目录

iOS逆向开发：越狱检测和反越狱检测

- 最新版本：`v0.7`
- 更新时间：`20221025`

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。

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。

更多其他电子书

本人 `crifan` 还写了其他 `100+` 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

iOS越狱检测概览

TODO：

- 把之前代码整理出 2套代码
  - iOSDetectJailbreak = iOS越狱检测
  - iOSBypassJailbreak = iOS反越狱检测

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
-

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

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

opendir

TODO：

【已解决】iOS越狱检测之打开文件之底层C函数：opendir

access系列

access

TODO：

- 【已解决】iOS越狱检测之打开文件：access

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

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

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

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

```
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];
}
```

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
-

iOS函数

NSURL

TODO：

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

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

---

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;
}
```

_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

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

其他饭越狱检测相关

TODO：

- 【记录】iOS反越狱检测：hook调试NSBundle的mainBundle的pathForResource

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
-

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：

- 【未解决】iOS反越狱检测之打开文件：access系列函数

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

fopen

TODO：

【已解决】iOS反越狱检测之打开文件：hook绕过fopen

stat

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
-

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

svc 0x80内联汇编

TODO：

- 目前无法实现
  - 【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
-

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

环境变量

是否可调试

TODO：

- 【已解决】iOS反越狱检测：是否可被调试

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

越狱相关进程

dyld动态库

dladdr

TODO：

- 【已解决】iOS反越狱检测：dladdr的hook绕过

dlopen+dlsym

TODO：

- 【已解决】iOS反越狱检测：逆向hook的dlopen+dlsym

_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

ObjC运行时

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

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 `20221025` 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: 20220315_1605
*/

#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/MuJiaBaiHuoTweak.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/MuJiaBaiHuoTweak.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

-