

Re: Dynamic Disassembler (determine main() location at runtime)

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2007-02/msg00101.htm>

- *From:* "Skywing [MVP]" <skywing_NO_SPAM@xxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 9 Feb 2007 14:31:51 -0500
-

1. You won't need to bruteforce this. Once you have the PE header, it is simple to determine the section header VA's by parsing the image header itself. As far as finding the PE header goes, GetModuleHandle is the proper way (HMODULE is a pointer to the PE header). There are other mechanisms (PSAPI, Toolhelp) for examining the loaded module list in a remote process.
2. .text is a default, but you can override this for any parts of code (in CL/LINK) with #pragma code_seg(). The memory manager sets pages as executable based on the section header for a region, which describes the protection flags (e.g. executable, read/write) for that region. Note that when using #pragma code_seg(), it is perfectly legal to have more than one section with executable code.
3. You can't really safely make this assumption. There is no reason that the data in the PE header should be wrong about the entrypoint address, though; you'll be using the same method to find the entrypoint as the OS itself does.

—

Ken Johnson (Skywing)
Windows SDK MVP
<http://www.nynaeve.net>

"Jeffrey Walton" <noloader@xxxxxxxxxx> wrote in message
news:1171047377.341297.17600@xx

1. I would enumerate the sections in the image (PE header) and look for the one with the lowest address that contains the IMAGE_SCN_MEM_EXECUTE characteristic.

It appears this is the method which will work somewhat reliably (however, one cannot use MEM_IMAGE).

I hate the thought of "brute forcing" a search staing at SYSTEM_INFORMATION minimum and Maximum address...

2. This information is available from processing the section header corresponding to '.text'. Please note that there is no requirement that the code section be named '.text', and there is furthermore no requirement that there must only be one section with code. Also, there could be 'dead space' after the section for alignment reasons.

Re: Dynamic Disassembler (determine main() location at runtime)

More bridges I have not crossed...

Please note that there is no requirement that the code section be named ``.text'`,

In this case, how does the loader/linker know the "executable" segment?

Also, there could be ``dead space'` after the section for alignment reasons.

I expect this. The current dead space seems to be 0x00 and 0xCC on disk. Windows 2000 allocates Virtual Memory on 64KB boundaries for the application (even though my current Intel x86 uses 4 KB pages). So, I may see more dead space with a new BYTE fill character.

3. For the entrypoint address invoked by the loader, look at ``AddressOfEntryPoint'` RVA field in the `IMAGE_OPTIONAL_HEADER`.

Interestingly, this seems to be placed at the 'End of the Executable' in Memory. Startup Code, when, a `CALL` to an earlier area of executable (main, wmain, etc).

BTW, I don't see why you would care if there is a jump stub for main or not; it is executable and will produce the same result regardless. The OS loader certainly doesn't care.

A flag in the sand – I helps with the discovery/learning of what is really going on. If I determine that in memory `main()` is located at `0x00401E20`, and I'm working around `0x004182E6`, I can probably say I'm off the mark.

Jeff

On Feb 8, 12:09 pm, "Skywing [MVP]" <skywing_NO_SP...@xxxxxxxxxxxxxxxxxxxx> wrote:

1. I would enumerate the sections in the image (PE header) and look for the one with the lowest address that contains the `IMAGE_SCN_MEM_EXECUTE` characteristic.
2. This information is available from processing the section header corresponding to ``.text'`. Please note that there is no requirement that the code section be named ``.text'`, and there is furthermore no requirement that there must only be one section with code. Also, there could be ``dead space'` after the section for alignment reasons.
3. For the entrypoint address invoked by the loader, look at

Re: Dynamic Disassembler (determine main() location at runtime)

`AddressOfEntryPoint' RVA field in the IMAGE_OPTIONAL_HEADER.
For other functions, without them being exported, you would have to rely on either debug symbols or code flow analysis.

BTW, I don't see why you would care if there is a jump stub for main or not; it is executable and will produce the same result regardless. The OS loader certainly doesn't care.

--

Ken Johnson (Skywing)
Windows SDK MVP <http://www.nynaeve.net> "Jeffrey Walton"
<noloa...@xxxxxxxxxx> wrote in message

news:1170891286.106224.274650@xx

> Hi Ken,

>> ... but you didn't really describe what problem you were having
>> other than listing some observations about how there might be a jump >>
>> stub
>> for main (perhaps due to incremental linking)...
> Basically, I desire three pieces of information. The 'in memory' is
> the caveat:

- > * First code page in memory
- > * Size of .text section in memory
- > * Location of function main()

> One – First code page in memory:
> Open Question

> Two – Size of .text section in memory
> I think_ this is the same as on disk (considering the padding I am
> observing based on page size boundaries).

> Three – Location of function
> I think_ I can find an arbitrary function (my example uses main())
> whether in Debug or Release.

> Any comments would be greatly appreciated.

> Jeff

```
> int _tmain(int argc, _TCHAR* argv[])  
> {  
> const UINT PATH_SIZE = 2 * MAX_PATH;  
> TCHAR szFilename[ PATH_SIZE + 1 ] = { 0 };  
> __try {  
> //////////////////////////////////////  
> //////////////////////////////////////
```

Re: Dynamic Disassembler (determine main() location at runtime)

```
> if( 0 == GetModuleFileName( NULL, szFilename, PATH_SIZE ) )
> {
> std::cout << _T("Error Retrieving Process Filename") <<
> std::endl;
> __leave;
> }
> std::cout << _T("File: ") << szFilename << std::endl <<
> std::endl;
> //////////////////////////////////////
> //////////////////////////////////////
> PVOID pfnMain = (PVOID) &_tmain;
> std::cout << _T("Original main(): ") << pfnMain << std::endl;
> //////////////////////////////////////
> //////////////////////////////////////
> {
> PBYTE pPossibleJump = static_cast<PBYTE>(pfnMain);
> BYTE opcode = *pPossibleJump;
> if( 0xE9 /* Jump */ != opcode )
> {
> std::cout << _T("main() is not a jump opcode... No
> fixup applied");
> std::cout << std::endl << std::endl;
> }
> else
> {
> DWORD dwJump =
> *( reinterpret_cast<PDWORD>(pPossibleJump+1) );
> pfnMain = pPossibleJump + dwJump + sizeof(opcode) +
> sizeof(dwJump);

> std::cout << _T("main() is a jump opcode... fixup
> applied") << std::endl;
> std::cout << _T("Calculated main() at ") << pfnMain;
> std::cout << std::endl << std::endl;
> }
> }
> ...
> }
> __except( EXCEPTION_EXECUTE_HANDLER ) {
> std::tcout << _T("Caught Exception") << std::endl;
> }
> }

> On Feb 7, 4:38 pm, "Skywing [MVP]"
> <skywing_NO_SP...@xxxxxxxxxxxxxxxxxxxx> wrote:
>> What is the exact problem that you are trying to solve here? Your >> post
>> makes sense, but you didn't really describe what problem you were >>
having
>> other than listing some observations about how there might be a jump >>
stub
>> for main (perhaps due to incremental linking)...
```

Re: Dynamic Disassembler (determine main() location at runtime)

>> --

>> Ken Johnson (Skywing)

>> Windows SDK MVP

>><http://www.nynaeve.net>

> "Jeffrey Walton" <noloa...@xxxxxxxx> wrote in message:

>>news:1170882501.458467.173940@xx

>>> Hi All,

>>> I apologize for the cross post (I hope three is not considered too
>>> bad). I wanted to enlist help from the kernel folks, and the power
>>> debuggers...

>>> SNIP