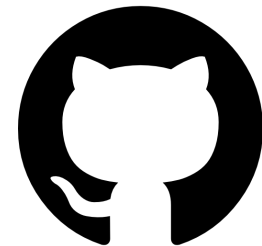


LLDB Kernel Module Improvement

Sheng-Yi Hong

Who am I?

- Student in NTNU CS
- Interest in Kernel and Toolchain
- Bhyve Raw TCP console
- LLDB Kernel Module



aokblast

Outline

- Introduction
- What we already have?
- What I have done?
- Demo
- Ongoing work
- Conclusion

Introduction



LLVM Review

Old	New
gcc	clang
Libstdc++	Libc++
libgcc	compiler-rt
libgcc_s	libunwind
GNU ld	lld
gdb	lldb

LLVM Review

Unchanged	Not shipped
libc	llvm-lto
elf-toolchain	bolt
	libclc

LLDB Architecture

1. Target - Builtin
2. ObjectFile
3. Process
4. ABI
5. DynamicLoader

What we already have?

1. Target - Builtin
2. ObjectFile - ✓
3. Process - ✓
4. ABI - ✓
5. DynamicLoader

Dynamic Loader Plugin

Kernel Module is like shared library

- Shared same address space
- Load when needed
- Load by dynamic loader

Goals of this Plugin:

- Parse all loaded Kernel Module
- Make symbols load address
- Run in tier-1 platform

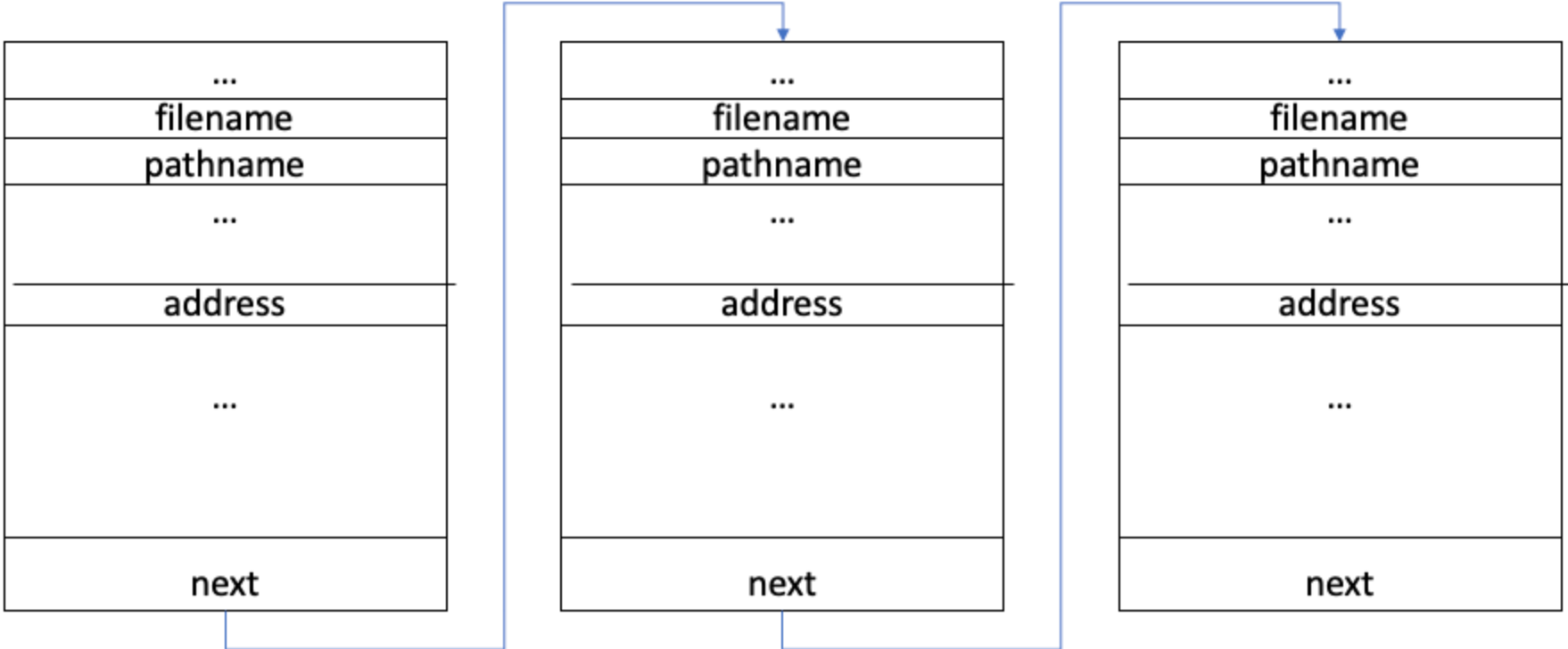
Design

1. Called by ProcessFreeBSDKernel
2. Find and Verify coredump information
3. Parsing Kernel Loaded Module Address
4. Handle Relocatable file and Shared Object

Find and Verify Coredump

- Currently, according to the load address in kernel binary
- Problem
 - When we have kASLR in the future
- Some other method available:
 - Search near PC

Parsing Kernel Module



Structure of linker_files

Parsing Kernel Module

```
// DynamicLoaderFreeBSDKernel.cpp

while (current_kld != 0) {
    // Read kld_filename, load_addr, pathname
    kmods_list.emplace_back();
    KModImageInfo &kmod_info = kmods_list.back();
    kmod_info.SetName(kld_filename);
    kmod_info.SetLoadAddress(kld_load_addr);
    kmod_info.SetPath(kld_pathname);

    current_kld =
        m_process->ReadPointerFromMemory(current_kld
+ kld_off_next, error);
    if (kmod_info.GetName() == "kernel")
        kmods_list.pop_back();
    if (error.Fail())
        return false;
}
```

Append Kernel Module into LLDB Module list

- Attach the symbol file
- For Dynamic Library
 - Verify ELF file and adjust section addr
- For relocatable kernel module
 - SetLoadAddress
- Put it into loaded module list

Single line cost me 2 weeks.

- Without this, the plugin will not work
 - Don't find any ref on network
 - No compile err or link err
 - In a path I seldom think about

```
lib/clang/include/Plugins/Plugins.def
```

↑	@@ -41,6 +41,7 @@	LLDB_PLUGIN(ArchitectureAArch64)
41	41	LLDB_PLUGIN(DisassemblerLLVMC)
42	42	LLDB_PLUGIN(DynamicLoaderPosixDYLD)
43	43	LLDB_PLUGIN(DynamicLoaderStatic)
	44	+ LLDB_PLUGIN(DynamicLoaderFreeBSDKernel)
44	45	LLDB_PLUGIN(InstructionARM)
45	46	LLDB_PLUGIN(InstructionARM64)
46	47	LLDB_PLUGIN(InstructionMIPS)

```
↓
```

Modification beyond the Dyld Plugin

- LoadAddress of Kernel Module

```
// ObjectFileELF.cpp

+ if (GetType() == ObjectFile::eTypeObjectFile)
+ {
+   for (I : m_section_headers) {
+     const ELFSectionHeaderInfo &header = *I;
+     if (header.sh_flags & SHF_ALLOC)
+       return Address(GetSectionList()-
+ >FindSectionByID(SectionIndex(I)), 0);
+   }
+   return LLDB_INVALID_ADDRESS;
+ }
```


Modification beyond the Dyld Plugin

- DebugInfo for Relocatable file

```
// ObjectFileELF.cpp

if ((ObjectType == eTypeObjectFile ... ) {

    NextVMAddress =
        llvm::alignTo(NextVMAddress,
            std::max<addr_t>(H.sh_addralign, 1));
    Address = NextVMAddress;
    NextVMAddress += Size;
}
```

Modification beyond the Dyld Plugin

- DebugInfo for Relocatable file

```
// ObjectFileELF.cpp

if ((ObjectType == eTypeObjectFile) ||
    (ObjectType == eTypeDebugInfo && H.sh_addr == 0)) {

    NextVMAddress =
        llvm::alignTo(NextVMAddress,
            std::max<addr_t>(H.sh_addralign, 1));
    Address = NextVMAddress;
    NextVMAddress += Size;
}
```

Modification beyond the Dyld Plugin

- Check if the binary is kernel
 - No way way to detect it directly in ELF
 - special “.interp” in FreeBSD (/red/herring”)

```
    case llvm::ELF::ET_EXEC:  
    -    // 2 - Executable file  
    -    // TODO: is there any way to detect that  
    an executable is a kernel  
    -    // related executable by inspecting the  
    program headers, section headers,  
    -    // symbols, or any other flag bits???
```

You can find the code in llvm-project

The screenshot shows a GitHub pull request interface. At the top, the title is "[lldb][FreeBSD] Add dynamic loader handle class for FreeBSD Kernel #67106". To the right of the title are "Edit" and "<> Code" buttons. Below the title, a purple "Merged" badge is followed by the text "emaste merged 1 commit into llvm:main from aokblast:lldb_dynamic_loader_fbsdkernel on Oct 4, 2023".

Below this, there are statistics: "Conversation 51", "Commits 1", "Checks 2", and "Files changed 6". To the right of these is a green progress bar showing "+1,014 -7".

A comment from user "aokblast" is visible, dated "Sep 22, 2023". The comment text reads: "This commit is moved from [llvm-phabricator](#). The implementation support parsing kernel module for FreeBSD Kernel and has been test on x86-64 and arm64. In summary, this class parse the linked list resides in the kernel memory that record all kernel module and load the debug symbol file to facilitate debug process". Below the comment are reaction icons (smiley face and fire) and a count of "2".

At the bottom left, a bot comment from "llvmbot" states "added the [lldb](#) label on Sep 22, 2023".

On the right side, the "Reviewers" section lists "bulbazord", "augusto2112", "emaste", and "clayborg". "emaste" and "clayborg" have green checkmarks next to their names, indicating they have approved the pull request. Below this is the "Assignees" section, which says "No one assigned".

Quick Demo

<https://reurl.cc/OGOLk9>

Ongoing work

- Programmer may load incompatible kernel coredump with kernel binary
- Use `.note.gnu.build-id` section
- ID in binary should be same as kernel coredump, if not, refused to load the `DynamicLoader`
- Extended to the kernel module

Thank you