XPC Client Validation? Music to my ears!



Hacking macOS audio plugin installers for fun and thesis.

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % echo "$USER"
```

Dipl.-Ing. Florian Haselsteiner, BSc

Senior Product Security Engineer @ Bitpanda

CWES, CPTS, OSCP(+), COAE

Some CVEs, some privately disclosed vulnerabilities

I like Macs
I like to mess with unnecessarily deep frequencies



Some important notes:

„Wicked Wicked…“ ~ General Levy
„If you focus, you can see the speaker pulsate…“ ~ SP:MC
„BIM BIM“ ~ Casual Gabberz

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % env
```

Talk about the Apple in the room

Some things about XPC

A little history about audio plugin installers

Chapter 1 – Who needs authentication anyways?
Chapter 2 – HTB taught me 777 things
Chapter 3 – I used to know how to use openssl!
Chapter 4 – Codesigning is optional...
Chapter 5 – A race you can barely lose.

Closing thoughts

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % systemstats
```

These are known exploit techniques!

- Kudos to **Csaba Fitzl**, **Wojciech Reguła** and co.
- As I just found out, hacktricks has a good docu about this as well!

This research was performed as part of my master's thesis.

- Looking at macOS security, especially IPC
- Since I mess with electronic music from time to time, I had some things on my device

All exploitable vulnerabilities identified during the process have been disclosed responsibly via:



Vulnerability Lab

https://sec-consult.com/de/vulnerability-lab/

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % uname -a
```

It is **NOT a "FaNcY LINuX"**

 vs  = ??

Compared to other operating systems, there are quite a few additional security considerations:

- SIP
- Codesigning
- TCC
- Gatekeeper
- XProtect
- AMFI
- ...



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % man csrutil
```

System Integrity Protection (**SIP**)

root (uid 0) != full access

Protects system files, processes and other stuff

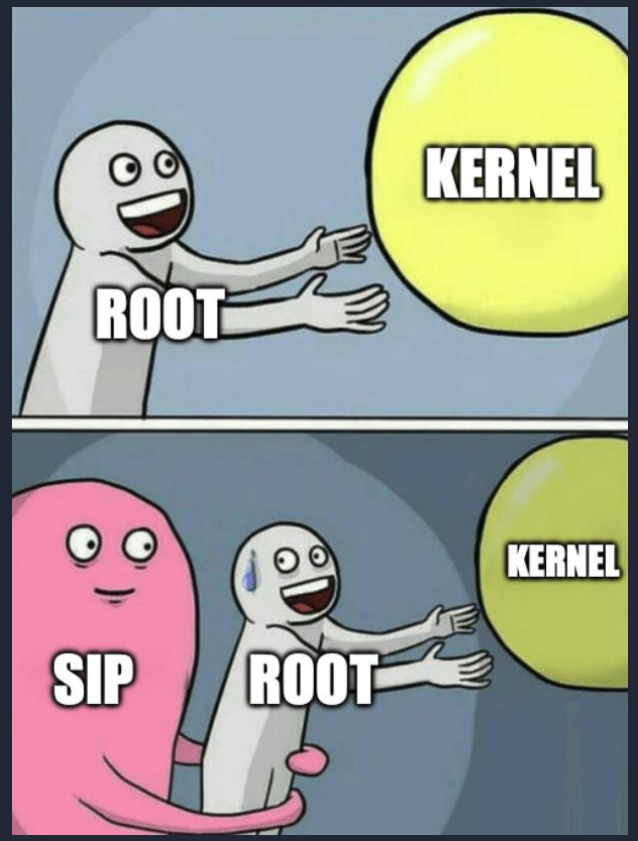
Adds kernel-enforced restrictions that **limit** what even **root processes** can modify or access

A cornerstone of macOS security

Enforces a **security boundary between processes**

- Foundation for macOS to enforce code-signing policy

There is more to it, but for now it does not matter



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % man codesign
```

Heavily utilized in Apple's operating systems

More than identify the signer
Configures **runtime protection**
    e.g. Hardened Runtime
Also grants **additional capabilities to binaries**
    So-called entitlements



Entitlements can:
    Grant access to privacy-relevant data
    Grant access to SIP-protected resources
    But more important for now:
        Can relax specific runtime protections!

Heavily used to **authenticate and authorize processes**

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % man xpc
```

IPC Framework by Apple

- Based on Mach messages
- Therefore, some other additional constraints apply

A big thing

At a high level, clients send dictionaries to server processes.

- **Server needs to establish whether the client is authorized** <- this is where the fun is…

XPC.h (C) vs NSXPC (ObjC)

- Sending dictionaries between processes.
- NSXPC abstracts this and acts as RPC

Often used with privileged helpers

- LaunchDaemons (System daemons)
- Used to perform privileged actions on behalf of non-privileged applications

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % ls -la /Library/PrivilegedHelperTools
```

Developers can utilize the concept of a privileged helper

- Developer is in charge
- Must write their own code

Installed together with the main application

LaunchDaemons (plist config stored in /Library/LaunchDaemons/)

Managed by Launchd (which is great!)

- Run in the system launchd domain and typically expose XPC service to unprivileged applications



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % ls -la /Library/PrivilegedHelperTools
```

Installed together with the main application

LaunchDaemons (Plist config stored in /Library/LaunchDaemons/)

Managed by Launchd (which is great!)

Run in the system launchd domain and typically expose XPC service to unprivileged applications



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % ls -la /Library/PrivilegedHelperTools
```

Installed together with the main application

LaunchDaemons (Plist config stored in /Library/LaunchDaemons/)

Managed by Launchd (which is great!)

Run in the system launchd domain and typically expose XPC service to unprivileged applications



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % ls -la /Library/PrivilegedHelperTools
```

To validate the client, the client has to be identified
- Using PID
- Using audit tokens

- At a minimum, client validation should consider:
  - "anchor apple generic" <- Often used
  - Developer ID / Team ID <- Often used
  - Bundle ID <- rarely used
  - Version <- very rarely used
  - No dangerous entitlements <- very rarely used
  - Hardened runtime <- very very rare (requires private APIs to be secure)



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % history
```

Audio plugin installers install audio plugins
- VST, VST3, AU

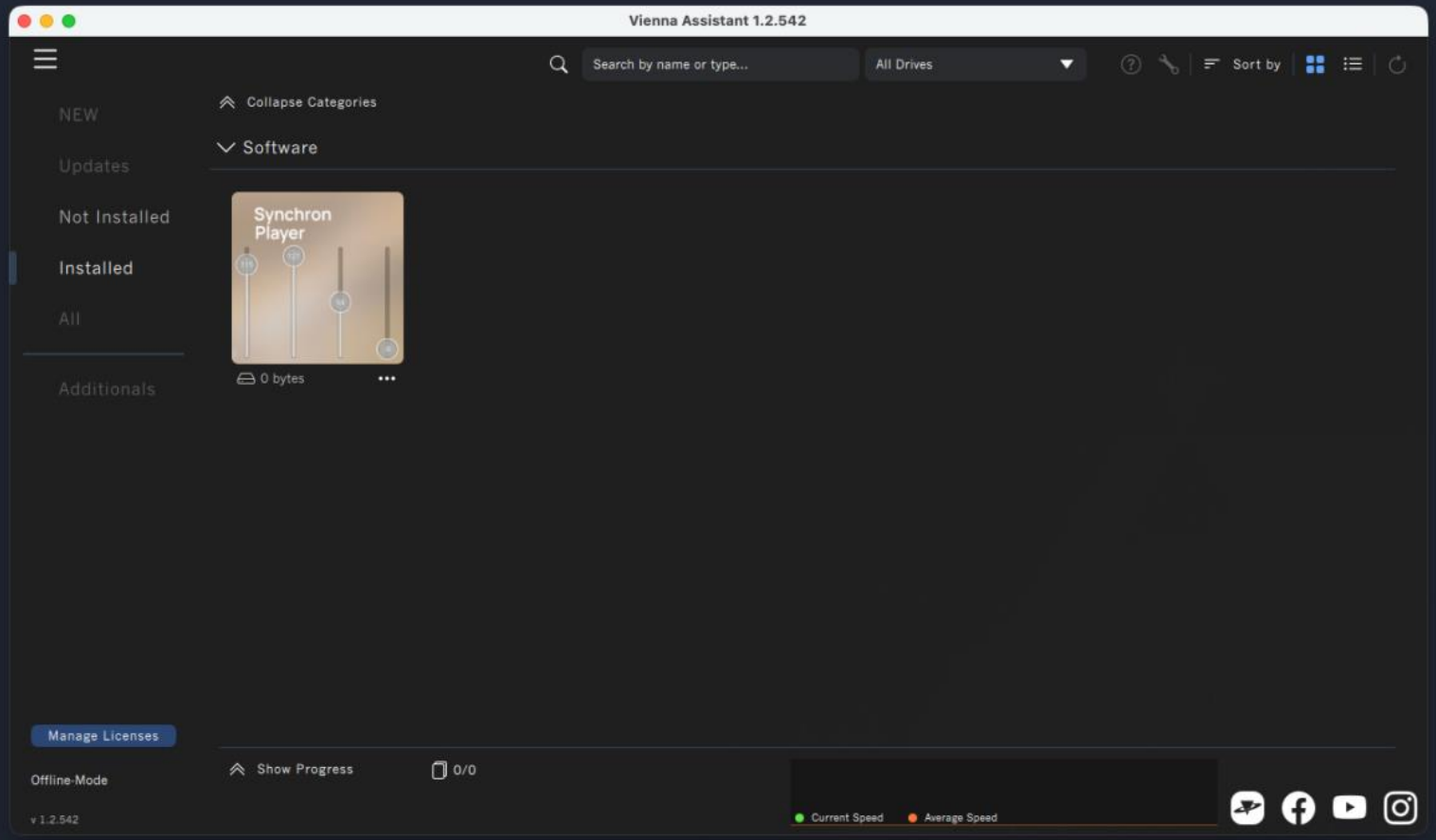
Default location is /Library/Audio/Plug-Ins/
- Owned by root
- Often utilize a privileged helper to write to this path

Such software has already been hacked in similar ways:
- Universal Audio (UA Connect) CVE-2024-8272
- Acustica Audio (Aquarius Desktop) CVE-2025-65842
- Plugin Alliance (PA Installation Manager) CVE-2025-55076

CVE-2025-65842 & CVE-2025-55076
- **Have not been patched** (at least until 02.2026)



```
user — -zsh — 85×23
user@Users-Virtual-Machine ~ % open /Applications/Vienna\ Assistant/Vienna\ Assistant.app
```



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x1000060b0 --stop-address=0x100006130 ./at.co.vsl.viennaassistant.helpertool
```

NSXPC uses the function **shouldAcceptNewConnection()** to determine if the connection should be allowed

- If allowed, the function resumes the connection and then returns YES (Objective-C for True)

```
1000060b0    bool -[HelperTool listener:shouldAcceptNewConnection:](struct HelperTool* self, SEL sel, id listener, id shouldAcceptNewConnection)

1000060b0    {
1000060b0        _NSLog(@"New Listener");
1000060fc        [shouldAcceptNewConnection setExportedInterface:[clsRef_NSXPCInterface
1000060fc            interfaceWithProtocol:protoRef_HelperToolProtocol]];
10000610c        [shouldAcceptNewConnection setExportedObject:self];
100006120        [shouldAcceptNewConnection resume];
100006130        return 1;
1000060b0    }
```

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x1000060b0 --stop-address=0x100006130 ./at.co.vsl.viennaassistant.helpertool
```

NSXPC uses the function **shouldAcceptNewConnection()** to determine if the connection should be allowed

- If allowed, the function resumes the connection and then returns YES (Objective-C for True)

```
1000060b0    bool -[HelperTool listener:shouldAcceptNewConnection:](struct HelperTool* self, SEL sel, id listener, id shouldAcceptNewConnection)

1000060b0    {
1000060b0        _NSLog(@"New Listener");
1000060fc        [shouldAcceptNewConnection setExportedInterface:[clsRef_NSXPCInterface
1000060fc            interfaceWithProtocol:protoRef_HelperToolProtocol]];
10000610         [shouldAcceptNewConnection setExportedObject:self];
10000612         [shouldAcceptNewConnection resume];
10000613         return 1;
1000060b0    }
```

Helper exposed runUninstaller:

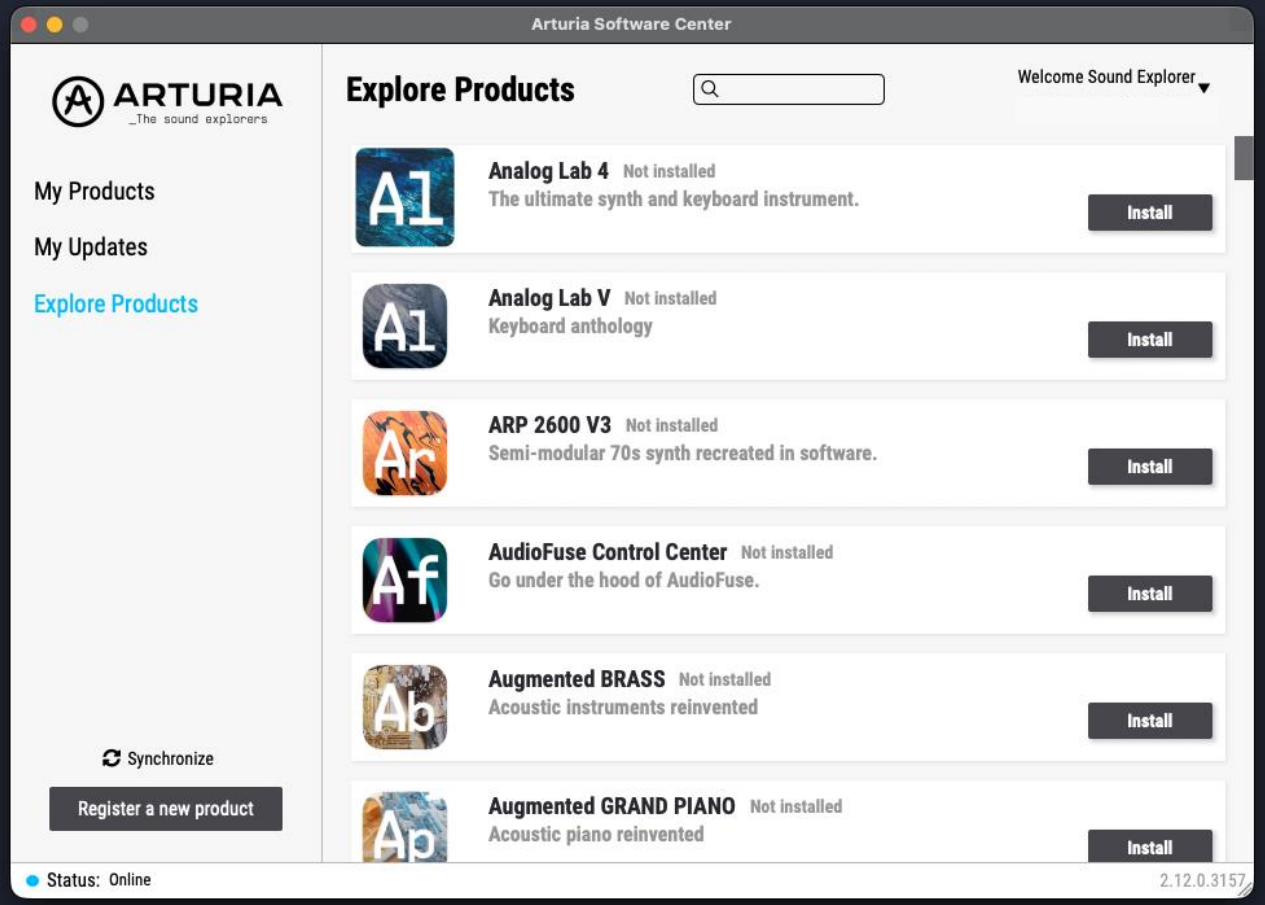
- Run any binary
- e.g. /bin/bash
- Also passes args to the binary

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % python3 meme.py
```



```
user — -zsh — 85×23
user@Users-Virtual-Machine ~ % open /Applications/Arturia/Arturia\ Software\ Center.app
```



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x10000bdbc --stop-address=0x10000be30 ./com.Arturia.InstallHelper
```

The XPC C API works differently

- Use **xpc_connection_set_event_handler()** to set the handler function for
  - Connection
  - Messages

Use **xpc_dictionary_get_<data_type>**

- Used to retrieve values from the XPC message dictionary

connection_handler2 does:

- First check if connection is error
- If not read data from dict and execute desired actions

No further client validation!

Everyone can connect!

```
10000bdbc   xpc_type_t connection_handler2(void* arg1, xpc_object_t arg2)

10000bdbc   {
10000bdbc       void* x24 = *(uint64_t*)((char*)arg1 + 0x20);
10000bde4       xpc_type_t result = _xpc_get_type(arg2);
10000bde4
10000bdf4       if (result == __xpc_type_error)
10000bdf4       {
10000be78           if (arg2 == __xpc_error_connection_invalid)
10000be78           {
10000bf00               *(uint8_t*)((char*)x24 + 0x80) = 1;
10000bf08               return std::promise<void>::set_value();
10000be78           }
10000be78
10000be90           if (arg2 == __xpc_error_termination_imminent
10000be90               && !(uint32_t)*(uint8_t*)((char*)x24 + 0x80))
10000be90           {
10000be98               *(uint8_t*)((char*)x24 + 0x80) = 1;
10000beb8               /* tailcall */
10000beb8               return _xpc_connection_cancel(*(uint64_t*)((char*)x24 + 0x70));
10000be90           }
10000bdf4       }
10000bdf4       else
10000bdf4       {
10000be0c           int32_t* key;
10000be0c
10000be0c           if ((int32_t)data_1000186d7 < 0)
10000be0c               key = (*(uint64_t*)data_1000186c0);
10000be0c           else
10000be0c               key = &data_1000186c0;
10000be0c
10000be14           char* x0_2 = _xpc_dictionary_get_string(arg2, key);
10000be30           int32_t* key_1;
10000be30
10000be30           if ((int32_t)data_1000186ef < 0)
10000be30               key_1 = (*(uint64_t*)data_1000186d8);
10000be30           else
```

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % xed arturia_softwareCenter_exploit.m
```

Basically RPC

- Using the dict key message
- Split by blank space
  - [0] command
  - [<=1] args

Two helpful “commands“:

FINISHM(string path)

Symlinks /Library/Arturia/ to path

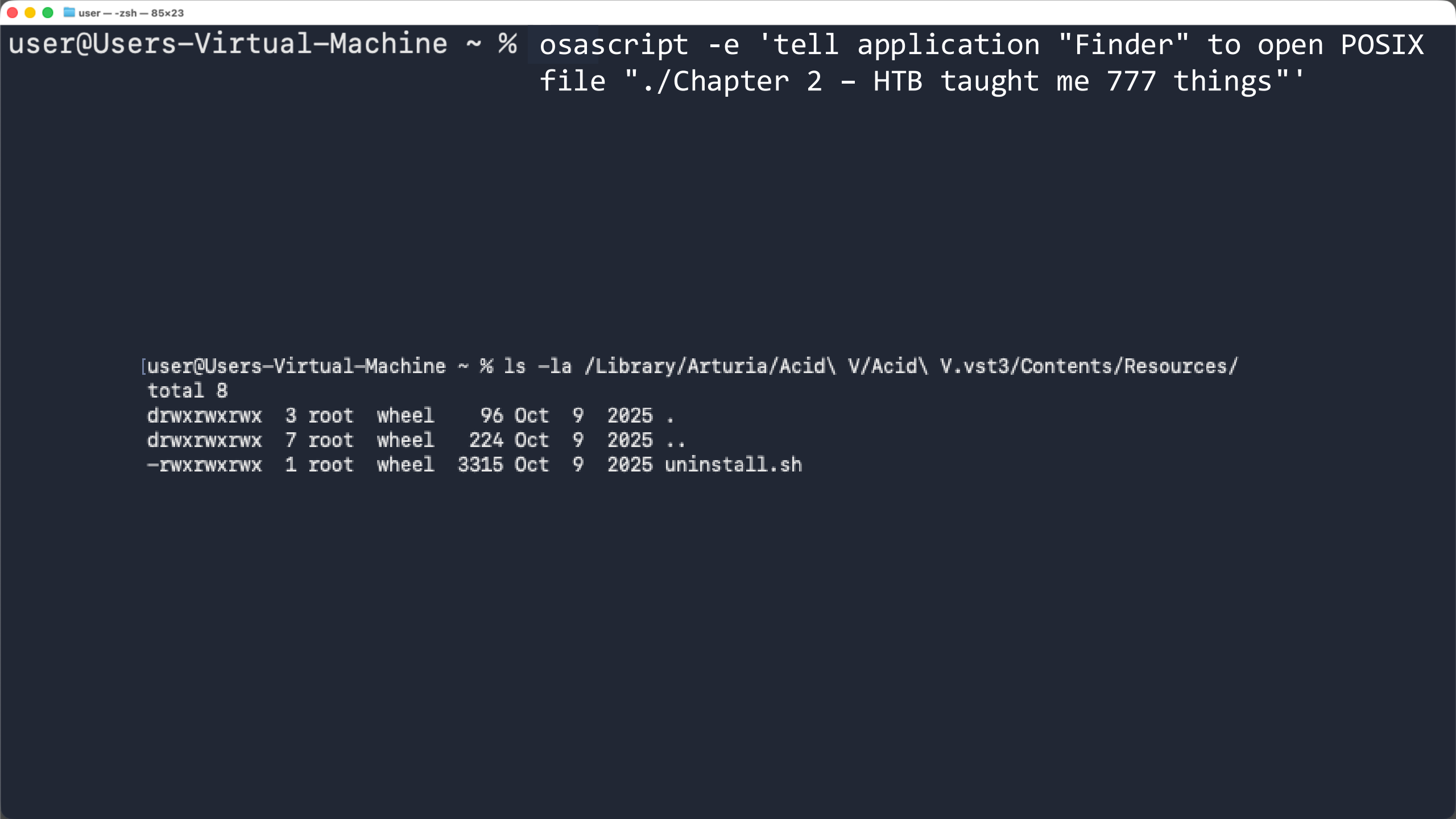
UNINSTA(string path)

Executes uninstall.sh script from /Library/Arturia/*

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % python3 meme.py
```




user — -zsh — 85×23
user@Users-Virtual-Machine ~ % osascript -e 'tell application "Finder" to open POSIX file "./Chapter 2 – HTB taught me 777 things"'
user@Users-Virtual-Machine ~ % ls -la /Library/Arturia/Acid\ V/Acid\ V.vst3/Contents/Resources/
total 8
drwxrwxrwx 3 root wheel   96 Oct 9 2025 .
drwxrwxrwx 7 root wheel   224 Oct 9 2025 ..
-rwxrwxrwx 1 root wheel 3315 Oct 9 2025 uninstall.sh

```
user — -zsh — 85×23
user@Users-Virtual-Machine ~ % osascript -e 'tell application "Finder" to open POSIX file "./Chapter 2 - HTB taught me 777 things"'

[user@Users-Virtual-Machine ~ % ls -la /Library/Arturia/Acid\ V/Acid\ V.vst3/Contents/Resources/
 total 8
 drwxrwxrwx  3 root  wheel    96 Oct  9  2025 .
 drwxrwxrwx  7 root  wheel   224 Oct  9  2025 ..
 -rwxrwxrwx  1 root  wheel  3315 Oct  9  2025 uninstall.sh
```

When installing a plugin the following path is created:

- /Library/Arturia/<pluginName>/<pluginName>.vst3/Contents/Resources
- Inside this directory uninstall.sh

Not directly a XPC issue

- However, it was exploitable via XPC
- On uninstall XPC message to Helper:
  - UNINSTA /Library/Arturia/<pluginName>/<pluginName>.vst3/Contents/Resources/uninstall.sh
- Remember there was no client validation...

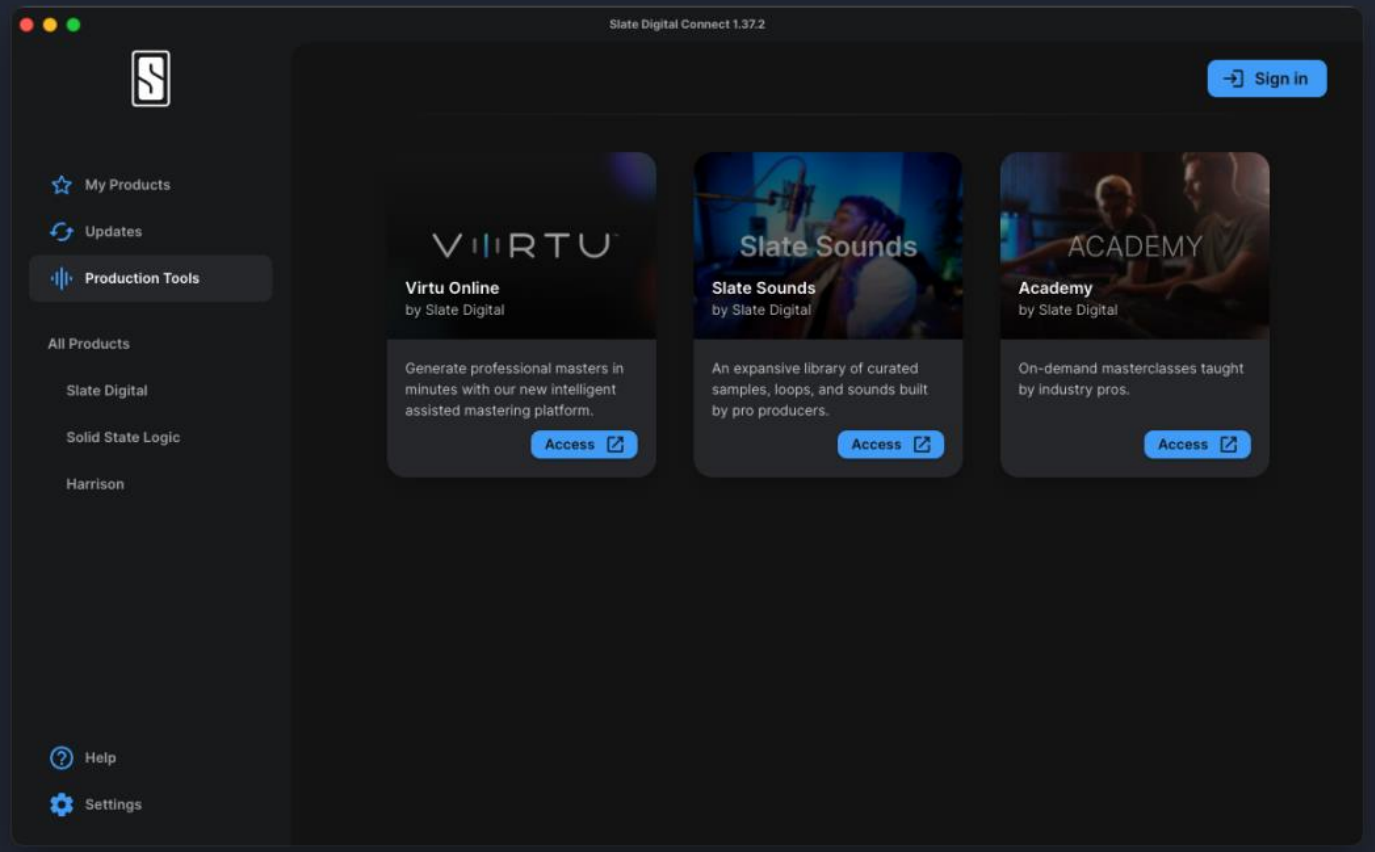


user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % python3 meme.py
```



```
user — -zsh — 85×23
user@Users-Virtual-Machine ~ % open /Applications/Slate\ Digital\ Connect.app
```



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x100002eb0 --stop-address=0x100002f40 ./com.slatedigital.connect.privileged.helper.tool
```

Helper utilizes isValidClient() to validate the XPC connection.

If isValidClient() returns true, the connection will be passed to the message handler.

```
100002eb0    int64_t _new_connection_handler(id arg1)

100002ec0        id obj = _objc_retain(obj: arg1)
100002ec0
100002ecc        if ((_isValidClient() & 1) == 0)
100002ed4            _xpc_connection_cancel(obj)
100002ed4
100002ee0        int64_t (* const var_48)() = __NSConcreteStackBlock
100002ef4        int64_t var_40 = 0xc2000000
100002f00        int64_t (* var_38)(void* arg1, id arg2) = ___new_connection_handler_block_invoke
100002f00        void* const var_30 =
100002f00            &___block_descriptor_40_e...33_v16 ?0 "NSObject<OS_xpc_object>"8l
100002f0c        id obj_1 = _objc_retain(obj)
100002f18        _xpc_connection_set_event_handler(obj_1, &var_48)
100002f20        _xpc_connection_resume(obj_1)
100002f28        _objc_release(obj)
100002f40        return _objc_release(obj: obj_1)
```

```
user — -zsh — 85×23
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x100002fbc --stop-address=0x100003064 ./com.slatedigital.connect.privileged.helper.tool
```

isValidClient() checks for a codesignature.

Using the security API.

Certificate requirement is a CFString

- We need to track it down
- Even though Binary Ninja already tells us what's inside.

```
100002fbc    int64_t _isValidClient()

100002fcc        SecRequirementRef cf_2 = nullptr
100002fcc        SecCodeRef var_28 = nullptr
100002fd4        int32_t valuePtr = _xpc_connection_get_pid()
100002fe0        CFAllocatorRef allocator = *_kCFAllocatorDefault
100002ff4        CFNumberRef values =
100002ff4            _CFNumberCreate(allocator, theType: kCFNumberSInt32Type, &valuePtr)
100003014        CFDictionaryRef cf = _CFDictionaryCreate(allocator, keys: _kSecGuestAttributePid,
100003014            &values, numValues: 1, keyCallBacks: nullptr, valueCallBacks: nullptr)
100003038        bool z
100003038
100003038        if (_SecCodeCopyGuestWithAttributes(__nullable: nullptr) == 0)
100003038            z = var_28 == 0
100003038        else
100003038            z = true
100003038
10000303c        int64_t result
10000303c        SecRequirementRef cf_1
10000303c
10000303c        if (z)
100003068        label_100003068:
100003068
100003068            if (cf != 0)
100003070                _CFRelease(cf)
100003070
100003078            if (var_28 != 0)
10000307c                _CFRelease(cf: var_28)
10000307c
100003084            result = 0
100003088            cf_1 = cf_2
100003088
10000308c            if (cf_1 != 0)
100003090                _CFRelease(cf: cf_1)
100003094                return 0
10000303c        else
100003054            if (_SecRequirementCreateWithString(
100003054                    text: &cfstr_certificate_leaf[subject.OU]_=_"3F5JHDQ8FZ", flags: 0,
100003054                    __nonnull: &cf_2) != 0)
100003054                goto label_100003068
100003054
100003064            if (_SecCodeCheckValidity(code: var_28, flags: 0, __nullable: cf_2) != 0)
100003064                goto label_100003068
```

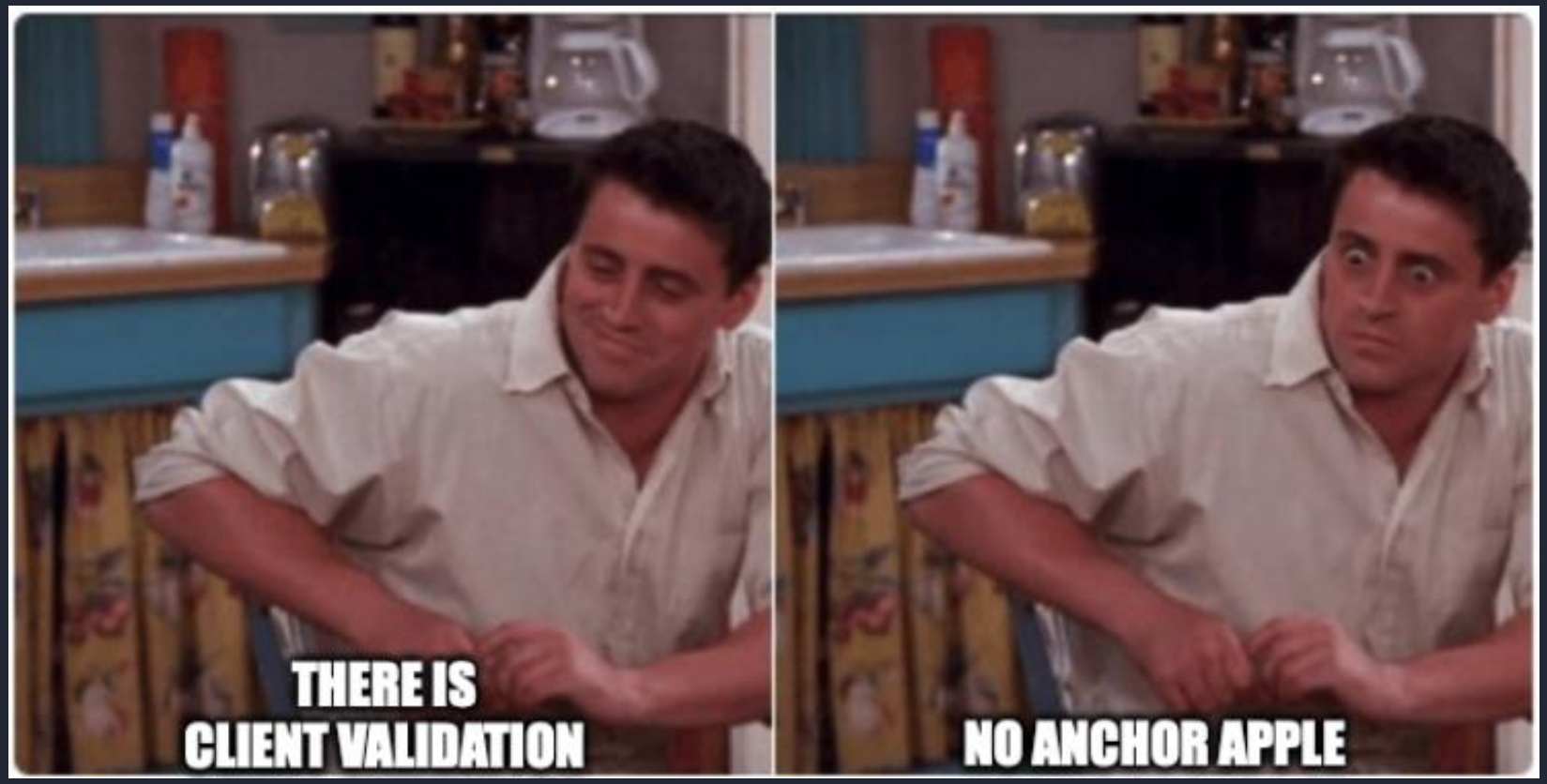
user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x1000041d8 --stop-address=0x1000041f8 ./com.slatedigital.connect.privileged.helper.tool
```

```
1000041d8 struct __NSConstantString cfstr_certificate_leaf[subject.OU]_=_"3F5JHDQ8FZ" =
1000041d8 {
1000041d8     void* isa = ___CFConstantStringClassReference
1000041e0     enum CFStringFlag flags = UTF8 | 0x7c0
1000041e8     char* data = cstr_certificate_leaf[subject.OU]_=_"3F5JHDQ8FZ" {"certificate leaf[subject.OU] = "3F5JHDQ8FZ""}
1000041f0     uint64_t length = 0x2b
1000041f8 }
```

user — -zsh — 85×23

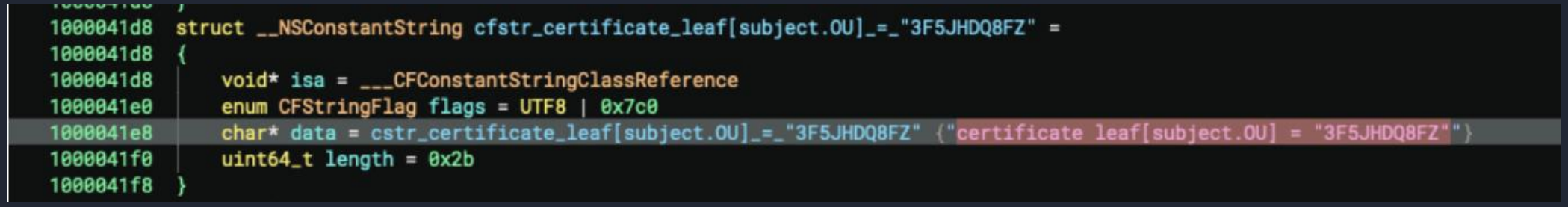
```
user@Users-Virtual-Machine ~ % python3 meme.py
```



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x1000041d8 --stop-address=0x1000041f8 ./com.slatedigital.connect.privileged.helper.tool
```

- At a minimum, client validation should consider:
  - ~~"anchor apple generic"~~ <- Often
  - **Developer ID / Team ID** <- Often
  - ~~Bundle ID~~ <- rare
  - ~~Version~~ <- very rare
  - ~~No dangerous entitlements~~ <- very rare
  - ~~Hardened runtime~~ <- very very rare (requires private APIs to be done securely!)



```
user@Users-Virtual-Machine ~ % man openssl
```

```
openssl genrsa -out codesign.key 4096; openssl req -new -x509 \
 -key codesign.key -out codesign.crt -days 3650 \
 -subj "/CN=My Self Signed Code Cert/OU=3F5JHDQ8FZ/O=Test Org/C=US" \
 -addext "keyUsage=digitalSignature" \
 -addext "extendedKeyUsage=codeSigning"

openssl pkcs12 -export -inkey codesign.key -in codesign.crt -out codesign.p12

codesign --sign "My Self Signed Code Cert" --force slateExploit
```

Then abuse the Command-request "function"

- Executes a shell command for us
- Very convenient!

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % osascript -e 'tell application "Finder" to open POSIX
                               file “./Chapter 4 – Codesigning is optional..."'
```

Hardened Runtime is great

- Protects against code injection
- Protects against debugger attaching
- Strengthens the security guarantees associated with code signing

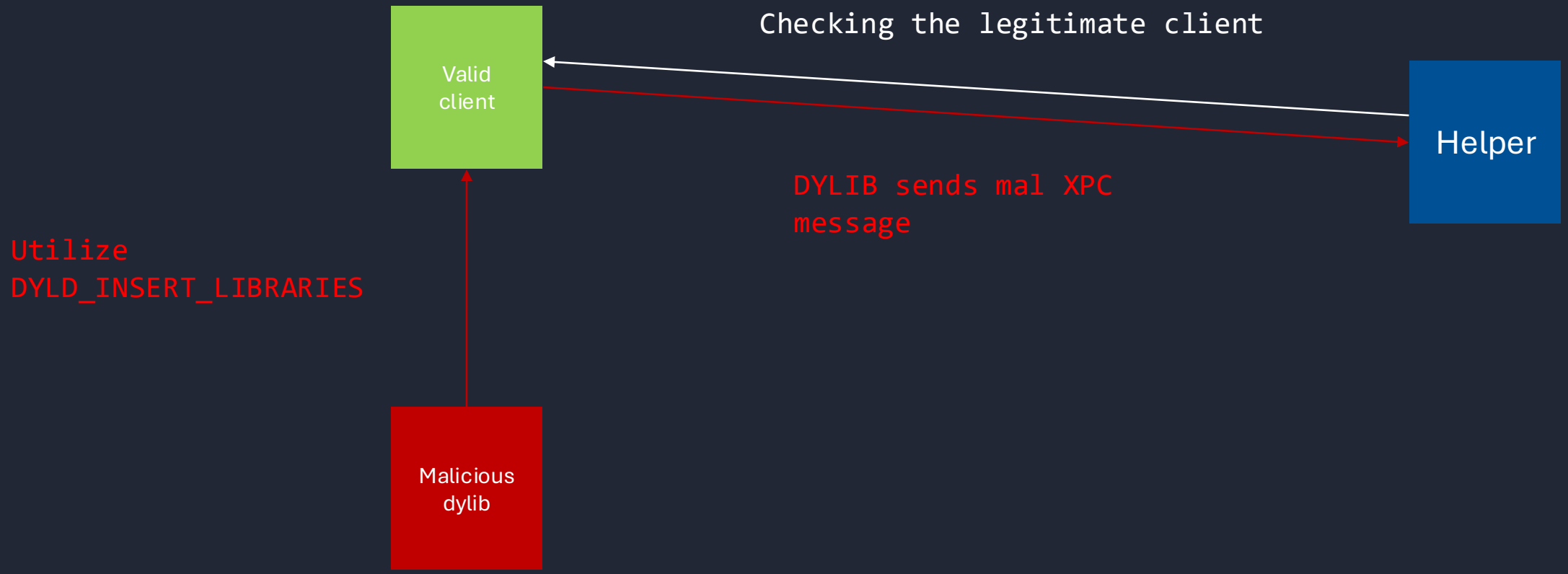
Hardened Runtime can be configured

- Set exceptions via Entitlements
- Most notably (for now):
  - **com.apple.security.cs.disable-library-validation**
  - **com.apple.security.cs.allow-dyld-environment-variables**

If code injection possible:

- The helper sees a legitimate, correctly signed client process, even though attacker-controlled code is executing inside that process

```
user@Users-Virtual-Machine ~ % osascript -e 'tell application "Finder" to open POSIX
file “./Chapter 4 – Codesigning is optional..."'
```



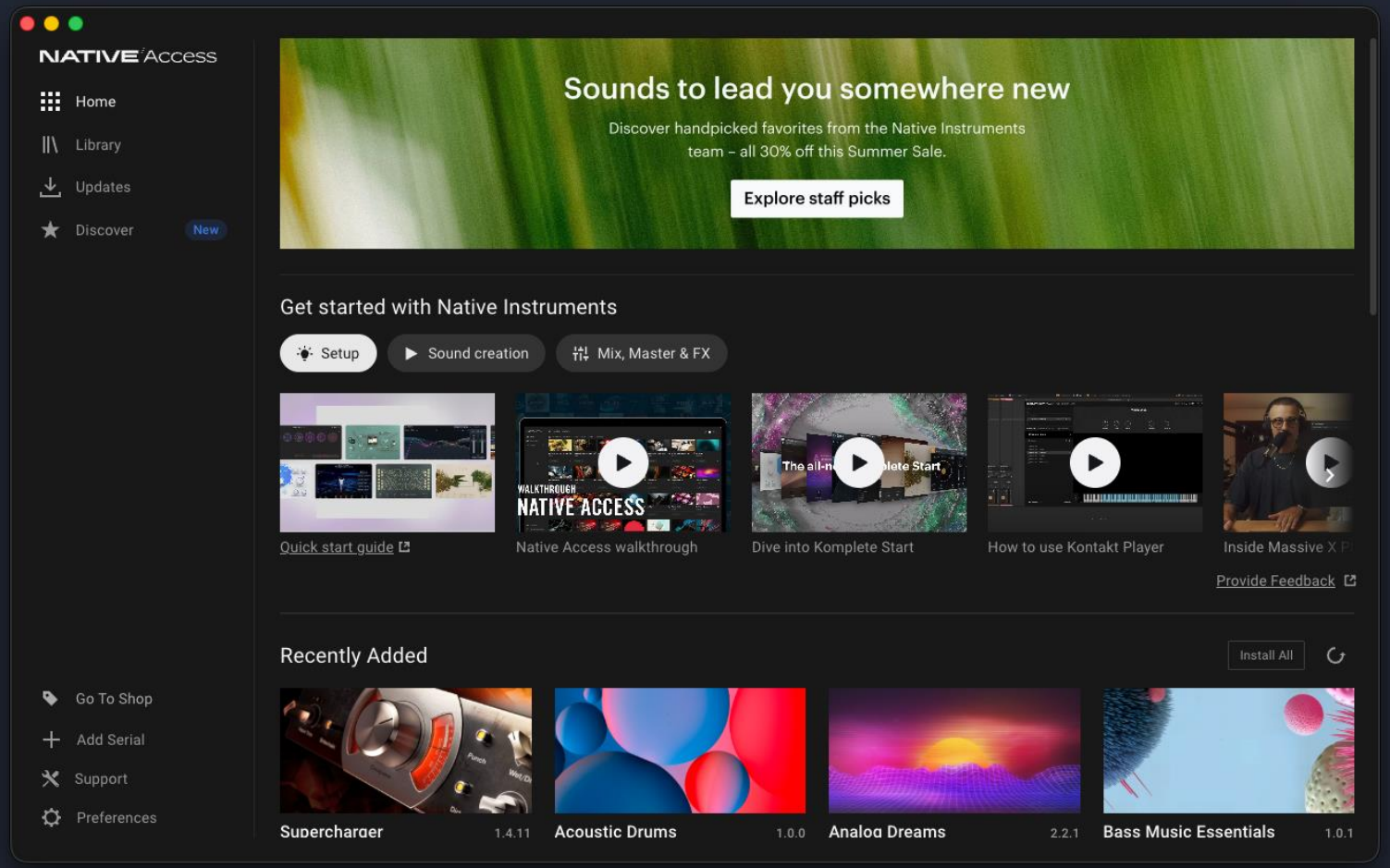
user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % osascript -e 'tell application "Finder" to open POSIX
                              file “./Chapter 4 – Codesigning is optional..."'
```



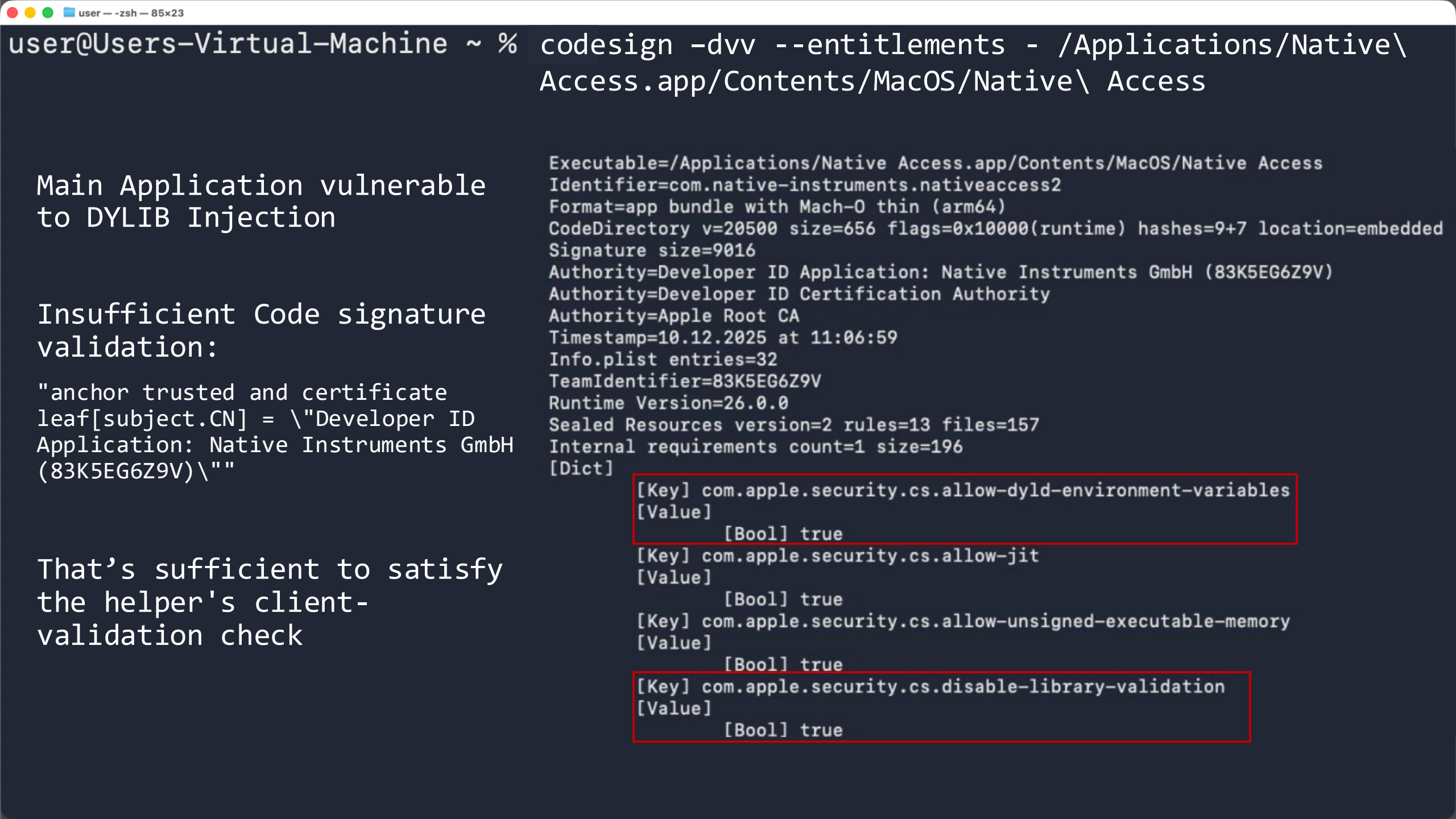
user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % open /Applications/Native\ Access.app
```



```
user@Users-Virtual-Machine ~ % codesign -dvv --entitlements - /Applications/Native\ Access.app/Contents/MacOS/Native\ Access

Executable=/Applications/Native Access.app/Contents/MacOS/Native Access
Identifier=com.native-instruments.nativeaccess2
Format=app bundle with Mach-O thin (arm64)
CodeDirectory v=20500 size=656 flags=0x10000(runtime) hashes=9+7 location=embedded
Signature size=9016
Authority=Developer ID Application: Native Instruments GmbH (83K5EG6Z9V)
Authority=Developer ID Certification Authority
Authority=Apple Root CA
Timestamp=10.12.2025 at 11:06:59
Info.plist entries=32
TeamIdentifier=83K5EG6Z9V
Runtime Version=26.0.0
Sealed Resources version=2 rules=13 files=157
Internal requirements count=1 size=196
[Dict]
        [Key] com.apple.security.cs.allow-dyld-environment-variables
        [Value]
                [Bool] true
        [Key] com.apple.security.cs.allow-jit
        [Value]
                [Bool] true
        [Key] com.apple.security.cs.allow-unsigned-executable-memory
        [Value]
                [Bool] true
        [Key] com.apple.security.cs.disable-library-validation
        [Value]
                [Bool] true
```



Main Application vulnerable to DYLIB Injection

Insufficient Code signature validation:

"anchor trusted and certificate leaf[subject.CN] = \"Developer ID Application: Native Instruments GmbH (83K5EG6Z9V)\""

That’s sufficient to satisfy the helper's client-validation check

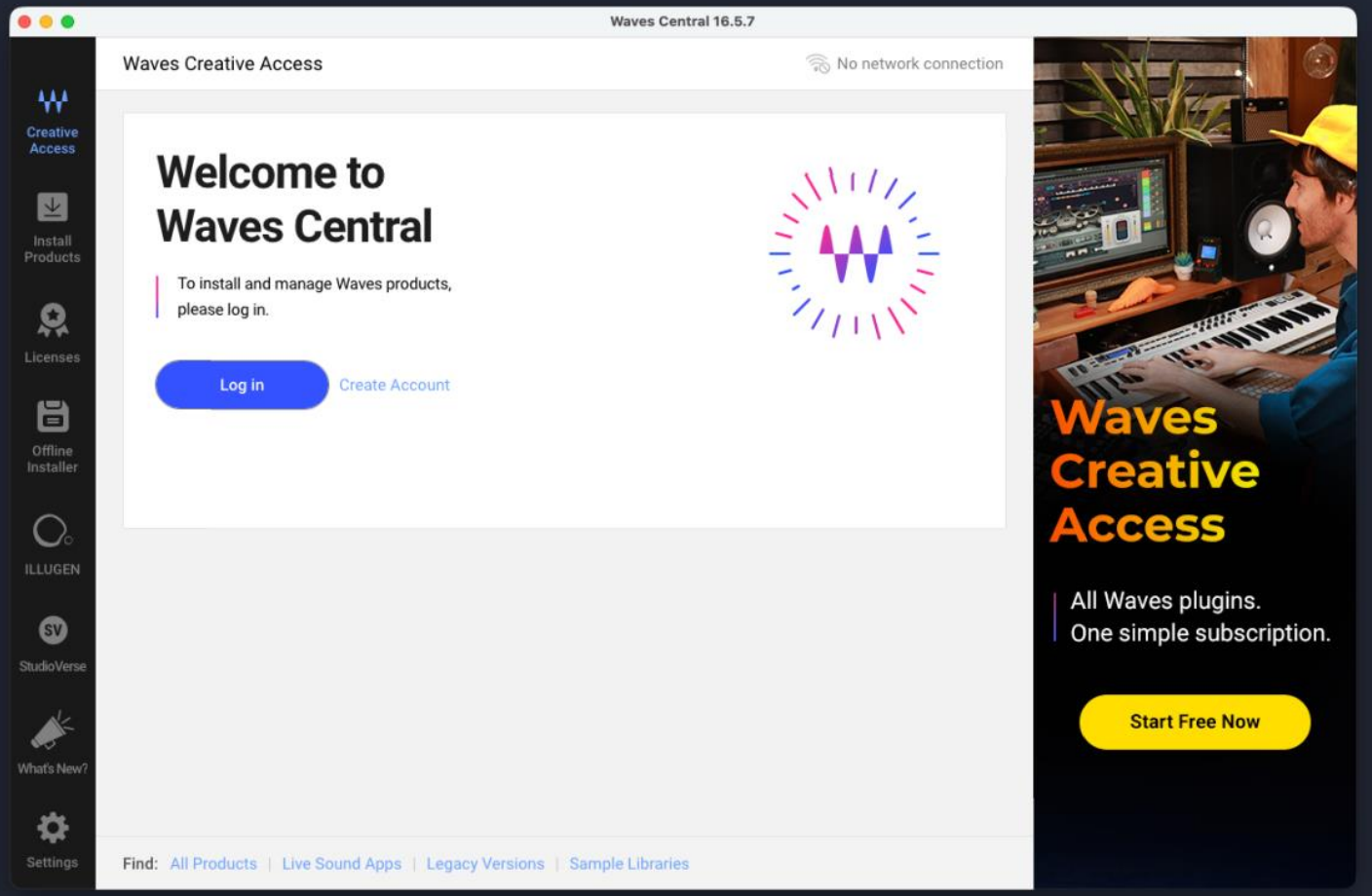
user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % clang -framework Foundation -dynamiclib -o
libxpcclient.dylib native_access_exploit_dylib.m;
DYLD_INSERT_LIBRARIES=./libxpcclient.dylib"/Applicatio
ns/Native Access.app/Contents/MacOS/Native Access"
```



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % open /Applications/Waves\ Central.app
```



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % codesign -dvv --entitlements - ./InstlHelperApplication
```

Legitimate XPC client vulnerable to DYLIB injection

No need to care about the validation

Because the legitimate client has to be permitted anyways

Enough to connect to the helper!

```
Executable=/Volumes/My Shared Files/Masterarbeit_Shared/PlanB/Targets/Targets/Clients/InstlHelperApplication
Identifier=com.waves.central.InstlHelperApplication
Format=Mach-O universal (x86_64 arm64)
CodeDirectory v=20500 size=1684 flags=0x10000(runtime) hashes=41+7 location=embedded
Signature size=8956
Authority=Developer ID Application: Waves Inc (GT6E3XD798)
Authority=Developer ID Certification Authority
Authority=Apple Root CA
Timestamp=12.02.2023 at 19:37:53
Info.plist=not bound
TeamIdentifier=GT6E3XD798
Runtime Version=11.1.0
Sealed Resources=none
Internal requirements count=1 size=200
[Dict]
        [Key] com.apple.security.inherit
        [Value]
                [Bool] true
        [Key] com.apple.security.network.client
        [Value]
                [Bool] true
        [Key] com.apple.security.network.server
        [Value]
                [Bool] true
        [Key] com.apple.security.files.bookmarks.app-scope
        [Value]
                [Bool] true
        [Key] com.apple.security.cs.disable-library-validation
        [Value]
                [Bool] true
        [Key] com.apple.security.files.bookmarks.document-scope
        [Value]
                [Bool] true
        [Key] com.apple.security.files.user-selected.read-write
        [Value]
                [Bool] true
        [Key] com.apple.security.personal-information.addressbook
        [Value]
                [Bool] true
        [Key] com.apple.security.cs.allow-dyld-environment-variables
        [Value]
                [Bool] true
        [Key] com.apple.security.cs.allow-unsigned-executable-memory
        [Value]
                [Bool] true
        [Key] com.apple.security.cs.disable-executable-page-protection
        [Value]
                [Bool] true
```

user — -zsh — 85×23

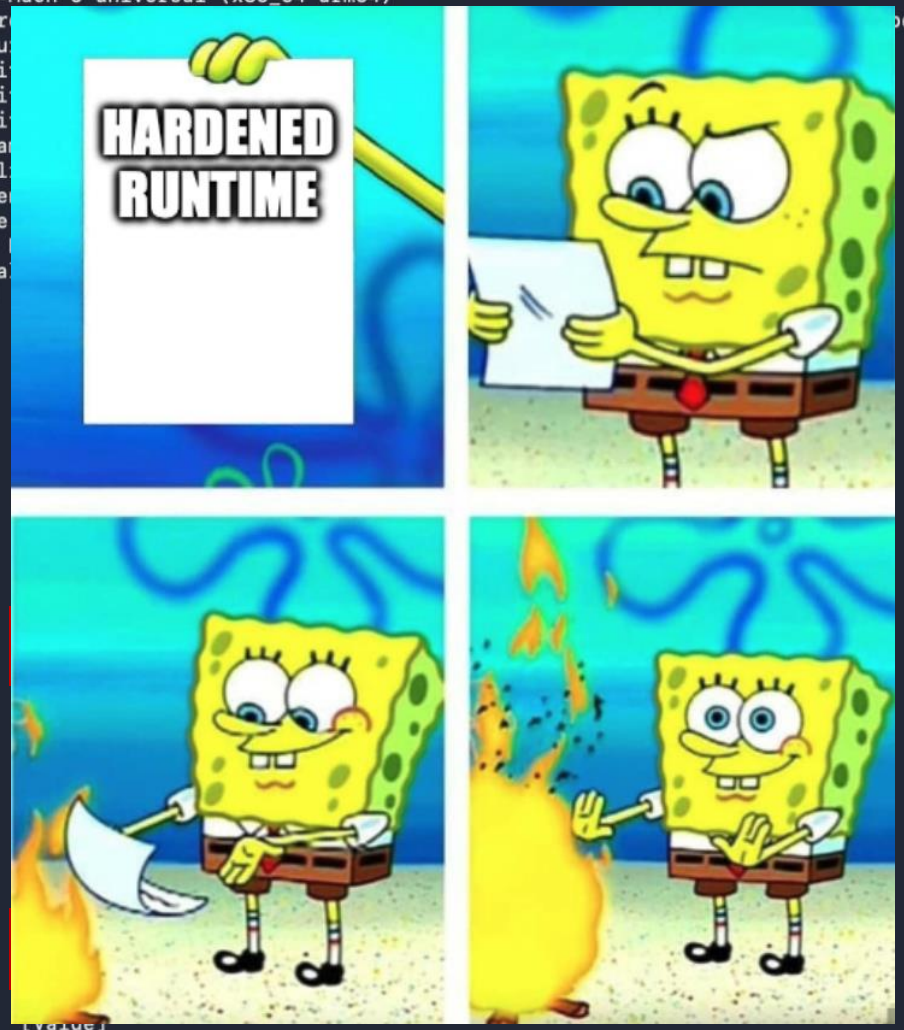
```
user@Users-Virtual-Machine ~ % codesign -dvv --entitlements - ./InstlHelperApplication
```

Legitimate XPC client vulnerable to DYLIB injection

Code execution as a service

- executeIrlFileWithPath
  - Executes plaintext files
  - Basically, bash commands
  - You can even define in which user context

```
Executable=/Volumes/My Shared Files/Masterarbeit_Shared/PlanB/Targets/Targets/Clients/InstlHelperApplication
Identifier=com.waves.central.InstlHelperApplication
Format=Mach-O universal (x86_64 arm64)
CodeDir
Signatu
Authori
Authori
Authori
Timesta
Info.pl
TeamIde
Runtime
Sealed
Interna
[Dict]
```




```
        [Bool] true
[Key] com.apple.security.cs.disable-executable-page-protection
[Value]
        [Bool] true
```

```
user@Users-Virtual-Machine ~ % osascript -e 'tell application "Finder" to open POSIX
                               file "./Chapter 5 - A race you can barely lose."'
```

When using the **Security API** to validate the client, you need to **identify the client.**

- Often the **PID** is used for that

The **PID is not stable** for the lifetime of a XPC message

- Since it can be reused
- Posix_spawn()

There can be a **race condition** between sending and validating the client.

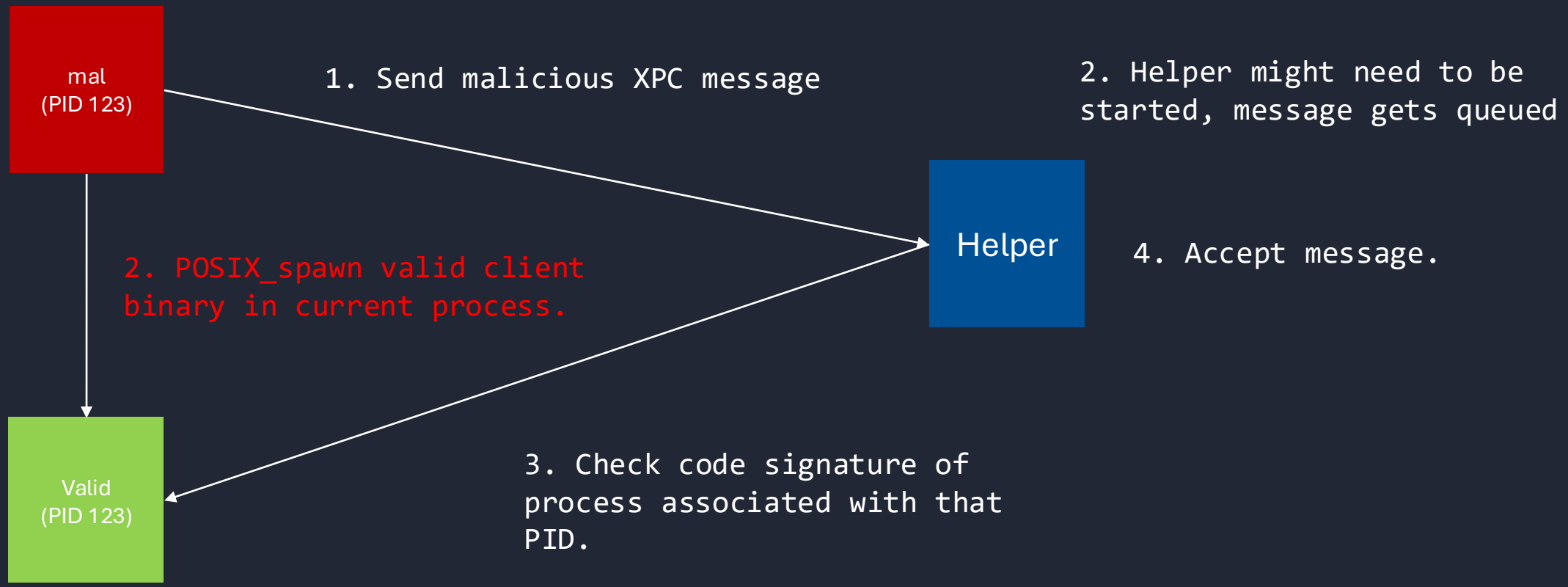
- Because the message has to be sent, which takes time, the validation may occure after the client identity has changed.
- The helper may need to be launched, increasing the race window.

If you don't want to be vulnerable to this:

- You can use XPC.h to validate the client (as discussed earlier)
- Or you need to use private APIs to utilize audit tokens.

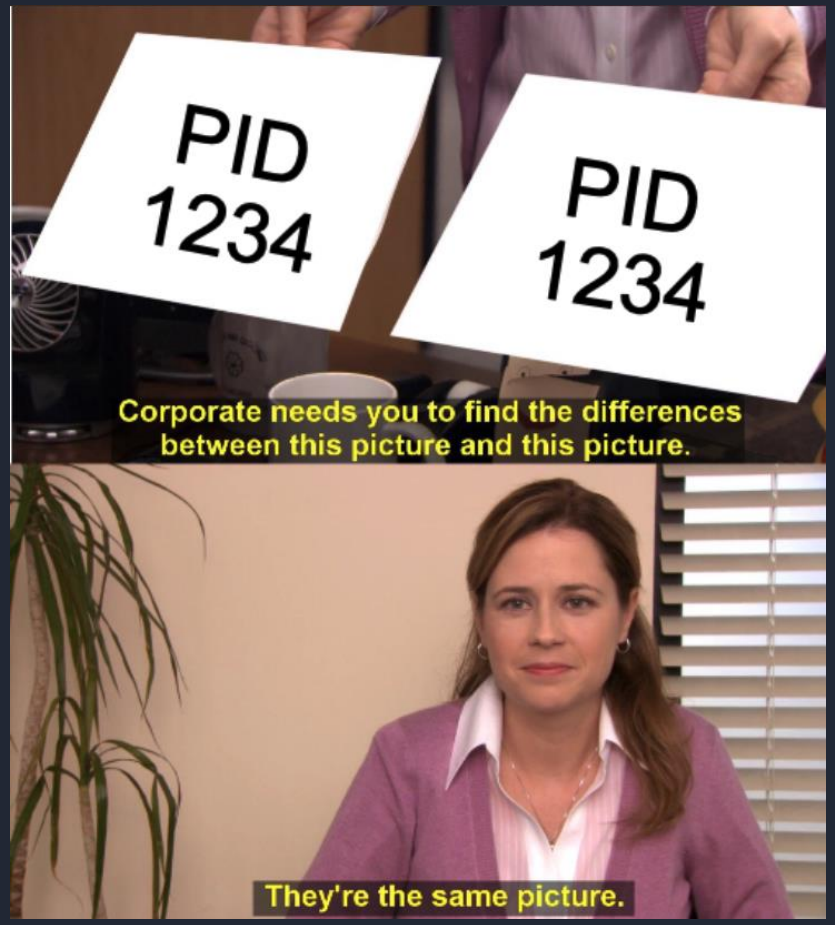
user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % osascript -e 'tell application "Finder" to open POSIX file “./Chapter 5 – A race you can barely lose."'
```



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % python3 meme.py
```



user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x100002fbc --stop-address=0x100003064 ./com.slatedigital.connect.privileged.helper.tool
```

Do you remember slate connect?

There was the isValidClient() function!

Take a closer look how the CS validation is performed.

YES - they use the PID to identify the client!

This is the primitive we need!

```
100002fbc    int64_t _isValidClient()

100002fcc        SecRequirementRef cf_2 = nullptr
100002fcc        SecCodeRef var_28 = nullptr
100002fd4        int32_t valuePtr = _xpc_connection_get_pid()
100002fe0        CFAllocatorRef allocator = *_kCFAllocatorDefault
100002ff4        CFNumberRef values =
100002ff4            _CFNumberCreate(allocator, theType: kCFNumberSInt32Type, &valuePtr)
100003014        CFDictionaryRef cf = _CFDictionaryCreate(allocator, keys: _kSecGuestAttributePid,
100003014            &values, numValues: 1, keyCallBacks: nullptr, valueCallBacks: nullptr)
100003038        bool z
100003038
100003038        if (_SecCodeCopyGuestWithAttributes(__nullable: nullptr) == 0)
100003038            z = var_28 == 0
100003038        else
100003038            z = true
100003038
1000303c        int64_t result
1000303c        SecRequirementRef cf_1
1000303c
1000303c        if (z)
100003068        label_100003068:
100003068
100003068            if (cf != 0)
100003070                _CFRelease(cf)
100003070
100003078            if (var_28 != 0)
1000307c                _CFRelease(cf: var_28)
1000307c
100003084            result = 0
100003088            cf_1 = cf_2
100003088
1000308c            if (cf_1 != 0)
100003090                _CFRelease(cf: cf_1)
100003094                return 0
1000303c        else
100003054            if (_SecRequirementCreateWithString(
100003054                    text: &cfstr_certificate_leaf[subject.OU]_=_"3F5JHDQ8FZ", flags: 0,
100003054                    __nonnull: &cf_2) != 0)
100003054                goto label_100003068
100003054
100003064            if (_SecCodeCheckValidity(code: var_28, flags: 0, __nullable: cf_2) != 0)
100003064                goto label_100003068
```

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x10000a60c --stop-address=0x10000a694 ./com.native-instruments.NativeAccess.Helper2
```

Native Access helper

Utilizes hasValidSignature(PID) to validate client.

From this call I would assume this is vulnerable

However, let's dig deeper.

```
10000a60c    int64_t ___main_block_invoke(int64_t arg1, xpc_object_t arg2)

10000a630        if (_xpc_get_type(object: arg2) != __xpc_type_connection)
10000a64c            return _syslog$DARWIN_EXTSN(5, "Unexpected type")
10000a64c
10000a65c        if ((hasValidSignature(_xpc_connection_get_pid(arg2)) & 1) == 0)
10000a66c            _syslog$DARWIN_EXTSN(5, "Refused connection from client with bad signature")
10000a674            _xpc_connection_cancel(arg2)
10000a674
10000a684        _xpc_connection_set_event_handler(arg2, &___block_literal_global)
10000a694        return _xpc_connection_activate(arg2) __tailcall
```

```
user — -zsh — 85×23
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x100005f60 --stop-address=0x100006050 ./com.native-instruments.NativeAccess.Helper2
```

hasValidSignature(PID)

PID is converted to a CFNumberRef (x0_2)

X0_2 is then used to get the Secure code signing information via SecCodeCopyGuestWithAttributes()

This is all we need!

```
100005f60    uint64_t hasValidSignature(int32_t arg1)

100005f60    {
100005f60        int32_t valuePtr = arg1;
100005f98        CFMutableDictionaryRef x0_1 = _CFDictionaryCreateMutable(
100005f98            *(uint64_t*)_kCFAllocatorDefault, 0, _kCFTypeDictionaryKeyCallBacks,
100005f98            _kCFTypeDictionaryValueCallBacks);
100005f98
100005f9c        if (!x0_1)
10000606c            return 0;
10000606c
100005fb0        CFNumberRef x0_2 = _CFNumberCreate(nullptr, kCFNumberIntType, &valuePtr);
100005fcc        _CFDictionarySetValue(x0_1, *(uint64_t*)_kSecGuestAttributePid, x0_2);
100005fe4        uint64_t result = 0;
100005fe4
100005fe8        if (!_SecCodeCopyGuestWithAttributes(nullptr))
100005fe8        {
100005ffc            result = 0;
100006004            SecRequirementRef var_48;
100006004            SecCodeRef var_40;
100006004
100006004            if (!_SecRequirementCreateWithString(
100006004                &_vtable_for_fmt::v11::format_error{for `std::runtime_error'}.
100006004                    ___CFConstantStringClassReference,
100006004                0, &var_48))
100006004            {
100006018                result = (uint64_t)(!_SecCodeCheckValidity(var_40, 0, var_48) ? 1 : 0);
100006020                _CFRelease(var_48);
100006004            }
100006004
100006028            _CFRelease(var_40);
100005fe8        }
100005fe8
100006030        _CFRelease(x0_2);
100006038        _CFRelease(x0_1);
100006050        return result;
100005f60    }
```

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % objdump -d --start-address=0x100005354 --stop-address=0x1000088ec ./com.waves.central.InstlHelper
```

Waves Central is written in Swift

Therefore, it uses NSXPC

Same issue as seen previously

PID is used to check the code signature

```
1000087d8    uint64_t _$s29com_waves_central_InstlHelper0E0C7isValid33_817A4106F53204769E97D313318C032FLL10connectionSbSo15NSXPCConnectionC_tFTf4nd_n(
1000087d8      id arg1)

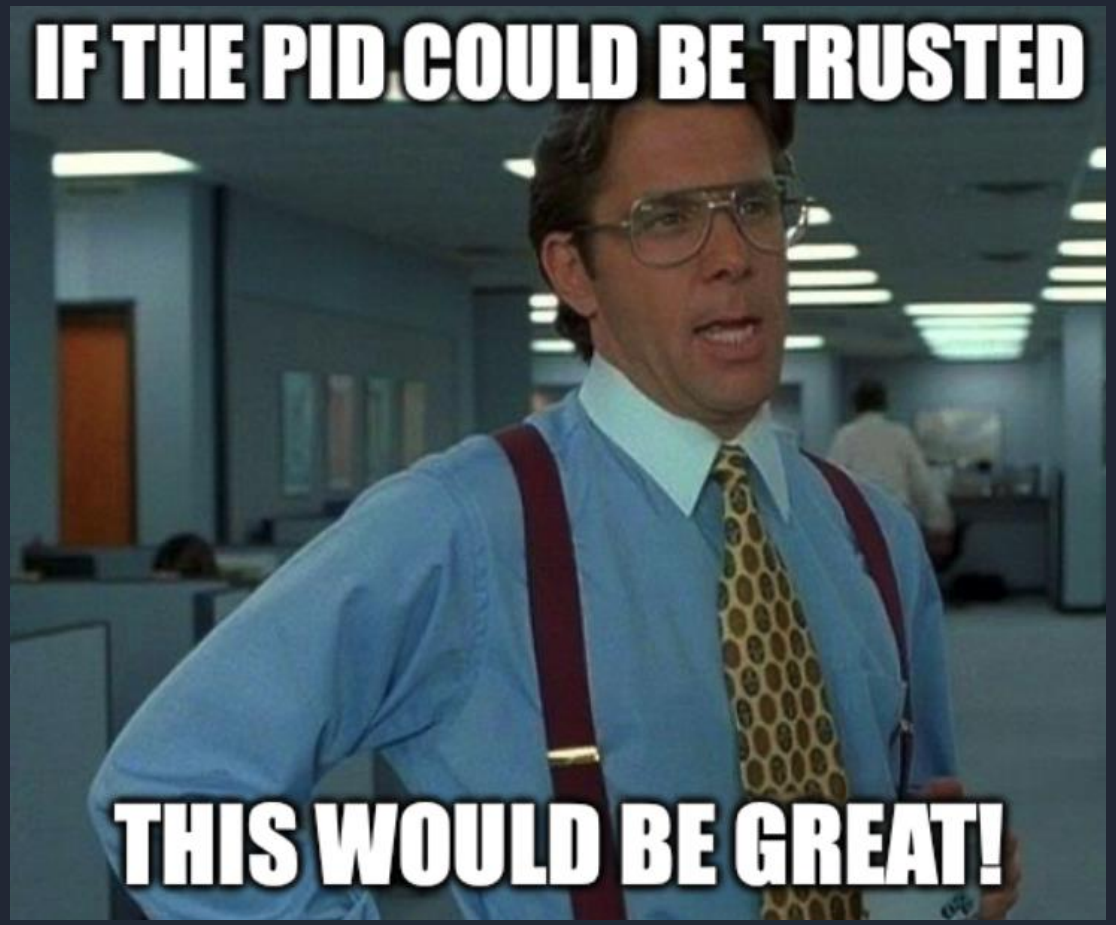
1000088ec    {
1000088ec        return (uint64_t)_$s29com_waves_central_I...ningMatches3pidSbs5Int32V_tKFZTf4nd_n(
1000088ec            [arg1 processIdentifier], 0) & 1;
1000088ec    }
```

```
100005354    uint64_t _$s29com_waves_central_InstlHelper13CodesignCheckV18codeSigningMatches3pidSbs5Int32V_tKFZTf4nd_n(int64_t (* pid)(), int64_t arg2 @ x2

100005354    {
100005354        SecStaticCodeRef obj =
100005354            _$s29com_waves_central_I...1D2500A5BLLSo03SeciJ3RefaSgyKFZTf4d_n(arg2);
10000536c        int64_t x19 = arg2;
10000536c
100005370        if (!arg2)
100005370        {
100005378            int64_t (* x20_1)();
100005378
100005378            if (!obj)
100005378            {
1000053a0                x20_1 = __swiftEmptyArrayStorage;
1000053a8                _swift_retain(__swiftEmptyArrayStorage);
1000053c0            label_1000053c0:
1000053c4                SecStaticCodeRef obj_1 =
1000053c4                    _$s29com_waves_central_I...DSo03SeciJ3RefaSgs5Int32V_tKFZTf4nd_n(
1000053c4                    (uint32_t)pid, x19);
1000053c4
1000053cc                if (!x19)
```

user — -zsh — 85×23

```
user@Users-Virtual-Machine ~ % python3 meme.py
```



```
sh-3.2# log stream
```

Software	No Client Validation	Permission Issues	DYLIB Injection	PID Reuse Attacks	No Anchor Apple
Vienna Assistant	CVE-2026-24068				
Arturia Software Center	CVE-2026-24062	CVE-2026-24063			
Native Access			CVE-2026-24070	CVE-2026-24071	
Waves Central			CVE-2026-24064	CVE-2026-24065	
Slate Connect				CVE-2026-24067	CVE-2026-24066

user — sudo ▸ bash — 85×23

sh-3.2# log stream

Secure XPC is not easy

- Due to limited documentation
- Developers are responsible for securing it

In my opinion Apple should take care of it

- Especially for Privileged Helper scenarios
  - Normally you only want one binary to connect to it
  - Why not enforce this by default?
  - Developers could opt out if needed

A word about disclosure:

If you develop software and release it, please create a security contact email or something. Also please ensure these are monitored.



user — sudo ▸ bash — 85×23

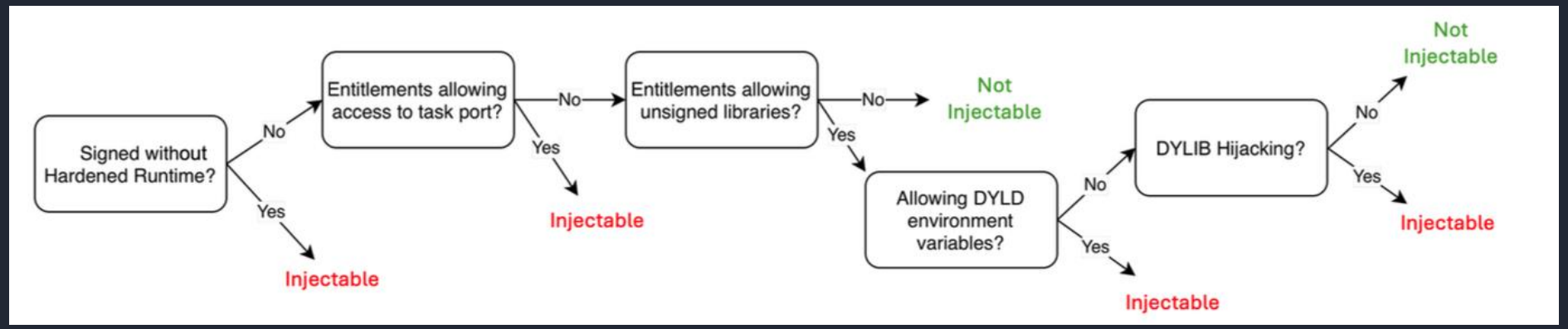
```
sh-3.2# log stream
```

How would I do it?

Sign the client binary with a custom entitlement. Something like:

com.test.version.123 = true

Ensure that the client binary cannot be injected into.



On the server side validate the client using:

xpc_connection_set_peer_codesigning_requirement(connection, requirement_string)

As requirement_string=

"anchor apple generic and entitlement [\“com.test.version.123\"] and certificate leaf[subject.OU] = \“<TEAM_ID>\"“