Exporting IDA Debug Information

Overview

- Who am I?
- What's the problem?
- What does this tool do?
- How does it work?
- Demo

about:me

Dynetics
The Power of Solutions®

File Edit Jump Search View Debugger Options Windows Help

No debugger

Library function | Data | Regular function | Unexplored | Instruction | External symbol

Functions window

Function name

sub_404130
sub_404140
sub_404160
sub_404180
j__free
sub_404260
sub_404500
sub_404550
sub_4045D0
sub_404820
sub_404890
sub_404A10
sub_404A20
sub_404A30
sub_404A60
sub_404A70
sub_404A80
sub_404A90
sub_404AA0
sub_404C00
sub_404CE0
sub_404DB0
sub_404E70
sub_404EB0
sub_404ED0
sub_404F40
sub_404FF0

Line 132 of 397

Graph overview

IDA View-A | Pseudocode-A | Hex View-1 | Structures

```
64    if ( 46 != v13 )
65      goto LABEL_20;
66    goto LABEL_8;
67  }
68  if ( !*(_BYTE *)(v10 + 1) )
69    return 1;
70  if ( 46 != v13 )
71  {
72LABEL_11:
73    if ( *(_BYTE *)(v10 + 1) == 46 && !*(_BYTE *)(v10 + 2) )
74      return 1;
75    goto LABEL_20;
76  }
77LABEL_8:
78  if ( *(_BYTE *)(v9 + 1) == 46 && !*(_BYTE *)(v9 + 2) )
79    return (unsigned int)-1;
80  if ( 46 == v14 )
81    goto LABEL_11;
82LABEL_20:
83  if ( v13 == 46 )
84  {
85    v11 = (const char *)(v9 + 1);
86    ++v10;
87    if ( v14 == 46 )
88      goto LABEL_22;
89    return (unsigned int)-1;
90  }
91  if ( v14 == 46 )
92    return 1;
93LABEL_22:
94  v36 = v11;
95  v37 = v10;
96  v16 = sub_40AD10((char **)&v36);
97  v17 = v16;
98  v18 = (__int64)v16;
99  v19 = sub_40AD10((char **)&v37);
100 v20 = v17 == 0LL;
101 v21 = v17;
102 v22 = v37;
103 v23 = (__int64)v19;
104 if ( v20 )
105   v18 = (__int64)v36;
106 v33 = (__int64)v36;
107 v24 = v18 - (_QWORD)v11;
```

0000AF35 sub_404A90:79

Output window

Function argument information has been propagated
The initial autoanalysis has been finished.

Why export information from IDA?

- An embedded device may have no way to connect IDA remotely
  - Manually referencing IDA is tedious
- Some platforms may have software debuggers that would be useful with debug info
- Some tools allow interesting dynamic analysis techniques not available with IDA
  - Ex: Reverse debugging

Use-case: QNX

- Provides a version of GDB for their platform on lots of architectures
  - Downside: it doesn't use the standard protocol
- Lots of connected components of mixed architecture
- Maybe no IP connections

With this plugin: export the debug info from IDA and import into gdb on the target.

Debug Info Formats

- *STABS*
  - Designed in the 1980s
  - Puts all info in symbol table
  - Not well standardized
- *DWARF*
  - Designed along with ELF
  - Used by most modern compilers
  - Binary format
- *Windows CodeView/Program Database*
  - Mostly undocumented, windows-only
- Many Others
  - COFF, OMF, IEEE-695

dwarfexport

dwarfexport is a plugin for IDA Pro that creates DWARF debug info using function names/variables locations/structures extracted from IDA.

It lets you create binaries as though you had built with debugging enabled.

Implementation

What do we need from IDA?

- Decompiled source
- 'step' points
- Global/local variable locations
- Type information

Decompilation

Intermediate Representation

- FunctionDecl *main*
  - FunctionCall *printf*
    - StringLiteral *%d*
    - NumericLiteral *10*

```
push    ecx
sub     esp, 14h
mov     [ebp+var_C], 0Ah
sub     esp, 8
push    [ebp+var_C]
push    offset format    ; "%d"
call    _printf
add     esp, 10h
```

IDA View-A | Pseudocode-A | Hex View-1

```
1 int __cdecl main(int argc, const char **argv, cons
2 {
3   printf("%d", 10);
4   return 0;
5 }
```

IDA AST

IDA View-A | Pseudocode-A | Hex View-1

```
1 int __cdecl main(int argc, const char **argv, cons
2 {
3   printf("%d", 10);
4   return 0;
5 }
```

```
printer.apply_to(cfunc.body, None)
0x804842e block
0x804842e   call
0x804842e     obj: _printf
0x8048429     obj: %d
0x804841c     num: 10
0x8048436   return
0x8048436     num: 0
```

IDA AST

```
1  void *__fastcall sub_404820(__int64
2  {
3    char v3; // r13@1
4    _QWORD *v4; // rbx@1
5    __int64 v5; // rax@1
6    __int64 v6; // rax@3
7    void *result; // rax@5
8
9    v3 = a3;
10   v4 = sub_411DC0(0x20uLL);
11   v5 = 0LL;
12   if ( a2 )
13     v5 = sub_411FA0(a2);
14   v4[1] = v5;
15   v6 = 0LL;
16   if ( a1 )
17     v6 = sub_411FA0(a1);
18   *v4 = v6;
19   result = ptr;
20   *((_BYTE *)v4 + 16) = v3;
21   ptr = v4;
22   v4[3] = result;
23   return result;
24 }
```

```
printer.apply_to(cfunc.body, None)
0x404831 block
0x404831   asg
0x404831     var: v3
0x404831     var: a3
0x40483d   asg
0x40483d     var: v4
0x404838     cast: __int64
0x404838       call
0x404838         obj: sub_411DC0
0x40482c         num: 32
0x404840   asg
0x404840     var: v5
0x404840     num: 0
0x404845   if
0x40484a     block
0x40484a       asg
0x40484a         var: v5
0x40484a         call
0x40484a           obj: sub_411FA0
0x404847           var: a2
0x404845     var: a2
0x40484f   asg
0x40484f     idx
0x40484f       var: v4
0x40484f       num: 1
0x40484f     var: v5
0x404853   asg
0x404853     var: v6
0x404853     num: 0
0x404858   if
0x40485d     block
0x40485d       asg
0x40485d         var: v6
0x40485d         call
0x40485d           obj: sub_411FA0
```

Step Points

```
1  void *__fastcall sub_404820(__int64
2  {
3    char v3; // r13@1
4    _QWORD *v4; // rbx@1
5    __int64 v5; // rax@1
6    __int64 v6; // rax@3
7    void *result; // rax@5
8
9    v3 = a3;
10   v4 = sub_411DC0(0x20uLL);
11   v5 = 0LL;
12   if ( a2 )
13     v5 = sub_411FA0(a2);
14   v4[1] = v5;
15   v6 = 0LL;
16   if ( a1 )
17     v6 = sub_411FA0(a1);
18   *v4 = v6;
19   result = ptr;
20   *((_BYTE *)v4 + 16) = v3;
21   ptr = v4;
22   v4[3] = result;
23   return result;
24 }
```

```
printer.apply_to(cfunc.body, None)
0x404831 block
0x404831   asg
0x404831     var: v3
0x404831     var: a3
0x40483d   asg
0x40483d     var: v4
0x404838     cast: __int64
0x404838       call
0x404838         obj: sub_411DC0
0x40482c         num: 32
0x404840   asg
0x404840     var: v5
0x404840     num: 0
0x404845   if
0x40484a     block
0x40484a       asg
0x40484a         var: v5
0x40484a         call
0x40484a           obj: sub_411FA0
0x404847           var: a2
0x404845     var: a2
0x40484f   asg
0x40484f     idx
0x40484f       var: v4
0x40484f       num: 1
0x40484f     var: v5
0x404853   asg
0x404853     var: v6
0x404853     num: 0
0x404858   if
0x40485d     block
0x40485d       asg
0x40485d         var: v6
0x40485d         call
0x40485d           obj: sub_411FA0
```

Local Variables

- Stack Variables:
  - Location is expressed as an offset from frame base address
  - *Note: There is no (complete) SDK interface for this*
- Register Variables:
  - Translate the IDA register number to dwarf number

Type Information

As the binary is traversed, maintain a mapping of `tinfo_t` to DWARF `die`:

- Extract each struct member name and type, as well as the offset from the struct start
- Handle array/pointer types

```
< 1><0x0000021d>    DW_TAG_structure_type
                      DW_AT_name                  lua_State_0
                      DW_AT_byte_size             0x000000d0
                      DW_AT_sibling               <0x000003b3>
< 2><0x0000022f>      DW_TAG_member
                        DW_AT_type                  <0x000003b3>
                        DW_AT_name                  next
                        DW_AT_data_member_location  DW_OP_plus_uconst 0
< 2><0x0000023c>      DW_TAG_member
                        DW_AT_type                  <0x000003f2>
                        DW_AT_name                  tt
                        DW_AT_data_member_location  DW_OP_plus_uconst 8
< 2><0x00000247>      DW_TAG_member
                        DW_AT_type                  <0x000003f2>
                        DW_AT_name                  marked
                        DW_AT_data_member_location  DW_OP_plus_uconst 9
< 2><0x00000256>      DW_TAG_member
                        DW_AT_type                  <0x000003fc>
                        DW_AT_name                  nci
                        DW_AT_data_member_location  DW_OP_plus_uconst 10
< 2><0x00000262>      DW_TAG_member
```

Demo

Other Uses

- Add debug info for shared libraries and create a fully debugged environment
- Reverse-debugging
  - Tested using 'rr' on linux
- Hardware Debugging
  - Software frontends for hardware debuggers must use some debug format
  - Green Hill 'MULTI' IDE can import DWARF info

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access

Debug

- lua [C/C++ Application]
  - lua [17241] [cores: 2]
    - Thread #1 [lua] 17241 [core: 2] (Suspended : Breakpoint)
      - main() at a.out.c:20,470 0x42907e
      - __libc_start_main() at 0x7ffff774343a
      - _start() at a.out.c:1 0x401a9a
  - gdb (7.12.1)

Variables | Breakpoints | Registers | Modules

Name	Type	Value
v3	int	65
status	int	32
L	struct lua_State_0 *	0x
next	struct GCObject_0 *	
tt	lu_byte	
marked	lu_byte	
nci	unsigned int __int16	
status	lu_byte	
top	struct TValue *	
value_	Value_0	
tt_	int	
l_G	struct global_State_0 *	
ci	struct CallInfo_0 *	
oldpc	const Instruction *	

lua.cpp | a.out.c

```
20461
20462
20463 int __cdecl main(int argc, const char **argv, const char **envp)
20464 {
20465   int v3; // eax@2
20466   int result; // ST10_4@3
20467   int status; // [rsp+14h] [rbp-Ch]@3
20468   lua_State_0 *L; // [rsp+18h] [rbp-8h]@1
20469
20470   L = luaL_newstate();
20471   if ( L )
20472   {
20473     lua_pushcclosure(L, (lua_CFunction)pmain, 0);
20474     lua_pushinteger(L, argc);
20475     lua_pushlightuserdata(L, argv);
20476     status = lua_pcallk(L, 2, 1, 0, 0LL, 0LL);
20477     result = lua_toboolean(L, -1);
```

Eclipse

lua - [~/ClionProjects/lua] - .../a.out.c - CLion 2016.1.3

File Edit View Navigate Code Refactor Run Tools VCS Window Help

lua 〉 a.out.c 〉

Project

- lua ~/ClionProjects/lua
  - a.out.c
  - a.out.dbg
  - CMakeLists.txt
  - main.cpp
- External Libraries

CMakeLists.txt × | a.out.c ×

File length (544691) exceeds configured limit for C/C++ (500000). Code insight features are not available.

main — Match Case — Regex — Words — 29 mat

```
}

int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // eax@2
  int result; // ST10_4@3
  int status; // [rsp+14h] [rbp-Ch]@3
  lua_State_0 *L; // [rsp+18h] [rbp-8h]@1

  L = luaL_newstate();
  if ( L )
  {
    lua_pushcclosure(L, (lua_CFunction)pmain, 0);
    lua_pushinteger(L, argc);
    lua_pushlightuserdata(L, argv);
    status = lua_pcallk(L, 2, 1, 0, 0LL, 0LL);
```

Debug: Build All | lua | lua

Debugger | Console

Frames

Thread-1

- main a.out.c:20471
- __libc_start_main
- _start a.out.c:1

Variables | GDB

- v3 = {int} 6557720
- status = {int} 32767
- L = {struct lua_State_0 * | 0x641018} 0x641018
  - [0] = {struct lua_State_0}
    - next = {struct GCObject_0 * | 0x0} nil
    - tt = {lu_byte} 8 '\b'
    - marked = {lu_byte} 1 '\001'
    - nci = {unsigned int __int16} 0
    - status = {lu_byte} 0 '\000'
    - top = {struct TValue * | 0x641640} 0x641640
    - l_G = {struct global_State_0 * | 0x6410e8} 0x6410e8
    - ci = {struct CallInfo_0 * | 0x641078} 0x641078

CLion

File Edit Selection View Go Debug Help

DEBUG (gdb) Lau

VARIABLES

Locals

v3: 6557720

status: 32767

L: 0x641018

next: 0x0

tt: 8 '\b'

marked: 1 '\001'

nci: 0

status: 0 '\000'

top: 0x641640

value_: 0

tt_: 0

l_G: 0x6410e8

WATCH

CALL STACK PAUSED ON STEP

main() a.out.c 20471:1

libc.so.6!__libc_start_m...

_start() a.out.c 1:1

BREAKPOINTS

a.out.c 20470

main.c 4

main.c 5

main.c a.out

main 1 of 29

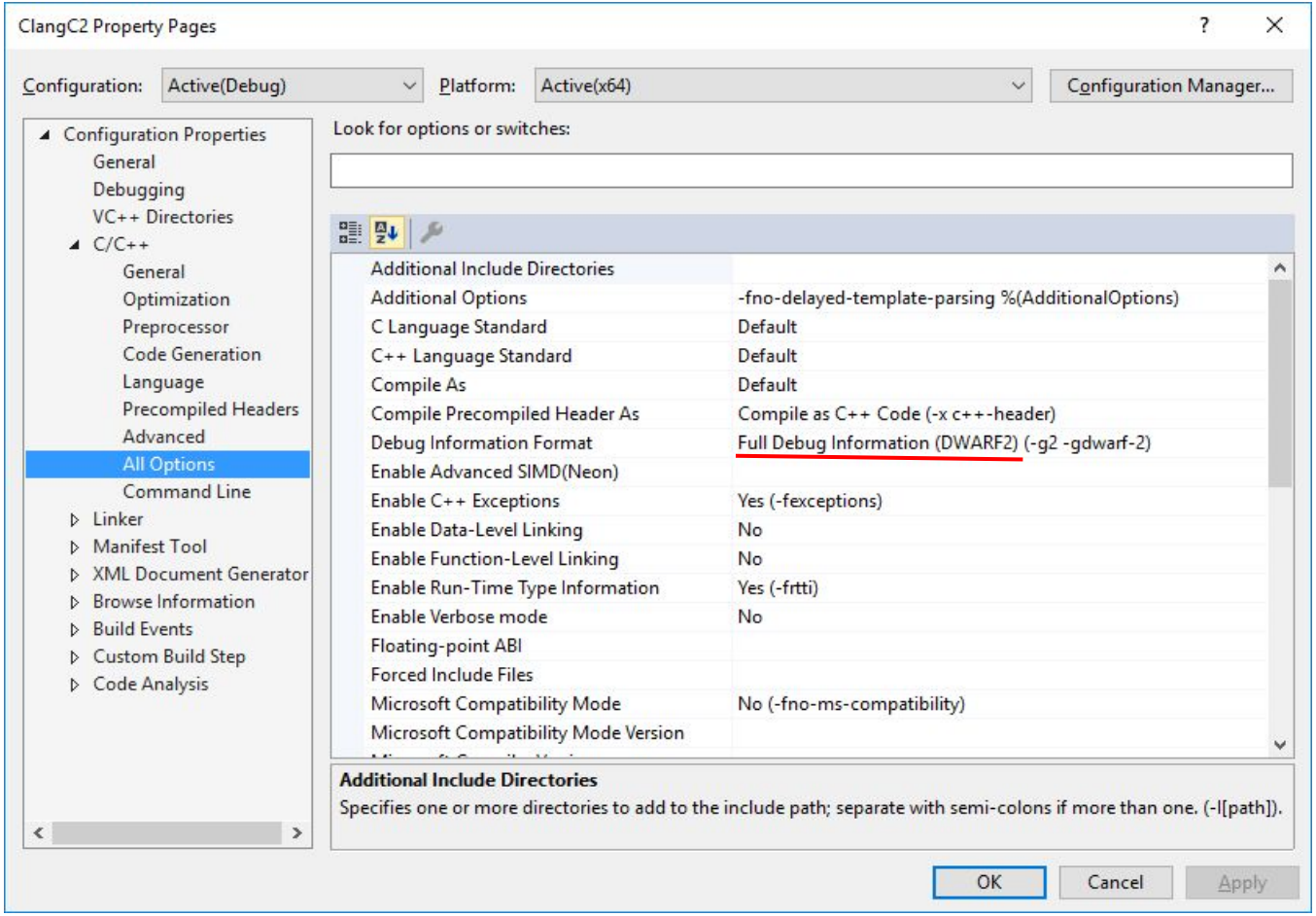
```
20455     {
20456       result = 0LL;
20457     }
20458   }
20459   return result;
20460 }
20461
20462
20463 int __cdecl main(int argc, const char **argv, const char **envp)
20464 {
20465   int v3; // eax@2
20466   int result; // ST10_4@3
20467   int status; // [rsp+14h] [rbp-Ch]@3
20468   lua_State_0 *L; // [rsp+18h] [rbp-8h]@1
20469
20470   L = luaL_newstate();
20471   if ( L )
20472   {
20473     lua_pushcclosure(L, (lua_CFunction)pmain, 0);
20474     lua_pushinteger(L, argc);
20475     lua_pushlightuserdata(L, argv);
20476     status = lua_pcallk(L, 2, 1, 0, 0LL, 0LL);
20477     result = lua_toboolean(L, -1);
20478     report(L, status);
20479     lua_close(L);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
=thread-group-added,id="i1"
GNU gdb (GDB) 7.12
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
```

0 0 main(int argc, const char ** argv, const char ** envp) Ln 20471, Col 1 Spaces: 2 UTF-8 LF C Linux

VS Code



Visual Studio(?)

Limitations

- DWARF debug info is not useful for windows utilities
- Limitations in IDA SDK may make some debug info inaccurate (for now)
- Register number translations must be added on a per-architecture basis
- Local variable values don't display correctly under GDB 8 (released June 4)

Questions?

github.com/alschwalm/dwarfexport

goo.gl/MlTkmV

Twitter/Github: @alschwalm

Email:
adamschwalm@gmail.com