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
Outline

- Introduction
- What we already have?
- What I have done?
- Demo
- Ongoing work
- Conclusion
Introduction
<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>clang</td>
</tr>
<tr>
<td>Libstdc++</td>
<td>Libc++</td>
</tr>
<tr>
<td>libgcc</td>
<td>compiler-rt</td>
</tr>
<tr>
<td>libgcc_s</td>
<td>libunwind</td>
</tr>
<tr>
<td>GNU ld</td>
<td>lld</td>
</tr>
<tr>
<td>gdb</td>
<td>lldb</td>
</tr>
</tbody>
</table>
## LLVM Review

<table>
<thead>
<tr>
<th>Unchanged</th>
<th>Not shipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>libc</td>
<td>llvm-lto</td>
</tr>
<tr>
<td>elf-toolchain</td>
<td>bolt</td>
</tr>
<tr>
<td></td>
<td>libclc</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LLDB Architecture

1. Target - Built in
2. Object File
3. Process
4. ABI
5. DynamicLoader
What we already have?

1. Target - Built-in
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
// 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
Modification beyond the Dyld Plugin

• LoadAddress of Kernel Module

```cpp
// 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

```cpp
// 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

```cpp
// 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”)

```cpp
  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
Quick Demo

https://reurl.cc/OGOLk9
Ongoing work

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