NSLog(@"[%d] %s", i, curImageName);
        }

        // for debug
        if (isNeedDebug){
            [self dbgPrintLibInfo: i];
        }
    }

//    NSString *loadedDylibListStr = [CrifanLibiOS nsStrListToStr:loadedDylibList];
    NSString *loadedDylibListStr = [CrifanLibiOS nsStrListToStr loadedDylibList isSortList:TRUE
 isAddIndexPrefix:TRUE];
    NSLog(@"dyld: loadedDylibListStr=%@", loadedDylibListStr);

    NSString *jbLibListStr = [CrifanLibiOS nsStrListToStr jbDylibList isSortList:TRUE isAddInde
xPrefix:TRUE];
    NSLog(@"dyld: jbDylibList=%@", jbLibListStr);

    NSString* dyldLibResultStr = @"";
    if (jbDylibList.count > 0){
        dyldLibResultStr = [NSString stringWithFormat: @"检测出越狱动态库 -> 越狱手机; 越狱动态库
列表:\n%@", jbLibListStr];
    } else{
        dyldLibResultStr = @"未检测出越狱动态库 -> 非越狱手机";
```

```
    }
    NSLog(@"dyld: dyldLibResultStr=%@", dyldLibResultStr);

    _detectResultTv.text = dyldLibResultStr;
}
```

_dyld_register_func_for_add_image

```
static NSString *checkImageResult = @"未发现越狱库 -> 非越狱手机";
NSMutableArray *checkImageFoundJbLibList = NULL;

+ (void)load {
  static dispatch_once_t onceToken;
  dispatch_once(&onceToken, ^{
      checkImageFoundJbLibList = [NSMutableArray array];
      _dyld_register_func_for_add_image(_check_image);
  });
}

static void _check_image(const struct mach_header *header, intptr_t slide) {
    Dl_info info;
    size_t dlInfoSize = sizeof(Dl_info);
    memset(&info, 0, dlInfoSize);

    dladdr(header, &info);
    const char *curImgName = info.dli_fname;
    if(curImgName != NULL) {
        if (isJailbreakDylib(curImgName)) {
            NSString *curImgNameNs = [NSString stringWithUTF8String:curImgName];
            [checkImageFoundJbLibList addObject:curImgNameNs];
            NSString *jbLibListStr = [CrifanLibiOS nsStrListToStr:checkImageFoundJbLibList isSo
rtList:TRUE isAddIndexPrefix:TRUE];
            checkImageResult = [NSString stringWithFormat:@"发现越狱动态库 -> 越狱手机\n%@", jbL
ibListStr];
            NSLog(@"%@", checkImageResult);
            // "Found Jailbreak dylib: /usr/lib/substitute-inserter.dylib -> 越狱手机"
        }
    }
    return;
}

- (IBAction)dyldRegImgBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"dlyd register image add/remove check");
    NSString *resultStr = @"TODO";

    resultStr = checkImageResult;

    NSLog(@"resultStr=%@", resultStr);
    _detectResultTv.text = resultStr;
}
```

2022-10-25 22:54:59

dylib

dladdr

TODO:

- 【部分解决】iOS越狱检测：用dladdr解析函数所属动态库
- 【无需解决】已越狱iOS且已反越狱但dladdr仍是系统库libsystem_kernel.dylib
- 【已解决】iOS反越狱检测：dyld的dladdr

```
- (IBAction)dladdrBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"dladdr check");

    const int DLADDR_FAILED = 0;

    const char* curSystemLib = NULL;
    char* curTestFuncName = NULL;
    Dl_info dylib_info;

    const char* SystemLib_kernel = "/usr/lib/system/libsystem_kernel.dylib";
    curSystemLib = SystemLib_kernel;
    curTestFuncName = "stat";
    int (*func_stat)(const char *, struct stat *) = stat;
    int ret = dladdr(func_stat, &dylib_info);

//    const char* SystemLib_c = "/usr/lib/system/libsystem_c.dylib";
//    curSystemLib = SystemLib_c;
//    curTestFuncName = "fopen";
//    FILE* (*func_fopen)(const char *filename, const char *mode) = fopen;
//    int ret = dladdr(func_fopen, &dylib_info);

    NSLog(@"dladdr ret=%d", ret);

    NSString* dladdrResultStr = @"";
    if (DLADDR_FAILED != ret){
        NSString* conclusionStr = @"";
        const char* libName = dylib_info.dli_fname;
        NSLog(@"dladdr dli_fname=%s, dli_fbase=%p, dli_sname=%s, dli_saddr=%p", libName, dylib_
info.dli_fbase, dylib_info.dli_sname, dylib_info.dli_saddr);

        if (0 == strcmp(libName, curSystemLib)){
            conclusionStr = @"是系统库 -> 非越狱手机";
        } else {
            conclusionStr = @"不是系统库 -> 越狱手机";
        }

        dladdrResultStr = [NSString stringWithFormat @"解析成功 -> %s 所属动态库: %s -> %@", cur
TestFuncName, libName, conclusionStr];
    } else{
```

```
        NSLog(@"dladdr failed: ret=%d", ret);
        dladdrResultStr = [NSString stringWithFormat @"无法解析，返回值=%d", ret];
    }
    NSLog(@"dladdr: %@", dladdrResultStr);

    _detectResultTv.text = dladdrResultStr;
}
```

dlopen+dlsym

TODO：

- 【已解决】iOS越狱检测：正向的dlopen+dlsym
- 【记录】iOS中尝试dlopen和dlsym的system函数传入其他参数确保运行正常
  - 【已解决】iOS中调用system返回值始终是32512
- 【已解决】iOS反越狱检测：dyld的dlopen和dlsym

```
- (IBAction)dlopenDlsymBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"dlopen + dlsym check");

    typedef void (*function_common) (void *para);
//    typedef void (*lib_MSHookFunction)(void *symbol, void *hook, void **old);

    char* dylibPathList[] = {
//        // for debug
//        "/usr/lib/libstdc++.dylib",
//        "/usr/lib/libstdc++.6.dylib",
//        "/usr/lib/libstdc++.6.0.9.dylib",

        // common: tweak plugin libs
        "/usr/lib/libsubstrate.dylib",

        // Cydia Substrate libs
        "/Library/MobileSubstrate/MobileSubstrate.dylib",
        "/usr/lib/substrate/SubstrateInserter.dylib",
        "/usr/lib/substrate/SubstrateLoader.dylib",
        "/usr/lib/substrate/SubstrateBootstrap.dylib",

        // Substitute libs
        "/usr/lib/libsubstitute.dylib",
        "/usr/lib/substitute-inserter.dylib",
        "/usr/lib/substitute-loader.dylib",

        // Other libs
        "/usr/lib/tweakloader.dylib",
    };
    const int StrSize = sizeof(const char *);
    const int DylibLen = sizeof(dylibPathList) / StrSize;

    char* libFuncNameList[] = {
```

```
        "MSGetImageByName",
        "MSFindSymbol",
        "MSHookFunction",
        "MSHookMessageEx",

        "SubGetImageByName",
        "SubFindSymbol",
        "SubHookFunction",
        "SubHookMessageEx",
    };
    const int LibFuncLen = sizeof(libFuncNameList) / StrSize;

//    NSMutableArray *detectedJbDylibList = [NSMutableArray array];
//    NSMutableArray *detectedJbFuncNameList = [NSMutableArray array];
    NSMutableArray *detectedJbLibAndFuncList = [NSMutableArray array];

    for(int libIdx = 0; libIdx < DylibLen; libIdx++) {
        char* curDylib = dylibPathList[libIdx];
        void *curLibHandle = dlopen(curDylib, RTLD_GLOBAL | RTLD_NOW);
        if (NULL == curLibHandle) {
            char* errStr = dlerror();
            NSLog(@"Failed to open dylib %s, error: %s", curDylib, errStr);
        } else {
//            NSString* curDylibNs = [NSString stringWithFormat:@"%s", curDylib];
//            [detectedJbDylibList addObject:curDylibNs];

            for(int funcIdx = 0; funcIdx < LibFuncLen; funcIdx++) {
                char* curFuncName = libFuncNameList[funcIdx];
                function_common funcInLib = dlsym(curLibHandle, curFuncName);
                if (NULL != funcInLib){
                    NSLog(@"Found func %s=%p in dylib %s\n", curFuncName, funcInLib, curDylib);

//                    NSString* curFuncNameNs = [NSString stringWithFormat:@"%s", curFuncName];
//                    [detectedJbFuncNameList addObject:curFuncNameNs];
                    NSString* curLibAndFuncNs = [NSString stringWithFormat @"%s -> %s", curDylib
, curFuncName];
                    [detectedJbLibAndFuncList addObject curLibAndFuncNs];
                }
            }

            dlclose(curLibHandle);
        }
    }

    NSString* finalResult = @"";
//    BOOL isJb = (detectedJbDylibList.count > 0) || (detectedJbFuncNameList.count > 0);
//    NSString *detectedJbDylibListStr = [CrifanLibiOS nsStrListToStr:detectedJbDylibList isSor
tList:FALSE isAddIndexPrefix:TRUE];
//    NSString *detectedJbFuncNameListStr = [CrifanLibiOS nsStrListToStr:detectedJbFuncNameList
 isSortList:FALSE isAddIndexPrefix:TRUE];
//    NSString* detectedLibAndFuncNameStr = [NSString stringWithFormat:@"越狱库=%@\n库函数=%@",
detectedJbDylibListStr, detectedJbFuncNameListStr] ;

    BOOL isJb = (detectedJbLibAndFuncList.count > 0);
    NSString *detectedJbLibAndFuncListStr = [CrifanLibiOS nsStrListToStr detectedJbLibAndFuncLi
st isSortList FALSE isAddIndexPrefix TRUE];
```

```
    NSString *detectedLibAndFuncNameStr = [NSString stringWithFormat:@"越狱库和库函数=%@", detec
tedJbLibAndFuncListStr];

    if (isJb){
        finalResult = [NSString stringWithFormat:@"检测出越狱库或库函数 -> 越狱手机\n%@", detecte
dLibAndFuncNameStr] ;
    } else {
        finalResult = @"未检测出越狱库和库函数 -> 非越狱手机";
    }
    NSLog(@"finalResult=%@", finalResult);
    _detectResultTv.text = finalResult;
}
```

ObjC运行时

TODO：

- 【已解决】iOS越狱检测：objc_copyImageNames检测image
- 【无法解决】iOS越狱检测和反越狱检测：objc_getClass
- 【无需解决】iOS越狱检测和反越狱检测：NSClassFromString

---

objc_copyImageNames

```
- (IBAction)objCopyBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"objc_copyImageNames check");
    unsigned int outImageCount = 0;
    const char **imageList = objc_copyImageNames(&outImageCount);
    NSLog(@"outImageCount=%d, imageList=%p", outImageCount, imageList);

    NSMutableArray *jbImageList = [NSMutableArray array];

    if ((outImageCount > 0) && (imageList != NULL)) {
        for (int i = 0; i < outImageCount; i++) {
            const char* curImagePath = imageList[i];
            bool isJbPath = isJailbreakPath(curImagePath);
            NSLog(@"[%d] %s -> isJbPath=%s", i, curImagePath, boolToStr(isJbPath));
            if (isJbPath) {
                NSString* curImagePathNs = [NSString stringWithFormat:@"%s", curImagePath];
                [jbImageList addObject:curImagePathNs];
            }
        }
    }

    NSString *jbImageListStr = [CrifanLibiOS nsStrListToStr:jbImageList isSortList:TRUE isAddIn
dexPrefix:TRUE];
    NSLog(@"jbImageListStr=%@", jbImageListStr);

    NSString* resultStr = @"";
    if (jbImageList.count > 0) {
        resultStr = [NSString stringWithFormat:@"检测出越狱库image -> 越狱手机\n%@", jbImageListS
tr] ;
    } else {
        resultStr = @"未检测出越狱库image -> 非越狱手机";
    }
    NSLog(@"resultStr=%@", resultStr);
    _detectResultTv.text = resultStr;
}
```

2022-10-25 22:26:40

app本身

重签名

TODO：

- 【已解决】iOS防护：签名校验重签名检测

```
- (IBAction)reCodeSignBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"re-CodeSign check");
    NSString *resultStr = @"TODO";

//    NSString *embeddedPath = [[NSBundle mainBundle] pathForResource:@"embedded" ofType:@"mobileprovision"]; // embeddedPath    __NSCFString *    @"/private/var/containers/Bundle/Application/4366136E-242E-4C5D-9CC8-CF100A0B6FB2/ShowSysInfo.app/embedded.mobileprovision"    0x0000000282c11830
//    if (![[NSFileManager defaultManager] fileExistsAtPath:embeddedPath]) {
//        return;
//    }

//    // 读取application-identifier  注意描述文件的编码要使用:NSASCIIStringEncoding
//    NSStringEncoding fileEncoding = NSASCIIStringEncoding;
////    NSStringEncoding fileEncoding = NSUTF8StringEncoding;
//    NSString *embeddedProvisioning = [NSString stringWithContentsOfFile:embeddedPath encoding:fileEncoding error:nil];
//    NSArray<NSString *> *embeddedProvisioningLines = [embeddedProvisioning componentsSeparatedByCharactersInSet:[NSCharacterSet newlineCharacterSet]];
//    for (int i = 0; i < embeddedProvisioningLines.count; i++) {
//        if ([embeddedProvisioningLines[i] rangeOfString:@"application-identifier"].location != NSNotFound) {
//            NSString *identifierString = embeddedProvisioningLines[i + 1]; // 类似: <string>L2ZY2L7GYS.com.xx.xxx</string>
//            NSRange fromRange = [identifierString rangeOfString:@"<string>"];
//            NSInteger fromPosition = fromRange.location + fromRange.length;
//            NSInteger toPosition = [identifierString rangeOfString:@"</string>"].location;
//            NSRange range;
//            range.location = fromPosition;
//            range.length = toPosition - fromPosition;
//            NSString *fullIdentifier = [identifierString substringWithRange:range];
//            NSScanner *scanner = [NSScanner scannerWithString:fullIdentifier];
//            NSString *teamIdString;
//            [scanner scanUpToString:@"." intoString:&teamIdString];
//            NSRange teamIdRange = [fullIdentifier rangeOfString:teamIdString];
//            NSString *appIdentifier = [fullIdentifier substringFromIndex:teamIdRange.length + 1];
//            // 对比签名teamID或者identifier信息
// //           if (![appIdentifier isEqualToString:identifier] || ![teamId isEqualToString:teamIdString]) {
//
//            if (![appIdentifier isEqualToString: curAppId]) {
//                // exit(0)
```

```
//                asm(
//                    "mov X0,#0\n"
//                    "mov w16,#1\n"
//                    "svc #0x80"
//                    );
//            }
//            break;
//        }
//    }

    BOOL isExistCodesign = [CrifanLibiOS isCodeSignExist];

    if (isExistCodesign) {
//        NSString* curAppId = @"com.crifan.ShowSystemInfo";
        NSString  selfAppId = @"3WRHBBSBW4.*";
        NSString  gotAddId = [CrifanLibiOS getAppId];
//        BOOL isSelfId = [CrifanLibiOS isSelfAppId: curAppId];
//        BOOL isSelfId = FALSE;
        if ([gotAddId isEqualToString  selfAppId]) {
//            isSelfId = TRUE;
            resultStr = [NSString stringWithFormat: @"embedded.mobileprovision中是自己app的ID: %
@ -> 合法app", selfAppId];
        } else {
//            isSelfId = FALSE;
            resultStr = [NSString stringWithFormat: @"embedded.mobileprovision中的app的ID是%@ !
= 自己的AppId %@ -> 非法app", gotAddId, selfAppId];
        }
    } else {
        resultStr = @"不存在embedded.mobileprovision";
    }

    NSLog(@"resultStr=%@", resultStr);
    _detectResultTv.text = resultStr;
}
```

__RESTRICT

TODO：

- 【未解决】iOS越狱检测手段：__RESTRICT

已安装app

```
- (IBAction)lsapplicationBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"LSApplication check");
    NSString *resultStr = @"TODO";

    Class LSApplicationWorkspace_class = objc_getClass("LSApplicationWorkspace");
    NSObject *workspace = [LSApplicationWorkspace_class performSelector:@selector(defaultWorksp
ace)];
    NSArray *allAppList = [workspace performSelector:@selector(allApplications)]; //这样就能获取
到手机中安装的所有App

    resultStr = [NSString stringWithFormat:@"已安装app总数: %d", [allAppList count]];
    resultStr = [NSString stringWithFormat:@"%@\n非系统app列表: ", resultStr];

    for (int i=0; i<[allAppList count]; i++) {
//        LSApplicationProxy *appProxy = [allAppList objectAtIndex:i];
//        LSApplicationProxy_class *appProxy = [allAppList objectAtIndex:i];
//        NSString* bundleId =[appProxy applicationIdentifier];
//        NSString* name = [appProxy localizedName];

        id appProxy = [allAppList objectAtIndex:i];
        NSString *bundleId = [appProxy performSelector:@selector(applicationIdentifier)];
        NSString *name = [appProxy performSelector:@selector(localizedName)];
        NSString *version = [appProxy performSelector:@selector(bundleVersion)];
        NSObject *description = [appProxy performSelector:@selector(description)];
        NSArray *plugInKitPlugins = [appProxy performSelector:@selector(plugInKitPlugins)];
        if([bundleId hasPrefix:@"com.apple."]) {
            resultStr = [NSString stringWithFormat:@"%@\n[%d] bundleId=%@, name=%@, version=%@
, description=%@, plugInKitPlugins=%@", resultStr, i, bundleId, name, version, description, plu
gInKitPlugins];
        }
    }

//    Class LSApplicationProxy_class = object_getClass(@"LSApplicationProxy");
//
//    for (LSApplicationProxy_class in allAppList) {
//        NSString *bundleId = [LSApplicationProxy_class performSelector:@selector(applicationI
dentifier)];
//        NSString *version = [LSApplicationProxy_class performSelector:@selector(bundleVersion
)];
//    }

    NSLog(@"resultStr=%@", resultStr);
    _detectResultTv.text = resultStr;
}
```

SSH相关

TODO：

- 【未解决】iOS越狱检测之ssh相关
- 【未解决】iOS中如何用C语言代码实现ssh调用

```
- (IBAction)sshBtnClicked:(UIButton *)sender {
    _curBtnLbl.text = sender.titleLabel.text;
    NSLog(@"ssh check");
    const char* sshCmd = "ssh root@127.0.0.1";
//    int systemRet = system(sshCmd);
    int systemRet = iOS_system(sshCmd);
    NSLog(@"sshCmd=%s -> systemRet=%d", sshCmd, systemRet);

    _detectResultTv.text = @"TODO";
}
```

- 调用的函数：`iOS_system`
  - https://github.com/crifan/crifanLib/blob/master/c/crifanLib.c

```
/*================================================================================
 iOS: implement deprecated system()
================================================================================*/

int iOS_system(const char* command){
    const int SYSTEM_FAIL = -1;
    int systemRet = SYSTEM_FAIL;

//    if (NULL == command) {
//        return systemRet;
//    }

    typedef int (*function_system) (const char *command);
    char* dyLibSystem = "/usr/lib/libSystem.dylib";
    void* libHandle = dlopen(dyLibSystem, RTLD_GLOBAL | RTLD_NOW);
    if (NULL == libHandle) {
        char* errStr = dlerror();
        printf("Failed to open %s, error: %s", dyLibSystem, errStr);
    } else {
        function_system libSystem_system = dlsym(libHandle, "system");
        if (NULL != libSystem_system){
            systemRet = libSystem_system(command);
//            return systemRet;
        }
        dlclose(libHandle);
    }

    return systemRet;
}
```

getsectiondata

iOS反越狱检测

TODO：

- 【未解决】iOS反越狱检测：反越狱插件tweak
- 【已解决】iOS反越狱检测：参考学习借鉴开源代码项目
- 【未解决】iOS反越狱检测：优化逻辑调用被hook的orig函数
- 【未解决】越狱iOS如何实现反越狱检测防越狱检测屏蔽越狱检测
- 其他反越狱检测相关
  - 【记录】iOS反越狱检测：hook调试NSBundle的mainBundle的pathForResource

现已公开发布源码：

crifan/iOSBypassJailbreak: 越狱iOS的hook插件，实现反越狱检测

本教程后续章节中贴出的iOS反越狱检测的代码片段，均摘录自其中。

URL Scheme

TODO:

- 【已解决】iOS反越狱检测：绕过cydia://的url scheme
- 【未解决】iOS反越狱检测：能否打开特定开头的URL scheme

```
const char* CydiaPrefix = "cydia://";

/*================================================================================
 Hook: UIApplication canOpenURL:
================================================================================*/

/*
 hook url scheme, eg: cydia://
 */

%hook UIApplication

const char* CydiaPrefix = "cydia://";

- (BOOL)canOpenURL:(NSURL *)url
{
    iosLogDebug("url=%{public}@", url);
    bool couldOpen = false;
    bool isCydia = false;

    if (cfgHookEnable_misc) {
        NSString *urlNSStr = [url absoluteString];
        const char* urlStr = [urlNSStr UTF8String];
        char* urlStrLower = strToLowercase(urlStr);
        iosLogDebug("urlStrLower=%s", urlStrLower);
        isCydia = strStartsWith(urlStrLower, CydiaPrefix);
        free(urlStrLower);
        iosLogDebug("isCydia=%{public}s", boolToStr(isCydia));

        if(isCydia){
            couldOpen = false;
        } else{
    //        couldOpen = %orig(url);
            couldOpen = %orig;
        }
    } else {
        couldOpen = %orig;
    }

    // for debug
//    if (isCydia) {
        iosLogInfo("url=%{public}@ -> isCydia=%{public}s -> couldOpen=%{public}s", url, boolToS
tr(isCydia), boolToStr(couldOpen));
//    }
```

```
    return couldOpen;
}

end
```

文件

文件打开

TODO：

- 【已解决】iOS程序崩溃：strdup报错Thread 1 EXC_BAD_ACCESS code 2 address
- 【未解决】iOS反越狱检测：优化findRealImageCount改为调用_dyld_get_image_vmaddr_slide计算逻辑
- 【已解决】iOS报错：libsystem_malloc.dylib nanov2_allocate_from_block VARIANT mp
- 【已解决】iOS的tweak中open报错：Address of overloaded function open does not match required type void
- 【已解决】iOS的tweak中open的hook报错：%orig requires arguments when hooking variadic functions

C函数

TODO:

整理下面多个帖子的内容

open系列

open

TODO:

- 【基本解决】iOS反越狱检测之打开文件：open
- 【已解决】iOS反越狱检测：tweak插件中hook绕过open函数

```
/*================================================================================
 Hook: open()
================================================================================*/

/*
 TODO: maybe need support more version:
        int creat(const char *pathname, mode_t mode);
        int openat(int dirfd, const char *pathname, int flags);
        int openat(int dirfd, const char *pathname, int flags, mode_t mode);
        int openat2(int dirfd, const char *pathname, const struct open_how *how, size_t size);
 refer:
    https://man7.org/linux/man-pages/man2/open.2.html
 */

// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man2/open.2.html
// normally max number of open parameter is 1
int MaxSupportArgNum_open = 2;
int open(const char *path, int oflag, ...);

%hookf(int, open, const char *path, int oflag, ...){
    iosLogDebug("path=%{public}s, oflag=%d", path, oflag);
    bool isJbPath = false;

    if(cfgHookEnable_openFileC_open){
        isJbPath = isJailbreakPath(path);
        iosLogDebug("isJbPath=%{bool}d", isJbPath);

    //    // for debug: for system() call, temp allow /bin/sh
    //    if (0 == strcmp(path, "/bin/sh")){
    //        os_log(OS_LOG_DEFAULT, "hook_open_system: temp allow /bin/sh");
    //        isJbPath = false;
    //    }

        if (isJbPath){
```

```
            iosLogInfo("path=%{public}s -> isJbPath=%{bool}d", path, isJbPath);
            errno = ENOENT;
            iosLogDebug("set errno=%d", errno);
            return OPEN_FAILED;
        }
    }

    // Setting up some variables to get all the parameters from open
    mode_t curPara, paraList[MaxSupportArgNum_open];
    va_list argList;
    int curParaNum = 0;

    va_start(argList, oflag);
//    while ((curPara = (mode_t) va_arg(argList, mode_t))) {
//        paraList[curParaNum] = curPara;
//        curParaNum += 1;
//        os_log(OS_LOG_DEFAULT, "hook_open: [%d] curPara=%d", curParaNum, curPara);
//    }
//    unsigned int firstArg = va_arg(argList, mode_t)
//    curPara = (mode_t) va_arg(argList, mode_t);
    curPara = (mode_t) va_arg(argList, unsigned int);
    // maxium is only extra 1 para, so no need while
    if (0 != (int)curPara) {
        paraList[curParaNum] = curPara;
        curParaNum += 1;
        iosLogDebug("[%d] para=%d", curParaNum, curPara);
    }
    va_end(argList);

    iosLogDebug("curParaNum=%d, argList=%{public}s", curParaNum, argList);

//    return %orig;
    int openRetValue = OPEN_FAILED;
    if (0 == curParaNum){
        openRetValue = %orig(path, oflag);
        if (isJbPath) {
            iosLogInfo("%spath=%{public}s, oflag=0x%x -> isJbPath=%{bool}d -> openRetValue=%d",
HOOK_PREFIX(cfgHookEnable_openFileC_open), path, oflag, isJbPath, openRetValue);
        }
    } else if (1 == curParaNum){
        mode_t mode = paraList[0];
//        os_log(OS_LOG_DEFAULT, "hook_open: mode=0x%x", mode);
        openRetValue = %orig(path, oflag, mode);
        if (isJbPath) {
            iosLogInfo("%spath=%{public}s, oflag=0x%x, mode=0x%x -> isJbPath=%{bool}d -> openRe
tValue=%d", HOOK_PREFIX(cfgHookEnable_openFileC_open), path, oflag, mode, isJbPath, openRetValue
);
        }
    }
    return openRetValue;
}
```

fopen

TODO：

【已解决】iOS反越狱检测之打开文件：hook绕过fopen

```
/*======================================================================
 Hook: fopen()
======================================================================*/

// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/fopen.3.html
FILE *fopen(const char *filename, const char *mode);

%hookf(int, fopen, const char *filename, const char *mode){
    iosLogDebug("filename=%{public}s, mode=%{public}s", filename, mode);
    int retValue = FOPEN_OPEN_FAILED;
    bool isJbPath = false;

    if(cfgHookEnable_openFileC_fopen){
        isJbPath = isJailbreakPath(filename);
        if (isJbPath){
            retValue = FOPEN_OPEN_FAILED;
        } else {
            retValue = %orig;
        }
    } else {
        retValue = %orig;
    }

    // for debug
    if (isJbPath) {
        iosLogInfo("filename=%{public}s, mode=%{public}s -> isJbPath=%s -> retValue=%d", filename, mode, boolToStr(isJbPath), retValue);
    }
    return retValue;
}
```

opendir + __opendir2

```
/*======================================================================
 Hook: opendir()
======================================================================*/

// NOTES: !!! hook opendir will cause app crash.
//  detail log: SubstituteLog: SubHookFunction: substitute_hook_functions returned SUBSTITUTE_ERR_FUNC_BAD_INSN_AT_START

//// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/opendir.3.html
//DIR* opendir(const char *dirname);
//
//%hookf(DIR*, opendir, const char *dirname){
//    iosLogInfo("dirname=%s", dirname);
```

```
////    return %orig;
//    return %orig(dirname);
//}

/*============================================================================
 Hook: __opendir2()
============================================================================*/

// https://opensource.apple.com/source/Libc/Libc-186/gen.subproj/opendir.c.auto.html
// https://opensource.apple.com/source/Libc/Libc-320/include/dirent.h.auto.html
DIR *__opendir2(const char *name, int flags);

%hookf(DIR*, __opendir2, const char *name, int flags){
    iosLogDebug("name=%{public}s, flags=0x%x", name, flags);
    DIR* openedDir = OPENDIR_FAILED;
    bool isJbPath = false;

    if (cfgHookEnable_openFileC___opendir2){
        isJbPath = isJailbreakPath(name);
        if (isJbPath) {
            openedDir = OPENDIR_FAILED;
        } else {
//            openedDir = %orig(name, flags);
            openedDir = %orig;
        }
    } else {
//        openedDir = %orig(name, flags);
        openedDir = %orig;
    }

    // for debug
    if (isJbPath) {
        iosLogInfo("name=%{public}s, flags=0x%x -> isJbPath=%{bool}d -> openedDir=%p", name, fl
ags, isJbPath, openedDir);
    }

    return openedDir;
}
```

access系列

TODO：

- 【未解决】iOS反越狱检测之打开文件：access系列函数

access

```
/*============================================================================
 Hook: access()
============================================================================*/

// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneO
S/man2/access.2.html
```

```
int access(const char *path, int amode);

%hookf(int, access, const char *path, int amode){
    iosLogDebug("path=%{public}s, amode=0x%x", path, amode);
    int retValue = ACCESS_FAILED;
    bool isJbPath = false;

    if (cfgHookEnable_openFileC) {
        isJbPath = isJailbreakPath(path);
        iosLogDebug("isJbPath=%{bool}d", isJbPath);
        if (isJbPath){
            iosLogDebug("isJbPath=%{bool}d, %{public}s", isJbPath, path);
            errno = ENOENT;
            iosLogDebug("set errno=%d", errno);
            retValue = ACCESS_FAILED;
        } else {
            retValue = %orig;
        }
    } else {
        retValue = %orig;
    }

    // for debug
    if(isJbPath){
        iosLogInfo("path=%{public}s, amode=0x%x -> isJbPath=%{bool}d, retValue=%d", path, amode,
 isJbPath, retValue);
    }
    return retValue;
}
```

faccessat

```
/*============================================================================
 Hook: faccessat()
============================================================================*/

// https://man7.org/linux/man-pages/man2/access.2.html
// https://linux.die.net/man/2/faccessat
int faccessat(int dirfd, const char *pathname, int mode, int flags);

%hookf(int, faccessat, int dirfd, const char *pathname, int mode, int flags){
    iosLogDebug("dirfd=%d, pathname=%{public}s, mode=0x%x, flags=0x%x", dirfd, pathname, mode,
flags);
    int retInt = ACCESS_FAILED;
    bool isJbPath = false;

    if(cfgHookEnable_openFileC_faccessat){
        const char* absPath = NULL;
        bool isAbsPath = strStartsWith(pathname, "/");
        iosLogDebug("isAbsPath=%{bool}d", isAbsPath);
        if (isAbsPath) {
            absPath = pathname;
        } else {
```

```
            // is relative path
            if (dirfd == AT_FDCWD){
                iosLogDebug("dirfd is AT_FDCWD=%d", AT_FDCWD);

                // pathname is interpreted relative to the current working directory of the cal
ling process (like access())
                // TODO: try get current working directory -> avoid caller pass the special pat
h, finnaly is jailbreak path
                // eg: current working directory is "/usr/xxx/yyy/", then pass in "../../libexe
c/cydia/zzz"
                // finnal path is "/usr/libexec/cydia/zzz", match jailbreak path: "/usr/libexec
/cydia/", is jaibreak path
                // but use "../../libexec/cydia/zzz" can not check whether is jailbreak path
            } else {
                // get file path from dir fd
                char filePath[PATH_MAX];
                bool isGetPathOk = getFilePath(dirfd, filePath);
                iosLogDebug("isGetPathOk=%s", boolToStr(isGetPathOk));
                if (isGetPathOk) {
                    char* fullPath = strPathJoin(filePath, pathname)
                    iosLogDebug("fullPath=%{public}s", fullPath);
                    absPath = fullPath;
                }
            }
        }

        if (NULL != absPath){
            isJbPath = isJailbreakPath(absPath);
            iosLogDebug("absPath=%{public}s -> isJbPath=%{bool}d", absPath, isJbPath);
            if (isJbPath) {
                iosLogDebug("hook jailbreak path: %s", absPath);
                retInt = ACCESS_FAILED;
            } else {
                retInt = %orig;
            }
        } else {
            retInt = %orig;
        }
    } else {
        retInt = %orig;
    }

    // for debug
    if (isJbPath) {
        iosLogInfo("%sdirfd=%d, pathname=%{public}s, mode=0x%x, flags=0x%x -> isJbPath=%{bool}d
, retInt=%d",
                   HOOK_PREFIX(cfgHookEnable_openFileC_faccessat), dirfd, pathname, mode, flags,
 isJbPath, retInt);
    }

    return retInt;
}
```

realpath

```
/*==========================================================================
 Hook: realpath()
==========================================================================*/

// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/realpath.3.html
char* realpath(const char* file_name, char* resolved_name);

%hookf(char*, realpath, const char* file_name, char* resolved_name){
    iosLogDebug("file_name=%{public}s, resolved_name=%p", file_name, resolved_name);
    char* resolvedPath = REALPATH_FAILED;
    bool isJbPath = false;

    if (cfgHookEnable_openFileC) {
        isJbPath = isJailbreakPath(file_name);
        if (isJbPath) {
            resolvedPath = REALPATH_FAILED;
        } else {
//            resolvedPath = %orig(file_name, resolved_name);
            resolvedPath = %orig;
        }
    } else {
        resolvedPath = %orig;
    }

    // for debug
    if (isJbPath) {
        iosLogInfo("file_name=%{public}s, resolved_name=%{public}s -> isJbPath=%{bool}d -> resolvedPath=%{public}s", file_name, resolved_name, isJbPath, resolvedPath);
    }

    return resolvedPath;
}
```

stat系列

TODO：

- 【已解决】iOS反越狱检测之stat：支持路径是否带斜杠结尾以及包含点和两个点
- 【已解决】iOS反越狱之stat函数测试机文件列表对于非越狱手是否正常
- 【部分解决】iOS反越狱检测的其他版本stat函数：stat64
- 【已解决】iOS反越狱检测：hook绕过stat函数的实现机制和方式

---

stat

```
/*==========================================================================
 Hook: stat()
==========================================================================*/
```

```
int stat(const char *pathname, struct stat *buf);

%hookf(int, stat, const char *pathname, struct stat *buf){
    int statRet = STAT_FAILED;
    bool isJbPath = false;

    if (cfgHookEnable_openFileC) {
        isJbPath = isJailbreakPath(pathname);
        iosLogDebug("pathname=%{public}s -> isJbPath=%s", pathname, boolToStr(isJbPath));

        if (isJbPath){
            statRet = STAT_FAILED;
        } else {
            statRet = %orig;
        }
    } else {
        statRet = %orig;
    }

    // for debug
    if(isJbPath){
        iosLogInfo("pathname=%{public}s -> isJbPath=%s -> statRet=%d", pathname, boolToStr(isJb
Path), statRet);
    }

    return statRet;
}
```

lstat

```
/*============================================================================
 Hook: lstat()
============================================================================*/

// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneO
S/man2/lstat.2.html
int lstat(const char* path, struct stat* buf);

%hookf(int, lstat, const char* path, struct stat* buf){
    iosLogDebug("path=%{public}s, buf=%p", path, buf);
    int lstatRet = STAT_FAILED;
    bool isJbPath = false;

    if (cfgHookEnable_openFileC) {
        isJbPath = isJailbreakPath(path);
        iosLogDebug("isJbPath=%{bool}d", isJbPath);

        if (isJbPath){
            lstatRet = STAT_FAILED;
        } else {
            lstatRet = %orig;
        }
    } else {
```

```
        lstatRet = %orig;
    }

    // for debug
    if(isJbPath){
        iosLogInfo("path=%{public}s -> isJbPath=%s -> lstatRet=%d", path, boolToStr(isJbPath),
lstatRet);
    }

    return lstatRet;
}
```

fstat

```
/*==========================================================================
 Hook: fstat()
==========================================================================*/

// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneO
S/man2/fstat.2.html

int fstat(int fd, struct stat *buf);

%hookf(int, fstat, int fd, struct stat *buf){
    iosLogDebug("fd=%d, buf=%p", fd, buf);
    int retInt = STAT_FAILED;
    bool isJbPath = false;

    char parsedPath[PATH_MAX];
    memset(parsedPath, 0, PATH_MAX);

    if (cfgHookEnable_openFileC) {
        // get file path from fd
        bool isGetPathOk = getFilePath(fd, parsedPath);
        iosLogDebug("isGetPathOk=%s, parsedPath=%s", boolToStr(isGetPathOk), parsedPath);
        if (isGetPathOk) {
            isJbPath = isJailbreakPath(parsedPath);
            iosLogDebug("isJbPath=%{bool}d", isJbPath);

            if (isJbPath){
                retInt = STAT_FAILED;
            } else {
                retInt = %orig;
            }
        } else {
            // can not get path -> can not check is jailbreak or not -> not hook
            retInt = %orig;
        }
    } else {
        retInt = %orig;
    }

    // for debug
    if(isJbPath){
```

```
        iosLogInfo("fd=%d,buf=%p -> parsedPath=%{public}s -> isJbPath=%s -> retInt=%d", fd, buf,
 parsedPath, boolToStr(isJbPath), retInt);
    }

    return retInt;
}
```

fstatat

```
/*=============================================================================
 Hook: fstatat()
=============================================================================*/

// https://man7.org/linux/man-pages/man3/fstatat.3p.html
// https://linux.die.net/man/2/fstatat

//int fstatat(int dirfd, const char *pathname, struct stat *buf, int flags);
//int fstatat(int fd, const char *restrict path, struct stat *restrict buf, int flag);
int fstatat(int fd, const char* pathname, struct stat* buf, int flags);

%hookf(int, fstatat, int fd, const char* pathname, struct stat* buf, int flags){
    iosLogDebug("fd=%d, pathname=%{public}s, buf=%p, flags=0x%x", fd, pathname, buf, flags);
    int fstatatRet = STATFS_FAILED;
    bool isJbPath = false;

    if(cfgHookEnable_openFileC){
        const char* absPath = NULL;
        bool isAbsPath = strStartsWith(pathname, "/");
        iosLogDebug("isAbsPath=%{bool}d", isAbsPath);
        if (isAbsPath) {
            absPath = pathname;
        } else {
            // is relative path
            if (fd == AT_FDCWD){
                iosLogDebug("fd is AT_FDCWD=%d", AT_FDCWD);

                // pathname is interpreted relative to the current working directory of the cal
ling process (like access())
                // TODO: try get current working directory -> avoid caller pass the special pat
h, finnaly is jailbreak path
                // eg: current working directory is "/usr/xxx/yyy/", then pass in "../../libexe
c/cydia/zzz"
                // finnal path is "/usr/libexec/cydia/zzz", match jailbreak path: "/usr/libexec
/cydia/", is jaibreak path
                // but use "../../libexec/cydia/zzz" can not check whether is jailbreak path
            } else {
                // get file path from dir fd
                char filePath[PATH_MAX];
                bool isGetPathOk = getFilePath(fd, filePath);
                iosLogDebug("isGetPathOk=%s", boolToStr(isGetPathOk));
                if (isGetPathOk) {
                    char* fullPath = strPathJoin(filePath, pathname)
                    iosLogDebug("fullPath=%{public}s", fullPath);
```

```
                    absPath = fullPath;
                }
            }
        }

        if (NULL != absPath){
            isJbPath = isJailbreakPath(absPath);
            iosLogDebug("absPath=%{public}s -> isJbPath=%{bool}d", absPath, isJbPath);
            if (isJbPath) {
                iosLogDebug("hook jailbreak path: %s", absPath);
                fstatatRet = STATFS_FAILED;
            } else {
                fstatatRet = %orig;
            }
        } else {
            fstatatRet = %orig;
        }
    } else {
        fstatatRet = %orig;
    }

    // for debug
    if (isJbPath) {
        iosLogInfo("fd=%d, pathname=%{public}s, buf=%p, flags=0x%x -> isJbPath=%{bool}d -> fsta
tatRet=%d", fd, pathname, buf, flags, isJbPath, fstatatRet);
    }

    return fstatatRet;
}
```

statfs

```
/*============================================================================
 Hook: statfs()
============================================================================*/

// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneO
S/man2/statfs.2.html

int statfs(const char *path, struct statfs *buf);

%hookf(int, statfs, const char *path, struct statfs *buf){
    iosLogDebug("path=%{public}s, buf=%p", path, buf);
    int statfsRet = STATFS_FAILED;
    bool isJbPath = false;

    if (cfgHookEnable_openFileC) {
        isJbPath = isJailbreakPath(path);
        iosLogDebug("isJbPath=%{bool}d", isJbPath);

        if (isJbPath){
            statfsRet = STATFS_FAILED;
        } else {
            statfsRet = %orig;
```

```
        }
    } else {
        statfsRet = %orig;
    }

    // for debug
    if(isJbPath){
        iosLogInfo("found jailbreak path: path=%{public}s -> isJbPath=%s -> statfsRet=%d", path,
 boolToStr(isJbPath), statfsRet);
    }

    return statfsRet;
}
```

fstatfs

```
/*============================================================================
 Hook: fstatfs()
============================================================================*/

// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man2/statfs64.2.html

int fstatfs(int fd, struct statfs *buf);

%hookf(int, fstatfs, int fd, struct statfs *buf){
    iosLogDebug("fd=%d, buf=%p", fd, buf);
    int fstatfsRet = STATFS_FAILED;
    bool isJbPath = false;

    char parsedPath[PATH_MAX];
    memset(parsedPath, 0, PATH_MAX);

    if (cfgHookEnable_openFileC) {
        // get file path from fd
        bool isGetPathOk = getFilePath(fd, parsedPath);
        iosLogDebug("isGetPathOk=%s, parsedPath=%s", boolToStr(isGetPathOk), parsedPath);
        if (isGetPathOk) {
            isJbPath = isJailbreakPath(parsedPath);
            iosLogDebug("isJbPath=%{bool}d", isJbPath);

            if (isJbPath){
                fstatfsRet = STATFS_FAILED;
            } else {
                fstatfsRet = %orig;
            }
        } else {
            // can not get path -> can not check is jailbreak or not -> not hook
            fstatfsRet = %orig;
        }
    } else {
        fstatfsRet = %orig;
    }
```

```
    // for debug
    if(isJbPath){
        iosLogInfo("fd=%d,buf=%p -> parsedPath=%{public}s -> isJbPath=%s -> fstatfsRet=%d", fd,
 buf, parsedPath, boolToStr(isJbPath), fstatfsRet);
    }

    return fstatfsRet;
}
```

syscall

TODO：

- 【已解决】iOS去hook绕过syscall函数异常：可变参数计算个数再次异常12个13个
- 【已解决】iOS的tweak插件去hook函数syscall出现递归调用死循环
- 【已解决】iOS的tweak插件Logos的%orig的实现原理如何规避绕开原函数的递归调用
- 【未解决】iOS反越狱检测：syscall

syscall的fork

- 【已解决】iOS反越狱检测：syscall的fork

TODO：

syscall的stat和stat64

TODO：

- 【已解决】iOS反越狱检测hook绕过：syscall的stat和stat64

```
/*==============================================================================
 Define
==============================================================================*/

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

/*==============================================================================
 Const
==============================================================================*/

/*==============================================================================
 Hook: syscall()
==============================================================================*/

/*
    https://www.theiphonewiki.com/wiki/Kernel_Syscalls
    TODO: support syscall(access_extended)
 */


int syscall(int, ...);

// normally max number of syscall parameter is not exceed 8
// refer: https://opensource.apple.com/source/xnu/xnu-4570.1.46/bsd/kern/syscalls.master
int MaxSupportArgNum_syscall = 16;
```

```
%hookf(int, syscall, int number, ...){
    iosLogDebug("number=%d", number);

    int syscallRetValue = -1;

    // Setting up some variables to get all the parameters from syscall
    void *paraPtr, *paraList[MaxSupportArgNum_syscall];
//    char *paraPtr, *paraList[MaxSupportArgNum_syscall];
    va_list argList;
    int curParaNum = 0;

    if (cfgHookEnable_syscall) {
        // #define    SYS_fork           2
        bool isFork = (SYS_fork == number);
        if (isFork){
            iosLogInfo("number=%d -> return %d", number, FORK_FAILED);
            return FORK_FAILED;
        }

        // #define    SYS_open           5
        bool isOpen = (SYS_open == number);
        if (isOpen){
            //int open(const char *path, int oflag, ...);
            // ->
    //        int open(const char *pathname, int flags);
    //        int open(const char *pathname, int flags, mode_t mode);

            //5    AUE_OPEN_RWTC   ALL     { int open(user_addr_t path, int flags, int mode) NO
_SYSCALL_STUB; }
            va_start(argList, number);
            const char * fisrtPath = va_arg(argList, const char *);
            int secondFlags = va_arg(argList, int);
    //        mode_t thirdMode = va_arg(argList, mode_t);
            mode_t thirdMode = (mode_t)va_arg(argList, unsigned int);
            va_end(argList);
            iosLogDebug("fisrtPath=%{public}s, secondFlags=%d, thirdMode=%d", fisrtPath, second
Flags, thirdMode);

            bool isJbPath = isJailbreakPath(fisrtPath);
            iosLogDebug("isJbPath=%{bool}d", isJbPath);
            if (isJbPath){
                errno = ENOENT;
                iosLogDebug("set errno=%d", errno);
                syscallRetValue = OPEN_FAILED;
            } else {
                syscallRetValue = %orig(number, fisrtPath, secondFlags, thirdMode);
            }
            iosLogInfo("SYS_open: number=%d -> isJbPath=%{bool}d, fisrtPath=%{public}s -> sysca
llRetValue=%d", number, isJbPath, fisrtPath, syscallRetValue);
            return syscallRetValue;
        }

        // #define    SYS_ptrace         26
        bool isPtrace = (SYS_ptrace == number);
        if (isPtrace){
```

```
            // https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man2/ptrace.2.html
            // int ptrace(int request, pid_t pid, caddr_t addr, int data);
            va_start(argList, number);
            int request = va_arg(argList, int);
            int pid = va_arg(argList, int);
            char* addr = va_arg(argList, char*);
            int data = va_arg(argList, int);
            va_end(argList);

            iosLogInfo("request=%d, pid=%d, addr=%p, data=%d", request, pid, addr, data);

            if (PT_DENY_ATTACH == request){
                syscallRetValue = PTRACE_FAILED;
            } else {
                syscallRetValue = %orig(request, pid, addr, data);
            }

            iosLogInfo("SYS_ptrace: request=%d, pid=%d, addr=%p, data=%d -> syscallRetValue=%d", request, pid, addr, data, syscallRetValue);
            return syscallRetValue;
        }

        // #define    SYS_access         33
        bool isAccess = (SYS_access == number);
        if (isAccess) {
            // int access(const char *path, int amode);
            va_start(argList, number);
            const char* path = va_arg(argList, const char *);
            int amode = va_arg(argList, int);
            va_end(argList);

            iosLogDebug("isAccess=%{bool}d, path=%{public}s, amode=0x%x", isAccess, path, amode);

            bool isJbPath = isJailbreakPath(path);
            iosLogDebug("isJbPath=%{bool}d", isJbPath);
            if (isJbPath){
                syscallRetValue = ACCESS_FAILED;
            } else {
                syscallRetValue = %orig(number, path, amode);
            }
            iosLogInfo("SYS_access: number=%d -> path=%{public}s, amode=0x%x -> isJbPath=%{bool}d -> syscallRetValue=%d", number, path, amode, isJbPath, syscallRetValue);
            return syscallRetValue;
        }

        // #define    SYS_statfs         157
        bool isStatfs = (SYS_statfs == number);
        if (isStatfs) {
            // int statfs(const char *path, struct statfs *buf);
            va_start(argList, number);
            const char* path = va_arg(argList, const char *);
            struct statfs* buf = va_arg(argList, struct statfs*);
            va_end(argList);
```

```
            iosLogDebug("isStatfs=%{bool}d, path=%{public}s, buf=%p", isStatfs, path, buf);

            bool isJbPath = isJailbreakPath(path);
            iosLogDebug("isJbPath=%{bool}d", isJbPath);
            if (isJbPath){
                syscallRetValue = STATFS_FAILED;
            } else {
                syscallRetValue = %orig(number, path, buf);
            }
            iosLogInfo("SYS_statfs: number=%d -> path=%{public}s, buf=%p -> isJbPath=%{bool}d -
> syscallRetValue=%d", number, path, buf, isJbPath, syscallRetValue);
            return syscallRetValue;
        }

        // #define    SYS_fstatfs        158
        bool isFstatfs = (SYS_fstatfs == number);
        if (isFstatfs) {
            bool isGetPathOk = false;
            bool isJbPath = false;
            char parsedPath[PATH_MAX];
            memset(parsedPath, 0, PATH_MAX);

            // int fstatfs(int fd, struct statfs *buf);
            va_start(argList, number);
            int fd = va_arg(argList, int);
            struct stat* buf = va_arg(argList, struct stat*);
            va_end(argList);

            iosLogDebug("isFstatfs=%{bool}d, fd=%d, buf=%p", isFstatfs, fd, buf);

            isGetPathOk = getFilePath(fd, parsedPath);
            iosLogDebug("isGetPathOk=%s, parsedPath=%s", boolToStr(isGetPathOk), parsedPath);
            if (isGetPathOk) {
                isJbPath = isJailbreakPath(parsedPath);
                iosLogDebug("isJbPath=%{bool}d", isJbPath);

                if (isJbPath){
                    syscallRetValue = STATFS_FAILED;
                } else {
                    syscallRetValue = %orig(number, fd, buf);
                }
            } else {
                // can not get path -> can not check is jailbreak or not -> not hook
                syscallRetValue = %orig(number, fd, buf);
            }

            iosLogInfo("SYS_fstatfs: number=%d -> fd=%d, buf=%p -> isJbPath=%{bool}d -> syscall
RetValue=%d", number, fd, buf, isJbPath, syscallRetValue);
            return syscallRetValue;
        }

        // #define    SYS_stat           188
        // #define    SYS_stat64         338
        bool isStat = (SYS_stat == number);
        bool isStat64 = (SYS_stat64 == number);
        if (isStat || isStat64){
```

```
            //int     stat(const char *, struct stat *) __DARWIN_INODE64(stat);
            //int     stat64(const char *, struct stat64 *) __OSX_AVAILABLE_BUT_DEPRECATED(__MA
C_10_5, __MAC_10_6, __IPHONE_NA, __IPHONE_NA);
            va_start(argList, number);
            const char * fisrtPath = va_arg(argList, const char *);
            void *secondStat = va_arg(argList, void *);
            va_end(argList);

            iosLogDebug("isStat=%{bool}d, isStat64=%{BOOL}d, fisrtPath=%{public}s, secondStat=%
p", isStat, isStat64, fisrtPath, secondStat);

            bool isJbPath = isJailbreakPath(fisrtPath);
            iosLogDebug("isJbPath=%{bool}d", isJbPath);
            if (isJbPath){
                syscallRetValue =  OPEN_FAILED;
            } else {
                //        if (isStat){
        //            struct stat *statInfo = (struct stat *)secondStat;
        //            syscallRetValue = %orig(number, fisrtPath, statInfo);
        //        } else if(isStat64){
        //            struct stat64 *stat64Info = (struct stat64 *)secondStat;
        //            syscallRetValue = %orig(number, fisrtPath, stat64Info);
        //        }
                syscallRetValue = %orig(number, fisrtPath, secondStat);
            }
            iosLogInfo("SYS_stat/SYS_stat64: number=%d -> isJbPath=%{bool}d, fisrtPath=%{public
}s -> syscallRetValue=%d", number, isJbPath, fisrtPath, syscallRetValue);
            return syscallRetValue;
        }

        // #define    SYS_fstat          189
        bool isFstat = (SYS_fstat == number);
        if (isFstat) {
            bool isGetPathOk = false;
            bool isJbPath = false;
            char parsedPath[PATH_MAX];
            memset(parsedPath, 0, PATH_MAX);

            // int fstat(int fd, struct stat *buf);
            va_start(argList, number);
            int fd = va_arg(argList, int);
            struct stat* buf = (struct stat*)va_arg(argList, void *);
            va_end(argList);

            iosLogDebug("isFstat=%{bool}d, fd=%d, buf=%p", isFstat, fd, buf);

            isGetPathOk = getFilePath(fd, parsedPath);
            iosLogDebug("isGetPathOk=%{bool}d, parsedPath=%s", isGetPathOk, parsedPath);
            if (isGetPathOk) {
                isJbPath = isJailbreakPath(parsedPath);
                iosLogDebug("isJbPath=%{bool}d", isJbPath);

                if (isJbPath){
                    syscallRetValue =  STAT_FAILED;
                } else {
                    syscallRetValue = %orig(number, fd, buf);
```

```
            }
        } else {
            syscallRetValue = %orig(number, fd, buf);
        }

        iosLogInfo("SYS_fstat: number=%d -> fd=%d -> isGetPathOk=%{bool}d, parsedPath=%{pub
lic}s -> isJbPath=%{bool}d -> syscallRetValue=%d", number, fd, isGetPathOk, parsedPath, isJbPath
, syscallRetValue);
        return syscallRetValue;
    }

    // #define    SYS_lstat          190
    bool isLstat = (SYS_lstat == number);
    if (isLstat) {
        // int lstat(const char* path, struct stat* buf);
        va_start(argList, number);
        const char* fisrtPath = va_arg(argList, const char *);
        struct stat* secondBuf = (struct stat*)va_arg(argList, void *);
        va_end(argList);

        iosLogDebug("isLstat=%{bool}d, fisrtPath=%{public}s, secondBuf=%p", isLstat, fisrtP
ath, secondBuf);

        bool isJbPath = isJailbreakPath(fisrtPath);
        iosLogDebug("isJbPath=%{bool}d", isJbPath);
        if (isJbPath){
            syscallRetValue =  STAT_FAILED;
        } else {
            syscallRetValue = %orig(number, fisrtPath, secondBuf);
        }
        iosLogInfo("SYS_lstat: number=%d -> isJbPath=%{bool}d, fisrtPath=%{public}s -> sysc
allRetValue=%d", number, isJbPath, fisrtPath, syscallRetValue);
        return syscallRetValue;
    }

    // #define    SYS_fstatat        469
    bool isFstatat = (SYS_fstatat == number);
    if (isFstatat) {
        bool isJbPath = false;

        // int fstatat(int dirfd, const char *pathname, struct stat *buf, int flags);
        va_start(argList, number);
        int dirfd = va_arg(argList, int);
        const char *pathname = (const char *)va_arg(argList, void *);
        struct stat *buf = (struct stat*)va_arg(argList, void *);
        int flags = va_arg(argList, int);
        va_end(argList);

        iosLogDebug("isFstatat=%{bool}d, dirfd=%d, pathname=%{public}s, buf=%p, flags=%d",
isFstatat, dirfd, pathname, buf, flags);

        const char* absPath = NULL;
        bool isAbsPath = strStartsWith(pathname, "/");
        iosLogDebug("isAbsPath=%{bool}d", isAbsPath);
        if (isAbsPath) {
            absPath = pathname;
```

```
            } else {
                // is relative path
                if (dirfd == AT_FDCWD){
                    iosLogDebug("dirfd is AT_FDCWD=%d", AT_FDCWD);

                    // pathname is interpreted relative to the current working directory of the
 calling process (like access())
                    // TODO: try get current working directory -> avoid caller pass the special
 path, finnaly is jailbreak path
                    // eg: current working directory is "/usr/xxx/yyy/", then pass in "../../li
bexec/cydia/zzz"
                    // finnal path is "/usr/libexec/cydia/zzz", match jailbreak path: "/usr/lib
exec/cydia/", is jaibreak path
                    // but use "../../libexec/cydia/zzz" can not check whether is jailbreak path

                } else {
                    // get file path from dir fd
                    char filePath[PATH_MAX];
                    bool isGetPathOk = getFilePath(dirfd, filePath);
                    iosLogDebug("isGetPathOk=%s", boolToStr(isGetPathOk));
                    if (isGetPathOk) {
                        char* fullPath = strPathJoin(filePath, pathname)
                        iosLogDebug("fullPath=%{public}s", fullPath);
                        absPath = fullPath;
                    }
                }
            }

            if (NULL != absPath){
                isJbPath = isJailbreakPath(absPath);
                iosLogDebug("absPath=%{public}s -> isJbPath=%{bool}d", absPath, isJbPath);
                if (isJbPath) {
                    iosLogDebug("hook jailbreak path: %s", absPath);
                    syscallRetValue = STATFS_FAILED;
                } else {
                    syscallRetValue = %orig(number, dirfd, pathname, buf, flags);
                }
            } else {
                syscallRetValue = %orig(number, dirfd, pathname, buf, flags);
            }

            iosLogInfo("SYS_fstatat: number=%d -> dirfd=%d, pathname=%{public}s, buf=%p, flags=
0x%x -> isJbPath=%{bool}d -> syscallRetValue=%d", number, dirfd, pathname, buf, flags, isJbPath,
 syscallRetValue);
            return syscallRetValue;
        }

    }

    va_start(argList, number);
    while ((paraPtr = (void *) va_arg(argList, void *))) {
    //    while ((paraPtr = (char *) va_arg(argList, char *))) {
        paraList[curParaNum] = paraPtr;
        curParaNum += 1;
        iosLogDebug("[%d] paraPtr=%p", curParaNum, paraPtr);
    }
```

```
    va_end(argList);

//    iosLogDebug("argList=%{public}s", argList);
    iosLogDebug("curParaNum=%d", curParaNum);

//    return %orig;
//    return %orig(number, ...);
//    int retValue = %orig();

//    int retValue = callOriginSyscall(number, curParaNum, paraList);
////    int retValue = callOriginSyscall(number, curParaNum, (void *)paraList);
//    iosLogDebug("retValue=%d", retValue);
//    return retValue;

    int paraNum = curParaNum;

    if (0 == paraNum){
        syscallRetValue = %orig(number);
    } else if (1 == paraNum){
        void* para1 = paraList[0];
        iosLogDebug("para1=%p", para1);
        syscallRetValue = %orig(number, para1);
    } else if (2 == paraNum){
        void* para1 = paraList[0];
        void* para2 = paraList[1];
        iosLogDebug("para1=%p,para2=%p", para1, para2);
        syscallRetValue = %orig(number, para1, para2);
    } else if (3 == paraNum){
        void* para1 = paraList[0];
        void* para2 = paraList[1];
        void* para3 = paraList[2];
        iosLogDebug("para1=%p,para2=%p,para3=%p", para1, para2, para3);
        syscallRetValue = %orig(number, para1, para2, para3);
    } else if (4 == paraNum){
        void* para1 = paraList[0];
        void* para2 = paraList[1];
        void* para3 = paraList[2];
        void* para4 = paraList[3];
        iosLogDebug("para1=%p,para2=%p,para3=%p,para4=%p", para1, para2, para3, para4);
        syscallRetValue = %orig(number, para1, para2, para3, para4);
    } else if (5 == paraNum){
        void* para1 = paraList[0];
        void* para2 = paraList[1];
        void* para3 = paraList[2];
        void* para4 = paraList[3];
        void* para5 = paraList[4];
        iosLogDebug("para1=%p,para2=%p,para3=%p,para4=%p,para5=%p", para1, para2, para3, para4,
 para5);
        syscallRetValue = %orig(number, para1, para2, para3, para4, para5);
    } else if (6 == paraNum){
        void* para1 = paraList[0];
        void* para2 = paraList[1];
        void* para3 = paraList[2];
        void* para4 = paraList[3];
        void* para5 = paraList[4];
        void* para6 = paraList[5];
```

```
        iosLogDebug("para1=%p,para2=%p,para3=%p,para4=%p,para5=%p,para6=%p", para1, para2, para3
, para4, para5, para6);
        syscallRetValue = %orig(number, para1, para2, para3, para4, para5, para6);
    } else if (7 == paraNum){
        void* para1 = paraList[0];
        void* para2 = paraList[1];
        void* para3 = paraList[2];
        void* para4 = paraList[3];
        void* para5 = paraList[4];
        void* para6 = paraList[5];
        void* para7 = paraList[6];
        iosLogDebug("para1=%p,para2=%p,para3=%p,para4=%p,para5=%p,para6=%p,para7=%p", para1, pa
ra2, para3, para4, para5, para6, para7);
        syscallRetValue = %orig(number, para1, para2, para3, para4, para5, para6, para7);
    } else if (8 == paraNum){
        void* para1 = paraList[0];
        void* para2 = paraList[1];
        void* para3 = paraList[2];
        void* para4 = paraList[3];
        void* para5 = paraList[4];
        void* para6 = paraList[5];
        void* para7 = paraList[6];
        void* para8 = paraList[7];
        iosLogDebug("para1=%p,para2=%p,para3=%p,para4=%p,para5=%p,para6=%p,para7=%p,para8=%p",
para1, para2, para3, para4, para5, para6, para7, para8);
        syscallRetValue = %orig(number, para1, para2, para3, para4, para5, para6, para7, para8);

    } else if (9 == paraNum){
        void* para1 = paraList[0];
        void* para2 = paraList[1];
        void* para3 = paraList[2];
        void* para4 = paraList[3];
        void* para5 = paraList[4];
        void* para6 = paraList[5];
        void* para7 = paraList[6];
        void* para8 = paraList[7];
        void* para9 = paraList[8];
        iosLogDebug("para1=%p,para2=%p,para3=%p,para4=%p,para5=%p,para6=%p,para7=%p,para8=%p,pa
ra9=%p", para1, para2, para3, para4, para5, para6, para7, para8, para9);
        syscallRetValue = %orig(number, para1, para2, para3, para4, para5, para6, para7, para8,
 para9);
    }

    iosLogInfo("number=%d -> syscallRetValue=%d", number, syscallRetValue);
    return syscallRetValue;
}
```

svc 0x80内联汇编

TODO：

- 目前（暂时）无法实现
  - 【未解决】iOS反越狱检测：svc 0x80 call
  - 【未解决】iOS反越狱检测之：svc 0x80的open
  - 【未解决】iOS逆向反越狱检测：插件tweak中绕过内联汇编svc 0x80的系统调用

iOS函数

NSFileManager

TODO：

- 【已解决】iOS反越狱检测：NSFileManager的fileExistsAtPath

```
/*==========================================================================
 Hook: NSFileManager
==========================================================================*/

//@interface NSFileManager (TweakMethods)
//+ (BOOL) isJailbreakPath_iOS: (NSString*)curPath;
//@end

%hook NSFileManager

///*  Common Util Function */
//
//%new
//+ (BOOL) isJailbreakPath_iOS: (NSString*)curPath{
////- (BOOL) isJailbreakPath_iOS: (NSString*)curPath{
//    BOOL isJbPath = FALSE;
//
//    if (NULL != curPath){
//        const char* curPathStr = [curPath UTF8String];
////        isJbPath = isJailbreakPath(curPathStr);
//        const char* FILE_PREFIX = "file://";
//
////        const char* pathNoFilePrefix = removeHead(curPathStr, FILE_PREFIX);
//        char* toFreePtr = NULL;
//        const char* pathNoFilePrefix = removeHead(curPathStr, FILE_PREFIX, &toFreePtr);
//
//        isJbPath = isJailbreakPath(pathNoFilePrefix);
//
////        free(pathNoFilePrefix);
////        if (NULL != toFreePtr) {
//        iosLogDebug("now to free: toFreePtr=%p", toFreePtr);
//        free(toFreePtr);
////        }
//    }
//    iosLogDebug("curPath=%{public}@ -> isJbPath=%s", curPath, boolToStr(isJbPath));
//    return isJbPath;
//}

- (NSArray<NSString *> *)contentsOfDirectoryAtPath:(NSString *)path error:(NSError * _Nullable *)error
{
    iosLogDebug("path=%{public}@, *error=%@", path, ERROR_STR(error));
    NSArray<NSString *> * retContentList = NULL;
    BOOL isJbPath = FALSE;
```

```
    if (cfgHookEnable_openFileiOS) {
        if (NULL != path) {
            isJbPath = [JailbreakiOS isJailbreakPath_iOS: path];
            if (isJbPath){
                retContentList = NULL;
            } else {
//                retContentList = %orig(path, error);
                retContentList = %orig;
            }
        }
    } else {
        retContentList = %orig;
    }

    // for debug
   if (isJbPath){
        iosLogInfo("path=%{public}@, *error=%@ -> isJbPath=%{bool}d -> retContentList=%p", path,
 ERROR_STR(error), isJbPath, retContentList);
   }
    return retContentList;
}

- (BOOL)fileExistsAtPath:(NSString *)path
{
    iosLogDebug("path=%{public}@", path);
    bool isExists = FALSE;
    BOOL isJbPath = FALSE;

    if (cfgHookEnable_openFileiOS) {
        if (NULL != path){
            isJbPath = [JailbreakiOS isJailbreakPath_iOS: path];
            if(isJbPath){
                isExists = FALSE;
            } else{
//                isExists = %orig(path);
                isExists = %orig;
            }
        }
    } else {
        isExists = %orig;
    }

    // for debug
   if (isJbPath){
        iosLogInfo("path=%{public}@ -> isJbPath=%s -> isExists=%s", path, boolToStr(isJbPath),
boolToStr(isExists));
   }

    return isExists;
}

- (BOOL)fileExistsAtPath:(NSString *)path isDirectory:(BOOL *)isDirectory
{
    iosLogDebug("path=%{public}@, isDirectory=%p", path, isDirectory);
    BOOL isJbPath = FALSE;
    BOOL isExists = FALSE;
```

```
    if (cfgHookEnable_openFileiOS) {
        if (NULL != path) {
            isJbPath = [JailbreakiOS isJailbreakPath_iOS: path];
            if(isJbPath){
                isExists = FALSE;
            } else{
//                isExists = %orig(path, isDirectory);
                isExists = %orig;
            }
        }
    } else {
        isExists = %orig;
    }

    // for debug
   if (isJbPath){
        iosLogInfo("path=%{public}@, isDirectory=%p -> isJbPath=%s -> isExists=%s", path, isDir
ectory, boolToStr(isJbPath), boolToStr(isExists));
   }

    return isExists;
}

%end
```

NSURL

```
/*============================================================================
 Hook: NSURL
============================================================================*/

%hook NSURL

- (BOOL)checkResourceIsReachableAndReturnError:(NSError * _Nullable *)error{
    NSString* curUrlStr = [self absoluteString];
    iosLogDebug("curUrlStr=%{public}@, error=%p", curUrlStr, error);
    BOOL isJbPath = FALSE;
    BOOL isReachable = FALSE;

    if (cfgHookEnable_openFileiOS) {
        isJbPath = [JailbreakiOS isJailbreakPath_iOS: curUrlStr];
        if(isJbPath){
            isReachable = FALSE;
        } else{
//            isReachable = %orig(error);
            isReachable = %orig;
        }
    } else {
        isReachable = %orig;
    }

    // for debug
```

```
    if (isJbPath) {
        iosLogInfo("curUrlStr=%{public}@, error=%p -> isJbPath=%s -> isReachable=%s", curUrlStr,
 error, boolToStr(isJbPath), boolToStr(isReachable));
    }
     return isReachable;
}

end
```

文件写入

C函数

iOS函数

TODO：

【已解决】iOS反越狱检测之iOS层函数写入/private的hook绕过

工具类函数

```
bool shouldHookWritePath(const char* path);
bool shouldHookWritePath_NSString(NSString* pathNs);
bool shouldHookWritePath_NSURL(NSURL* url);

/*================================================================================
 Common Functions
================================================================================*/

// /private/testWriteToFile.txt -> true
// /private/var/mobile/Containers/Data/Application/EEFACEA4-2ADB-4D25-9DB4-B5D643EA8943/Documen
ts/bd.turing/ -> false
bool shouldHookWritePath(const char* path){
    const char* Path_Private = "/private/";
    const char* Path_FilePrivate = "file:///private/";

    bool shouldHook = false;

    char* purePath = toPurePath(path);
    iosLogDebug("path=%{public}s -> purePath=%s", path, purePath);
    bool isStartWithPrivate = strStartsWith(purePath, Path_Private);
    bool isStartWithFilePrivate = strStartsWith(purePath, Path_FilePrivate);
    iosLogDebug("isStartWithPrivate=%s, isStartWithFilePrivate=%s",boolToStr(isStartWithPrivate)
, boolToStr(isStartWithFilePrivate));

    if (isStartWithPrivate || isStartWithFilePrivate){
        // is /private/ path
        char* pathNoPrivateHead = NULL;
//        int origMallocStrPointerMovePrevLen = 0;
        char* toFreeRemoveHeadPathPrivate = NULL;
        char* toFreeRemoveHeadPathFilePrivate = NULL;

        if(isStartWithPrivate){
//            pathNoPrivateHead = removeHead(purePath, Path_Private);
//            origMallocStrPointerMovePrevLen = strlen(Path_Private);
            pathNoPrivateHead = removeHead(purePath, Path_Private, &toFreeRemoveHeadPathPrivate)
;
        }

        if(isStartWithFilePrivate){
//            pathNoPrivateHead = removeHead(purePath, Path_FilePrivate);
//            origMallocStrPointerMovePrevLen = strlen(Path_FilePrivate);
            pathNoPrivateHead = removeHead(purePath, Path_FilePrivate, &toFreeRemoveHeadPathFil
ePrivate);
```

```
        }
//        iosLogDebug("purePath=%s -> pathNoPrivateHead=%s, origMallocStrPointerMovePrevLen=%d"
, purePath, pathNoPrivateHead, origMallocStrPointerMovePrevLen);
        iosLogDebug("purePath=%s -> pathNoPrivateHead=%s, toFreeRemoveHeadPathPrivate=%p, toFre
eRemoveHeadPathFilePrivate=%p", purePath, pathNoPrivateHead, toFreeRemoveHeadPathPrivate, toFre
eRemoveHeadPathFilePrivate);

        // testWriteToFile.txt
        // var/mobile/Containers/Data/Application/EEFACEA4-2ADB-4D25-9DB4-B5D643EA8943/Document
s/xxx
        if (NULL != pathNoPrivateHead){
            char* foundSlash = strstr(pathNoPrivateHead, "/");
            iosLogDebug("foundSlash=%s", foundSlash);
            if (NULL != foundSlash){
                // var/mobile/Containers/Data/Application/EEFACEA4-2ADB-4D25-9DB4-B5D643EA8943/
Documents/xxx
                shouldHook = false;
            } else {
                // testWriteToFile.txt
                shouldHook = true;
            }

//            free(pathNoPrivateHead); // will crash !!!
//            char* toFreePtr = pathNoPrivateHead - origMallocStrPointerMovePrevLen;
//            iosLogDebug("pathNoPrivateHead=%p, toFreePtr=%p", pathNoPrivateHead, toFreePtr);
//            free(toFreePtr);
            if (NULL != toFreeRemoveHeadPathPrivate){
                free(toFreeRemoveHeadPathPrivate);
                iosLogDebug("has free toFreeRemoveHeadPathPrivate=%p", toFreeRemoveHeadPathPriv
ate);
            }

            if (NULL != toFreeRemoveHeadPathFilePrivate){
                free(toFreeRemoveHeadPathFilePrivate);
                iosLogDebug("has free toFreeRemoveHeadPathFilePrivate=%p", toFreeRemoveHeadPath
FilePrivate);
            }
        } else {
            shouldHook = false;
        }
    } else {
        // not /private/ path
        shouldHook = false;
    }

    free(purePath);

    // for debug
    if (shouldHook) {
        iosLogInfo("path=%{public}s -> shouldHook=%s", path, boolToStr(shouldHook));
        // /private/testWriteToFile.txt
    }

//    // for debug
//    shouldHook = false;
```

```
    return shouldHook;
}

bool shouldHookWritePath_NSString(NSString* pathNs){
    const char* pathStr = [pathNs UTF8String];
    BOOL shouldHook = shouldHookWritePath(pathStr);

//    // for debug
//    shouldHook = false;

    iosLogDebug("pathNs=%@ -> shouldHook=%s", pathNs, boolToStr(shouldHook));
    return shouldHook;
}

bool shouldHookWritePath_NSURL(NSURL* url){
    NSString *urlNSStr = [url absoluteString];
    const char* urlStr = [urlNSStr UTF8String];
    BOOL shouldHook = shouldHookWritePath(urlStr);

//    // for debug
//    shouldHook = false;

    iosLogDebug("url=%@ -> shouldHook=%s", url, boolToStr(shouldHook));
    return shouldHook;
}
```

NSString

```
/*=============================================================================
 Hook: NSString
=============================================================================*/

%hook NSString

- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile
{
    BOOL isWriteOk = FALSE;

    if(cfgHookEnable_writeFileiOS){
        if(shouldHookWritePath_NSString(path)){
            isWriteOk = FALSE;
            iosLogInfo("hooked path=%{public}@ -> isWriteOk=%s", path, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(path, useAuxiliaryFile);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(path, useAuxiliaryFile);
        isWriteOk = %orig;
    }

    iosLogDebug("%spath=%{public}@, useAuxiliaryFile=%s -> isWriteOk=%s", HOOK_PREFIX(cfgHookEn
able_writeFileiOS), path, boolToStr(useAuxiliaryFile), boolToStr(isWriteOk));
```

```
    return isWriteOk;
}

- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile encoding:(NSStringEncodi
ng)enc error:(NSError **)error{
    iosLogDebug("path=%@, useAuxiliaryFile=%s, enc=%ld, *error=%@", path, boolToStr(useAuxiliar
yFile), enc, ERROR_STR(error));
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSString(path)){
            isWriteOk = FALSE;
            iosLogInfo("hooked path=%{public}@ -> isWriteOk=%s", path, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(path, useAuxiliaryFile, enc, error);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(path, useAuxiliaryFile, enc, error);
        isWriteOk = %orig;
    }
    iosLogDebug("%spath=%{public}@, useAuxiliaryFile=%s, enc=%lu, *error=%@-> isWriteOk=%s", HO
OK_PREFIX(cfgHookEnable_writeFileiOS), path, boolToStr(useAuxiliaryFile), enc, ERROR_STR(error),
 boolToStr(isWriteOk));
    return isWriteOk;
}

- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)atomically{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isWriteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isWriteOk=%s", url, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(url, atomically);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(url, atomically);
        isWriteOk = %orig;
    }
    iosLogDebug("%surl=%{public}@, atomically=%s -> isWriteOk=%s", HOOK_PREFIX(cfgHookEnable_wr
iteFileiOS), url, boolToStr(atomically), boolToStr(isWriteOk));
    return isWriteOk;
}

- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)useAuxiliaryFile encoding:(NSStringEncoding)en
c error:(NSError **)error{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isWriteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isWriteOk=%s", url, boolToStr(isWriteOk));
        } else {
```

```
//            isWriteOk = %orig(url, useAuxiliaryFile, enc, error);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(url, useAuxiliaryFile, enc, error);
        isWriteOk = %orig;
    }
    iosLogDebug("%surl=%{public}@, useAuxiliaryFile=%s, enc=%lu, *error=%@-> isWriteOk=%s", HOO
K_PREFIX(cfgHookEnable_writeFileiOS), url, boolToStr(useAuxiliaryFile), enc, ERROR_STR(error),
boolToStr(isWriteOk));
    return isWriteOk;
}

%end
```

NSData

```
/*============================================================================
 Hook: NSData
============================================================================*/

%hook NSData

- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)atomically{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isWriteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isWriteOk=%s", url, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(url, atomically);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(url, atomically);
        isWriteOk = %orig;
    }
    iosLogDebug("%surl=%{public}@, atomically=%s -> isWriteOk=%s", HOOK_PREFIX(cfgHookEnable_wr
iteFileiOS), url, boolToStr(atomically), boolToStr(isWriteOk));
    return isWriteOk;
}

//- (BOOL)writeToFile:(NSString *)path options:(NSDataWritingOptions)writeOptionsMask error:(NS
Error **)errorPtr{
- (BOOL)writeToFile:(NSString *)path options:(NSDataWritingOptions)writeOptionsMask error:(NSEr
ror **)error{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSString(path)){
            isWriteOk = FALSE;
            iosLogInfo("hooked path=%{public}@ -> isWriteOk=%s", path, boolToStr(isWriteOk));
```

```
        } else {
//            isWriteOk = %orig(path, writeOptionsMask, error);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(path, writeOptionsMask, error);
        isWriteOk = %orig;
    }
    iosLogDebug("%spath=%{public}@, writeOptionsMask=0x%lx, *error=%@-> isWriteOk=%s", HOOK_PRE
FIX(cfgHookEnable_writeFileiOS), path, writeOptionsMask, ERROR_STR(error), boolToStr(isWriteOk))
;
    return isWriteOk;
}

//- (BOOL)writeToURL:(NSURL *)url options:(NSDataWritingOptions)writeOptionsMask error:(NSError
 **)errorPtr{
- (BOOL)writeToURL:(NSURL *)url options:(NSDataWritingOptions)writeOptionsMask error:(NSError **
)error{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isWriteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isWriteOk=%s", url, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(url, writeOptionsMask, error);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(url, writeOptionsMask, error);
        isWriteOk = %orig;
    }
    iosLogDebug("%surl=%{public}@, writeOptionsMask=0x%lx, *error=%@-> isWriteOk=%s", HOOK_PREF
IX(cfgHookEnable_writeFileiOS), url, writeOptionsMask, ERROR_STR(error), boolToStr(isWriteOk));
    return isWriteOk;
}

%end
```

NSArray

```
/*========================================================================
 Hook: NSArray
========================================================================*/

%hook NSArray

- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSString(path)){
            isWriteOk = FALSE;
```

```
            iosLogInfo("hooked path=%{public}@ -> isWriteOk=%s", path, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(path, useAuxiliaryFile);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(path, useAuxiliaryFile);
        isWriteOk = %orig;
    }
    iosLogDebug("%spath=%{public}@, useAuxiliaryFile=%s -> isWriteOk=%s", HOOK_PREFIX(cfgHookEn
able_writeFileiOS), path, boolToStr(useAuxiliaryFile), boolToStr(isWriteOk));
    return isWriteOk;
}

- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)atomically{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isWriteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isWriteOk=%s", url, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(url, atomically);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(url, atomically);
        isWriteOk = %orig;
    }
    iosLogDebug("%surl=%{public}@, atomically=%s -> isWriteOk=%s", HOOK_PREFIX(cfgHookEnable_wr
iteFileiOS), url, boolToStr(atomically), boolToStr(isWriteOk));
    return isWriteOk;
}

- (BOOL)writeToURL:(NSURL *)url error:(NSError **)error{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isWriteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isWriteOk=%s", url, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(url, error);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(url, error);
        isWriteOk = %orig;
    }
    iosLogDebug("%surl=%{public}@, *error=%@ -> isWriteOk=%s", HOOK_PREFIX(cfgHookEnable_writeF
ileiOS), url, ERROR_STR(error), boolToStr(isWriteOk));
    return isWriteOk;
}

%end
```

NSDictionary

```
/*======================================================================
 Hook: NSDictionary
======================================================================*/

%hook NSDictionary

- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSString(path)){
            isWriteOk = FALSE;
            iosLogInfo("hooked path=%{public}@ -> isWriteOk=%s", path, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(path, useAuxiliaryFile);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(path, useAuxiliaryFile);
        isWriteOk = %orig;
    }
    iosLogDebug("%spath=%{public}@, useAuxiliaryFile=%s -> isWriteOk=%s", HOOK_PREFIX(cfgHookEn
able_writeFileiOS), path, boolToStr(useAuxiliaryFile), boolToStr(isWriteOk));
    return isWriteOk;
}

- (BOOL)writeToURL:(NSURL *)url error:(NSError **)error{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isWriteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isWriteOk=%s", url, boolToStr(isWriteOk));
        } else {
//            isWriteOk = %orig(url, error);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(url, error);
        isWriteOk = %orig;
    }
    iosLogDebug("%surl=%{public}@, *error=%@ -> isWriteOk=%s", HOOK_PREFIX(cfgHookEnable_writeF
ileiOS), url, ERROR_STR(error), boolToStr(isWriteOk));
    return isWriteOk;
}

- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)atomically{
    BOOL isWriteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isWriteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isWriteOk=%s", url, boolToStr(isWriteOk));
```

```
        } else {
//            isWriteOk = %orig(url, atomically);
            isWriteOk = %orig;
        }
    } else {
//        isWriteOk = %orig(url, atomically);
        isWriteOk = %orig;
    }
    iosLogDebug("%surl=%{public}@, atomically=%s -> isWriteOk=%s", HOOK_PREFIX(cfgHookEnable_wr
iteFileiOS), url, boolToStr(atomically), boolToStr(isWriteOk));
    return isWriteOk;
}

%end
```

NSFileManager

```
/*============================================================================
 Hook: NSFileManager
============================================================================*/

%hook NSFileManager

- (BOOL)removeItemAtPath:(NSString *)path error:(NSError **)error {
    BOOL isDeleteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSString(path)){
            isDeleteOk = FALSE;
            iosLogInfo("hooked path=%{public}@ -> isDeleteOk=%s", path, boolToStr(isDeleteOk));
        } else {
//            isDeleteOk = %orig(path, error);
            isDeleteOk = %orig;
        }
    } else {
//        isDeleteOk = %orig(path, error);
        isDeleteOk = %orig;
    }
    iosLogDebug("%spath=%{public}@, *error=%@-> isDeleteOk=%s", HOOK_PREFIX(cfgHookEnable_write
FileiOS), path, ERROR_STR(error), boolToStr(isDeleteOk));
    return isDeleteOk;
}

//- (BOOL)removeItemAtURL:(NSURL *)URL error:(NSError **)error {
- (BOOL)removeItemAtURL:(NSURL *)url error:(NSError **)error {
    BOOL isDeleteOk = FALSE;

    if (cfgHookEnable_writeFileiOS) {
        if(shouldHookWritePath_NSURL(url)){
            isDeleteOk = FALSE;
            iosLogInfo("hooked url=%{public}@ -> isDeleteOk=%s", url, boolToStr(isDeleteOk));
        } else {
//            isDeleteOk = %orig(url, error);
            isDeleteOk = %orig;
```

```
        }
    } else {
//        isDeleteOk = %orig(url, error);
        isDeleteOk = %orig;
    }
    iosLogDebug("%s url=%{public}@, *error=%@-> isDeleteOk=%s", HOOK_PREFIX(cfgHookEnable_writeFileiOS), url, ERROR_STR(error), boolToStr(isDeleteOk));
    return isDeleteOk;
}

%end
```

环境变量

```
/*============================================================================
 Hook: getenv(DYLD_INSERT_LIBRARIES)
============================================================================*/

char * getenv(const char* name);
const char* DYLD_INSERT_LIBRARIES = "DYLD_INSERT_LIBRARIES";

%hookf(char *, getenv, const char* name){
    //    char* getenvRetStr = %orig(name);
    char* getenvRetStr = %orig;

    if (cfgHookEnable_misc) {
        //    iosLogDebug("name=%s", name);
        //    NSLog(@"getenv name");

        // "_CFXNOTIFICATIONREGISTAR2_ENABLED" will cause crash
        if (strStartsWith(name, "DYLD_")){
//        if (!strStartsWith(name, "_")){
//            iosLogInfo("not start with '_', name=%s", name);
            iosLogInfo("DYLD_ name=%s", name);
        }

        if(0 == strcmp(name, DYLD_INSERT_LIBRARIES)){
            iosLogInfo("name=%s -> getenvRetStr=%{public}s", name, getenvRetStr);
            getenvRetStr = NULL;
        } else {
            if (strStartsWith(name, "DYLD_")){
                iosLogInfo("name=%s -> getenvRetStr=%{public}s", name, getenvRetStr);
            }
        }
    }

    return getenvRetStr;
}
```

是否可调试

TODO：

- 【已解决】iOS反越狱检测：是否可被调试

sysctl

```
/*========================================================================
 Hook: sysctl
========================================================================*/

int sysctl(int *name, u_int namelen, void *oldp, size_t *oldlenp, void *newp, size_t newlen);

%hookf(int, sysctl, int *name, u_int namelen, void *oldp, size_t *oldlenp, void *newp, size_t n
ewlen){
    iosLogDebug("name=%p, namelen=%d, oldp=%p, oldlenp=%p, newp=%p, newlen=%ld", name, namelen,
 oldp, oldlenp, newp, newlen);

//    int sysctlRet = SYSCTL_FAIL;
//    sysctlRet = %orig(name, namelen, oldp, oldlenp, newp, newlen);
    int sysctlRet = %orig;

    if (cfgHookEnable_sysctl_sysctl) {
        // for Anti-Debug
        bool isGetpid = (name[0] == CTL_KERN && name[1] == KERN_PROC && name[2] == KERN_PROC_PID
);
        if (isGetpid) {
            struct kinfo_proc *info = NULL;
            info = (struct kinfo_proc *)oldp;
            int oldPFlag = info->kp_proc.p_flag;
            info->kp_proc.p_flag &= ~(P_TRACED);
            int newPFlag = info->kp_proc.p_flag;

            iosLogInfo("name=%p, namelen=%d, oldp=%p, oldlenp=%p, newp=%p, newlen=%ld -> isGetp
id=%s -> oldPFlag=0x%x, newPFlag=0x%x -> sysctlRet=%d", name, namelen, oldp, oldlenp, newp, new
len, boolToStr(isGetpid), oldPFlag, newPFlag, sysctlRet);
        }
    }

    return sysctlRet;
}
```

sysctlnametomib

```
/*========================================================================
 Hook: sysctlnametomib
========================================================================*/
```

```
// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/sysctlnametomib.3.html
int sysctlnametomib(const char *name, int *mibp, size_t *sizep);

%hookf(int, sysctlnametomib, const char *name, int *mibp, size_t *sizep){
//    iosLogInfo("name=%p, mibp=%p, sizep=%p", name, mibp, sizep);
    int retInt = SYSCTL_FAIL;
    retInt = %orig;
    iosLogInfo("name=%{public}s, mibp=%p, sizep=%p -> retInt=%d", name, mibp, sizep, retInt);
    return retInt;
}
```

system

TODO：

- 【已解决】iOS反越狱检测：system()
-

沙箱完整性校验

TODO：

- 【已解决】iOS反越狱检测：fork()进程即沙箱完整性检测
- 【已解决】iOS反越狱检测：fork()的hook绕过

```
/*============================================================================
 Hook: fork()
============================================================================*/

pid_t fork(void);

HOOKF(int, fork, void){
    int retForkValue = FORK_FAILED;
    if (cfgHookEnable_misc) {
        retForkValue = FORK_FAILED;
    } else {
        retForkValue = ORIG_fork();
    }
    iosLogInfo("retForkValue=%d", retForkValue);
    return retForkValue;
}
```

越狱相关进程

dyld动态库

_dyld系列

TODO：

- 【已解决】iOS反越狱检测：优化findRealImageCount改为调用_dyld_get_image_vmaddr_slide计算逻辑
- 【已解决】iOS反越狱检测：_dyld_image_count和_dyld_get_image_name返回hook后的值
- 【已解决】iOS反越狱检测：如何hook绕过_dyld_image_count和_dyld_get_image_name
- 【已解决】iOS反越狱检测：优化findRealImageCount改为调用_dyld_get_image_vmaddr_slide计算逻辑
- 【已解决】iOS反越狱检测：_dyld_get_image_name的hook绕过
- 【已解决】iOS反越狱检测：_dyld_image_count和_dyld_get_image_name改为普通hook逻辑
- 【已解决】iOS反越狱检测：dyld的_dyld_image_count和_dyld_get_image_name
- 【已解决】iOS反越狱检测：_dyld_register_func_for_add_image和_dyld_register_func_for_remove_image
-
- 【已解决】反越狱检测测试抖音：优化dyld的hook逻辑

相关工具函数

```
/*============================================================================
Hook: _dyld_image_count(), _dyld_get_image_name(), _dyld_get_image_header(), _dyld_get_image_vmaddr_slide()
==============================================================================*/

/*
https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/dyld.3.html
   _dyld_image_count,
   _dyld_get_image_header,
   _dyld_get_image_vmaddr_slide,
   _dyld_get_image_name,
   _dyld_register_func_for_add_image,
   _dyld_register_func_for_remove_image,
   NSVersionOfRunTimeLibrary,
   NSVersionOfLinkTimeLibrary,
   _NSGetExecutablePath
*/

uint32_t _dyld_image_count(void);
//uint32_t orig__dyld_image_count(void);
//uint32_t _logos_orig$_ungrouped$_dyld_image_count(void);
//uint32_t (*_logos_orig$_ungrouped$_dyld_image_count)(void);
//static uint32_t (*_logos_orig$_ungrouped$_dyld_image_count)(void);

const struct mach_header* _dyld_get_image_header(uint32_t image_index);
const char* _dyld_get_image_name(uint32_t image_index);
intptr_t _dyld_get_image_vmaddr_slide(uint32_t image_index);
```

```
void _dyld_register_func_for_add_image(void (*func)(const struct mach_header* mh, intptr_t vmad
dr_slide));
void _dyld_register_func_for_remove_image(void (*func)(const struct mach_header* mh, intptr_t v
maddr_slide));

int32_t NSVersionOfRunTimeLibrary(const char* libraryName);

int32_t NSVersionOfLinkTimeLibrary(const char* libraryName);

int _NSGetExecutablePath(char* buf, uint32_t* bufsize);

const int IMAGE_INDEX_FAKE_END = IMAGE_INDEX_FAKE_START + IMAGE_INDEX_MAX_VALID_NUMBER;

// Global Variable
int gOrigImageCount = -1;
int gHookedImageCount = -1;
int gRealOrigImageCount = -1; // after hooked, image name/header/slide got hooked image count -
> so need find real original image count

int* gJbDylibIdxList = NULL;
int gJbDylibIdxListLen = -1;

int* gHookedImgIdxList = NULL;
int gHookedImgIdxListLen = -1;

static int generateFakeImageIndex(int origImageIndex){
    int fakeImgIdx = origImageIndex + IMAGE_INDEX_FAKE_START;
    iosLogDebug("generateFakeImageIndex: origImageIndex=%d -> fakeImgIdx=%d", origImageIndex, fa
keImgIdx);
    return fakeImgIdx;
}

static bool isFakeImageIndex(int curImageIndex){
    bool isFakeIdx = (curImageIndex >= IMAGE_INDEX_FAKE_START) && (curImageIndex < IMAGE_INDEX_F
AKE_END);
    iosLogDebug("curImageIndex=%d -> isFakeIdx=%s", curImageIndex, boolToStr(isFakeIdx));
    return isFakeIdx;
}

static int fakeToRealImageIndex(int fakeImgageIndex){
    int realImageIndex = fakeImgageIndex - IMAGE_INDEX_FAKE_START;
    iosLogDebug("fakeImgageIndex=%d -> realImageIndex=%d", fakeImgageIndex, realImageIndex);
    return realImageIndex;
}

static void dbgPrintImgIdxList(int* imgIdxList){
    iosLogDebug("imgIdxList=%p", imgIdxList);

    if (NULL != imgIdxList){
        int curListIdx = 0;
        int curIdxValue = DYLD_IMAGE_INDEX_INVALID;
        curIdxValue = imgIdxList[curListIdx];
        if (DYLD_IMAGE_INDEX_INVALID == curIdxValue) {
            iosLogDebug("[%d] %d", curListIdx, curIdxValue);
        }
```

```
        while(DYLD_IMAGE_INDEX_INVALID != curIdxValue){
            iosLogDebug("[%d] %d", curListIdx, curIdxValue);

            ++curListIdx;
            curIdxValue = imgIdxList[curListIdx];
        }

        int listCount = curListIdx;
        iosLogDebug("end listCount=%d", listCount);
    }
}

static void getJbDylibImgIdxList(int origImageCount, int** outJbDylibIdxList, int* jbDylibIdxL
istLen){
    iosLogDebug("origImageCount=%d", origImageCount);

    int intSize = sizeof(int);
    int mallocCount = IMAGE_INDEX_MAX_JAILBREAK + 1;
    int mallocSize = intSize * mallocCount;
    iosLogDebug("intSize=%d, mallocCount=%d, mallocSize=%d", intSize, mallocCount, mallocSize);

    int curListIdx = 0;

    int* jbDylibIdxList = (int *)malloc(mallocSize);
    iosLogDebug("jbDylibIdxList=%p", jbDylibIdxList);

    if (NULL != jbDylibIdxList) {
        for (int origImgIdx = 0 ; origImgIdx < origImageCount; ++origImgIdx) {
            int fakeImgIdx = generateFakeImageIndex(origImgIdx);
            iosLogDebug("origImgIdx=%d, fakeImgIdx=%d", origImgIdx, fakeImgIdx);
            const char* curImageName = _dyld_get_image_name(fakeImgIdx);
            iosLogDebug("curImageName=%{public}s", curImageName);

            bool isJbDylib = isJailbreakDylib(curImageName);
            iosLogDebug("isJbDylib=%s", boolToStr(isJbDylib));

            if(isJbDylib){
                jbDylibIdxList[curListIdx] = origImgIdx;
                iosLogInfo("curImageName=%{public}s -> origImgIdx=%d, jbDylibIdxList[%d]=%d", cu
rImageName, origImgIdx, curListIdx, jbDylibIdxList[curListIdx]);
                ++curListIdx;
            }
        }

        int curListCount = curListIdx;

        if (jbDylibIdxListLen) {
            *jbDylibIdxListLen = curListCount;
            iosLogDebug("*jbDylibIdxListLen=%d", *jbDylibIdxListLen);
        }

        int curListEndIdx = curListCount;
        jbDylibIdxList[curListEndIdx] = DYLD_IMAGE_INDEX_INVALID;
        iosLogDebug("list end, jbDylibIdxList[%d]=%d", curListEndIdx, jbDylibIdxList[curListEndI
dx]);
```

```
        dbgPrintImgIdxList(jbDylibIdxList);

        if (outJbDylibIdxList) {
            // Note: here for 0 jailbreak dylib, also means get OK
            *outJbDylibIdxList = jbDylibIdxList;
        }
    }

    iosLogInfo("origImageCount=%d -> outJbDylibIdxList=%p, *outJbDylibIdxList=%p, jbDylibIdxList
=%p, *jbDylibIdxListLen=%d", origImageCount, outJbDylibIdxList, outJbDylibIdxList ? *outJbDylib
IdxList : NULL, jbDylibIdxList, jbDylibIdxListLen ? *jbDylibIdxListLen : 0);
}

static void initDylibImageIdxList(void) {
    // init for _dyld_image_count and related
    if (cfgCurDyldHookType == DYLD_HOOK_COMPLEX){
        getJbDylibImgIdxList(&gOrigImageCount, &gJbDylibIdxList, &gJbDylibIdxListLen);
        gHookedImageCount = gOrigImageCount - gJbDylibIdxListLen;
        iosLogInfo("gOrigImageCount=%d, gJbDylibIdxList=%p, gJbDylibIdxListLen=%d -> gHookedImag
eCount=%d", gOrigImageCount, gJbDylibIdxList, gJbDylibIdxListLen, gHookedImageCount);
    }
}

static void generateHookedImageIndexList(int* jbDylibIdxList, int jbDylibIdxListLen, int origIm
gageCount, int** outHookedImgIdxList, int* outHookedImgIdxListLen){
    int* hookedImgIdxList = (int*)malloc(sizeof(int) * (origImgageCount + 1));
    int curListIdx = 0;
    for(int curImgIdx = 0; curImgIdx < origImgageCount; curImgIdx++){
        bool isJbDylibIdx = false;
        if (jbDylibIdxListLen > 0){
            isJbDylibIdx = isIntInList(curImgIdx, jbDylibIdxList, jbDylibIdxListLen);
        }

        iosLogDebug("curImgIdx=%d, isJbDylibIdx=%s", curImgIdx, boolToStr(isJbDylibIdx));

        if( isJbDylibIdx){
            hookedImgIdxList[curListIdx] = curImgIdx;
            ++curListIdx;
        }
    }

    int hookedImgIdxListLen = curListIdx;
    // set end
    int hookedImgIdxListEndIdx = hookedImgIdxListLen;
    hookedImgIdxList[hookedImgIdxListEndIdx] = DYLD_IMAGE_INDEX_INVALID;

    dbgPrintImgIdxList(hookedImgIdxList);

    // return result
    *outHookedImgIdxList = hookedImgIdxList;
    *outHookedImgIdxListLen = hookedImgIdxListLen;

    iosLogInfo("-> outHookedImgIdxList=%p, *outHookedImgIdxList=%p, *outHookedImgIdxListLen=%d",
 outHookedImgIdxList, *outHookedImgIdxList, *outHookedImgIdxListLen);
}
```

```
static void reInitImgCountIfNeed(int curOrigCount) {
    if (curOrigCount != gOrigImageCount) {
        iosLogInfo("curOrigCount=%d != gOrigImageCount=%d, need init", curOrigCount, gOrigImageC
ount);
        gOrigImageCount = curOrigCount;
        initDylibImageIdxList();

        bool foundJbLib = (NULL != gJbDylibIdxList) && (gJbDylibIdxListLen > 0);
        if (foundJbLib) {
            generateHookedImageIndexList(gJbDylibIdxList, gJbDylibIdxListLen, gOrigImageCount, &
gHookedImgIdxList, &gHookedImgIdxListLen);
        }
    }
}

// static int hookedToOrigImageIndex(int hookedImageIndex, int* jbDylibIdxList, int jbDylibIdxL
istLen, int origImgageCount){
static int hookedToOrigImageIndex(int hookedImageIndex){
    int origImgIdx = DYLD_IMAGE_INDEX_INVALID;

    // int* hookedImgIdxList = NULL;
    // int hookedImgIdxListLen = 0;
    // generateHookedImageIndexList(jbDylibIdxList, jbDylibIdxListLen, origImgageCount, &hookedI
mgIdxList, &hookedImgIdxListLen);

    // int hookedImgIdxListMaxIdx = hookedImgIdxListLen - 1;
    int hookedImgIdxListMaxIdx = gHookedImgIdxListLen - 1;
    iosLogDebug("hookedImgIdxListMaxIdx=%d", hookedImgIdxListMaxIdx);

    if (hookedImageIndex <= hookedImgIdxListMaxIdx){
        // origImgIdx = hookedImgIdxList[hookedImageIndex];
        origImgIdx = gHookedImgIdxList[hookedImageIndex];
        iosLogDebug("hookedImageIndex=%d <= hookedImgIdxListMaxIdx=%d -> origImgIdx=%d", hookedI
mageIndex, hookedImgIdxListMaxIdx, origImgIdx);
    } else {
        origImgIdx = DYLD_IMAGE_INDEX_INVALID;
        iosLogDebug("hookedImageIndex=%d > hookedImgIdxListMaxIdx=%d -> origImgIdx=%d", hookedIm
ageIndex, hookedImgIdxListMaxIdx, origImgIdx);
    }

    // if (NULL != hookedImgIdxList){
    //     free(hookedImgIdxList);
    // }

    return origImgIdx;
}

static int findRealImageCount(void){
    iosLogDebug("%s", "");

    int realImageCount = 0;
    int hookedImageCount = _dyld_image_count();
//        int origImageCount = orig__dyld_image_count();
    iosLogDebug("hookedImageCount=%d", hookedImageCount);
```

```
    // find real count
    int curImgIdx = hookedImageCount;

    // use: _dyld_get_image_vmaddr_slide

    long retSlide = _dyld_get_image_vmaddr_slide(generateFakeImageIndex(curImgIdx));
    iosLogDebug("[%d] -> retSlide=%ld", curImgIdx, retSlide);
    while(DYLD_IMAGE_SLIDE_INVALID != retSlide){
        ++curImgIdx;
        retSlide = _dyld_get_image_vmaddr_slide(generateFakeImageIndex(curImgIdx));
        iosLogDebug("[%d] -> retSlide=%ld", curImgIdx, retSlide);
    }

    // // use: _dyld_get_image_name
    // const char* retImgName = _dyld_get_image_name(generateFakeImageIndex(curImgIdx));
    // iosLogDebug("[%d] -> retImgName=%s", curImgIdx, retImgName);
    // while(NULL != retImgName){
    //     ++curImgIdx;
    //     retImgName = _dyld_get_image_name(generateFakeImageIndex(curImgIdx));
    //     iosLogDebug("[%d] -> retImgName=%s", curImgIdx, retImgName);
    // }

    // // use: _dyld_get_image_header
    // const struct mach_header* retImgHeader = _dyld_get_image_header(generateFakeImageIndex(cu
rImgIdx));
    // iosLogDebug("[%d] -> retImgHeader=%p", curImgIdx, retImgHeader);
    // while(NULL != retImgHeader){
    //     ++curImgIdx;
    //     retImgHeader = _dyld_get_image_header(generateFakeImageIndex(curImgIdx));
    //     iosLogDebug("[%d] -> retImgHeader=%p", curImgIdx, retImgHeader);
    // }

    realImageCount = curImgIdx;
    iosLogDebug("realImageCount=%d", realImageCount);
    return realImageCount;
}

static void reInitAllRelated(void) {
    // if (gRealOrigImageCount <= 0) {
    //     // invalid, need reinit
    //     gRealOrigImageCount = findRealImageCount();
    //     iosLogInfo("gRealOrigImageCount=%d", gRealOrigImageCount);
    // }

    int curOrigCount = -1;

    int curHookedImageCount = _dyld_image_count();
    if (gJbDylibIdxListLen > 0) {
        curOrigCount = curHookedImageCount - gJbDylibIdxListLen;
    } else {
        curOrigCount = findRealImageCount();
    }
    iosLogDebug("curHookedImageCount=%d, gJbDylibIdxListLen=%d -> curOrigCount=%d", curHookedIma
geCount, gJbDylibIdxListLen, curOrigCount);

    if (curOrigCount != gOrigImageCount) {
```

```
        iosLogInfo("curOrigCount=%d != gOrigImageCount=%d -> reinit image index list", curOrigCo
unt, gOrigImageCount);

        reInitImgCountIfNeed(curOrigCount);
        iosLogInfo("after reinit, gOrigImageCount=%d, gHookedImgIdxList=%p, gHookedImgIdxListLen
=%d", gOrigImageCount, gHookedImgIdxList, gHookedImgIdxListLen);
    } else {
        iosLogDebug("gJbDylibIdxList=%p, gJbDylibIdxListLen=%d, gHookedImgIdxList=%p, gHookedImg
IdxListLen=%d", gJbDylibIdxList, gJbDylibIdxListLen, gHookedImgIdxList, gHookedImgIdxListLen);

        if ((NULL == gJbDylibIdxList) || (gJbDylibIdxListLen <= 0)) {
            reInitImgCountIfNeed(curOrigCount);
        }

        if ((NULL == gHookedImgIdxList) || (gHookedImgIdxListLen <= 0)) {
            generateHookedImageIndexList(gJbDylibIdxList, gJbDylibIdxListLen, curOrigCount, &gHo
okedImgIdxList, &gHookedImgIdxListLen);
        }
    }
}

static int getOrigImageIndex(int hookedImageIndex){
    iosLogDebug("hookedImageIndex=%d", hookedImageIndex);

    int origImgIdx = DYLD_IMAGE_INDEX_INVALID;

    // uint32_t origImageCount_byLogos = (*_logos_orig$_ungrouped$_dyld_image_count)();
    // os_log(OS_LOG_DEFAULT, "hook_dyld getOrigImageIndex: origImageCount_byLogos=%d", origImag
eCount_byLogos);

//     uint32_t origImageCount = findRealImageCount();
//     iosLogDebug("origImageCount=%d", origImageCount);
//
//    int* jbDylibImgIdxList = NULL;
//    int jbDylibImgIdxListLen = -1;
//    getJbDylibImgIdxList(origImageCount, &jbDylibImgIdxList, &jbDylibImgIdxListLen);
//    iosLogDebug("jbDylibImgIdxList=%p,jbDylibImgIdxListLen=%d", jbDylibImgIdxList, jbDylibImg
IdxListLen);

    reInitAllRelated();

    // if (jbDylibImgIdxListLen > 0){
    if (gJbDylibIdxListLen > 0){
        // check input image index validation
        // int origImgMaxIdx = origImageCount - 1;
        // int origImgMaxIdx = gRealOrigImageCount - 1;
        int origImgMaxIdx = gOrigImageCount - 1;
        // int hookedImgMaxIdx = origImgMaxIdx - jbDylibImgIdxListLen;
        int hookedImgMaxIdx = origImgMaxIdx - gJbDylibIdxListLen;
        iosLogDebug("origImgMaxIdx=%d, hookedImgMaxIdx=%d, hookedImageIndex=%d", origImgMaxIdx,
hookedImgMaxIdx, hookedImageIndex);

        if(hookedImageIndex > hookedImgMaxIdx){
            // invalid
            iosLogError("input image index invalid, hookedImageIndex=%d > hookedImgMaxIdx=%d", h
ookedImageIndex, hookedImgMaxIdx);
```

```
            origImgIdx = DYLD_IMAGE_INDEX_INVALID;
        } else {
            // valid
            // origImgIdx = hookedToOrigImageIndex(hookedImageIndex, jbDylibImgIdxList, jbDylibI
mgIdxListLen,  origImageCount);
            origImgIdx = hookedToOrigImageIndex(hookedImageIndex);
        }
    } else {
        // no jailbreak dylib image index list
        origImgIdx = hookedImageIndex;
    }

    iosLogDebug("hookedImageIndex=%d -> origImgIdx=%d", hookedImageIndex, origImgIdx);

    // if (NULL != jbDylibImgIdxList){
    //     free(jbDylibImgIdxList);
    // }

    return origImgIdx;
}

static int getRealOrOrigImageIndex(int inputImageIndex){
    iosLogDebug("inputImageIndex=%d", inputImageIndex);

    int realOrOrigImgIdx = DYLD_IMAGE_INDEX_INVALID;

    bool isFakeImgIdx = isFakeImageIndex(inputImageIndex);
    iosLogDebug("isFakeImgIdx=%s", boolToStr(isFakeImgIdx));

    if (isFakeImgIdx){
        realOrOrigImgIdx = fakeToRealImageIndex(inputImageIndex);
    } else {
        realOrOrigImgIdx = getOrigImageIndex(inputImageIndex);
    }

    iosLogDebug("inputImageIndex=%d -> realOrOrigImgIdx=%d", inputImageIndex, realOrOrigImgIdx);

    return realOrOrigImgIdx;
}
```

_dyld_image_count

```
%hookf(uint32_t, _dyld_image_count, void){
//    iosLogDebug();
    iosLogDebug("%s", "");

    uint32_t origCount = 0;
    int retImageCount = 0;

    if (cfgHookEnable_dyld){
        origCount = %orig();
        iosLogDebug("origCount=%d", origCount);
        retImageCount = origCount;
```

```
        if (cfgCurDyldHookType == DYLD_HOOK_COMPLEX){
//            int* jbDylibIdxList = NULL;
//            int jbDylibIdxListLen = -1;
//            getJbDylibImgIdxList(origCount, &jbDylibIdxList, &jbDylibIdxListLen);
//            iosLogDebug("jbDylibIdxList=%p, jbDylibIdxListLen=%d", jbDylibIdxList, jbDylibIdx
ListLen);
//            retImageCount = origCount - jbDylibIdxListLen;
//
//            if(NULL != jbDylibIdxList){
//                free(jbDylibIdxList);
//            }

            reInitImgCountIfNeed(origCount);

            // if ((NULL == gHookedImgIdxList) || (gHookedImgIdxListLen <= 0)) {
            //     generateHookedImageIndexList(gJbDylibIdxList, gJbDylibIdxListLen, origCount,
&gHookedImgIdxList, &gHookedImgIdxListLen);
            // }

            retImageCount = gHookedImageCount;
            // retImageCount = gOrigImageCount;
        }
    } else {
        origCount = %orig();
        retImageCount = origCount;
    }

    iosLogDebug("%sorigCount=%d -> retImageCount=%d", HOOK_PREFIX(cfgHookEnable_dyld), origCount,
 retImageCount);
    return retImageCount;
}
```

_dyld_get_image_name

```
%hookf(const char*, _dyld_get_image_name, uint32_t image_index){
    iosLogDebug("image_index=%d", image_index);
    const char* retImgName = NULL;

    if (cfgHookEnable_dyld){
        if (cfgCurDyldHookType == DYLD_HOOK_COMPLEX){
            int realOrOrigImgIdx = getRealOrOrigImageIndex(image_index);
            bool isValidImgIdx = (realOrOrigImgIdx >= 0);
            if (isValidImgIdx){
                const char* imgName = %orig(realOrOrigImgIdx);
                iosLogDebug("image_index=%d -> realOrOrigImgIdx=%d -> isValidImgIdx=%s -> imgNam
e=%{public}s", image_index, realOrOrigImgIdx, boolToStr(isValidImgIdx), imgName);
                retImgName = imgName;
            } else {
                iosLogError("fail to get real or origin image index for image_index=%d", image_i
ndex);
                retImgName = NULL;
            }
        } else {
```

```
            const char * firstImgName = NULL;
            char* randomDylibName = NULL;
            const char* imgName = %orig(image_index);
            bool isJbDylib = isJailbreakDylib(imgName);
            if (isJbDylib){
                if (cfgCurDyldHookType == DYLD_HOOK_SIMPLE_NULL) {
                    retImgName = NULL;
                } else if (cfgCurDyldHookType == DYLD_HOOK_SIMPLE_FIRST) {
                    firstImgName = _dyld_get_image_name(0);
                    // normally is app self
                    // eg: /private/var/containers/Bundle/Application/B6327617-9ED7-4DED-AFAC-4D
9C92D82377/Aweme.app/Aweme
                    retImgName = firstImgName;
                } else if (cfgCurDyldHookType == DYLD_HOOK_SIMPLE_RANDOM_NAME) {
                    char* randomName = randomStr(10, NULL);
                    asprintf(&randomDylibName, "/usr/lib/%s.dylib", randomName);
                    retImgName = randomDylibName;
                }
            } else {
                retImgName = imgName;
            }

            // for debug
            if (isJbDylib) {
                iosLogInfo("image_index=%d -> imgName=%{public}s -> isJbDylib=%s -> firstImgName
=%{public}s, randomDylibName=%{public}s -> retImgName=%{public}s", image_index, imgName, boolTo
Str(isJbDylib), firstImgName, randomDylibName, retImgName);
            }
        }
    } else {
        retImgName = %orig(image_index);
    }

    iosLogDebug("%simage_index=%d -> retImgName=%{public}s", HOOK_PREFIX(cfgHookEnable_dyld), im
age_index, retImgName);
    return retImgName;
}
```

_dyld_get_image_header

```
%hookf(const struct mach_header*, _dyld_get_image_header, uint32_t image_index){
    iosLogDebug("image_index=%d", image_index);
//    return %orig;
    const struct mach_header* retMachHeader = NULL;

    if (cfgHookEnable_dyld){
        if (cfgCurDyldHookType == DYLD_HOOK_COMPLEX){
            int realOrOrigImgIdx = getRealOrOrigImageIndex(image_index);
            bool isValidImgIdx = (realOrOrigImgIdx >= 0);
            if (isValidImgIdx){
                retMachHeader = %orig(realOrOrigImgIdx);
            } else {
                iosLogError("fail to get real or origin image index for image_index=%d", image_i
ndex);
```

```
                retMachHeader = NULL;
            }
            iosLogDebug("image_index=%d -> realOrOrigImgIdx=%d -> isValidImgIdx=%s -> retMachHea
der=%p", image_index, realOrOrigImgIdx, boolToStr(isValidImgIdx), retMachHeader);
        } else {
            bool isJbDylib = false;
            const struct mach_header* firstImgHeader = NULL;
            const char* imageName = _dyld_get_image_name(image_index);
            if (NULL == imageName){
                retMachHeader = NULL;
            } else {
                isJbDylib = isJailbreakDylib(imageName);
                if (isJbDylib){
                    if (cfgCurDyldHookType == DYLD_HOOK_SIMPLE_NULL) {
                        retMachHeader = NULL;
                    } else if ( (cfgCurDyldHookType == DYLD_HOOK_SIMPLE_FIRST) || (cfgCurDyldHoo
kType == DYLD_HOOK_SIMPLE_RANDOM_NAME) ) {
                        firstImgHeader = _dyld_get_image_header(0);
                        // normally is app self
                        retMachHeader = firstImgHeader;
                    }
                } else {
                    retMachHeader = %orig(image_index);
                }
            }

            // for debug
            if (isJbDylib) {
                iosLogInfo("image_index=%d -> imageName=%{public}s -> isJbDylib=%s -> firstImgHe
ader=%p -> retMachHeader=%p", image_index, imageName, boolToStr(isJbDylib), firstImgHeader, ret
MachHeader);
            }
        }
    } else {
        retMachHeader = %orig(image_index);
//        iosLogDebug("%simage_index=%d -> retMachHeader=%p", HOOK_PREFIX(cfgHookEnable_dyld),
image_index, retMachHeader);
    }

    iosLogDebug("%simage_index=%d -> retMachHeader=%p", HOOK_PREFIX(cfgHookEnable_dyld), image_i
ndex, retMachHeader);
    return retMachHeader;
}
```

_dyld_get_image_vmaddr_slide

```
%hookf(intptr_t, _dyld_get_image_vmaddr_slide, uint32_t image_index){
    iosLogDebug("image_index=%d", image_index);
//    return %orig;
    long retSlide = DYLD_IMAGE_SLIDE_INVALID;

    if (cfgHookEnable_dyld){
        if (cfgCurDyldHookType == DYLD_HOOK_COMPLEX){
            int realOrOrigImgIdx = getRealOrOrigImageIndex(image_index);
```

```
            bool isValidImgIdx = (realOrOrigImgIdx >= 0);
            if (isValidImgIdx){
                retSlide = %orig(realOrOrigImgIdx);
            } else {
                iosLogError("fail to get real or origin image index for image_index=%d", image_i
ndex);
                retSlide = DYLD_IMAGE_SLIDE_INVALID;
            }
            iosLogDebug("image_index=%d -> realOrOrigImgIdx=%d -> isValidImgIdx=%s -> retSlide=0
x%lx", image_index, realOrOrigImgIdx, boolToStr(isValidImgIdx), retSlide);
        } else {
            bool isJbDylib = false;
            long firtImgSlide = DYLD_IMAGE_SLIDE_INVALID;
            const char* imageName = _dyld_get_image_name(image_index);
            if (NULL == imageName){
                retSlide = DYLD_IMAGE_SLIDE_INVALID;
            } else {
                isJbDylib = isJailbreakDylib(imageName);
                if (isJbDylib){
                    if (cfgCurDyldHookType == DYLD_HOOK_SIMPLE_NULL) {
                        retSlide = DYLD_IMAGE_SLIDE_INVALID;
                    } else if ( (cfgCurDyldHookType == DYLD_HOOK_SIMPLE_FIRST) || (cfgCurDyldHoo
kType == DYLD_HOOK_SIMPLE_RANDOM_NAME) ) {
                        firtImgSlide = _dyld_get_image_vmaddr_slide(0);
                        // normally is app self
                        retSlide = firtImgSlide;
                    }
                } else {
                    retSlide = %orig(image_index);
                }
            }

            // for debug
            if (isJbDylib) {
                iosLogInfo("image_index=%d -> imageName=%{public}s -> isJbDylib=%s -> firtImgSli
de=0x%lx -> retSlide=0x%lx", image_index, imageName, boolToStr(isJbDylib), firtImgSlide, retSli
de);
            }
        }
    } else {
        retSlide = %orig(image_index);
//        iosLogDebug("%simage_index=%d -> retSlide=0x%lx", HOOK_PREFIX(cfgHookEnable_dyld), im
age_index, retSlide);
    }

    iosLogDebug("%simage_index=%d -> retSlide=0x%lx", HOOK_PREFIX(cfgHookEnable_dyld), image_ind
ex, retSlide);
    return retSlide;
}
```

其实无法hook，只能调试或注释掉

_dyld_register_func_for_add_image

```
%hookf(void, _dyld_register_func_for_add_image, void (*func)(const struct mach_header* mh, intptr_t vmaddr_slide)){
//    iosLogInfo("%sfunc=%p -> Omitted", HOOK_PREFIX(cfgHookEnable_dyld), func);
   iosLogInfo("%sfunc=%p", HOOK_PREFIX(cfgHookEnable_dyld), func);

//#ifndef XCODE_DEBUG
   %orig;
//    %orig(func);
//#endif
}
```

_dyld_register_func_for_remove_image

```
%hookf(void, _dyld_register_func_for_remove_image, void (*func)(const struct mach_header* mh, intptr_t vmaddr_slide)){
//    iosLogInfo("%sfunc=%p -> Omitted", HOOK_PREFIX(cfgHookEnable_dyld), func);
   iosLogInfo("%sfunc=%p", HOOK_PREFIX(cfgHookEnable_dyld), func);
   %orig;
}
```

其他调试内容

```
%hookf(int32_t, NSVersionOfRunTimeLibrary, const char* libraryName){
   int32_t rtLibVer = %orig;
   iosLogInfo("libraryName=%s -> rtLibVer=%d", libraryName, rtLibVer);
   return rtLibVer;
}

%hookf(int32_t, NSVersionOfLinkTimeLibrary, const char* libraryName){
   int32_t rtLtLibVer = %orig;
   iosLogInfo("libraryName=%s -> rtLtLibVer=%d", libraryName, rtLtLibVer);
   return rtLtLibVer;
}

%hookf(int, _NSGetExecutablePath, char* buf, uint32_t* bufsize){
   int extPathCpSize = %orig;
   iosLogInfo("buf=%{public}s,*bufsize=%d -> extPathCpSize=%d", buf, *bufsize, extPathCpSize);
   return extPathCpSize;
}
```

dylib

TODO：

- 【已解决】iOS反越狱检测：dladdr的hook绕过
- 【已解决】iOS反越狱检测：逆向hook的dlopen+dlsym

---

dladdr

```
/*============================================================================
 Hook: dladdr()
============================================================================*/
// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/dladdr.3.html


/*============================================================================
    hook dladdr()
============================================================================*/

void* generateHookedDladdrAddress(void *origAddr);

const long DLADDR_HOOKED_ADDRESS_BASE = 0xF00000000000;
//const unsigned long DLADDR_HOOKED_ADDRESS_MAX = 0xFFFF000000000000;

void* generateHookedDladdrAddress(void *origAddr) {
//    if ((long)origAddr < (long)DLADDR_HOOKED_ADDRESS_MAX) {
    void* hookedAddr = origAddr;
    if ((long)origAddr > (long)DLADDR_HOOKED_ADDRESS_BASE) {
        hookedAddr = origAddr;
    } else {
        hookedAddr = (void*)((long)origAddr + DLADDR_HOOKED_ADDRESS_BASE);
    }
    return hookedAddr;
}

static bool isHookedDladdrAddress(const void *addr){
    bool isHookedAddr = false;
    long addrLong = (long) addr;
//    if ((addrLong > DLADDR_HOOKED_ADDRESS_BASE) && (addrLong < DLADDR_HOOKED_ADDRESS_MAX)) {
    if (addrLong > DLADDR_HOOKED_ADDRESS_BASE) {
        isHookedAddr = true;
    }

    return isHookedAddr;
}

static void* hookedToOrigDladdrAddr(const void *hookedAddr){
    return (void*) ( (long)hookedAddr - DLADDR_HOOKED_ADDRESS_BASE );
```

```
}

int dladdr(const void *, Dl_info *);
//int dladdr(void *, Dl_info *);
//extern int dladdr(const void *, Dl_info *);

//%hookf(int, dladdr, void *addr, Dl_info *info){
%hookf(int, dladdr, const void *addr, Dl_info *info){
    iosLogDebug("addr=%p,info=%p", addr, info);
    int finalRet = DLADDR_FAILED;

    if (NULL == addr) {
        iosLogInfo("addr is %s", "NULL");
    } else {
        void* origAddr = (void*)addr;

        bool isHookedAddr = isHookedDladdrAddress(addr);
        if (isHookedAddr) {
            origAddr = hookedToOrigDladdrAddr(addr);

            iosLogDebug("addr=%p -> isHookedAddr=%s -> origAddr=%p", addr, boolToStr(isHookedAd
dr), origAddr);

            if (NULL == origAddr) {
                iosLogInfo("addr=%p -> isHookedAddr=%s -> origAddr=%p", addr, boolToStr(isHooke
dAddr), origAddr);
            }
        }

    //    int origRet = %orig;

        //        int origRet = DLADDR_FAILED;
//        if (NULL == origAddr) {
//            origRet = DLADDR_FAILED;
//        } else {
//            origRet = %orig(origAddr, info);
//        }

        int origRet = %orig(origAddr, info);
        finalRet = origRet;

        bool isNotHookedAddr = !isHookedAddr;
        bool isNeedHook = cfgHookEnable_dylib_dladdr && isNotHookedAddr;
        if (isNeedHook) {
            //    if (dladdrRetInt > 0) {
            if (DLADDR_FAILED != origRet) {
                if (NULL != info) {
                    const char* curImageName = info->dli_fname;
                    bool isJbDyib = isJailbreakDylib(curImageName);
                    if (isJbDyib) {
                        finalRet = DLADDR_FAILED;

                        iosLogInfo("addr=%p -> origRet=%d -> dli_fname=%{public}s, dli_fbase=%p
, dli_sname=%{public}s, dli_saddr=%p -> isJbDyib=%s -> finalRet=%d", addr, origRet, info->dli_f
name, info->dli_fbase, info->dli_sname, info->dli_saddr, boolToStr(isJbDyib), finalRet);
    //                    iosLogInfo("isJbDyib=%s", boolToStr(isJbDyib));
```

```
    //                    iosLogInfo("addr=%p -> origRet=%d", addr, origRet);
    //                    iosLogInfo("dli_fname=%{public}s, dli_fbase=%p, dli_sname=%{public}s,
 dli_saddr=%p", info->dli_fname, info->dli_fbase, info->dli_sname, info->dli_saddr);
    //                    iosLogInfo("finalRet=%d", finalRet);

                        size_t dlInfoSize = sizeof(Dl_info);
                        memset(info, 0, dlInfoSize);
                    }
                }
            }
        }
    }

    return finalRet;
}

/*
    TODO:
        https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iP
honeOS/man3/dyld.3.html
        https://man7.org/linux/man-pages/man3/dladdr.3.html
        may need support:
            int dladdr1(const void *addr, Dl_info *info, void **extra_info, int flags);
*/
```

dlopen

```
/*================================================================================
 Hook: dlopen()
================================================================================*/
// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneO
S/man3/dlopen.3.html
void* dlopen(const char* path, int mode);

%hookf(void*, dlopen, const char* path, int mode){
    iosLogDebug("path=%{public}s, mode=0x%x", path, mode);
    void* dlopenRetPtr = NULL;

    if (cfgHookEnable_dylib) {
        bool isJbDylib = isJailbreakDylib(path);
        if (isJbDylib) {
            dlopenRetPtr = NULL;
        } else {
//            dlopenRetPtr = %orig(path, mode);
            dlopenRetPtr = %orig;
        }

        if (isJbDylib) {
            iosLogInfo("path=%{public}s, mode=0x%x -> isJbDylib=%s -> dlopenRetPtr=%p", path, m
ode, boolToStr(isJbDylib), dlopenRetPtr);
        }
    } else {
//        dlopenRetPtr = %orig(path, mode);
```

```
        dlopenRetPtr = %orig;
    }

    return dlopenRetPtr;
}

////void* _dlopen(const char* path, int mode);
//void* __ZL15dlopen_internalPKciPv(const char* path, int mode);
//
////%hookf(void*, _dlopen, const char* path, int mode){
//%hookf(void*, __ZL15dlopen_internalPKciPv, const char* path, int mode){
//    iosLogInfo("path=%{public}s, mode=0x%x", path, mode);
//    return %orig;
//}
```

dlclose

```
/*============================================================================
 Hook: dlclose()
============================================================================*/

int dlclose(void* handle);

%hookf(int, dlclose, void* handle){
    bool isJbLib = false;

    Dl_info info;
    size_t dlInfoSize = sizeof(Dl_info);
    memset(&info, 0, dlInfoSize);

//        dladdr(mhp, &info);
    void* hookedAddr = generateHookedDladdrAddress(handle);
    dladdr(hookedAddr, &info);

    const char* curImgName = info.dli_fname;
    if(curImgName != NULL) {
        isJbLib = isJailbreakDylib(curImgName);
    }

    if (isJbLib) {
        iosLogInfo("handle=%p -> is jb lib: %s", handle, curImgName);
    }

    int closeRet = %orig;
    iosLogInfo("handle=%p -> closeRet=%d", handle, closeRet);
    return closeRet;
}
```

dlsym

```
/*============================================================================
```

```
 Hook: dlsym()
================================================================================*/
// https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/dlsym.3.html
void* dlsym(void* handle, const char* symbol);

%hookf(void*, dlsym, void* handle, const char* symbol) {
    iosLogDebug("handle=%p, symbol=%{public}s", handle, symbol);
    void* dlsymRetPtr = NULL;

    if (cfgHookEnable_dylib) {
        bool shouldHook = false;
        bool isJbFuncName = isJailbreakDylibFunctionName(symbol);
        bool isPtrace = 0 == strcmp(symbol, "ptrace");
        shouldHook = isJbFuncName || isPtrace;
        iosLogDebug("isPtrace=%s, shouldHook=%s", boolToStr(isPtrace), boolToStr(shouldHook));
    //    if (isJbFuncName) {
        if (shouldHook) {
            dlsymRetPtr = NULL;
        } else {
//            dlsymRetPtr = %orig(handle, symbol);
            dlsymRetPtr = %orig;
        }

    //    if (isJbFuncName) {
        if (shouldHook) {
    //        iosLogInfo("handle=%p, symbol=%{public}s -> isJbFuncName=%s -> dlsymRetPtr=%p", handle, symbol, boolToStr(isJbFuncName), dlsymRetPtr);
            iosLogInfo("handle=%p, symbol=%{public}s -> isJbFuncName=%s, isPtrace=%s -> shouldHook=%s -> dlsymRetPtr=%p", handle, symbol, boolToStr(isJbFuncName), boolToStr(isPtrace), boolToStr(shouldHook), dlsymRetPtr);
        }
    } else {
//        dlsymRetPtr = %orig(handle, symbol);
        dlsymRetPtr = %orig;
    }

    return dlsymRetPtr;
}
```

ObjC运行时

objc_copyImageNames

```
/*============================================================================
 Hook: objc_copyImageNames
============================================================================*/

//const char * _Nonnull * objc_copyImageNames(unsigned int *outCount);
const char ** objc_copyImageNames(unsigned int *outCount);

%hookf(const char **, objc_copyImageNames, unsigned int *outCount){
    iosLogInfo("outCount=%p", outCount);
    const char** imageList = %orig(outCount);
    iosLogInfo("*outCount=%d, imageList=%p", *outCount, imageList);
    if (cfgHookEnable_aweme) {
        // TODO: add support

        if ((*outCount > 0) && (imageList != NULL)) {
            for (int i = 0; i < *outCount; i++) {
                const char* curImagePath = imageList[i];
                bool isJbPath = isJailbreakPath(curImagePath);
                if (isJbPath) {
                    iosLogInfo("[%d] %s -> isJbPath=%s", i, curImagePath, boolToStr(isJbPath));
                }
            }
        }
    }
    return imageList;
}
```

最后更新：2022-11-06 16:09:58

app本身

TODO：

- 【未解决】iOS逆向：绕过重签名检测embedded.mobileprovision

NSBundle

```
/*=============================================================================
 Hook: debugging embedded.mobileprovision
=============================================================================*/

// NSString *embeddedPath = [[NSBundle mainBundle] pathForResource:@"embedded" ofType:@"mobilep
rovision"];
%hook NSBundle

- (NSString *)pathForResource:(NSString *)name ofType:(NSString *)ext {
    NSString *resPath = %orig(name, ext);

    if (cfgHookEnable_aweme) {
        if ([ext isEqualToString: @"mobileprovision"]){
            iosLogInfo("name=%{public}@, ext=%{public}@ -> resPath=%{public}@", name, ext, resP
ath);
            if ([name isEqualToString: @"embedded"]){
                resPath = NULL;
            }
        }
    }

    return resPath;
}

// https://developer.apple.com/documentation/foundation/nsbundle/1407973-bundlepath
// @property(readonly, copy) NSString *bundlePath;

- (NSString *)bundlePath {
    NSString *origBundlePath = %orig;
    BOOL shouldOmit = [origBundlePath containsString: @"Aweme"] || [origBundlePath containsStri
ng: @"/System/Library"];
    if (!shouldOmit){
        iosLogInfo("origBundlePath=%{public}@", origBundlePath);
    }
    return origBundlePath;
}

%end
```

getsectiondata

TODO：

- 【已解决】hook函数getsectiondata时不hook函数dladdr看看是否还是导致抖音崩溃

---

getsectiondata的hook

```
/*============================================================================
 Hook: getsectiondata
============================================================================*/

extern uint8_t *getsectiondata(
    const struct mach_header_64 *mhp,
    const char *segname,
    const char *sectname,
    unsigned long *size);

//extern uint8_t *getsectiondata(
//    const struct mach_header *mhp,
//    const char *segname,
//    const char *sectname,
//    unsigned long *size);

%hookf(uint8_t*, getsectiondata, const struct mach_header_64 *mhp, const char *segname, const c
har *sectname, unsigned long *size){
    iosLogDebug("mhp=%p,segname=%{public}s,sectname=%{public}s,size=%p", mhp, segname, sectname,
 size);

    uint8_t* origRetIntP = %orig;

    if (cfgHookEnable_macho) {
        bool isJbLib = false;
        bool isShowLog = false;

        Dl_info info;
        size_t dlInfoSize = sizeof(Dl_info);
        memset(&info, 0, dlInfoSize);

//        dladdr(mhp, &info);
        void* hookedAddr = generateHookedDladdrAddress((void*)mhp);
        dladdr(hookedAddr, &info);

        const char* curImgName = info.dli_fname;
        if(curImgName != NULL) {
            isJbLib = isJailbreakDylib(curImgName);
        }

        if (isJbLib) {
    //        isShowLog = true;
            if( size && (*size > 0) ) {
```

```
                isShowLog = true;

//#ifdef XCODE_DEBUG
                // Note: MUST filter out following log, otherwise Aweme will crash

//                // getsectiondata: mhp=0x114af0000,segname=__TEXT,sectname=__swift5_replace,s
ize=0x16fbf7df8 ===> *size=6169788088, curImgName=/Library/MobileSubstrate/DynamicLibraries/App
SyncUnified-FrontBoard.dylib, isJbLib=True
                if (
                    strstr(curImgName, "AppSyncUnified") && \
                    (0==strcmp(segname, "__TEXT"))
//                    ( (0==strcmp(sectname, "__swift5_replace")) || (0==strcmp(sectname, "__s
wift5_types")) ) \
                ) {
                    isShowLog = false;
                }

                // "/Library/MobileSubstrate/DynamicLibraries/   Choicy.dylib"
                if (strstr(curImgName, "Choicy")) {
                    isShowLog = false;
                }

                // /usr/lib/librocketbootstrap.dylib
                if (strstr(curImgName, "librocketbootstrap")) {
                    isShowLog = false;
                }
//#endif

                if (isShowLog) {
                    iosLogInfo("mhp=%p,segname=%{public}s,sectname=%{public}s,size=%p ===> *siz
e=%lu, curImgName=%{public}s, isJbLib=%s", mhp, segname, sectname, size, size ? *size : 0, curI
mgName, boolToStr(isJbLib));
                }
            }
        }

        if (isJbLib) {
            origRetIntP = NULL;
            if (NULL != size) {
                *size = 0;
            }
        }

    //    if (NULL != size) {
    //        if (*size > 0) {
    //            isShowLog = true;
    //        }
    //    }

    //    if (isShowLog) {
    //        iosLogInfo("mhp=%p,segname=%{public}s,sectname=%{public}s,size=%p ===> *size=%lu,
 curImgName=%{public}s, isJbLib=%s", mhp, segname, sectname, size, size ? *size : 0, curImgName
, boolToStr(isJbLib));
    //    }
    }
```

```
//    // for debug
//    if (origRetIntP != NULL) {
//        printf("origRetIntP=%p", origRetIntP);
//    }

    return origRetIntP;
}
```

getsectbynamefromheader_64

```
const struct section_64* getsectbynamefromheader_64(const struct mach_header_64 *mhp, const char
 *segname, const char *sectname);

%hookf(const struct section_64 *, getsectbynamefromheader_64, const struct mach_header_64 *mhp,
const char *segname, const char *sectname){
    const struct section_64* retSection64 = %orig;

    bool isJbLib = false;

    Dl_info info;
    size_t dlInfoSize = sizeof(Dl_info);
    memset(&info, 0, dlInfoSize);

//        dladdr(mhp, &info);
    void* hookedAddr = generateHookedDladdrAddress((void *)mhp);
    dladdr(hookedAddr, &info);

    const char* curImgName = info.dli_fname;
    if(curImgName != NULL) {
        isJbLib = isJailbreakDylib(curImgName);
    }

    if (isJbLib) {
        iosLogInfo("mhp=%p,segname=%{public}s,sectname=%{public}s -> retSection64=%p -> isJbLib
=%s", mhp, segname, sectname, retSection64, boolToStr(isJbLib));
        retSection64 = NULL;
    } else {
        iosLogDebug("mhp=%p,segname=%{public}s,sectname=%{public}s -> retSection64=%p", mhp, se
gname, sectname, retSection64);
    }

    return retSection64;
}
```

其他相关

```
// https://opensource.apple.com/source/cctools/cctools-895/include/mach-o/getsect.h.auto.html

/*=============================================================================
```

```
 Hook: getsegbyname
*******************************************************************************/

// Note: if add log, Aweme will crash

uint8_t* getsegmentdata(const struct mach_header_64 *mhp, const char *segname, unsigned long *s
ize);

%hookf(uint8_t*, getsegmentdata, const struct mach_header_64 *mhp, const char *segname, unsigned
 long *size){
//    iosLogInfo("mhp=%p,segname=%{public}s,size=%p", mhp, segname, size);
    uint8_t* retSegData = %orig;
//    iosLogInfo("mhp=%p,segname=%{public}s,*size=%lu -> retSegCmd=%p", mhp, segname, *size, re
tSegData);
    return retSegData;
}

/*******************************************************************************
 Hook: getsectdatafromFramework
*******************************************************************************/

const struct section_64* getsectbyname(const char *segname, const char *sectname);

%hookf(const struct section_64*, getsectbyname, const char *segname, const char *sectname){
    const struct section_64* retSection = %orig;
    iosLogInfo("segname=%{public}s,sectname=%{public}s -> retSection=%p", segname, sectname, re
tSection);
    return retSection;
}

/*******************************************************************************
 Hook: getsegbyname
*******************************************************************************/

const struct segment_command_64* getsegbyname(const char *segname);

%hookf(const struct segment_command_64*, getsegbyname, const char *segname){
    const struct segment_command_64* retSegCmd = %orig;
    iosLogInfo("segname=%{public}s -> retSegCmd=%p", segname, retSegCmd);
    return retSegCmd;
}

/*******************************************************************************
 Hook: getsectbynamefromheaderwithswap_64
*******************************************************************************/

const struct section* getsectbynamefromheaderwithswap_64(struct mach_header_64 *mhp, const char
 *segname, const char *sectname, int fSwap);

%hookf(const struct section*, getsectbynamefromheaderwithswap_64, struct mach_header_64 *mhp, c
onst char *segname, const char *sectname, int fSwap){
    const struct section* retSection = %orig;
    iosLogInfo("mhp=%p,segname=%{public}s,sectname=%{public}s,fSwap=%d -> retSection=%p", mhp,
segname, sectname, fSwap, retSection);
    return retSection;
}
```

```
/*============================================================================
 Hook: getsectdata
============================================================================*/

extern char* getsectdata(const char *segname, const char *sectname, unsigned long *size);

%hookf(char*, getsectdata, const char *segname, const char *sectname, unsigned long *size){
    char* sectDataStr = %orig;
    iosLogInfo("segname=%{public}s,sectname=%{public}s,*size=%lu -> sectDataStr=%s", segname, s
ectname, *size, sectDataStr);
    return sectDataStr;
}

/*============================================================================
 Hook: getsectdatafromheader_64
============================================================================*/

char* getsectdatafromheader_64(const struct mach_header_64 *mhp, const char *segname, const char
 *sectname, uint64_t *size);

%hookf(char*, getsectdatafromheader_64, const struct mach_header_64 *mhp, const char *segname,
const char *sectname, uint64_t *size){
    char* retSectDataStr = %orig;
    iosLogInfo("mhp=%p,segname=%{public}s,sectname=%{public}s,*size=%llu -> retSectData=%{publi
c}s", mhp, segname, sectname, *size, retSectDataStr);
    return retSectDataStr;
}

/*============================================================================
 Hook: getsectdatafromFramework
============================================================================*/

char* getsectdatafromFramework(const char *FrameworkName, const char *segname, const char *sect
name, unsigned long *size);

%hookf(char *, getsectdatafromFramework, const char *FrameworkName, const char *segname, const
char *sectname, unsigned long *size){
    char* sectDataFrameworkStr = %orig;
    iosLogInfo("FrameworkName=%{public}s,segname=%{public}s,sectname=%{public}s,*size=%lu -> se
ctDataFrameworkStr=%s", FrameworkName, segname, sectname, *size, sectDataFrameworkStr);
    return sectDataFrameworkStr;
}

/*============================================================================
 Hook: getsectbynamefromheader getsectbynamefromheader_64
============================================================================*/

// Not found: Aweme call getsectbynamefromheader
const struct section* getsectbynamefromheader(const struct mach_header *mhp, const char *segname
, const char *sectname);

%hookf(const struct section*, getsectbynamefromheader, const struct mach_header *mhp, const char
 *segname, const char *sectname){
    const struct section* retSection = %orig;
    iosLogInfo("mhp=%p,segname=%{public}s,sectname=%{public}s -> retSection=%p", mhp, segname,
```

```
sectname, retSection);
    return retSection;
}
```

内核级反越狱

TODO：

- 【未解决】研究和尝试内核级反越狱检测：KernBypass

通用内容

此处整理，正向的iOS越狱检测和逆向的iOS反越狱检测，都用得到的部分，通用的内容。

app启动过程

TODO：

- 越狱检测和反越狱检测 会涉及到的：启动的阶段和过程
  - 【未解决】iOS的app的启动流程启动过程

越狱路径相关

TODO：

- 【已解决】反越狱插件中解决内存泄漏OOM：isPathInList
- 【未解决】反越狱相关路径：/Library/MobileSubstrate/DynamicLibraries/
- 【已解决】越狱iOS中动态库/usr/lib/libsubstrate.dylib是哪个插件的

越狱文件列表

TODO：

- 【已解决】iOS越狱检测：越狱文件列表
- 【记录】iOS越狱文件列表更新和维护
- 【已解决】iOS反越狱：优化越狱文件列表的NSString版本从JailbreakFileList.c文件中生成

最新内容已整理至独立的文件

- `JailbreakPathList`
  - 最新版详见
    - https://github.com/crifan/crifanLib/blob/master/c/JailbreakPathList.c
    - https://github.com/crifan/crifanLib/blob/master/c/JailbreakPathList.h

---

截止目前 `20221104` 的最新版是：

- `JailbreakPathList.c`

```
/*
    File: JailbreakPathList.c
    Function: crifan's common jailbreak file path list
    Author: Crifan Li
    Latest: https://github.com/crifan/crifanLib/blob/master/c/JailbreakPathList.c
    Updated: 20221104_1730
*/

#include "JailbreakPathList.h"

/*================================================================================
 Jailbreak Path List
================================================================================*/

// when use isJailbreakPath_realpath, should/could disable KEEP_SOFT_LINK -> internally will co
nvert soft link to real link, so no need soft link
// when use isJailbreakPath_pureC, shold enable KEEP_SOFT_LINK -> to include other soft link ja
ilbreak path for later compare
#define KEEP_SOFT_LINK

const char* jailbreakDylibFuncNameList[] = {
    "MSGetImageByName",
    "MSFindSymbol",
    "MSHookFunction",
    "MSHookMessageEx",

    "SubGetImageByName",
    "SubFindSymbol",
    "SubHookFunction",
    "SubHookMessageEx",
};
```

```
const char* jailbreakPathList_Dylib[] = {
//char* jailbreakPathList_Dylib[] = {
    // common: tweak plugin libs
    "/Library/Frameworks/Cephei.framework/Cephei", // -> /usr/lib/CepheiUI.framework/CepheiUI ?

#ifdef KEEP_SOFT_LINK
    "/Library/Frameworks/CydiaSubstrate.framework/CydiaSubstrate", // -> /usr/lib/libsubstrate.
dylib
#endif

    "/Library/MobileSubstrate/DynamicLibraries/   Choicy.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/0Shadow.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/afc2dService.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/afc2dSupport.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/AppSyncUnified-FrontBoard.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/AppSyncUnified-installd.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/ChoicySB.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/dygz.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/LiveClock.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/MobileSafety.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/PreferenceLoader.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/RocketBootstrap.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/Veency.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/xCon.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/zorro.dylib",
    "/Library/MobileSubstrate/DynamicLibraries/zzzzHeiBaoLib.dylib",

    "/usr/lib/libsubstrate.dylib",

    // Cydia Substrate libs
    "/Library/MobileSubstrate/MobileSubstrate.dylib",
    "/usr/lib/CepheiUI.framework/CepheiUI",
    "/usr/lib/substrate/SubstrateInserter.dylib",
    "/usr/lib/substrate/SubstrateLoader.dylib",
    "/usr/lib/substrate/SubstrateBootstrap.dylib",

    // Substitute libs
    "/usr/lib/libsubstitute.dylib",
#ifdef KEEP_SOFT_LINK
    "/usr/lib/libsubstitute.0.dylib", // -> /usr/lib/libsubstitute.dylib
#endif
    "/usr/lib/substitute-inserter.dylib",
    "/usr/lib/substitute-loader.dylib",
#ifdef KEEP_SOFT_LINK
    "/Library/Frameworks/CydiaSubstrate.framework/SubstrateLoader.dylib", // -> /usr/lib/substi
tute-loader.dylib
#endif

    // Other libs
    "/private/var/lib/clutch/overdrive.dylib",
    "/usr/lib/frida/frida-agent.dylib",

#ifdef KEEP_SOFT_LINK
    "/usr/lib/libapt-inst.2.0.dylib",
    "/usr/lib/libapt-pkg.5.0.dylib",
    "/usr/lib/libapt-private.0.0.dylib",
```

```
#endif
    "/usr/lib/libapt-inst.2.0.0.dylib",
    "/usr/lib/libapt-pkg.5.0.2.dylib",
    "/usr/lib/libapt-private.0.0.0.dylib",

    "/usr/lib/libcycript.dylib",
    "/usr/lib/librocketbootstrap.dylib",
    "/usr/lib/tweakloader.dylib",
};

const char* jailbreakPathList_Other[] = {
//char* jailbreakPathList_Other[] = {
    "/Applications/Activator.app",
    "/Applications/ALS.app",
    "/Applications/blackra1n.app",
    "/Applications/Cydia.app",
    "/Applications/FakeCarrier.app",
    "/Applications/Filza.app",
    "/Applications/FlyJB.app",
    "/Applications/Icy.app",
    "/Applications/iFile.app",
    "/Applications/Iny.app",
    "/Applications/IntelliScreen.app",
    "/Applications/MTerminal.app",
    "/Applications/MxTube.app",
    "/Applications/RockApp.app",
    "/Applications/SBSettings.app",
    "/Applications/SubstituteSettings.app"
    "/Applications/SubstituteSettings.app/Info.plist",
    "/Applications/SubstituteSettings.app/SubstituteSettings",
    "/Applications/Snoop-itConfig.app",
    "/Applications/WinterBoard.app",

#ifdef KEEP_SOFT_LINK
    "/bin/sh",
#endif
    "/bin/bash",

#ifdef KEEP_SOFT_LINK
    // Note: etc -> private/etc/ !!!
    "/etc/alternatives/sh",
    "/etc/apt",
    "/etc/apt/preferences.d/checkra1n",
    "/etc/apt/preferences.d/cydia",
    "/etc/clutch.conf",
    "/etc/clutch_cracked.plist",
    "/etc/dpkg/origins/debian",
    "/etc/rc.d/substitute-launcher",
    "/etc/ssh/sshd_config",
#endif

    "/Library/Activator",
    "/Library/Flipswitch",
    "/Library/dpkg/",

    "/Library/Frameworks/CydiaSubstrate.framework/",
```

```
    "/Library/Frameworks/CydiaSubstrate.framework/Headers/"
    "/Library/Frameworks/CydiaSubstrate.framework/Headers/CydiaSubstrate.h",
    "/Library/Frameworks/CydiaSubstrate.framework/Info.plist",

    "/Library/LaunchDaemons/ai.akemi.asu_inject.plist",
    "/Library/LaunchDaemons/com.openssh.sshd.plist",
    "/Library/LaunchDaemons/com.rpetrich.rocketbootstrapd.plist",
    "/Library/LaunchDaemons/com.saurik.Cydia.Startup.plist",
    "/Library/LaunchDaemons/com.tigisoftware.filza.helper.plist",
    "/Library/LaunchDaemons/dhpdaemon.plist",
    "/Library/LaunchDaemons/re.frida.server.plist",

    // for debug: try avoid 抖音(Aweme) crash
    "/Library/MobileSubstrate/",
    "/Library/MobileSubstrate/DynamicLibraries/",

    "/Library/MobileSubstrate/DynamicLibraries/   Choicy.plist",
    "/Library/MobileSubstrate/DynamicLibraries/afc2dService.plist",
    "/Library/MobileSubstrate/DynamicLibraries/afc2dSupport.plist",
    "/Library/MobileSubstrate/DynamicLibraries/AppSyncUnified-FrontBoard.plist",
    "/Library/MobileSubstrate/DynamicLibraries/AppSyncUnified-installd.plist",
    "/Library/MobileSubstrate/DynamicLibraries/ChoicySB.plist",
    "/Library/MobileSubstrate/DynamicLibraries/dygz.plist",
    "/Library/MobileSubstrate/DynamicLibraries/LiveClock.plist",
    "/Library/MobileSubstrate/DynamicLibraries/MobileSafety.plist",
    "/Library/MobileSubstrate/DynamicLibraries/PreferenceLoader.plist",
    "/Library/MobileSubstrate/DynamicLibraries/RocketBootstrap.plist",
    "/Library/MobileSubstrate/DynamicLibraries/Veency.plist",
    "/Library/MobileSubstrate/DynamicLibraries/xCon.plist",
    "/Library/MobileSubstrate/DynamicLibraries/zorro.plist",
    "/Library/MobileSubstrate/DynamicLibraries/zzzzHeiBaoLib.plist",

    "/Library/PreferenceBundles/SubstitutePrefs.bundle/",
    "/Library/PreferenceBundles/SubstitutePrefs.bundle/Info.plist",
    "/Library/PreferenceBundles/SubstitutePrefs.bundle/SubstitutePrefs",

    "/Library/PreferenceLoader/Preferences/SubstituteSettings.plist",

    "/private/etc/alternatives/sh",
    "/private/etc/apt",
    "/private/etc/apt/preferences.d/checkra1n",
    "/private/etc/apt/preferences.d/cydia",
    "/private/etc/clutch.conf",
    "/private/etc/clutch_cracked.plist",
    "/private/etc/dpkg/origins/debian",
    "/private/etc/rc.d/substitute-launcher",
    "/private/etc/ssh/sshd_config",

    "/private/var/cache/apt/",
    "/private/var/cache/clutch.plist",
    "/private/var/cache/clutch_cracked.plist",
    "/private/var/db/stash",
    "/private/var/evasi0n",
    "/private/var/lib/apt/",
    "/private/var/lib/cydia/",
    "/private/var/lib/dpkg/",
```

```
    "/private/var/mobile/Applications/", //TODO: non-jailbreak can normally open?
    "/private/var/mobile/Library/Filza/",
    "/private/var/mobile/Library/Filza/pasteboard.plist",
    "/private/var/mobile/Library/Cydia/",
    "/private/var/mobile/Library/Preferences/com.ex.substitute.plist",
    "/private/var/mobile/Library/SBSettingsThemes/",
    "/private/var/MobileSoftwareUpdate/mnt1/System/Library/PrivateFrameworks/DictionaryServices
.framework/SubstituteCharacters.plist",
    "/private/var/root/Documents/Cracked/",
    "/private/var/stash",
    "/private/var/tmp/cydia.log",

    "/System/Library/LaunchDaemons/com.saurik.Cydia.Startup.plist",
    "/System/Library/LaunchDaemons/com.ikey.bbot.plist",
    "/System/Library/PrivateFrameworks/DictionaryServices.framework/SubstituteCharacters.plist",


#ifdef KEEP_SOFT_LINK
    // Note: /User -> /var/mobile/
    "/User/Applications/", //TODO: non-jailbreak can normally open?
    "/User/Library/Filza/",
    "/User/Library/Filza/pasteboard.plist",
    "/User/Library/Cydia/",
#endif

    "/usr/bin/asu_inject",
    "/usr/bin/cycc",
    "/usr/bin/cycript",
#ifdef KEEP_SOFT_LINK
    "/usr/bin/cynject", // -> /usr/bin/sinject
    "/usr/bin/Filza", // -> /usr/libexec/filza/Filza
#endif
    "/usr/bin/scp",
    "/usr/bin/sftp",
    "/usr/bin/ssh",
    "/usr/bin/ssh-add",
    "/usr/bin/ssh-agent",
    "/usr/bin/ssh-keygen",
    "/usr/bin/ssh-keyscan",
    "/usr/bin/sshd",
    "/usr/bin/sinject",

    "/usr/include/substrate.h",

    "/usr/lib/cycript0.9/",
    "/usr/lib/cycript0.9/com/",
    "/usr/lib/cycript0.9/com/saurik/"
    "/usr/lib/cycript0.9/com/saurik/substrate/",
    "/usr/lib/cycript0.9/com/saurik/substrate/MS.cy",
    "/usr/libexec/filza/Filza",
    "/usr/libexec/substituted",
    "/usr/libexec/sinject-vpa",

    "/usr/lib/substrate/",
```

```
    "/usr/lib/TweakInject",

    "/usr/libexec/cydia/",
    "/usr/libexec/sftp-server",
    "/usr/libexec/substrate",
    "/usr/libexec/substrated",
    "/usr/libexec/ssh-keysign",

    "/usr/local/bin/cycript",

    "/usr/sbin/frida-server",
    "/usr/sbin/sshd",

#ifdef KEEP_SOFT_LINK
    // /var -> /private/var/

    // TODO: add more /var/xxx path
    "/var/cache/apt/",
    "/var/cache/clutch.plist",
    "/var/cache/clutch_cracked.plist",
    "/var/db/stash",
    "/var/evasi0n",
    "/var/lib/apt/",
    "/var/lib/cydia/",
    "/var/lib/dpkg/",

    "/var/mobile/Applications/", //TODO: non-jailbreak can normally open?
    "/var/mobile/Library/Filza/",
    "/var/mobile/Library/Filza/pasteboard.plist",
    "/var/mobile/Library/Cydia/",
    "/var/mobile/Library/Preferences/com.ex.substitute.plist",
    "/var/mobile/Library/SBSettingsThemes/",
    "/var/MobileSoftwareUpdate/mnt1/System/Library/PrivateFrameworks/DictionaryServices.framewo
rk/SubstituteCharacters.plist",
    "/var/root/Documents/Cracked/",
    "/var/stash",
    "/var/tmp/cydia.log",

#endif
};

const int StrSize = sizeof(const char *);
const int jailbreakPathListLen_Dylib = sizeof(jailbreakPathList_Dylib) / StrSize;
const int jailbreakPathListLen_Other = sizeof(jailbreakPathList_Other) / StrSize;

//int jailbreakPathListLen = sizeof(jailbreakPathList) / StrSize;
const int jailbreakPathListLen = jailbreakPathListLen_Dylib + jailbreakPathListLen_Other;

const int jailbreakDylibFuncNameListLen = sizeof(jailbreakDylibFuncNameList) / StrSize;

const char** getJailbreakPathList(void){
    int strPtrMaxIdx = jailbreakPathListLen; // 133
    int strPtrNum = strPtrMaxIdx + 1; // 134
    int singleSize = sizeof(const char *); // 8
    size_t mallocSize = singleSize * strPtrNum; // 1072
    const char** jailbreakPathStrPtrList = malloc(mallocSize);
```

```
    // jailbreakPathStrPtrList=0x000000011e840c00

    // set each string
    for(int curStrIdx = 0; curStrIdx < jailbreakPathListLen_Dylib; curStrIdx++){
        const char* curStrPtr = jailbreakPathList_Dylib[curStrIdx];
        jailbreakPathStrPtrList[curStrIdx] = curStrPtr;
    }

    for(int curStrIdx = 0; curStrIdx < jailbreakPathListLen_Other; curStrIdx++){
        int totalIndex = jailbreakPathListLen_Dylib + curStrIdx;
        const char* curStrPtr = jailbreakPathList_Other[curStrIdx];
        jailbreakPathStrPtrList[totalIndex] = curStrPtr;
    }
    // set end
    jailbreakPathStrPtrList[strPtrMaxIdx] = NULL;

    return jailbreakPathStrPtrList;
}


/*================================================================================
 Jailbreak Function
================================================================================*/

bool isPathInList(
      const char* inputPath,
//      char* inputPath,
      const char** pathList,
//      char** pathList,
      int pathListLen,
      bool isConvertToPurePath, // is convert to pure path or not
      bool isCmpSubFolder // is compare sub foder or not
){
    bool isInside = false;
    if (!inputPath) {
        return isInside;
    }

    char* inputOrigOrPurePath = NULL;
    if (isConvertToPurePath){
        inputOrigOrPurePath = toPurePath(inputPath);
    }else{
        inputOrigOrPurePath = strdup(inputPath);
    }

    char* matchedPath = NULL;

    char* curPathNoEndSlash = NULL;
    char * curPathWithEndSlash = NULL;
    for (int i 0; i < pathListLen; i++) {
        const char* curPath = pathList[i];
//        char* curPath = pathList[i];
        if (isPathEaqual(inputOrigOrPurePath, curPath)){
            isInside = true;
            matchedPath = (char *)curPath;
            break;
```

```
        }

        if (isCmpSubFolder){
            // check sub folder
            // "/Applications/Cydia.app/Info.plist" belong to "/Applications/Cydia.app/", shoul
d bypass
            // but avoid: '/usr/bin/ssh-keyscan' starts with '/usr/bin/ssh'
            curPathNoEndSlash = removeEndSlash(curPath);
            curPathWithEndSlash = NULL;
            asprintf(&curPathWithEndSlash, "%s/", curPathNoEndSlash);

            if (strStartsWith(inputOrigOrPurePath, curPathWithEndSlash)){
                isInside = true;
                matchedPath = (char *)curPath;
                break;
            }
        }

        if(NULL != curPathNoEndSlash){
            free(curPathNoEndSlash);
            curPathNoEndSlash = NULL;
        }

        if(NULL != curPathWithEndSlash){
            free(curPathWithEndSlash);
            curPathWithEndSlash = NULL;
        }
    }

    if (NULL != inputOrigOrPurePath){
        free(inputOrigOrPurePath);
    }

    return isInside;
}

bool isPathInJailbreakPathList(const char *curPath){
    bool isInJbPathList = false;

    const char** jailbreakPathList = getJailbreakPathList();
    if(jailbreakPathList) {
        isInJbPathList = isPathInList(curPath, jailbreakPathList, jailbreakPathListLen, true, t
rue);
        // final: free char** self
        free(jailbreakPathList);
    }

    return isInJbPathList;
}

bool isJailbreakPath_pureC(const char *curPath){
    bool isJbPath = false;
    if (!curPath) {
        return isJbPath;
    }
```

```
    isJbPath = isPathInJailbreakPathList(curPath);

    return isJbPath;
}

bool isJailbreakPath_realpath(const char *curPath){
    bool isJbPath = false;
    if (!curPath) {
        return isJbPath;
    }

    char gotRealPath[PATH_MAX];
    bool isParseRealPathOk = parseRealPath(curPath, gotRealPath);
//    os_log(OS_LOG_DEFAULT, "isJailbreakPath: isParseRealPathOk=%{bool}d", isParseRealPathOk);

    char curRealPath[PATH_MAX];
    if (isParseRealPathOk) {
        strcpy(curRealPath, gotRealPath);
    } else {
        strcpy(curRealPath, curPath);
    }
//    os_log(OS_LOG_DEFAULT, "isJailbreakPath: curRealPath=%{public}s", curRealPath);
    isJbPath = isPathInJailbreakPathList(curRealPath);

    return isJbPath;
}

// "/Applications/Cydia.app" -> true
bool isJailbreakPath(const char *pathname){
    if (!pathname) {
        return false;
    } else {
    //    return isJailbreakPath_realpath(pathname);
        return isJailbreakPath_pureC(pathname);
    }
}

// "/Library/MobileSubstrate/MobileSubstrate.dylib" -> true
bool isJailbreakDylib(const char *pathname){
    bool isJbDylib = false;

    if (NULL != pathname){
        isJbDylib = isPathInList(pathname, jailbreakPathList_Dylib, jailbreakPathListLen_Dylib,
 true, false);
    }

    return isJbDylib;
}

// "MSHookFunction" -> true
bool isJailbreakDylibFunctionName(const char *libFuncName){
    bool isJbDylibFuncName = false;

    if (NULL != libFuncName){
        isJbDylibFuncName = isPathInList(libFuncName, jailbreakDylibFuncNameList, jailbreakDyli
bFuncNameListLen, false, false);
```

```
    }

    return isJbDylibFuncName;
}
```

- `JailbreakPathList.h`

```
/*
    File: JailbreakPathList.h
    Function: crifan's common jailbreak file path list header file
    Author: Crifan Li
    Latest: https://github.com/crifan/crifanLib/blob/master/c/JailbreakPathList.h
    Updated: 20211230_1049
*/

// This will not work with all C++ compilers, but it works with clang and gcc
#ifdef __cplusplus
extern "C" {
#endif

#ifndef JailbreakPathList_h
#define JailbreakPathList_h

#include <stdbool.h>

#include "CrifanLib.h"

extern const int jailbreakPathListLen;
extern const char* jailbreakPathList_Dylib[];
extern const char* jailbreakPathList_Other[];
//extern char* jailbreakPathList_Dylib[];
//extern char* jailbreakPathList_Other[];
extern const int jailbreakPathListLen_Dylib;
extern const int jailbreakPathListLen_Other;

//extern const char* jailbreakPathList[];
const char** getJailbreakPathList(void);
//char** getJailbreakPathList(void);

bool isPathInJailbreakPathList(const char *curPath);
bool isJailbreakPath_pureC(const char *curPath);
bool isJailbreakPath_realpath(const char *pathname);
bool isJailbreakPath(const char *pathname);
bool isJailbreakDylib(const char *pathname);
bool isJailbreakDylibFunctionName(const char *libFuncName);

bool isPathInList(
      const char* inputPath,
//      char* inputPath,
      const char** pathList,
//      char** pathList,
      int pathListLen,
      bool isConvertToPurePath, // is convert to pure path or not
      bool isCmpSubFolder // is compare sub foder or not
```

```
);

#endif /* JailbreakPathList_h */

#ifdef __cplusplus
}
#endif
```

其他心得

TODO：

- 【已解决】iOS中Choicy中变量的定义：kCFCoreFoundationVersionNumber
-

附录

下面列出相关参考资料。

参考资料

-