分析某加固的Anti-Frida保护

分析样本：某邦加固企业版

首先得感谢龙哥在分析过程中提出了大量非常专业的指导与建议，bug属于我，荣耀属于白龙Lilac!

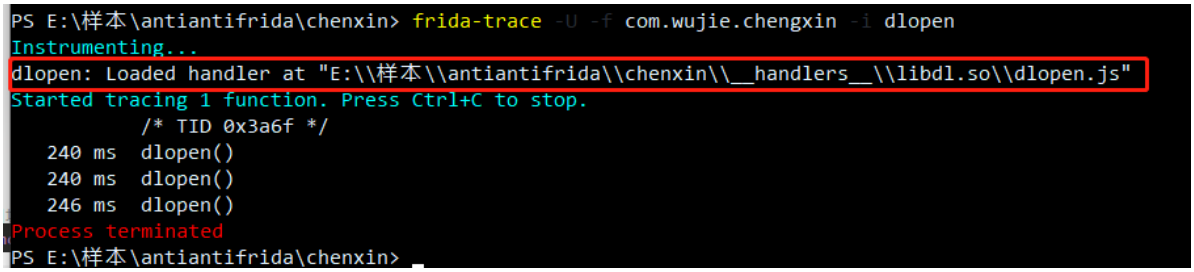
分析过程

找到检测所在的so

我们可以通过`frida-trace`快速进行系统函数的`hook`，首先我们需要知道load so的函数一般为`dlopen`和`android_dlopen_ext`，所以先执行

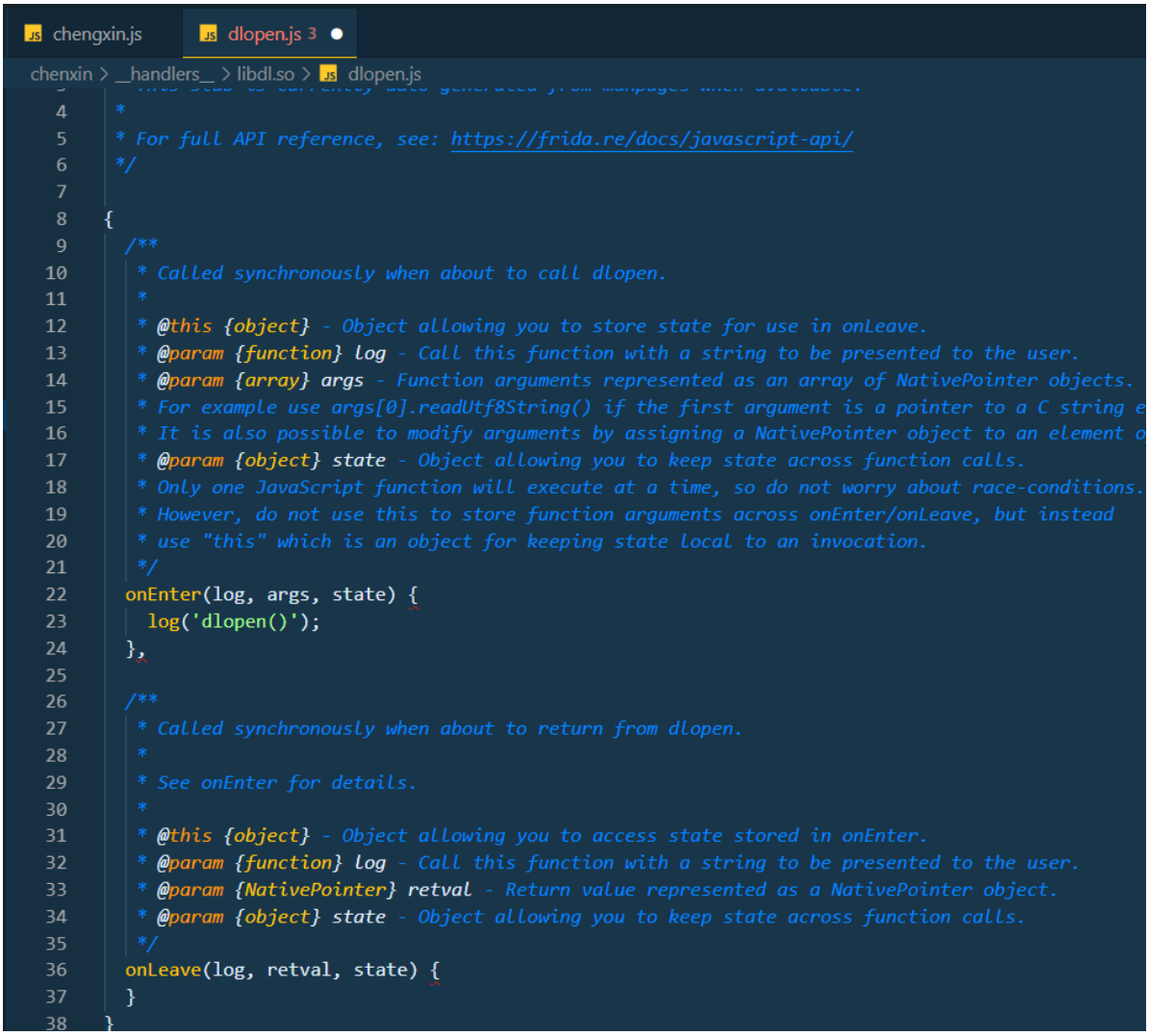
`frida-trace -U -f com.wujie.chengxin -i dlopen`

可以观察到以下输出：



可以看到这里只显示了调用dlopen，但是参数没有输出，dlopen的第一个参数即为所需load的so的名字（`args[0].readCString()`）我们可以去提示的路径下修改dlopen.js脚本

修改前：


chengxin.js
dlopen.js 3
chenxin > __handlers__ > libdl.so > dlopen.js
*
* For full API reference, see: https://frida.re/docs/javascript-api/
*/
{
/**
* Called synchronously when about to call dlopen.
*
* @this {object} - Object allowing you to store state for use in onLeave.
* @param {function} log - Call this function with a string to be presented to the user.
* @param {array} args - Function arguments represented as an array of NativePointer objects.
* For example use args[0].readUtf8String() if the first argument is a pointer to a C string e
* It is also possible to modify arguments by assigning a NativePointer object to an element o
* @param {object} state - Object allowing you to keep state across function calls.
* Only one JavaScript function will execute at a time, so do not worry about race-conditions.
* However, do not use this to store function arguments across onEnter/onLeave, but instead
* use "this" which is an object for keeping state local to an invocation.
*/
onEnter(log, args, state) {
log('dlopen()');
},
/**
* Called synchronously when about to return from dlopen.
*
* See onEnter for details.
*
* @this {object} - Object allowing you to access state stored in onEnter.
* @param {function} log - Call this function with a string to be presented to the user.
* @param {NativePointer} retval - Return value represented as a NativePointer object.
* @param {object} state - Object allowing you to keep state across function calls.
*/
onLeave(log, retval, state) {
}
}

修改后：

```
chengxin.js   dlopen.js 3
chenxin > __handlers__ > libdl.so > dlopen.js
 4    *
 5    * For full API reference, see: https://frida.re/docs/javascript-api/
 6    */
 7
 8   {
 9     /**
10      * Called synchronously when about to call dlopen.
11      *
12      * @this {object} - Object allowing you to store state for use in onLeave.
13      * @param {function} log - Call this function with a string to be presented to the user.
14      * @param {array} args - Function arguments represented as an array of NativePointer objects.
15      * For example use args[0].readUtf8String() if the first argument is a pointer to a C string e
16      * It is also possible to modify arguments by assigning a NativePointer object to an element o
17      * @param {object} state - Object allowing you to keep state across function calls.
18      * Only one JavaScript function will execute at a time, so do not worry about race-conditions.
19      * However, do not use this to store function arguments across onEnter/onLeave, but instead
20      * use "this" which is an object for keeping state local to an invocation.
21      */
22     onEnter(log, args, state) {
23       log('dlopen(): '+args[0].readCString());
24     },
25
26     /**
27      * Called synchronously when about to return from dlopen.
28      *
29      * See onEnter for details.
30      *
31      * @this {object} - Object allowing you to access state stored in onEnter.
32      * @param {function} log - Call this function with a string to be presented to the user.
33      * @param {NativePointer} retval - Return value represented as a NativePointer object.
34      * @param {object} state - Object allowing you to keep state across function calls.
35      */
36     onLeave(log, retval, state) {
37     }
38   }
```

再次输出一下：

```
PS E:\样本\antiantifrida\chenxin> frida-trace -U -f com.wujie.chengxin -i dlopen
Instrumenting...
dlopen: Loaded handler at "E:\\样本\\antiantifrida\\chenxin\\__handlers__\\libdl.so\\dlopen.js"
Started tracing 1 function. Press Ctrl+C to stop.
           /* TID 0x3e92 */
   241 ms  dlopen(): libc.so
   242 ms  dlopen(): libc.so
   247 ms  dlopen(): libdatajar.so
Process terminated
PS E:\样本\antiantifrida\chenxin>
```

现在dlopen的参数就显示出来了，但是这里load的三个so显然是系统的so而非app的so，所以我们再hook android_dlopen_ext看看:

```
PS E:\样本\antiantifrida\chenxin> frida-trace -U -f com.wujie.chengxin -i android_dlopen_ext
Instrumenting...
android_dlopen_ext: Loaded handler at "E:\\样本\\antiantifrida\\chenxin\\__handlers__\\libdl.so\\android_dlopen_ext.js"
Started tracing 1 function. Press Ctrl+C to stop.
           /* TID 0x406c */
   211 ms  android_dlopen_ext(): /system/framework/oat/arm/org.apache.http.legacy.odex
   216 ms  android_dlopen_ext(): /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/oat/arm/base.odex
   222 ms  android_dlopen_ext(): /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/lib/arm/libDexHelper.so
Process terminated
PS E:\样本\antiantifrida\chenxin>
```

可以看到当load到libDexHelper.so的时候，frida被杀掉了，所以我们初步可以判定做检测的位置在libDexHelper.so中

检测点一

首先我们可以通过hook字符串比较函数（比如strstr和strcmp等函数）来观察是否传入了frida相关的字符串进行比较

```
Interceptor.attach(Module.findExportByName(null, "strstr"),{
    onEnter: function(args){
        if(args[0].readCString().indexOf("frida")!==-1     ||args[1].readCString().indexOf("frida")!==-1      ||
          args[0].readCString().indexOf("gum-js-loop")!==-1||args[1].readCString().indexOf("gum-js-loop")!==-1||
          args[0].readCString().indexOf("gmain")!==-1      ||args[1].readCString().indexOf("gmain")!==-1      ||
          args[0].readCString().indexOf("linjector")!==-1  ||args[1].readCString().indexOf("linjector")!==-1){
            console.log('\nstrstr(' +
              's1="' + args[0].readCString() + '"' +
              ', s2="' + args[1].readCString() + '"' +
            ')');
        }
    },
    onLeave: function(retval){

    }
});
```

```
PS E:\样本\antiantifrida\chenxin> frida-trace -U -f com.wujie.chengxin -i strstr
Instrumenting...
strstr: Loaded handler at "E:\\样本\\antiantifrida\\chenxin\\__handlers__\\libc.so\\strstr.js"
Started tracing 1 function. Press Ctrl+C to stop.
           /* TID 0x4dcd */
   246 ms
strstr(s1="/data/local/tmp/re.frida.server/linjector-62", s2="com.wujie.chengxin")
   258 ms
strstr(s1="d0e51000-d1c72000 r-xp 00000000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="lib/libart.so")
   258 ms
strstr(s1="d1c72000-d1cca000 r--p 00e20000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="lib/libart.so")
   258 ms
strstr(s1="d1cca000-d1cd8000 rw-p 00e78000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="lib/libart.so")
   310 ms
strstr(s1="d0e51000-d1c72000 r-xp 00000000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="/lib/libdexfile.so")
   310 ms
strstr(s1="d1c72000-d1cca000 r--p 00e20000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="/lib/libdexfile.so")
   310 ms
strstr(s1="d1cca000-d1cd8000 rw-p 00e78000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="/lib/libdexfile.so")
   397 ms
strstr(s1="d0e51000-d1c72000 r-xp 00000000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="/lib/libart.so")
   397 ms
strstr(s1="d1c72000-d1cca000 r--p 00e20000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="/lib/libart.so")
   398 ms
strstr(s1="d1cca000-d1cd8000 rw-p 00e78000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="/lib/libart.so")
   441 ms
strstr(s1="d0e51000-d1c72000 r-xp 00000000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="/lib/libart.so")
   441 ms
strstr(s1="d1c72000-d1cca000 r--p 00e20000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
", s2="/lib/libart.so")
   441 ms
strstr(s1="d1cca000-d1cd8000 rw-p 00e78000 103:06 4063246                      /data/local/tmp/re.frida.server/frida-agent-32.so
```

通过这些字符串的特征，可以知道它们来自maps，而Frida的一大特征就是在注入到app中后，app的maps中会有frida-agent.so的内存分布。所以这里我们可以通过伪造maps来绕过这里的检测（在/data/data/pkgname/路径下创建一个新的maps，并读取按行读取原始的maps，如果某一行中存在tmp字符串时，就跳过这一行，否则写入新的maps）在脚本里加入以下代码：

```
const openPtr = Module.getExportByName('libc.so', 'open');
const open = new NativeFunction(openPtr, 'int', ['pointer', 'int']);
var readPtr = Module.findExportByName("libc.so", "read");
var read = new NativeFunction(readPtr, 'int', ['int', 'pointer', "int"]);

var fakePath = "/data/data/com.pkgname/maps";
var file = new File(fakePath, "w");
var buffer = Memory.alloc(512);

Interceptor.replace(openPtr, new NativeCallback(function (pathnameptr, flag) {
```

```
    var pathname = Memory.readUtf8String(pathnameptr);
    var realFd = open(pathnameptr, flag);
    console.log("open:", pathname)
    if (pathname.indexOf("maps") >= 0) {
        // console.log("open maps:", pathname);
        while(parseInt(read(realFd, buffer, 512)) !== 0){
            var oneLine = Memory.readCString(buffer);
            if(oneLine.indexOf("tmp")===-1){
                file.write(oneLine);
            }else {
                console.log(oneLine);
            }
        }
        var filename = Memory.allocUtf8String(fakePath);
        return open(filename, flag);
    }
    var fd = open(pathnameptr, flag);
    // Thread.sleep(1)
    return fd;
}, 'int', ['pointer', 'int']));
```

执行后的输出如下：

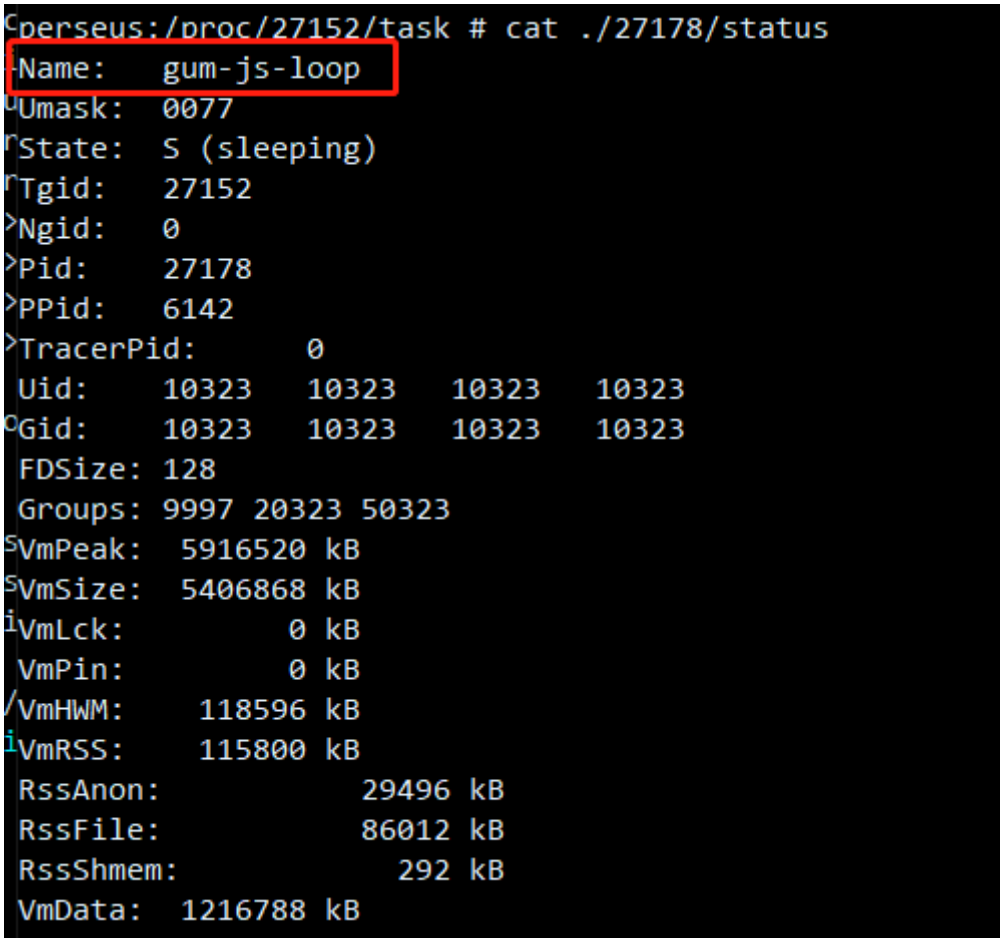
```
d1fe9000-d1ff7000 rw-p 00e78000 103:06 4063243                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1ff7000-d203b000 rw-p 00000000 00:00 0
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/base.apk
open: /proc/self/maps
-d1170000 r-xs 02000000 00:05 66941807                               /memfd:/jit-cache (deleted)
d1170000-d1f91000 r-xp 00000000 103:06 4063243                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1f91000-d1fe9000 r--p 00e20000 103:06 4063243                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1fe9000-d1ff7000 rw-p 00e78000 103:06 4063243                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1ff7000-d203b000 rw-p 00000000 00:00 0
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/base.apk
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/base.apk
open: /apex/com.android.runtime/lib/bionic/libc.so
open: /proc/26096/maps
-d1170000 r-xs 02000000 00:05 66941807                               /memfd:/jit-cache (deleted)
d1170000-d1f91000 r-xp 00000000 103:06 4063243                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1f91000-d1fe9000 r--p 00e20000 103:06 4063243                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1fe9000-d1ff7000 rw-p 00e78000 103:06 4063243                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1ff7000-d203b000 rw-p 00000000 00:00 0
open: /proc/self/task/26096/status
open: /proc/self/task/26101/status
open: /proc/self/task/26102/status
open: /proc/self/task/26103/status
open: /proc/self/task/26104/status
open: /proc/self/task/26105/status
open: /proc/self/task/26107/status
open: /proc/self/task/26112/status
open: /proc/self/task/26113/status
open: /proc/self/task/26115/status
open: /proc/self/task/26116/status
open: /proc/self/task/26117/status
open: /proc/self/task/26118/status
open: /proc/self/task/26119/status
open: /proc/self/task/26120/status
open: /proc/self/task/26121/status
open: /proc/self/task/26122/status
open: /proc/self/task/26123/status
open: /proc/self/task/26124/status
open: /proc/self/task/26125/status
Process terminated
[MIX 3::com.wujie.chengxin]->

Thank you for using Frida!
PS E:\样本\antiantifrida\chenxin>
```

通过观察log可以发现，之前maps中frida相关的字符串没有出现在strstr的参数中，说明我们已经过掉了这个检测点，但是frida仍然被杀掉了，并且被杀掉之前app打开了`/proc/self/task/pid/status`文件，所以我们需要再去观察一下这些status文件。

检测点二

通过观察发现，当app中注入了frida，那么frida的特征会在status文件中的Name字段有所体现，这里我们可以通过其它没有检测frida的app做一个验证，如下图所示：

```
perseus:/proc/27152/task # cat ./27177/status
Name:   gdbus
Umask:  0077
State:  S (sleeping)
Tgid:   27152
Ngid:   0
Pid:    27177
PPid:   6142
TracerPid:      0
Uid:    10323   10323   10323   10323
Gid:    10323   10323   10323   10323
FDSize: 128
Groups: 9997 20323 50323
VmPeak:  5916520 kB
VmSize:  5406868 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:    118596 kB
VmRSS:    115800 kB
RssAnon:           29496 kB
RssFile:           86012 kB
RssShmem:            292 kB
VmData:  1216788 kB
VmStk:      8192 kB
VmExe:        28 kB
VmLib:    186144 kB
perseus:/proc/27152/task # cat ./27176/status
Name:   gmain
Umask:  0077
State:  S (sleeping)
Tgid:   27152
Ngid:   0
Pid:    27176
PPid:   6142
TracerPid:      0
Uid:    10323   10323   10323   10323
Gid:    10323   10323   10323   10323
FDSize: 128
Groups: 9997 20323 50323
VmPeak:  5916520 kB
VmSize:  5406868 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:    118596 kB
VmRSS:    115800 kB
RssAnon:           29496 kB
RssFile:           86012 kB
RssShmem:            292 kB
VmData:  1216788 kB
VmStk:      8192 kB
VmExe:        28 kB
VmLib:    186144 kB
```



所以我们可以通过上面伪造maps的方法去伪造task，修改脚本如下：

```
const openPtr = Module.getExportByName('libc.so', 'open');
const open = new NativeFunction(openPtr, 'int', ['pointer', 'int']);
var readPtr = Module.findExportByName("libc.so", "read");
var read = new NativeFunction(readPtr, 'int', ['int', 'pointer', "int"]);

var fakePath = "/data/data/com.pkgname/maps";
var file = new File(fakePath, "w");
var buffer = Memory.alloc(512);

var fakePath2 = "/data/data/com.pkgname/task";
var file2 = new File(fakePath2, "w");
var buffer2 = Memory.alloc(512);

Interceptor.replace(openPtr, new NativeCallback(function (pathnameptr, flag) {
    var pathname = Memory.readUtf8String(pathnameptr);
    var realFd = open(pathnameptr, flag);
    console.log("open:", pathname)
    if (pathname.indexOf("maps") >= 0 || pathname.indexOf("task") >= 0) {
        var temp = pathname.indexOf("maps") >= 0 ? 1:2;
        switch(temp){
            case 1:
                // console.log("open maps:", pathname);
                while(parseInt(read(realFd, buffer, 512)) !== 0){
                    var oneLine = Memory.readCString(buffer);
                    if(oneLine.indexOf("tmp")===-1){
                        file.write(oneLine);
                    }else {
                        // console.log(oneLine);
                    }
```

```
                }
                var filename = Memory.allocUtf8String(fakePath);
                return open(filename, flag);
                break;
            case 2:
                // console.log("open task:", pathname);
                while(parseInt(read(realFd, buffer2, 512)) !== 0){
                    var oneLine = Memory.readCString(buffer2);
                    var replaceStr = "123"
                    if(oneLine.indexOf("gum-js-loop")!=-1){
                        oneLine = oneLine.replace("gum-js-loop", replaceStr)
                    }
                    if(oneLine.indexOf("gmain")!=-1){
                        oneLine = oneLine.replace("gmain", replaceStr)
                    }
                    file2.write(oneLine);
                    // console.log(oneLine)
                }
                var filename = Memory.allocUtf8String(fakePath2);
                return open(filename, flag);
                break;
        }
    }
    var fd = open(pathnameptr, flag);
    // Thread.sleep(1)
    return fd;
}, 'int', ['pointer', 'int']));
```

上面的脚本执行完成后，其实frida还是会被断下来，但是查看输出的log我们可以看到替换是成功的：

```
Mems_allowed:   1
Mems_allowed_list:      0
voluntary_ctxt_switches:        53
nonvoluntary_ctxt_switches:     3
PTE:            988 kB
VmPMD:           16 kB
VmSwap:     19556 k
Name:   123
Umask:  0077
State:  S (sleeping)
Tgid:   22820
Ngid:   0
Pid:    22849
PPid:   6143
TracerPid:      0
Uid:    10341   10341   10341   10341
Gid:    10341   10341   10341   10341
FDSize: 128
Groups: 3003 9997 20341 50341
VmPeak:  2010596 kB
VmSize:  2010228 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:    148848 kB
VmRSS:    148848 kB
```

但细心一点我们会发现，还有一个frida相关的字符串出现在了log里（pool-frida）所以我们需要修改一下脚本：

```
case 2:
    // console.log("open task:", pathname);
    while(parseInt(read(realFd, buffer2, 512)) !== 0){
        var oneLine = Memory.readCString(buffer2);
        var replaceStr = "123"
        if(oneLine.indexOf("pool-frida")!=-1){
            oneLine = oneLine.replace("pool-frida", replaceStr)
        }
        if(oneLine.indexOf("gum-js-loop")!=-1){
            oneLine = oneLine.replace("gum-js-loop", replaceStr)
        }
        if(oneLine.indexOf("gmain")!=-1){
            oneLine = oneLine.replace("gmain", replaceStr)
        }
        file2.write(oneLine);
        // console.log(oneLine)
    }
    var filename = Memory.allocUtf8String(fakePath2);
    return open(filename, flag);
    break;
```

然后，执行！然后frida又双叒叕被终止了，所以还是继续看输出。

检测点三

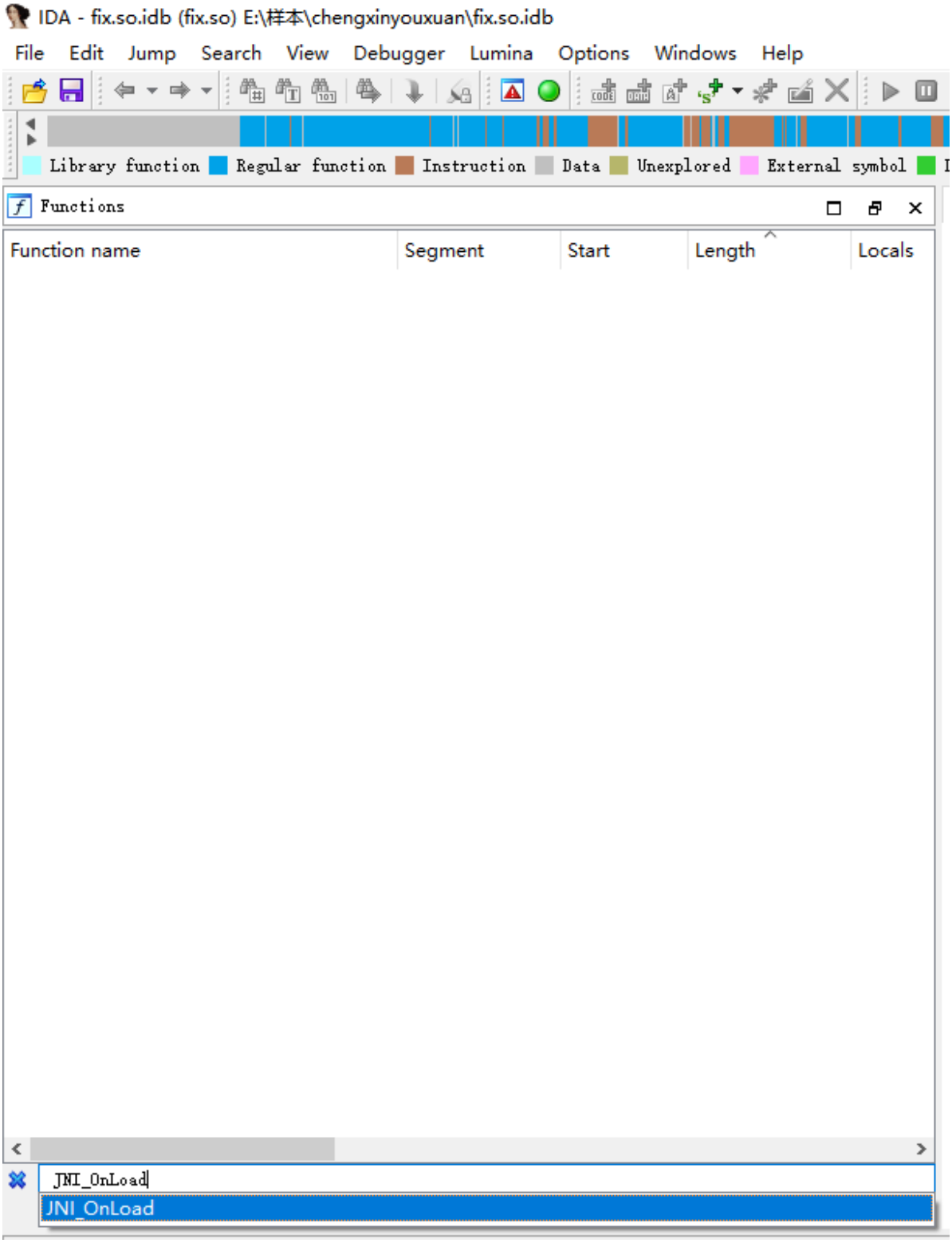
通过输出可以看到上面还有个关于strstr函数且参数为frida相关字符串的调用：

```
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/oat/arm/base.vdex
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/base.apk
open: /system/framework/arm/boot.art
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/oat/arm/base.art
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/lib/arm/libDexHelper.so
open: /proc/8423/cmdline

strstr(s1="/data/local/tmp/re.frida.server/linjector-99", s2="com.wujie.chengxin")
open: /system/lib/libc.so
open: /proc/self/maps
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/oat/arm/base.odex
open: /data/user/0/com.wujie.chengxin/.cache/classes.dve
open: /system/lib/libc.so
open: /proc/self/maps
open: /proc/self/maps
open: /proc/self/maps
open: /proc/self/maps
open: /proc/self/maps
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/base.apk
open: /proc/self/maps
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/base.apk
open: /data/app/com.wujie.chengxin-WZCaCd7ATEUlEIc9Fl7Xdg==/base.apk
open: /apex/com.android.runtime/lib/bionic/libc.so
open: /proc/8423/maps
open: /proc/self/task/8423/status
open: /proc/self/task/8427/status
open: /proc/self/task/8429/status
open: /proc/self/task/8430/status
open: /proc/self/task/8431/status
open: /proc/self/task/8432/status
open: /proc/self/task/8434/status
open: /proc/self/task/8439/status
open: /proc/self/task/8440/status
open: /proc/self/task/8441/status
open: /proc/self/task/8442/status
open: /proc/self/task/8443/status
open: /proc/self/task/8444/status
open: /proc/self/task/8445/status
open: /proc/self/task/8446/status
open: /proc/self/task/8447/status
open: /proc/self/task/8448/status
open: /proc/self/task/8449/status
open: /proc/self/task/8450/status
open: /proc/self/task/8451/status
open: /proc/self/task/8452/status
open: /proc/self/task/8453/status
open: /proc/self/task/8454/status
open: /proc/self/task/8455/status
Process terminated
[MIX 3::com.wujie.chengxin]->
```

我们打印一下这个strstr的调用栈看看

```
PS E:\样本\antiantifrida\chenxin> frida -U -f com.wujie.chengxin -l .\chengxin.js --no-pause
     ____
    / _  |   Frida 15.0.18 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
Spawned `com.wujie.chengxin`. Resuming main thread!
[MIX 3::com.wujie.chengxin]->
strstr(s1="/data/local/tmp/re.frida.server/linjector-102", s2="com.wujie.chengxin")
0xca12891d libDexHelper.so!JNI_OnLoad+0x14a8
0xf6c55100
```

所以我们再去看看图中这个JNI_OnLoad+0x14a8位置的代码逻辑（此处省略从内存中dump xxx.so以及修复so过程）

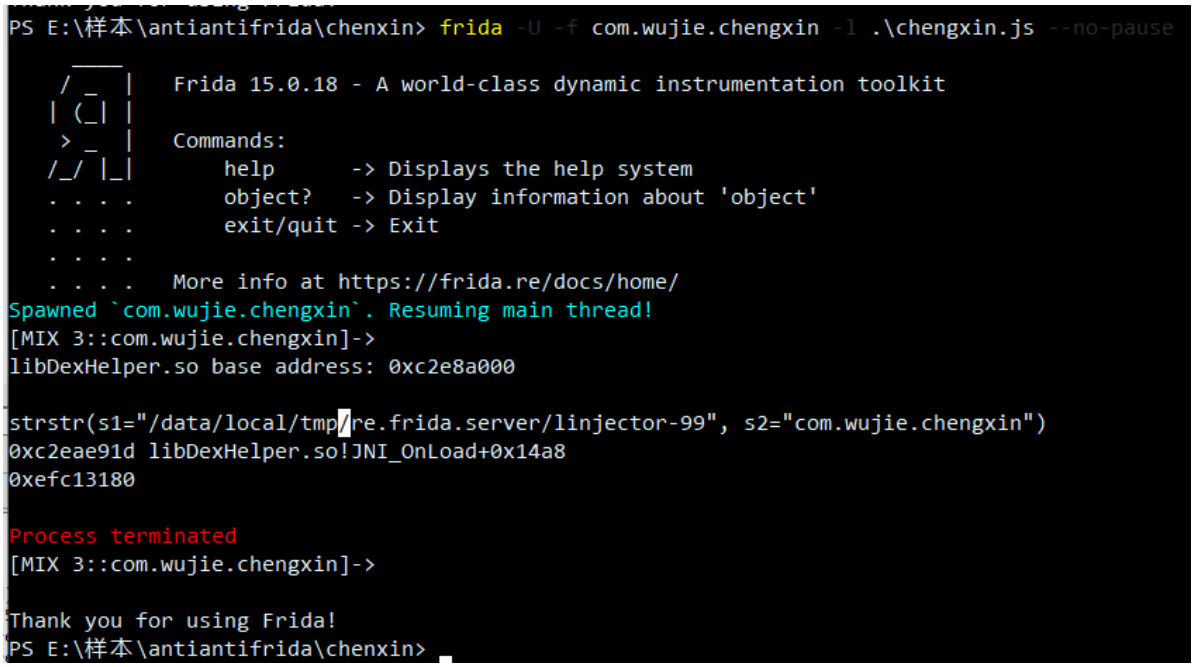


通过在ida里搜索可以看到JNI_OnLoad函数被抹掉了，所以我们得去打印libDexHelper.so的基址，然后计算偏移来获取strstr的调用点，修改hook strstr部分的脚本为：

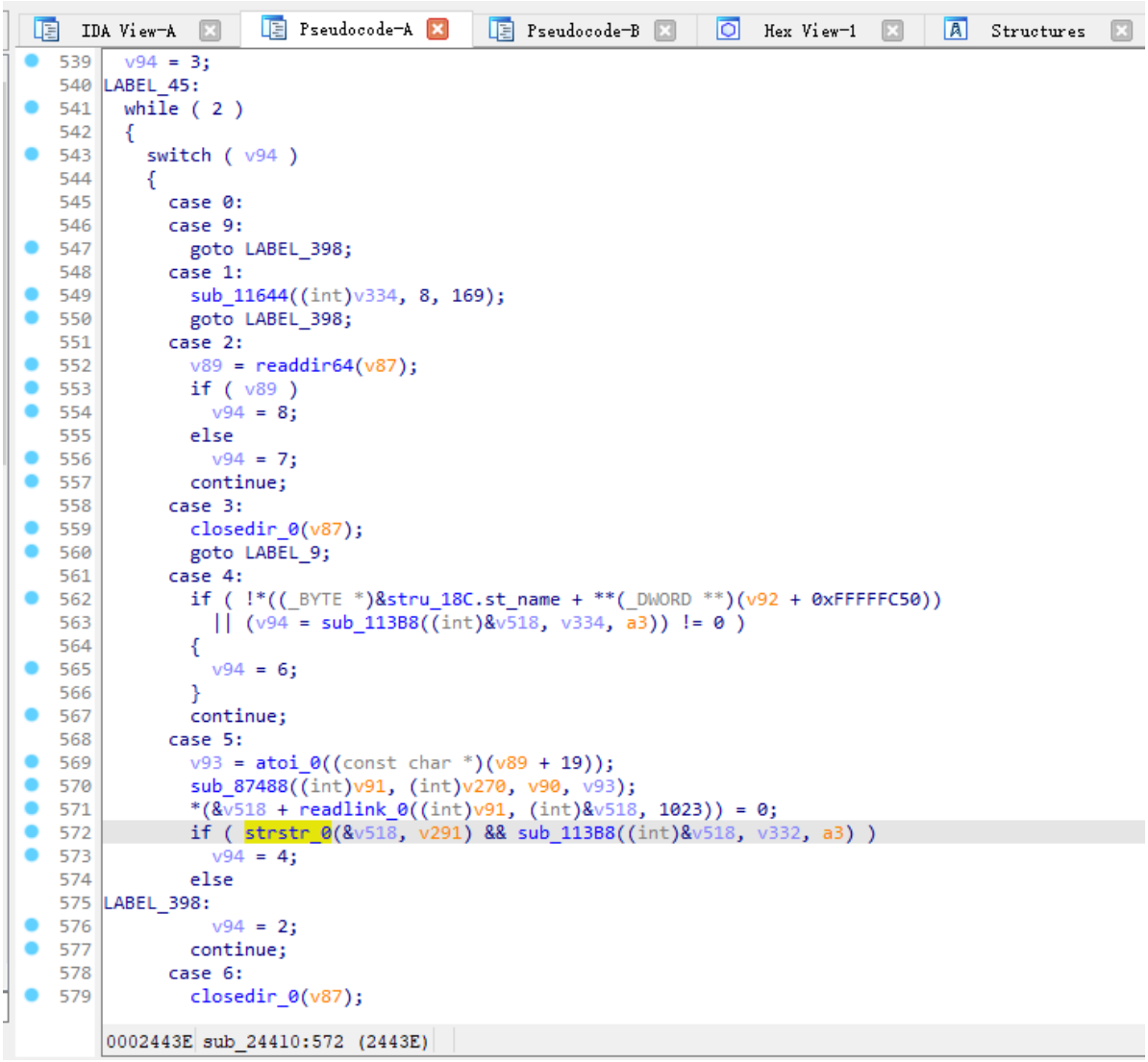
```
Interceptor.attach(Module.findExportByName(null, "strstr"),{
    onEnter: function(args){
        if(args[0].readCString().indexOf("frida")!==-1
||args[1].readCString().indexOf("frida")!==-1      ||
            args[0].readCString().indexOf("gum-js-
loop")!==-1||args[1].readCString().indexOf("gum-js-loop")!==-1||
            args[0].readCString().indexOf("gmain")!==-1
 ||args[1].readCString().indexOf("gmain")!==-1      ||
```

```
            args[0].readCString().indexOf("linjector")!==-1
  ||args[1].readCString().indexOf("linjector")!==-1){
            console.log("\nlibDexHelper.so base address:
"+Module.findBaseAddress("libDexHelper.so"))
            console.log('\nstrstr(' +
                's1="' + args[0].readCString() + '"' +
                ', s2="' + args[1].readCString() + '"' +
            ')');
            console.log(Thread.backtrace(this.context,
Backtracer.ACCURATE).map(DebugSymbol.fromAddress).join('\n') + '\n');
        }
    },
    onLeave: function(retval){

    }
});
```

执行：



所以偏移为：0xc2eae91d-0xc2e8a000，在ida里跳转后，往上找一找能发现一个strstr函数调用（其实计算出来的偏移在ida中跳转后并不是strstr的调用点，但肯定是在这个函数里的，所以需要上下找一找）：



这里strstr函数上面就是一个readlink函数，并且参数v518也和readlink似乎有点关联，所以我们去hook这个readlink看看，在我们的脚本里补充：

```
var iii,jjj;
Interceptor.attach(Module.findExportByName(null, "readlink"),{
    onEnter: function(args){
      console.log("call readlink")
      iii = args[0]
      jjj = args[1]
      // console.log(Thread.backtrace(this.context,
Backtracer.ACCURATE).map(DebugSymbol.fromAddress).join('\n') + '\n');

    },
    onLeave: function(retval){
        console.log("leave readlink: "+iii.readCString()+" "+jjj.readCString())
    }
});
```

果然，我们可以看到frida出现了：

```
leave readlink: /proc/4085/fd/44 anon_inode:[eventfd]
leave readlink: /proc/4085/fd/45 socket:[68595041]fd]
leave readlink: /proc/4085/fd/46 anon_inode:[eventfd]
leave readlink: /proc/4085/fd/47 anon_inode:[eventfd]
leave readlink: /proc/4085/fd/48 socket:[68621502]fd]
leave readlink: /proc/4085/fd/49 socket:[68620885]
leave readlink: /proc/4085/fd/50 socket:[68620894]
leave readlink: /proc/4085/fd/51 /dev/proc_kperfevents.apk
leave readlink: /proc/4085/fd/52 anon_inode:[eventfd]s
leave readlink: /proc/4085/fd/53 /data/local/tmp/re.frida.server/linjector-105-ext-res.apk

libDexHelper.so base address: 0xc2e80000

strstr(s1="/data/local/tmp/re.frida.server/linjector-105", s2="com.wujie.chengxin")
0xc2ea491d libDexHelper.so!JNI_OnLoad+0x14a8
0xefc13180

leave readlink: /proc/4085/fd/54 /data/data/com.wujie.chengxin/mapsnjector-105
leave readlink: /proc/4085/fd/55 socket:[68620928]jie.chengxin/maps
leave readlink: /proc/4085/fd/56 /data/data/com.wujie.chengxin/task
leave readlink: /proc/4085/fd/57 anon_inode:[eventfd].chengxin/task
leave readlink: /proc/4085/fd/58 anon_inode:[eventpoll]hengxin/task
leave readlink: /proc/4085/fd/59 /proc/4123/timerslack_nsngxin/task
leave readlink: /proc/4085/fd/60 /proc/4124/timerslack_ns
leave readlink: /proc/4085/fd/61 /proc/4125/timerslack_ns
leave readlink: /proc/4085/fd/62 /proc/4085/task/4085/delayxin/task
leave readlink: /proc/4085/fd/63 /data/system/theme/iconsay
leave readlink: /proc/4085/fd/64 /data/system/theme/icons
leave readlink: /proc/4085/fd/65 /data/system/theme/framework-miui-resctor-105
leave readlink: /proc/4085/fd/66 /data/system/theme/framework-miui-res
leave readlink: /proc/4085/fd/67 /proc/4085/cmdline/framework-miui-res
leave readlink: /proc/4085/fd/68 /system/framework/oat/arm/org.apache.http.legacy.vdex.apk
```

readlink函数的定义如下：

linux下readlink函数详解

相关函数： stat, lstat, symlink
表头文件： #include <unistd.h>
定义函数：int readlink(const char *path, char *buf, size_t bufsiz);
函数说明：readlink()会将参数path的符号连接内容到参数buf所指的内存空间，返回的内容不是以NULL作字符串结尾，但会将字符串的字符数返回。若参数bufsiz小于符号连接的内容长度，过长的内容会被截断

返回值 ：执行成功则传符号连接所指的文件路径字符串，失败返回-1， 错误代码存于errno
错误代码：

EACCESS	取文件时被拒绝，权限不够
EINVAL	参数bufsiz为负数
EIO	O存取错误
ELOOP	欲打开的文件有过多符号连接问题
ENAMETOOLONG	参数path的路径名称太长
ENOENT	参数path所指定的文件不存在
ENOMEM	核心内存不足
ENOTDIR	参数path路径中的目录存在但却非真正的目录

所以，我们需要在readlink函数返回的时候替换掉buf中关于frida的内容，这样就能绕过这个检测了，脚本如下：

```
var aaa,bbb,ccc;
var ss = false
Interceptor.attach(Module.findExportByName(null, "readlink"),{
```

```
    onEnter: function(args){
        aaa = args[0];
        bbb = args[1];
        ccc = args[2];
    },
    onLeave: function(retval){
      if(bbb.readCString().indexOf("frida")!==-1      ||
         bbb.readCString().indexOf("gum-js-loop")!==-1||
         bbb.readCString().indexOf("gmain")!==-1      ||
         bbb.readCString().indexOf("linjector")!==-1){
          console.log('\nreadlink(' +
                's1="' + aaa.readCString() + '"' +
                ', s2="' + bbb.readCString() + '"' +
                ', s3="' + ccc + '"' +
              ')');

          bbb.writeUtf8String("/system/framework/boot.art")
          console.log("replce with: "+bbb.readCString())
          retval.replace(0x1A)
          console.log("retval: "+retval)
          }
      }
});
```

再次执行：

```
PS E:\样本\antiantifrida\chenxin> frida -U -f com.wujie.chengxin -l .\chengxin.js --no-pause
     ____
    / _  |   Frida 15.0.18 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
Spawned `com.wujie.chengxin`. Resuming main thread!
[MIX 3::com.wujie.chengxin]->
readlink(s1="/proc/4843/fd/53", s2="/data/local/tmp/re.frida.server/linjector-109-ext-res.apk", s3="0x3ff"
replce with: /system/framework/boot.art
retval: 0x1a

readlink(s1="/proc/4843/fd/53", s2="/data/local/tmp/re.frida.server/linjector-109-ext-res.apk", s3="0x3ff"
replce with: /system/framework/boot.art
retval: 0x1a

readlink(s1="/proc/self/fd/53", s2="/data/local/tmp/re.frida.server/linjector-109", s3="0x100")
replce with: /system/framework/boot.art
retval: 0x1a
[MIX 3::com.wujie.chengxin]->
[MIX 3::com.wujie.chengxin]->
[MIX 3::com.wujie.chengxin]->
[MIX 3::com.wujie.chengxin]-> Process.id
4843
[MIX 3::com.wujie.chengxin]->
```

成功！(ps：将上面的脚本拼接起来就可以复现)

常见Frida检测点总结

通过maps检测

扫描/proc/self/maps文件中的内存分布，寻找是否存在打开了/data/local/tmp路径下的so（Frida在运行时会先确定/data/local/tmp路径下是否有re.frida.server文件夹，若没有则创建该文件夹并存放frida-agent.so等文件）

```
perseus:/data/local/tmp/re.frida.server # ls -ll
total 37848
-rwxr-xr-x 1 root root 15226516 2021-09-26 16:21:26.412616821 +0800 frida-agent-32.so
-rwxr-xr-x 1 root root 20847304 2021-09-26 16:21:26.512616821 +0800 frida-agent-64.so
-rwx------ 1 root root  2665124 2021-09-26 16:21:26.522616821 +0800 frida-helper-32
prw-rw-rw- 1 root root        0 2021-09-26 16:21:26.662616821 +0800 linjector-1
prw-rw-rw- 1 root root        0 2021-09-26 16:21:27.952616820 +0800 linjector-2
prw-rw-rw- 1 root root        0 2021-09-26 16:21:55.252616810 +0800 linjector-3
prw-rw-rw- 1 root root        0 2021-09-26 16:21:52.762616811 +0800 linjector-7
perseus:/data/local/tmp/re.frida.server #
```

在maps里表现为：

```
perseus:/proc/10884 # cat maps|grep frida --context=3
cae51000-cce51000 r--s 02000000 00:05 66344170                           /memfd:/jit-cache (deleted)
cce51000-cee51000 rw-s 00000000 00:05 66344170                           /memfd:/jit-cache (deleted)
cee51000-d0e51000 r-xs 02000000 00:05 66344170                           /memfd:/jit-cache (deleted)
d0e51000-d1c72000 r-xp 00000000 103:06 4063246                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1c72000-d1cca000 r--p 00e20000 103:06 4063246                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1cca000-d1cd8000 rw-p 00e78000 103:06 4063246                           /data/local/tmp/re.frida.server/frida-agent-32.so
d1cd8000-d1d1c000 rw-p 00000000 00:00 0                                  [anon:.bss]
d1d1e000-d1d34000 rw-p 00000000 00:00 0                                  [anon:dalvik-large object space allocation]
d1d39000-d1d3e000 rw-p 00000000 00:00 0                                  [anon:dalvik-large object space allocation]
perseus:/proc/10884 #
```

通过task检测

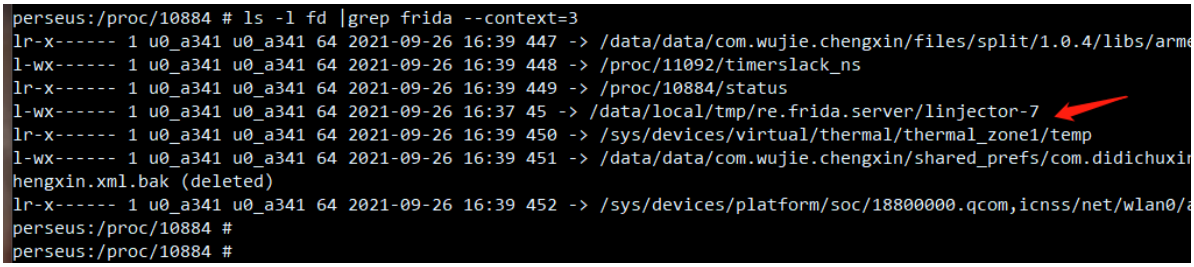
扫描task目录下所有/task/pid/status中的Name字段寻找是否存在frida注入的特征，具体线程名为`gmain`、`gdbus`和`gum-js- loop`，一般情况下这三个线程在第11--13的位置，此外在frida运行脚本过程中，还会存在一个Name字段为`pool-frida`的线程。

```
perseus:/proc/10884 # cd task/
perseus:/proc/10884/task # ls -ll
total 0
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:52.752616811 +0800 10884
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10888
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10901
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10903
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10904
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10905
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10906
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10913
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10914
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10915
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10916
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10918
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10919
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10921
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10925
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10926
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:53.842616810 +0800 10927
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:54.082616810 +0800 10928
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:54.142616810 +0800 10931
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:54.142616810 +0800 10933
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:54.742616810 +0800 10935
dr-xr-xr-x 7 u0_a341 u0_a341 0 2021-09-26 16:21:54.742616810 +0800 10940
```

```
perseus:/proc/10884/task # cat ./10916/status |grep Name
Name:   gmain
perseus:/proc/10884/task #
perseus:/proc/10884/task # cat ./10918/status |grep Name
Name:   gdbus
perseus:/proc/10884/task # cat ./10919/status |grep Name
Name:   gum-js-loop
```

通过fd检测

通过readlink查看`/proc/self/fd`和`/proc/self/task/pid/fd`下所有打开的文件，检测是否有frida相关文件。



通过D-BUS检测

Frida是通过D-Bus协议进行通信的，所以可以遍历/proc/net/tcp文件，向每个开放的端口发送 D-Bus 的认证消息`AUTH`，如果端口回复了`REJECT`，那么这个端口就是frida-server（具体案例后边补上）