What’s a Ghidra and why shoud you care?

Chris Eagle

Kernelcon 2019

whoami

- Full time reverse engineer
  - Long time Ida Pro user
- Part time faculty member
  - Naval Postgraduate School, Monterey, CA
- Author
  - The Ida Pro Book

What to expect

- This talk is a high level overview of Ghidra
- This talk is not a tutorial on Ghidra
- Assumes some knowledge of disasemblers and their uses

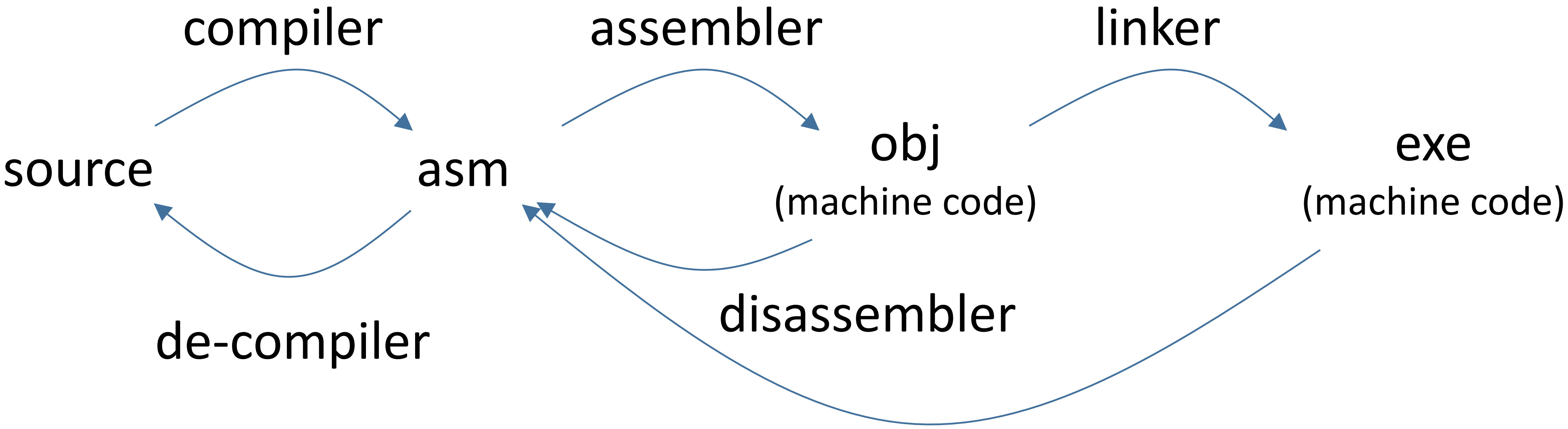
What’s Ghidra

- An internal NSA reverse engineering tool
  - Closest public equivalent is Ida Pro
- Released in binary form at RSA 2019 (3/5/19)
  - Version 9.0.0
  - Remote code execution “bug” found almost immediately
  - Version 9.0.1 followed shortly thereafter
- Source released on github on 4/4/19
  - Coincides with release of 9.0.2

Where to get it

- Main site: https://ghidra-sre.org/
    - Links to binary downloads
- github: https://github.com/NationalSecurityAgency/ghidra
    - Currently 160 open issues

Reverse Engineering Tool Chains



Some existing tools

- Ida Pro – commercial
- Binary Ninja – commercial
- Radare2 – open source
- Hopper – commercial
- Comparison chart
  - https://rada.re/r/cmp.html

How did we get here?

- At least seven years in the making
- Desire for community contributions
- Taxpayer dollars at work
- Provide a free tool for academic use

Ghidra highlights

- Large number of supported processor types
- Decompilers for supported architectures
- Undo
- Collaboration server
- Scriptable / extensable
- Written in Java – requires Java 11 or greater
  - Maybe not a highlight

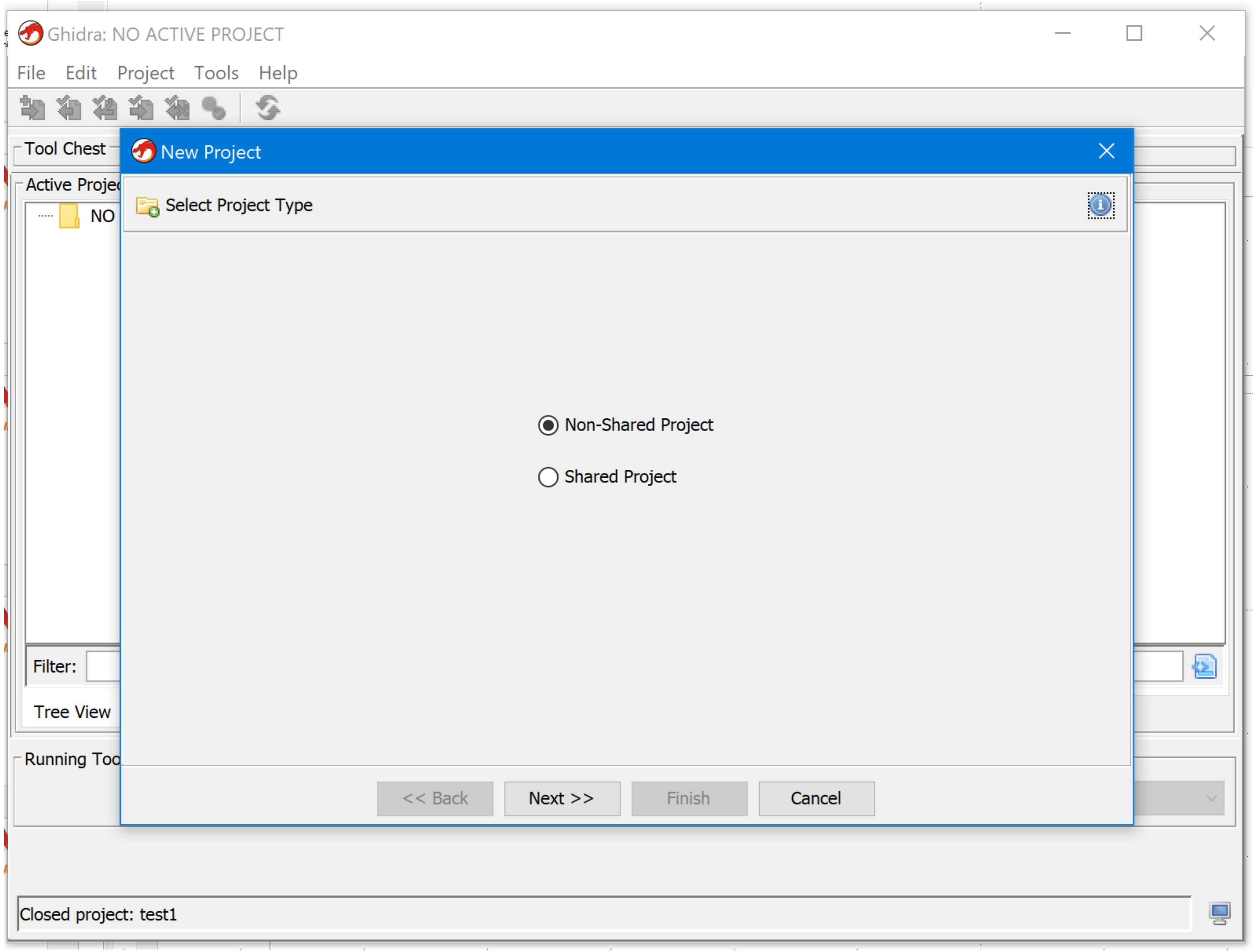
Quick walkthrough

- Basic workflow
- Highlight some features
- Note: A lot of people know a lot more about Ghidra than I do
  - Especially the scripting side of it

Basic workflow

- Create new project
  - Private
  - Shared – requires running server
    - Server included with Ghidra
- Add files to project
  - Drag and drop or open
- Perform automated analysis
- Utilize provided tools for additional analysis

Create project



New Project

Select Project Type

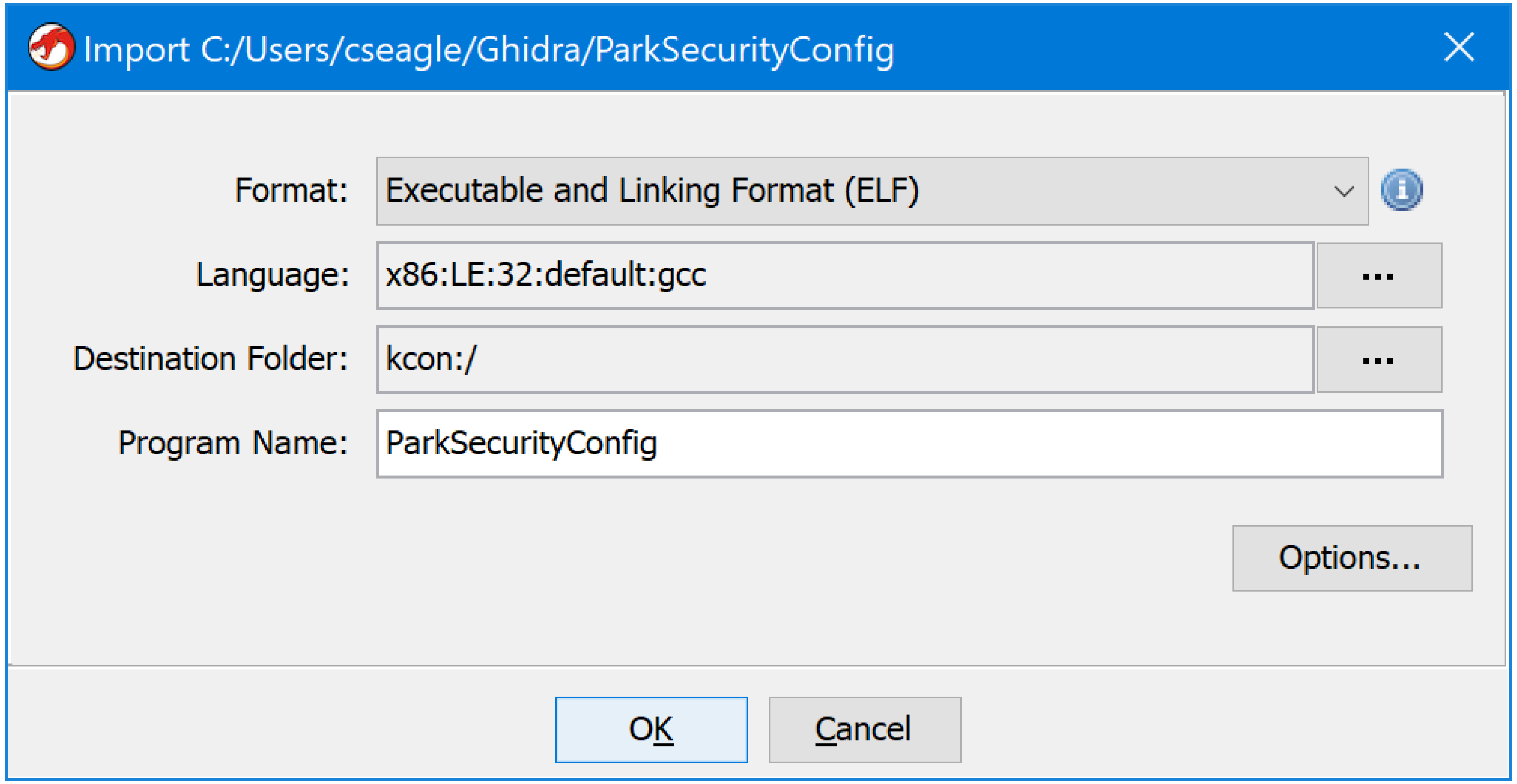
● Non-Shared Project

○ Shared Project

Closed project: test1

Import binaries

- Performs basic file identification
- Can also load dependent shared libraries



Import C:/Users/cseagle/Ghidra/ParkSecurityConfig

Format: Executable and Linking Format (ELF)

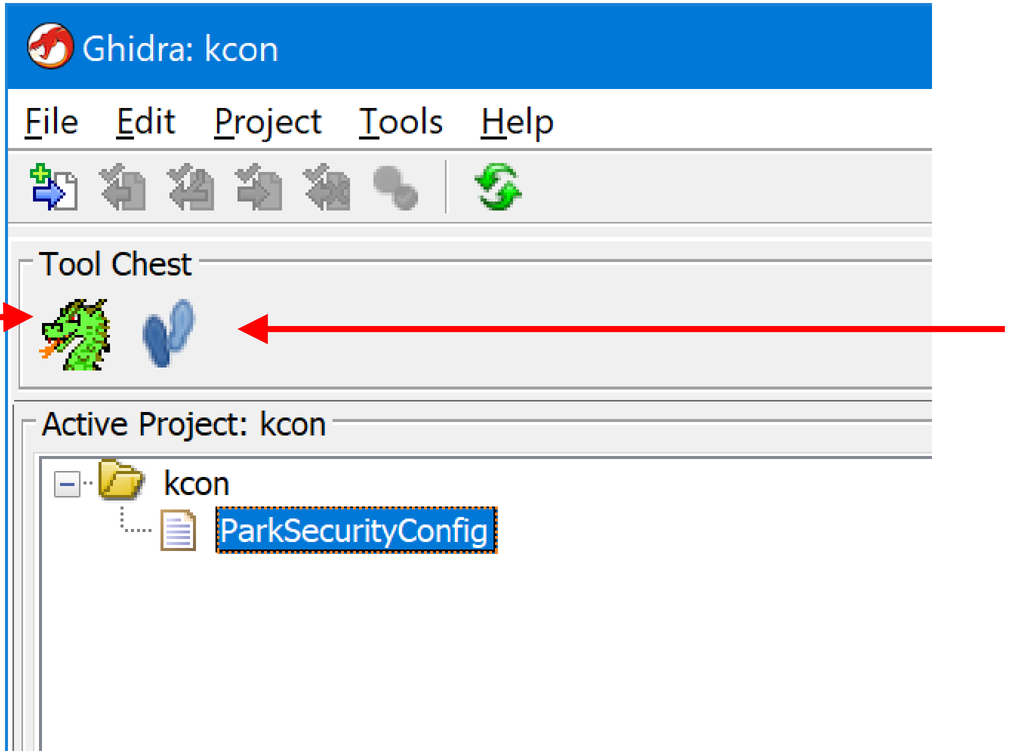
Language: x86:LE:32:default:gcc

Destination Folder: kcon:/

Program Name: ParkSecurityConfig

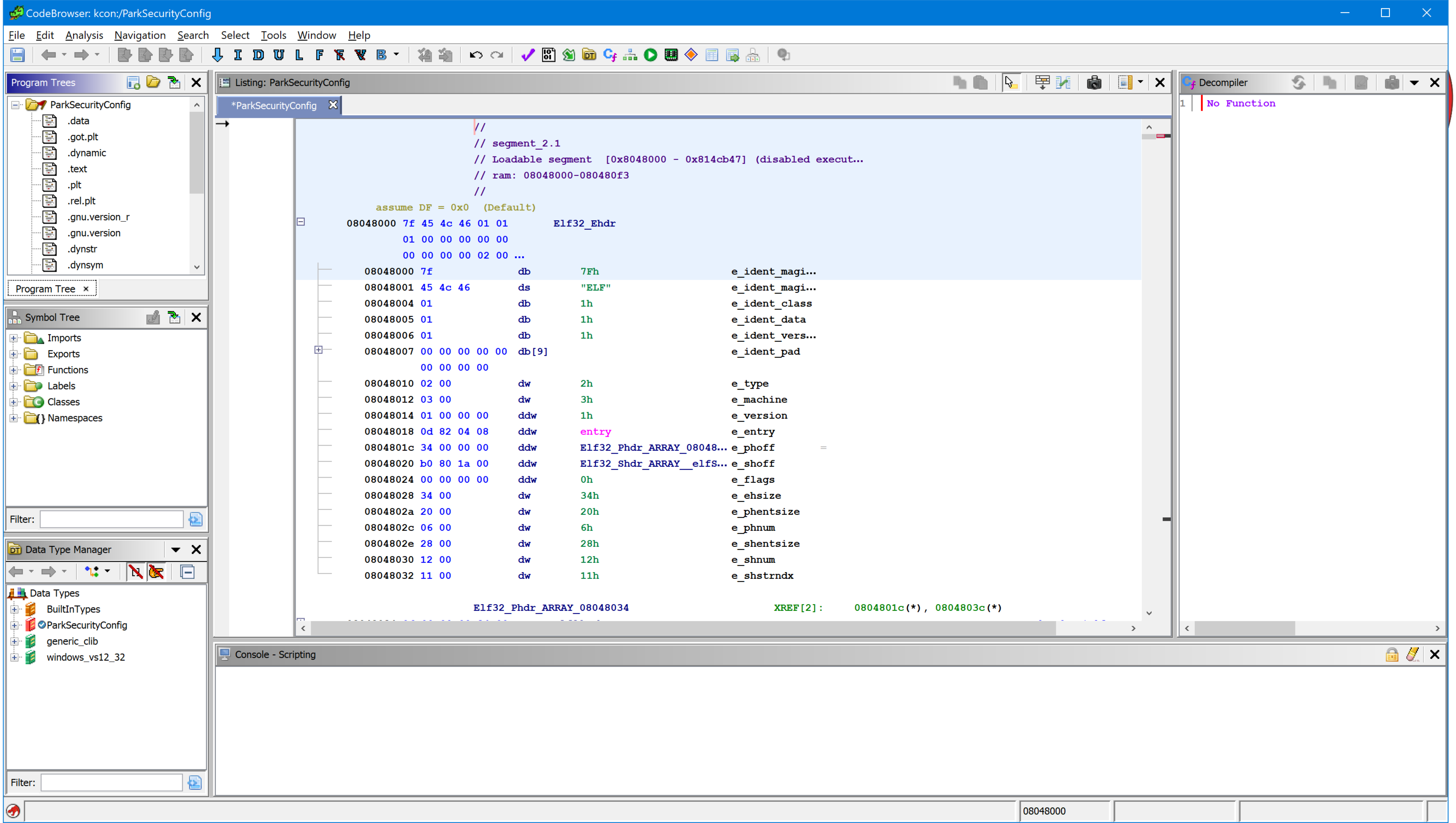
Analysis

- Ghidra is tool based, default is disassembler
- Initial analysis is much like Ida, but positions at file header



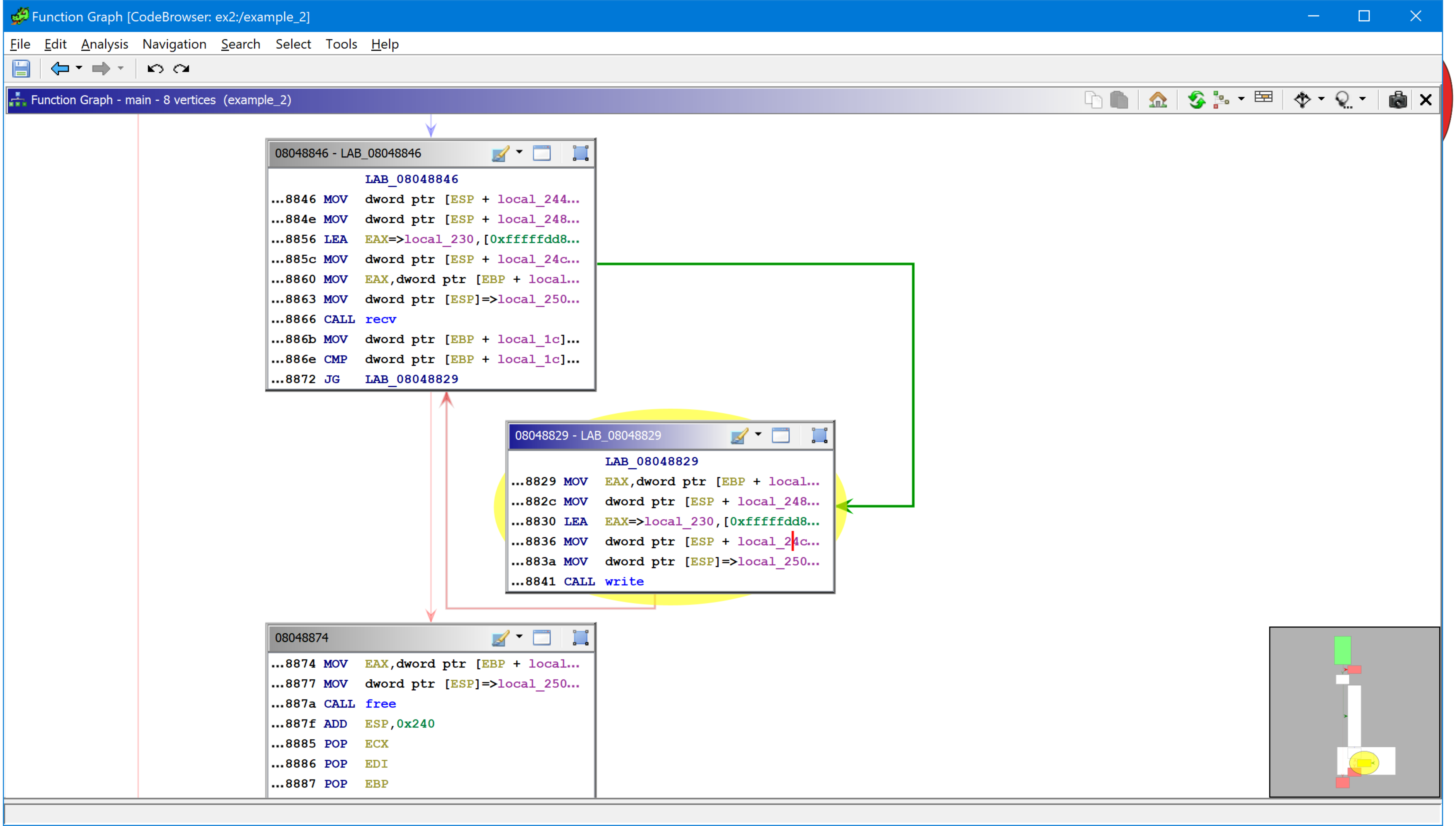
Disassembler

Version tracker



```
                              //
                              // segment_2.1
                              // Loadable segment  [0x8048000 - 0x814cb47] (disabled execut...
                              // ram: 08048000-080480f3
                              //
                  assume DF = 0x0  (Default)
             08048000 7f 45 4c 46 01 01        Elf32_Ehdr
                      01 00 00 00 00 00
                      00 00 00 00 02 00 ...
                08048000 7f             db        7Fh                    e_ident_magi...
                08048001 45 4c 46       ds        "ELF"                  e_ident_magi...
                08048004 01             db        1h                     e_ident_class
                08048005 01             db        1h                     e_ident_data
                08048006 01             db        1h                     e_ident_vers...
                08048007 00 00 00 00 00 db[9]                            e_ident_pad
                         00 00 00 00
                08048010 02 00          dw        2h                     e_type
                08048012 03 00          dw        3h                     e_machine
                08048014 01 00 00 00    ddw       1h                     e_version
                08048018 0d 82 04 08    ddw       entry                  e_entry
                0804801c 34 00 00 00    ddw       Elf32_Phdr_ARRAY_08048... e_phoff
                08048020 b0 80 1a 00    ddw       Elf32_Shdr_ARRAY__elfS... e_shoff
                08048024 00 00 00 00    ddw       0h                     e_flags
                08048028 34 00          dw        34h                    e_ehsize
                0804802a 20 00          dw        20h                    e_phentsize
                0804802c 06 00          dw        6h                     e_phnum
                0804802e 28 00          dw        28h                    e_shentsize
                08048030 12 00          dw        12h                    e_shnum
                08048032 11 00          dw        11h                    e_shstrndx

                              Elf32_Phdr_ARRAY_08048034              XREF[2]:     0804801c(*), 0804803c(*)
```

```
1  No Function
```

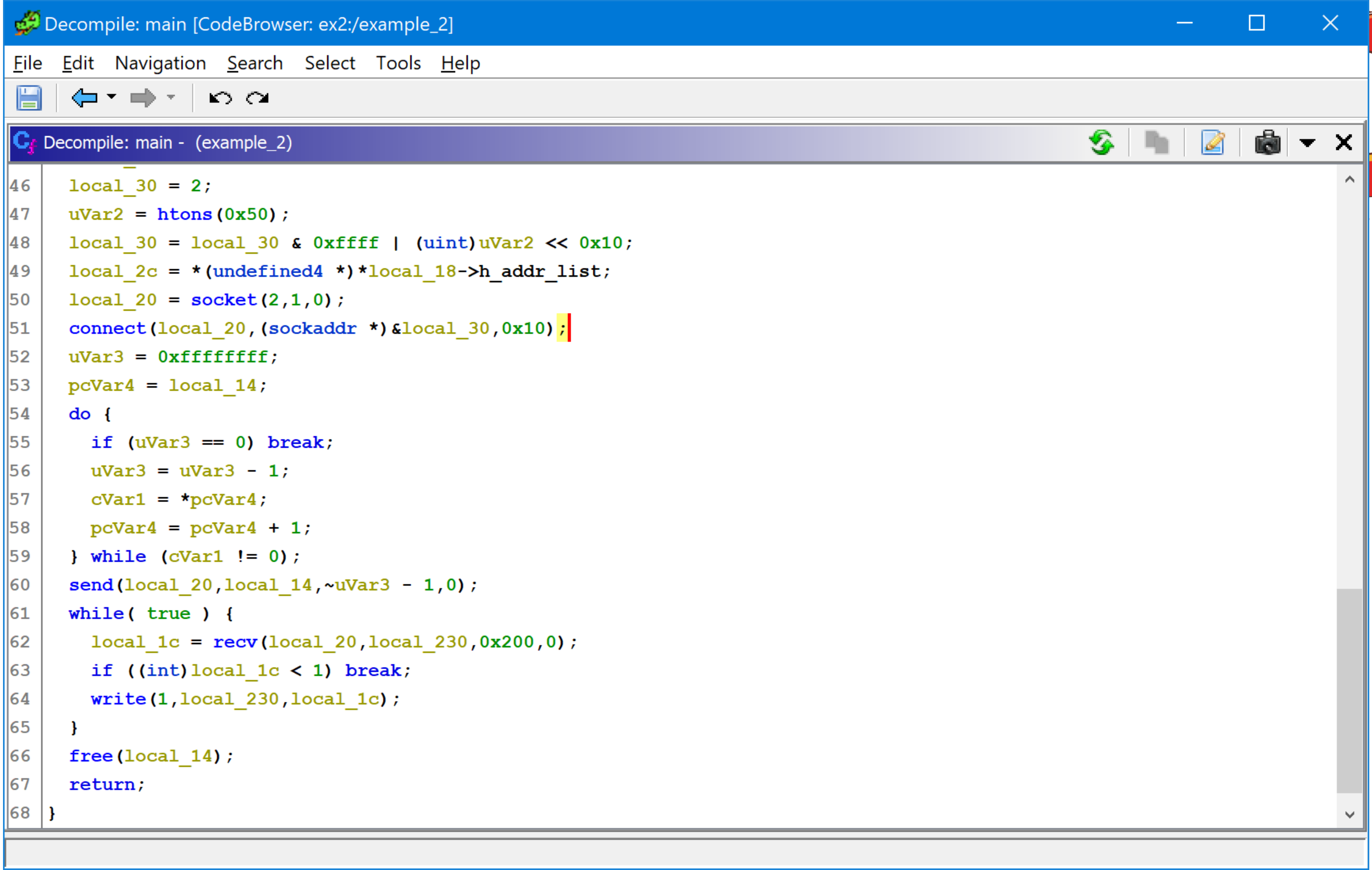


Function Graph [CodeBrowser: ex2:/example_2]
File
Edit
Analysis
Navigation
Search
Select
Tools
Help
Function Graph - main - 8 vertices (example_2)

```
08048846 - LAB_08048846

              LAB_08048846
...8846 MOV   dword ptr [ESP + local_244...
...884e MOV   dword ptr [ESP + local_248...
...8856 LEA   EAX=>local_230,[0xffffdd8...
...885c MOV   dword ptr [ESP + local_24c...
...8860 MOV   EAX,dword ptr [EBP + local...
...8863 MOV   dword ptr [ESP]=>local_250...
...8866 CALL  recv
...886b MOV   dword ptr [EBP + local_1c]...
...886e CMP   dword ptr [EBP + local_1c]...
...8872 JG    LAB_08048829
```

```
08048829 - LAB_08048829

              LAB_08048829
...8829 MOV   EAX,dword ptr [EBP + local...
...882c MOV   dword ptr [ESP + local_248...
...8830 LEA   EAX=>local_230,[0xffffdd8...
...8836 MOV   dword ptr [ESP + local_24c...
...883a MOV   dword ptr [ESP]=>local_250...
...8841 CALL  write
```

```
08048874

...8874 MOV   EAX,dword ptr [EBP + local...
...8877 MOV   dword ptr [ESP]=>local_250...
...887a CALL  free
...887f ADD   ESP,0x240
...8885 POP   ECX
...8886 POP   EDI
...8887 POP   EBP
```


Decompile: main [CodeBrowser: ex2:/example_2]
File Edit Navigation Search Select Tools Help
Decompile: main - (example_2)

```
  local_30 = 2;
  uVar2 = htons(0x50);
  local_30 = local_30 & 0xffff | (uint)uVar2 << 0x10;
  local_2c = *(undefined4 *)*local_18->h_addr_list;
  local_20 = socket(2,1,0);
  connect(local_20,(sockaddr *)&local_30,0x10);
  uVar3 = 0xffffffff;
  pcVar4 = local_14;
  do {
    if (uVar3 == 0) break;
    uVar3 = uVar3 - 1;
    cVar1 = *pcVar4;
    pcVar4 = pcVar4 + 1;
  } while (cVar1 != 0);
  send(local_20,local_14,~uVar3 - 1,0);
  while( true ) {
    local_1c = recv(local_20,local_230,0x200,0);
    if ((int)local_1c < 1) break;
    write(1,local_230,local_1c);
  }
  free(local_14);
  return;
}
```

Scripting

- Scripting with Java (not javascript) is well supported
- Support for Eclipse integration and debugging
- Javadoc for Ghidra APIs is included with Ghidra distro
- Python scripting supported via Jython
- Nice blog post on developing with Python:
  - https://www.somersetrecon.com/blog/2019/ghidra-plugin-development-for-vulnerability-research-part-1

Some observations

- Longer analysis times on large binaries
- Stack listings are in reverse order compared to Ida
  - No separate stack view ?
- Trouble with switch idioms (jump tables)
- Better data type editor
  - Structure creation
- No debugger (in work?)

Impact of Ghidra release

- Perhaps drive price decreases?
- Perhaps more features?
  - Undo or collaboration in Ida?
- Already have seen Hex-Rays offer a free educational license
- Unclear how market share will shake out
- Huge win for education and independent tinkerers

That’s All

Thank you and questions