iOS和macOS沙盒技术分析

the**** / 2019-02-01 09:03:00 / 浏览数 1908

众所周知，macOS的沙盒一直是一个神秘的东西，我喜欢利用各种工具并从Jonathan Levin的《*OS Internals》等参考书再或者苹果官方自己都不太清楚的文档中收集的知识来分析它。苹果的安全机制并不是最好的，这不是什么新鲜事。沙盒技术有很长的历史，macOS的

0x01 背景

苹果首次使用沙盒技术是在其OS X 10.5(Leopard)中，它被称为"SeatBelt"（安全带）。正如这个词语的意思一样，它就像在汽车旅行中为了安全而系上安全带一样，强制开发人员在应用程序上使用沙盒技术(MAC)Framework，沙盒的想法肯定不错，但离成功还很远。MACF框架是苹果设备的整个安全模型构建的基础。

在OS X 10.7中，苹果公司吸取了OS X 10.5的教训，沙盒现在已经不再任由开发人员在应用程序上是否使用，默认情况下是强制执行的。即使是今天在macOS Mojave上，苹果仍然强制使用沙盒，基于应用程序拥有的权限（com.apple.security.app-sandbox）。如果应用程序具有此权限，它将被放置在沙盒中，而不是考虑

需要注意的是，与iOS的沙盒相比，macOS更容易操作。在iOS上，第三方程序均不可能逃脱沙盒，除非你使用沙盒逃逸技术，而大多数这种情况下是由内核漏洞或沙盒漏洞containers/bundl/中的程序。任何在/var/中的东西都要被沙盒化，因为你不能直接在其他地方安装你的程序，除非你越狱了。在macOS上，只有Appstore中的程序是沙盒的。如果你从开

0x02 工作原理

沙盒的唯一目的是限制程序访问系统的各种资源，比如系统调用、文件或任何东西，这是为了恶意程序肆意破坏系统。在iOS上，我可以骗你安装一个恶意的程序，但这个做

沙盒实际上是一项非常好的技术，这也就是为什么它一直沿用到今天的原因。假如你在Windows上打开了一个从非法来源上下载的恶意程序，而该程序若想删除System32

苹果官方说过:沙盒是一种在内核层面强制实施的访问控制技术(在内核层面，用户或任何受到损害的程序通常都无法控制)。沙盒可以确保它拦截沙盒程序执行的所有操作，并

在macOS上，沙箱本身不是单个文件或单个进程，它被分割成多个组件，比如位于/usr/libexec/sandboxd目录中的userland daemon，这是com.apple.security.sandbox是kext (Kernel Extension)，还有依赖于AppContainer.Framework的AppSandbox私有框架。正如你所见，多个组件一起工作来实现本文所诉的程序沙箱。

在终端中运行kextstat | grep"sand"命令，可以看到macOS上的kext处于活动状态。

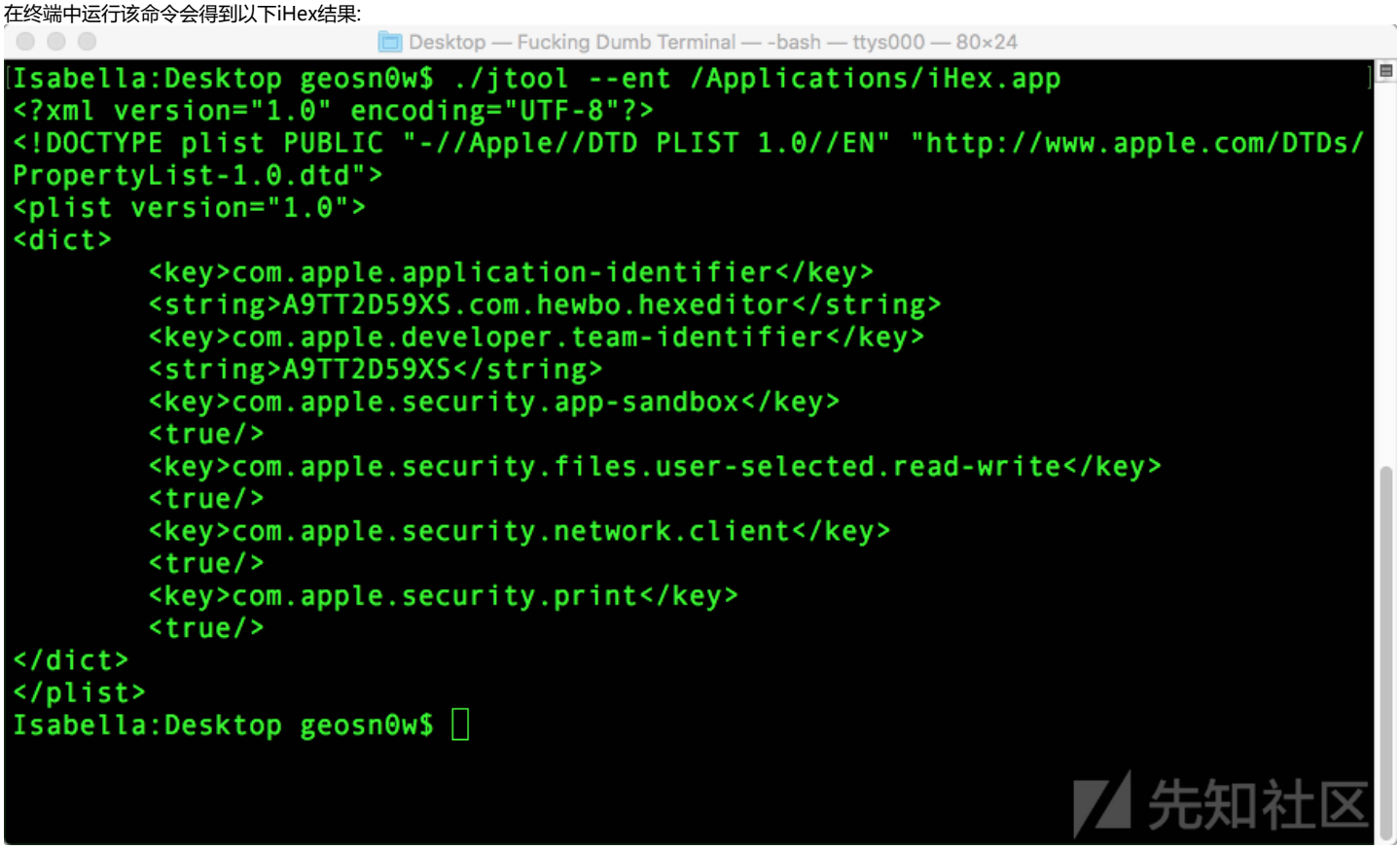
```
Isabella:/ geosn0w$ kextstat | grep "sand"
  381 0xffffff7f811a3000 0x210000x21000com.apple.security.sandbox (300.0) BDFF700A-6746-3643-A0A1-852628695B04 <37 30 18 7 6 5
Isabella:/ geosn0w$
```

沙箱是多个MACF策略模块之一。AMFI (Apple Mobile File Integrity)的协同设计是另一个模块。

0x03 测试:根据授权决定macOS上的应用程序是否沙盒化

正如之前所提到的，该应用被沙盒化的一个明显迹象是应用程序二进制文件中是否需要com.apple.security.app-sandbox权限。我们可以使用很多工具检查macOS上Levin的jtool这个工具，运行命令./jtool--ent /Applications/AppName.在终端app中，我们可以看到程序所拥有的全部权限。以iHex为例，Appstore中的只需要OpenBoardView权限。DMG格式如下：

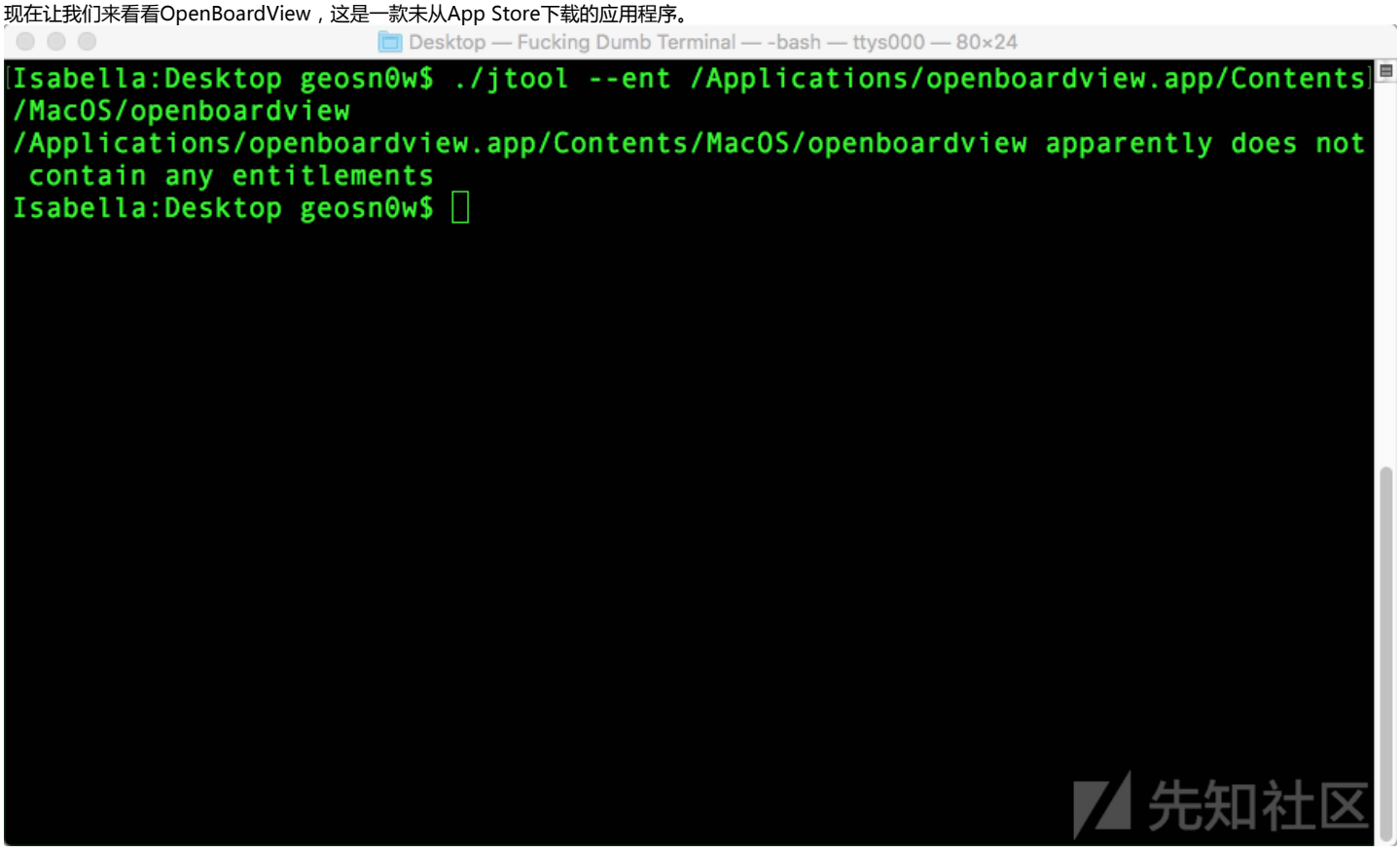
在终端中运行该命令会得到以下iHex结果:



需要注意的是，权限是存在的，并且密钥被设置为`true`，此程序将被沙盒化。现在，正如你所见，这些权利是以类似于XML的格式列出的，它们实际上位于`.PLIST or Property List`文件中，而属性列表文件只不过是美化的XML。PLISTs可以采用二进制格式，可以使用命令`plutil -convert xml1 -o`将其转换为可读的格式。

使用`Jtool`可以替换程序的权限，但之后需要对程序进行伪造签名。总之，这是一种解除macOS应用程序沙盒的方法。这在iOS上并不容易做到，因为沙盒是基于应用程序的

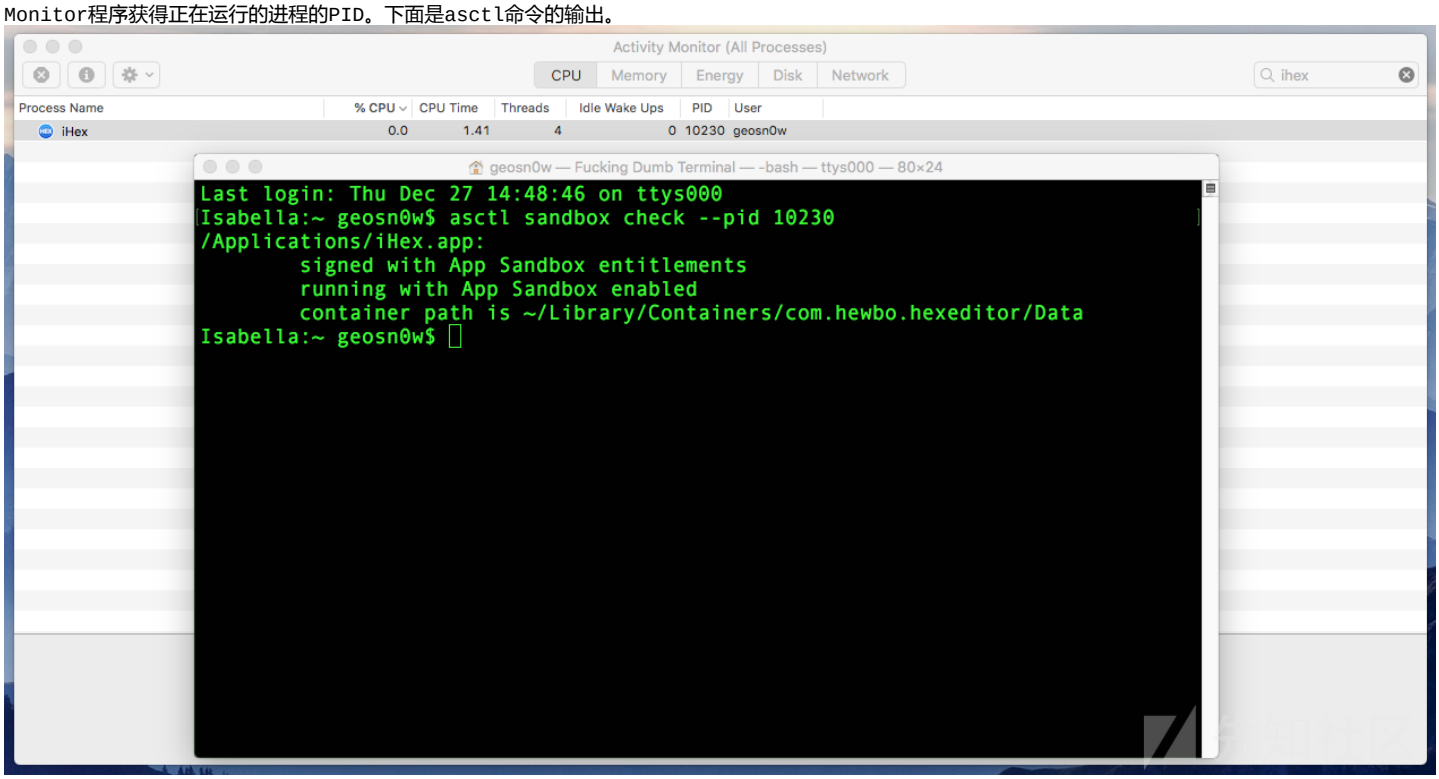
现在让我们来看看OpenBoardView，这是一款未从App Store下载的应用程序。



如你所见，程序没有任何权限。它不会被沙盒化，这意味着它可以比任何应用程序商店应用程序访问更多的源代码。

`com.apple.security.app-sandbox` 的权限并不是iHEX开发人员自己添加的,它是由苹果官方在App Store审核的过程中自动添加的。

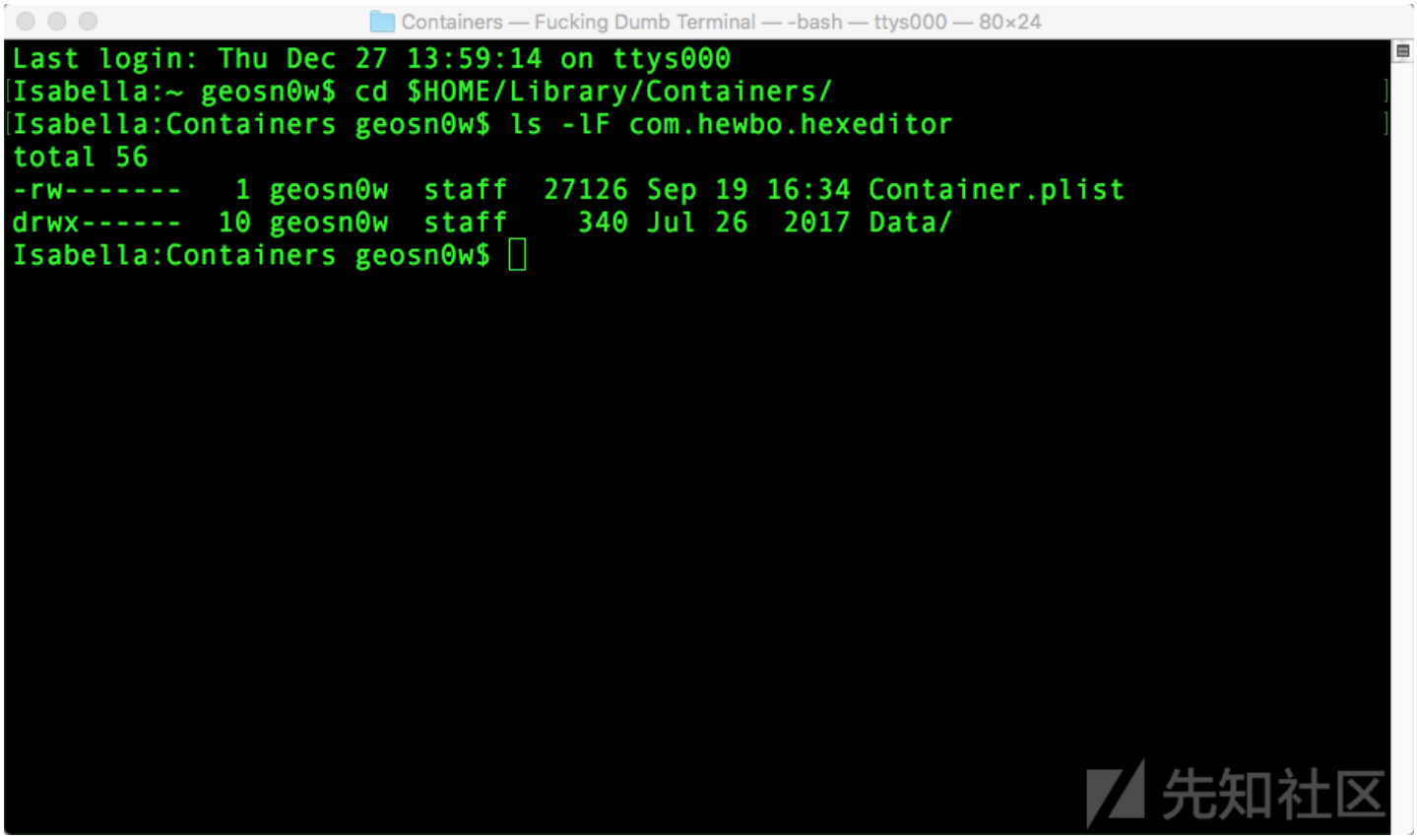
另一种检查程序是否被沙盒化的方法是运行`asctl sandbox check --pid XYZ`命令，其中XYZ是程序的PID(Process ID)。可以从macOS上的Activity Monitor程序获得正在运行的进程的PID。下面是asctl命令的输出。



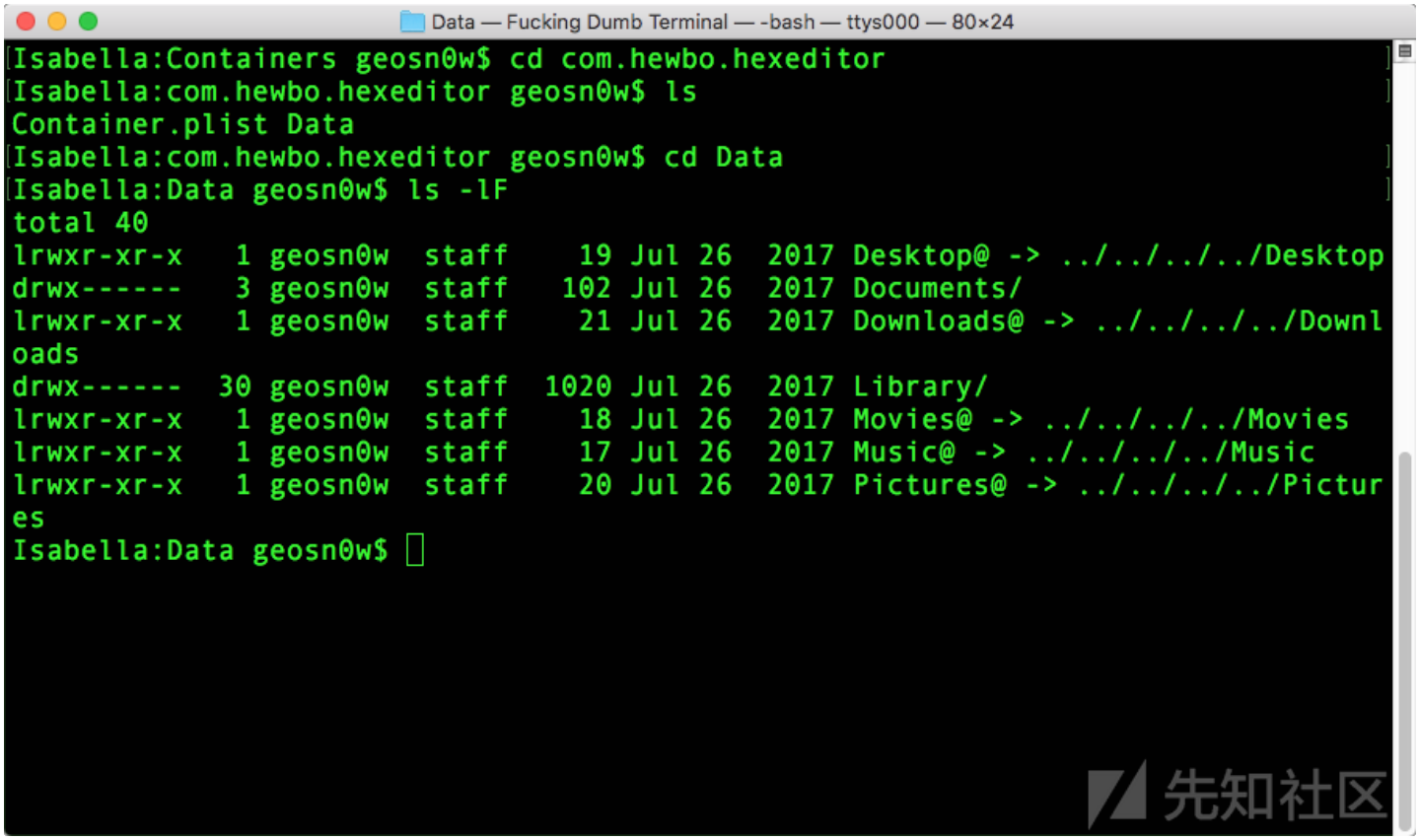
0x04 执行流程

进入沙盒容器中，也就是放置在`$HOME/Library/Containers/`上的文件夹。此文件夹是为任何沙盒程序创建的，而不管实际二进制文件安装在何处。文件夹遵循简单的约

找到iHEX 的 `Container` ，将目录切到上面提到的路径，然后运行`ls -lF com.hewbo.hexeditor`。`com.hewbo.hexeditor`是iHex的`CFBundleIndentifier`(在.app文件夹中可以找到`Info.Plist`)。



可以看到app的容器包含一个`Data`文件夹和前面提到的`Container.Plist`文件。数据文件夹非常有趣，如果将目录切到它，可以看到它模拟了用户的主目录。当然，所有



0x05 沙盒化

在内部启动应用程序时，内核将调用mac_execve函数，可以在XNU源代码中看到。__mac_execve几乎会加载二进制文件，但它也会检查MAC label，看看是否应该强制执行沙箱。

```
/*
* __mac_execve
*
* Parameters:    uap->fname        File name to exec
*        uap->argp        Argument list
*        uap->envp        Environment list
*        uap->mac_p        MAC label supplied by caller
*
* Returns:    0            Success
*        EINVAL            Invalid argument
*        ENOTSUP            Not supported
*        ENOEXEC            Executable file format error
*    exec_activate_image:EINVAL    Invalid argument
*    exec_activate_image:EACCES    Permission denied
*    exec_activate_image:EINTR    Interrupted function
*    exec_activate_image:ENOMEM    Not enough space
*    exec_activate_image:EFAULT    Bad address
*    exec_activate_image:ENAMETOOLONG    Filename too long
*    exec_activate_image:ENOEXEC    Executable file format error
*    exec_activate_image:ETXTBSY    Text file busy [misuse of error code]
*    exec_activate_image:EBADEXEC    The executable is corrupt/unknown
*    exec_activate_image:???
*    mac_execve_enter:???
*
* TODO:    Dynamic linker header address on stack is copied via suword()
*/
int
__mac_execve(proc_t p, struct __mac_execve_args *uap, int32_t *retval)
{
   char *bufp = NULL;
   struct image_params *imgp;
   struct vnode_attr *vap;
   struct vnode_attr *origvap;
   int error;
   char alt_p_comm[sizeof(p->p_comm)] = {0};    /* for PowerPC */
```

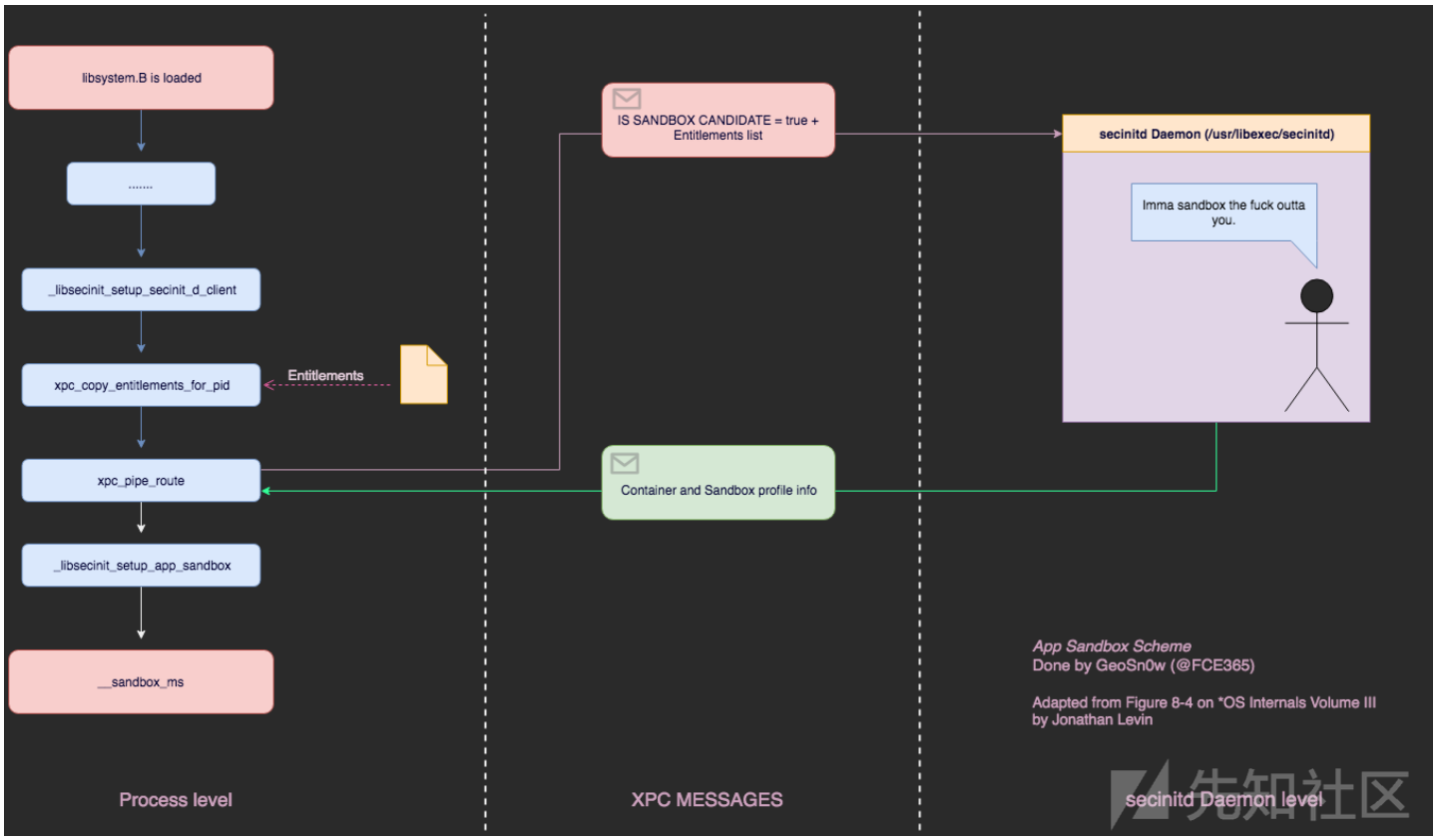
```
    int is_64 = IS_64BIT_PROCESS(p);
    struct vfs_context context;

    context.vc_thread = current_thread();
    context.vc_ucred = kauth_cred_proc_ref(p);    /* XXX must NOT be kauth_cred_get() */

    /* Allocate a big chunk for locals instead of using stack since these
     * structures a pretty big.
     */
    MALLOC(bufp, char *, (sizeof(*imgp) + sizeof(*vap) + sizeof(*origvap)), M_TEMP, M_WAITOK | M_ZERO);
    imgp = (struct image_params *) bufp;
    if (bufp == NULL) {
        error = ENOMEM;
        goto exit_with_error;
    }
    vap = (struct vnode_attr *) (bufp + sizeof(*imgp));
    origvap = (struct vnode_attr *) (bufp + sizeof(*imgp) + sizeof(*vap));

    /* Initialize the common data in the image_params structure */
    imgp->ip_user_fname = uap->fname;
    imgp->ip_user_argv = uap->argp;
    imgp->ip_user_envv = uap->envp;
    imgp->ip_vattr = vap;
    imgp->ip_origvattr = origvap;
    imgp->ip_vfs_context = &context;
    imgp->ip_flags = (is_64 ? IMGPF_WAS_64BIT : IMGPF_NONE) | ((p->p_flag & P_DISABLE_ASLR) ? IMGPF_DISABLE_ASLR : IMGPF_NONE);
    imgp->ip_p_comm = alt_p_comm;        /* for PowerPC */
    imgp->ip_seg = (is_64 ? UIO_USERSPACE64 : UIO_USERSPACE32);

#if CONFIG_MACF
    if (uap->mac_p != USER_ADDR_NULL) {
        error = mac_execve_enter(uap->mac_p, imgp);
        if (error) {
            kauth_cred_unref(&context.vc_ucred);
            goto exit_with_error;
        }
    }
#endif

    error = exec_activate_image(imgp);

    kauth_cred_unref(&context.vc_ucred);

    /* Image not claimed by any activator? */
    if (error == -1)
        error = ENOEXEC;

    if (error == 0) {
        exec_resettextvp(p, imgp);
        error = check_for_signature(p, imgp);
    }
    if (imgp->ip_vp != NULLVP)
        vnode_put(imgp->ip_vp);
    if (imgp->ip_strings)
        execargs_free(imgp);
#if CONFIG_MACF
    if (imgp->ip_execlabelp)
        mac_cred_label_free(imgp->ip_execlabelp);
    if (imgp->ip_scriptlabelp)
        mac_vnode_label_free(imgp->ip_scriptlabelp);
#endif
    if (!error) {
        struct uthread    *uthread;

        /* Sever any extant thread affinity */
        thread_affinity_exec(current_thread());

        DTRACE_PROC(exec__success);
        uthread = get_bsdthread_info(current_thread());
```

```
        if (uthread->uu_flag & UT_VFORK) {
            vfork_return(p, retval, p->p_pid);
            (void)thread_resume(imgp->ip_new_thread);
        }
    } else {
        DTRACE_PROC1(exec__failure, int, error);
    }

exit_with_error:
    if (bufp != NULL) {
        FREE(bufp, M_TEMP);
    }

    return(error);
}
```

当进程启动时，在其生命周期中很早就会加载libSystem.B。因为所有的APIs都依赖于它。在执行过程中的某个时刻，libSystem.B.initializer将落入_libsecini

secinitd
守护进程将承认这样一个事实:如果存在权限，沙盒应该被强制执行,那么它将调用AppSandbox.Framework来创建沙盒配置文件。创建概要文件之后，secinitd将返回一
message，其中包含CONTAINER_ID_KEY■CONTAINER_ROOT_PATH_KEY■SANDBOX_PROFILE_DATA_KEY和其他数据。该信息将由_libsecinit_setup_app_sandb

流程如下：



0x06 实验:跟踪运行时创建的程序沙盒

使用LLDB可以调试一个沙盒程序，并查看到底发生了什么，包括从进程传递到secinitd守护进程的XPC消息。即将深入了解Terminal和LLDB，下面的清单可能很难理解。
起初，打开终端并调用lldb。如果没有安装LLDB，请安装Xcode，因为它附带了您需要的所有调试工具。首先在xpc_pipe_routine和__sandbox_ms处下断点。

```
Last login: Thu Dec 27 16:44:59 on ttys000
Isabella:~ geosn0w$ lldb /Applications/iHex.app/Contents/MacOS/iHex
(lldb) target create "/Applications/iHex.app/Contents/MacOS/iHex"
Traceback (most recent call last):
 File "<string>", line 1, in <module>
 File "/Applications/Xcode.app/Contents/SharedFrameworks/LLDB.framework/Resources/Python/lldb/__init__.py", line 98, in <modul
   import six
ImportError: No module named six
Traceback (most recent call last):
 File "<string>", line 1, in <module>
NameError: name 'run_one_line' is not defined
Traceback (most recent call last):
```

```
 File "<string>", line 1, in <module>
Current executable set to '/Applications/iHex.app/Contents/MacOS/iHex' (x86_64).
(lldb) b xpc_pipe_routine
Breakpoint 1: where = libxpc.dylib`xpc_pipe_routine, address = 0x0000000000005c40
(lldb) b __sandbox_ms
Breakpoint 2: where = libsystem_kernel.dylib`__mac_syscall, address = 0x000000000001c648
(lldb) run
Process 12594 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
   frame #0: 0x00007fff6a75ec40 libxpc.dylib`xpc_pipe_routine
libxpc.dylib`xpc_pipe_routine:
->  0x7fff6a75ec40 <+0>: pushq  %rbp
   0x7fff6a75ec41 <+1>: movq   %rsp, %rbp
   0x7fff6a75ec44 <+4>: pushq  %r15
   0x7fff6a75ec46 <+6>: pushq  %r14
Target 0: (iHex) stopped.
(lldb) c
Process 12594 resuming
Process 12594 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
   frame #0: 0x00007fff6a75ec40 libxpc.dylib`xpc_pipe_routine
libxpc.dylib`xpc_pipe_routine:
->  0x7fff6a75ec40 <+0>: pushq  %rbp
   0x7fff6a75ec41 <+1>: movq   %rsp, %rbp
   0x7fff6a75ec44 <+4>: pushq  %r15
   0x7fff6a75ec46 <+6>: pushq  %r14
Target 0: (iHex) stopped.
```

然后在libxpc.dylib中停在xpc_pipe_.routine。做一个backtrace来看看发生了什么，可以通过bt命令来实现这一点。

```
(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
 * frame #0: 0x00007fff6a75ec40 libxpc.dylib`xpc_pipe_routine
   frame #1: 0x00007fff6a75eaad libxpc.dylib`_xpc_interface_routine + 167
   frame #2: 0x00007fff6a7650b5 libxpc.dylib`_xpc_uncork_domain + 529
   frame #3: 0x00007fff6a75ad85 libxpc.dylib`_libxpc_initializer + 1053
   frame #4: 0x00007fff680aa9c8 libSystem.B.dylib`libSystem_initializer + 126
   frame #5: 0x0000000100582ac6 dyld`ImageLoaderMachO::doModInitFunctions(ImageLoader::LinkContext const&) + 420
   frame #6: 0x0000000100582cf6 dyld`ImageLoaderMachO::doInitialization(ImageLoader::LinkContext const&) + 40
   ...
   frame #18: 0x0000000010056d3d4 dyld`dyldbootstrap::start(macho_header const*, int, char const**, long, macho_header const*,
   frame #19: 0x0000000010056d1d2 dyld`_dyld_start + 54
(lldb) c
Process 12594 resuming
Process 12594 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
   frame #0: 0x00007fff6a75ec40 libxpc.dylib`xpc_pipe_routine
libxpc.dylib`xpc_pipe_routine:
->  0x7fff6a75ec40 <+0>: pushq  %rbp
   0x7fff6a75ec41 <+1>: movq   %rsp, %rbp
   0x7fff6a75ec44 <+4>: pushq  %r15
   0x7fff6a75ec46 <+6>: pushq  %r14
Target 0: (iHex) stopped.
```

很明显这个不是我们所需要的，这是libxpc.dylib的_xpc_uncork_domain函数。我们需要xpc_pipe_create■按c继续并再次回溯。

```
(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
 * frame #0: 0x00007fff6a75ec40 libxpc.dylib`xpc_pipe_routine
   frame #1: 0x00007fff6a75eaad libxpc.dylib`_xpc_interface_routine + 167
   frame #2: 0x00007fff6a75e5d3 libxpc.dylib`bootstrap_look_up3 + 185
   frame #3: 0x00007fff6a75e4ff libxpc.dylib`bootstrap_look_up2 + 41
   frame #4: 0x00007fff6a7609d7 libxpc.dylib`xpc_pipe_create + 60
   frame #5: 0x00007fff6a500485 libsystem_info.dylib`_mbr_xpc_pipe + 261
   frame #6: 0x00007fff6a50033f libsystem_info.dylib`_mbr_od_available + 15
   frame #7: 0x00007fff6a4fffe5 libsystem_info.dylib`mbr_identifier_translate + 645
   frame #8: 0x00007fff6a4ffbf5 libsystem_info.dylib`mbr_identifier_to_uuid + 53
   frame #9: 0x00007fff6a4ffbba libsystem_info.dylib`mbr_uid_to_uuid + 42
   frame #10: 0x00007fff6a734db4 libsystem_secinit.dylib`_libsecinit_setup_secinitd_client + 728
   frame #11: 0x00007fff6a734a7b libsystem_secinit.dylib`_libsecinit_initialize_once + 13
```

```
   frame #12: 0x00007fff6a3d5db8 libdispatch.dylib`_dispatch_client_callout + 8
   frame #13: 0x00007fff6a3d5d6b libdispatch.dylib`dispatch_once_f + 41
   frame #14: 0x00007fff680aa9d2 libSystem.B.dylib`libSystem_initializer + 136
   ....
   frame #29: 0x0000000010056d1d2 dyld`_dyld_start + 54
```

找到所需的xpc_pipe_create函数。可以使用p (char *)
xpc_copy_description($rsi)查看通过XPC管道发送的消息，这调试非常有用。使用RSI寄存器作为消息的第二个参数(第一个参数是管道)。

```
(lldb) p (char *) xpc_copy_description($rsi)
(char *) $0 = 0x0000000101101fa0 "<dictionary: 0x10100c430> { count = 9, transaction: 0, voucher = 0x0, contents =\n\t"subsyst
```

这也不是所需要的。这只是一个握手信息，继续。

```
(lldb) c
Process 12594 resuming
Process 12594 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
   frame #0: 0x00007fff6a75ec40 libxpc.dylib`xpc_pipe_routine
libxpc.dylib`xpc_pipe_routine:
->  0x7fff6a75ec40 <+0>: pushq  %rbp
   0x7fff6a75ec41 <+1>: movq   %rsp, %rbp
   0x7fff6a75ec44 <+4>: pushq  %r15
   0x7fff6a75ec46 <+6>: pushq  %r14
Target 0: (iHex) stopped.
...
(lldb) c
Process 12594 resuming
Process 12594 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
   frame #0: 0x00007fff6a75ec40 libxpc.dylib`xpc_pipe_routine
libxpc.dylib`xpc_pipe_routine:
->  0x7fff6a75ec40 <+0>: pushq  %rbp
   0x7fff6a75ec41 <+1>: movq   %rsp, %rbp
   0x7fff6a75ec44 <+4>: pushq  %r15
   0x7fff6a75ec46 <+6>: pushq  %r14
Target 0: (iHex) stopped.
(lldb) p (char *) xpc_copy_description($rsi)
(char *) $5 = 0x0000000102821a00 "<dictionary: 0x1010051b0> { count = 11, transaction: 0, voucher = 0x0, contents =\n\t"SECINI
(lldb)
```

包含程序的权限以及它是否是沙盒的候选项的宝贵信息。正如所见，SECINITD_REGISTRATION_MESSAGE_IS_SANDBOX_CANDIDATE_KEY设置为bool
true，并且确实拥有com.apple.security.app-sandbox权限。

可以看到了进程发送给secinitd的内容，看是否正在创建沙盒。使用设置的第二个断点，即__sandbox_ms上的断点，继续(c)直到找到它。

```
(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
 * frame #0: 0x00007fff6a55f648 libsystem_kernel.dylib`__mac_syscall
   frame #1: 0x00007fff6a731bc9 libsystem_sandbox.dylib`sandbox_container_path_for_pid + 63
   frame #2: 0x00007fff6a4edd0c libsystem_coreservices.dylib`_dirhelper_init + 159
   frame #3: 0x00007fff6a71cf00 libsystem_platform.dylib`_os_once + 33
   frame #4: 0x00007fff6a4ee754 libsystem_coreservices.dylib`_dirhelper + 1873
   frame #5: 0x00007fff6a4604e9 libsystem_c.dylib`confstr + 525
   frame #6: 0x00007fff6a7354a5 libsystem_secinit.dylib`_libsecinit_setup_app_sandbox + 474 # As you can see, the Sandbox is s
   frame #7: 0x00007fff6a734a82 libsystem_secinit.dylib`_libsecinit_initialize_once + 20
   frame #8: 0x00007fff6a3d5db8 libdispatch.dylib`_dispatch_client_callout + 8
   frame #9: 0x00007fff6a3d5d6b libdispatch.dylib`dispatch_once_f + 41
   frame #10: 0x00007fff680aa9d2 libSystem.B.dylib`libSystem_initializer + 136
   frame #11: 0x0000000100582ac6 dyld`ImageLoaderMachO::doModInitFunctions(ImageLoader::LinkContext const&) + 420
   frame #12: 0x0000000100582cf6 dyld`ImageLoaderMachO::doInitialization(ImageLoader::LinkContext const&) + 40
   frame #13: 0x0000000010057e218 dyld`ImageLoader::recursiveInitialization(ImageLoader::LinkContext const&, unsigned int, char
   frame #14: 0x0000000010057e1ab dyld`ImageLoader::recursiveInitialization(ImageLoader::LinkContext const&, unsigned int, char
   frame #15: 0x0000000010057e1ab dyld`ImageLoader::recursiveInitialization(ImageLoader::LinkContext const&, unsigned int, char
   frame #16: 0x0000000010057e1ab dyld`ImageLoader::recursiveInitialization(ImageLoader::LinkContext const&, unsigned int, char
   frame #17: 0x0000000010057e1ab dyld`ImageLoader::recursiveInitialization(ImageLoader::LinkContext const&, unsigned int, char
   frame #18: 0x0000000010057e1ab dyld`ImageLoader::recursiveInitialization(ImageLoader::LinkContext const&, unsigned int, char
   frame #19: 0x0000000010057e1ab dyld`ImageLoader::recursiveInitialization(ImageLoader::LinkContext const&, unsigned int, char
   frame #20: 0x0000000010057d34e dyld`ImageLoader::processInitializers(ImageLoader::LinkContext const&, unsigned int, ImageLoa
   frame #21: 0x0000000010057d3e2 dyld`ImageLoader::runInitializers(ImageLoader::LinkContext const&, ImageLoader::InitializerTi
```

```
   frame #22: 0x0000000010056e567 dyld`dyld::initializeMainExecutable() + 196
   frame #23: 0x0000000100573239 dyld`dyld::_main(macho_header const*, unsigned long, int, char const**, char const**, char co
   frame #24: 0x0000000010056d3d4 dyld`dyldbootstrap::start(macho_header const*, int, char const**, long, macho_header const*,
   frame #25: 0x0000000010056d1d2 dyld`_dyld_start + 54
(lldb)
```

接下来，调用libsystem_secinit的_libsecinit_setup_app_sandbox。这意味着沙盒已经创建好了，将在开始的时候把程序放入沙盒中。接下来的几个continue
' -[NSApplication init]启动应用程序。

```
(lldb) c
Process 13280 resuming
Process 13280 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
   frame #0: 0x00007fff6a55f648 libsystem_kernel.dylib`__mac_syscall
libsystem_kernel.dylib`__mac_syscall:
->  0x7fff6a55f648 <+0>:  movl   $0x200017d, %eax          ; imm = 0x200017D
   0x7fff6a55f64d <+5>:  movq   %rcx, %r10
   0x7fff6a55f650 <+8>:  syscall
   0x7fff6a55f652 <+10>: jae    0x7fff6a55f65c            ; <+20>
Target 0: (iHex) stopped.
(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
 * frame #0: 0x00007fff6a55f648 libsystem_kernel.dylib`__mac_syscall
   frame #1: 0x00007fff6a731646 libsystem_sandbox.dylib`sandbox_check_common + 322
   frame #2: 0x00007fff6a7318f9 libsystem_sandbox.dylib`sandbox_check_by_audit_token + 177
   frame #3: 0x00007fff43ae952e LaunchServices`_LSIsAuditTokenSandboxed + 149
   frame #4: 0x00007fff6a3d5db8 libdispatch.dylib`_dispatch_client_callout + 8
   frame #5: 0x00007fff6a3d5d6b libdispatch.dylib`dispatch_once_f + 41
   frame #6: 0x00007fff439c7ed1 LaunchServices`_LSIsCurrentProcessSandboxed + 178
   frame #7: 0x00007fff43ae92ec LaunchServices`_LSCheckMachPortAccessForAuditToken + 72
   frame #8: 0x00007fff43ae9448 LaunchServices`_LSCheckLSDServiceAccessForAuditToken + 153
   frame #9: 0x00007fff439c097a LaunchServices`_LSRegisterSelf + 64
   frame #10: 0x00007fff439b9a7c LaunchServices`_LSApplicationCheckIn + 5420
   frame #11: 0x00007fff40d7192c HIServices`_RegisterApplication + 4617
   frame #12: 0x00007fff40d7064c HIServices`GetCurrentProcess + 24
   frame #13: 0x00007fff417cf4ab HIToolbox`MenuBarInstance::GetAggregateUIMode(unsigned int*, unsigned int*) + 63
   frame #14: 0x00007fff417cf435 HIToolbox`MenuBarInstance::IsVisible() + 51
   frame #15: 0x00007fff3fa71197 AppKit`_NSInitializeAppContext + 35
   frame #16: 0x00007fff3fa70590 AppKit`-[NSApplication init] + 443
   frame #17: 0x00007fff3fa701e6 AppKit`+[NSApplication sharedApplication] + 138
   frame #18: 0x00007fff3fa718b2 AppKit`NSApplicationMain + 356
   frame #19: 0x0000000100001c04 iHex`___lldb_unnamed_symbol1$$iHex + 52
(lldb)
```

至此，程序沙盒化完成！

原文：https://geosn0w.github.io/A-Long-Evening-With-macOS's-Sandbox/