INFINUM

neverstop

Audio Unit debugging and reverse engineering

VLAHO POLUTA
FLUTTER TEAM LEAD, INFINUM
vlaho.poluta@infinum.com

JOSIP CAVAR
IOS ENGINEER, INFINUM
josip.cavar@infinum.com

01 Problem

CoreAudio comes with a set of built-in Audio Units. Unfortunately, they don't always work as expected, which raises the question:

How can we debug and understand closed-source AUs?

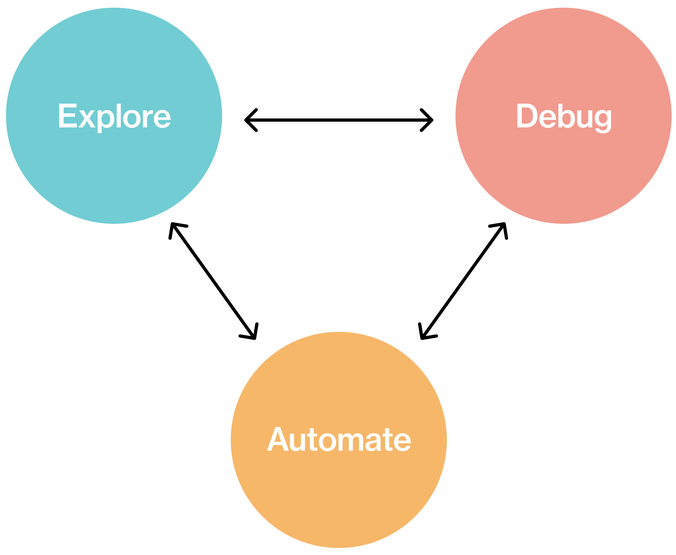
02 Introduction

Built-in CoreAudio AUs are reusable pieces of code that we can import and use. When they misbehave, it usually requires certain level of understanding of internal workings of audio unit in order to understand and workaround problem. Therefore, a methodology for understanding and debugging them becomes essential.

Traditional method of filing bug report and waiting for the fix can last long time, if it ever happens. It also requires new OS release, leaving us with the same problems and limitations on older operating systems.

In our work, we've developed a simple methodology which provided us with a toolbox to understand and workaround a number of problems we've faced.

03 Methodology



Our methodology comprises three interconnected methods suitable for different use cases:

1. Explore
When working with closed-source AUs, we often face the challenge of not knowing where to begin. Hopper Disassembler proves invaluable in helping us identify the code areas of interest. Hopper's fuzzy symbol search enables us to find internal methods of AUs in a user friendly way.

2. Debug
Once we've identified the code area of interest, we can employ symbolic breakpoints and step through the code methodically to comprehend problematic behaviour.

3. Automate
Debugging line by line can be a cumbersome process. We leverage Facebook's Fishhook to rebind MachO symbols, allowing us to access valuable data, including private logs. These logs can provide critical insights for issue resolution.

All three methods are interconnected, and the choice of where to begin depends on the specific use case.

04 Results

Through the consistent application of this approach, we successfully fine-tuned AUSampler to align with our specific requirements. Our efforts yielded a comprehensive list of private functionalities that we could utilise.

This in-depth exploration enabled us to identify, document and workaround numerous issues, which we subsequently reported to Apple.

Examples of our findings include:

- Discovered a number of AUSampler private properties
- This enabled us to find workarounds for public functionality having issues
- Identify distinct instrument placement process for macOS sandboxed applications despite the lack of documentation
- Highlighting voice count limitation issue, which Apple addressed in iOS 16 following our detailed report
- Identify process of AUSampler zone configuration when filename and AUPreset are in conflict

```
let sampler = AVAudioUnitSampler()
try! sampler.loadPreset(at: preset)
```

```
SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '~/Library/Containers/com.app.app/Data/Library/
Audio'

SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '/Library/Audio'

SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '/Library/Application Support/GarageBand/
Instrument Library/Sampler'

SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '/Library/Application Support/Logic/'

SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '/Library/Application Support/GarageBand/
Instrument Library/Sampler'

SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '/Library/Application Support/Logic/'

SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '~/Library/Containers/com.app.app/Data/
Downloads/Audio'

SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '/Library/Application Support/GarageBand/
Instrument Library/Sampler'

SampleManager.cpp:201 SampleManager::SearchForResource:
looking in '/Library/Application Support/Logic/'

SampleManager.cpp:441 about to throw -43: Failed to locate
sample
```

Figure 1.
Excerpt from private AUSampler logs failing to load preset due to looking in location that is not documented

Description	Property ID	AUPreset Path
Set Voice Count	4131	Instrument.voice count
Get Voice Count	4104	Instrument.coarse tune
Transpose	4123	Instrument.voice count
Panic	4111	N/A

Table 1.
A subset of AUSampler private properties that can be used with *AudioUnitSetProperty* and *AudioUnitGetProperty*

```
int __ZN7Sampler16InitializeMemoryEv() {
    VoiceZone::InitializeAllocationPool(0x80);
    rax = AudioStreamer::Initialize();
    return rax;
}
```

Code 1.
Hopper disassembly of *Sampler::InitializeMemory* on iOS 15

```
int __ZN7Sampler16InitializeMemoryEv() }
    VoiceZone::InitializeAllocationPool(0x400);
    if (((r30 ^ r30 * 0x2) & 0x40000000) != 0x0) {
            asm { brk          #0xc471 };
            r0 = loc_1ed9c8d44();
    }
    else {
            r0 = AudioStreamer::Initialize();
    }
    return r0;
}
```

Code 2.
Hopper disassembly of *Sampler::InitializeMemory* on iOS 16

05 Conclusions

Understanding and debugging closed source code is a lot of work and effort. But it is sometimes the most reasonable option to move forward. This simple toolbox allowed us to do that with some structure, each method providing different level of control.

In this work, we presented a simple methodology that can help us debug, understand and sometimes workaround problems that we encounter as audio developers working within existing ecosystems.

Our work is focused on working within Apple's CoreAudio framework covering Audio Units and AVAudioEngine, but the methodology is more widely applicable on a range of problems.

06 References