Three Heads Are Better Than One: Mastering NSA's Ghidra Reverse Engineering Tool

Alexei Bulazel
@0xAlexei

Jeremy Blackthorne
@0xJeremy

github.com/0xAlexei/INFILTRATE2019

Disclaimer

This materials is based on the publicly released Ghidra, there is no classified information in this presentation

Alexei Bulazel @0xAlexei

- Senior Security Researcher at River Loop Security
- Research presentations and publications:
  - Presentations at REcon (MTL & BRX), SummerCon, DEFCON, Black Hat, etc.
  - Academic publications at USENIX WOOT and ROOTS
  - Cyber policy in Lawfare, etc.
- Collaborated with Jeremy on research at RPI, MIT Lincoln Laboratory, and Boston Cybernetics Institute
- Proud RPISEC alumnus

RPISEC

Jeremy Blackthorne @0xJeremy

- Instructor at the Boston Cybernetics Institute
- PhD candidate at RPI focused on environmental keying
- Former researcher at MIT Lincoln Laboratory
- United States Marine Corps 2002 - 2006
- RPISEC alumnus

RPISEC

Outline

1. Intro
2. Interactive Exercises
    a. Manual Static Analysis
    b. Scripting Ghidra
3. P-Code & SLEIGH
4. Discussion
5. Conclusion

Participating

1. Install OpenJDK 11, add its `bin` directory to your `PATH`
   - `jdk.java.net/11`
2. Download Ghidra
   - `ghidra-sre.org`
   - `github.com/NationalSecurityAgency/ghidra/releases`
3. Download our demo scripts and binaries
   - `github.com/0xAlexei/INFILTRATE2019`

Ghidra

- Java-based interactive reverse engineering tool developed by US National Security Agency - similar in functionality to IDA Pro, Binary Ninja, etc…
  - Static analysis only currently, debugger support promised to be coming soon
  - Runs on Mac, Linux, and Windows

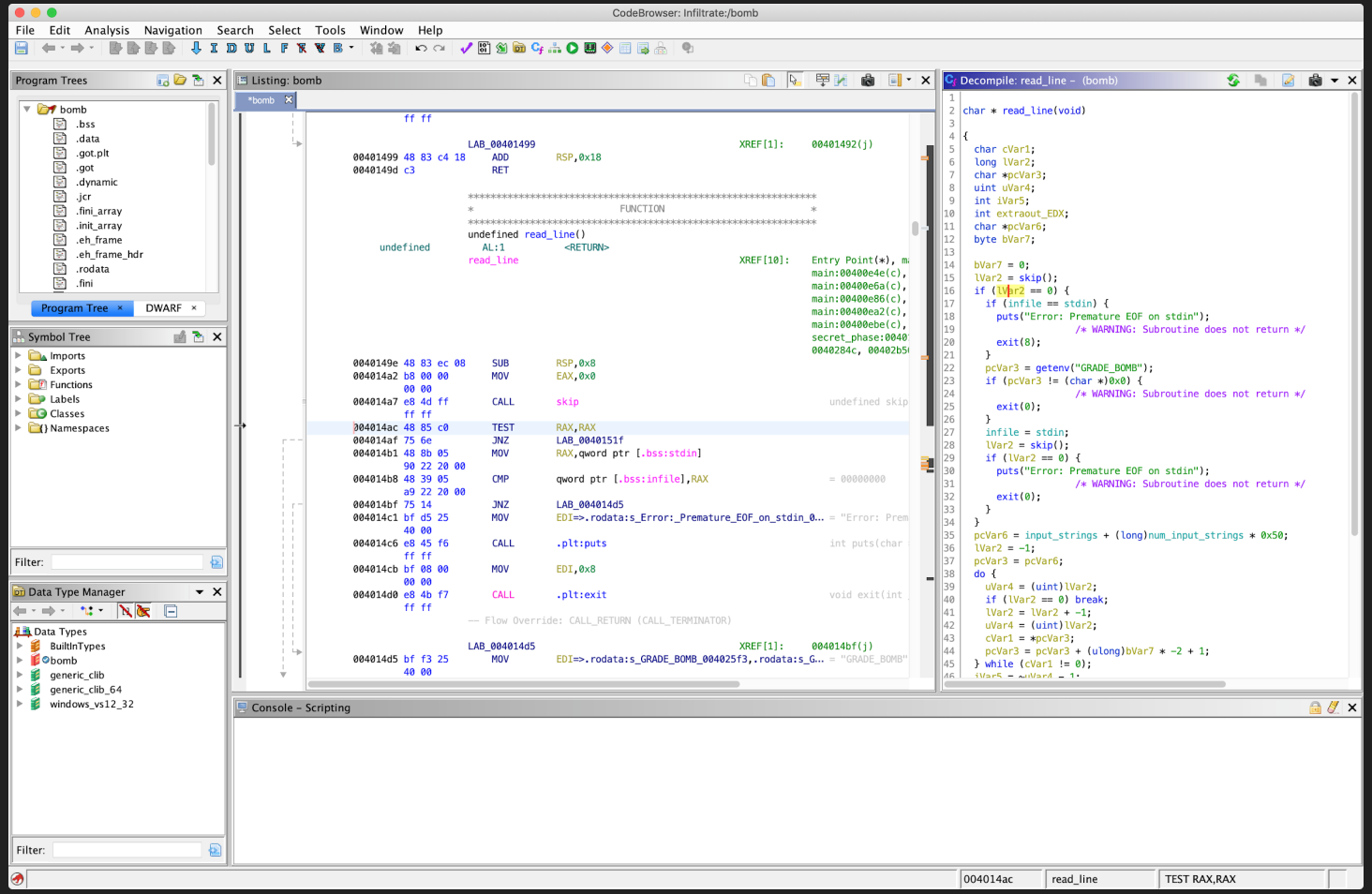
- All credit for creating Ghidra goes to the developers at NSA
- Released open source at RSA in March 2019
  - 1.2M+ lines of code
- NSA has not discussed the history of the tool, but comments in source files go as far back as February 1999


```
$ grep -r "1999" * --include *.java
src/Generic-src/ghidra/util/exception/NotYetImplementedException.java: * @version 1999/02/05
src/SoftwareModeling-src/ghidra/program/model/address/AddressOutOfBoundsException.java: * @version 1999-03-31
src/SoftwareModeling-src/ghidra/program/model/scalar/ScalarOverflowException.java: * @version 1999-03-31
src/SoftwareModeling-src/ghidra/program/model/scalar/ScalarFormat.java: * @version 1999/02/04
```

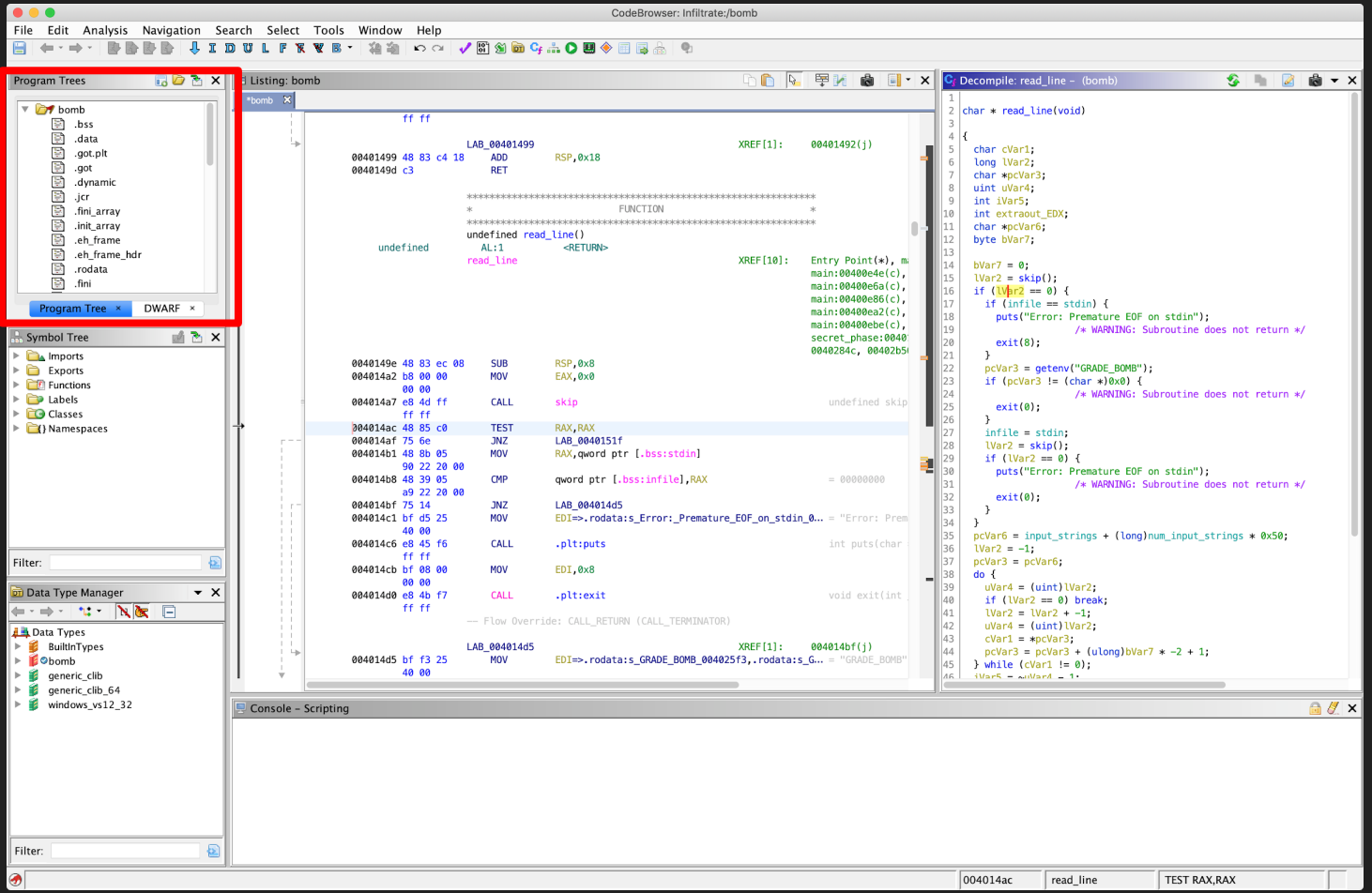
Outline

1. Intro
2. Interactive Exercises
   a. Manual Static Analysis
   b. Scripting Ghidra
3. P-Code & SLEIGH
4. Discussion
5. Conclusion

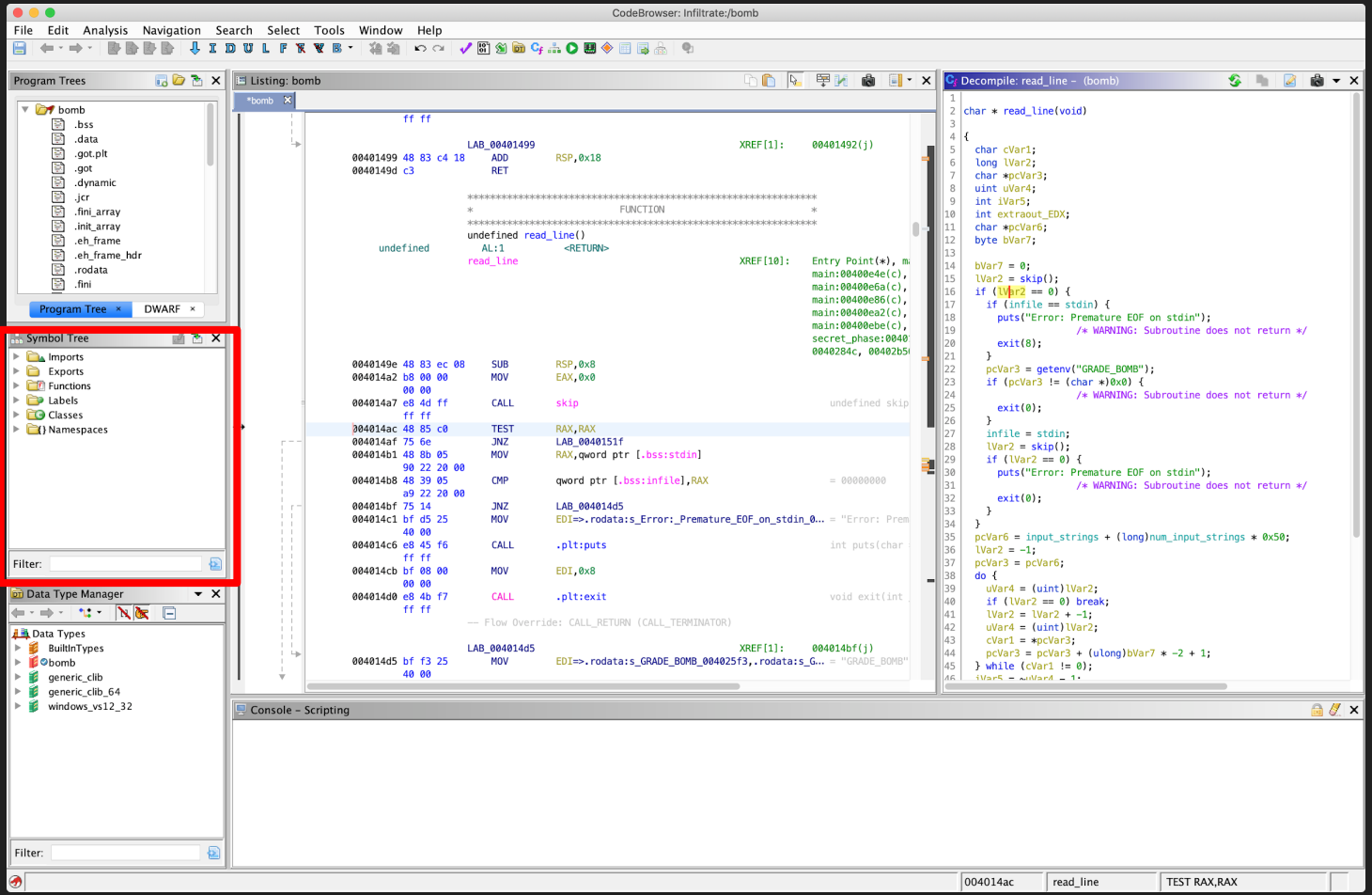
Default UI - CodeBrowser



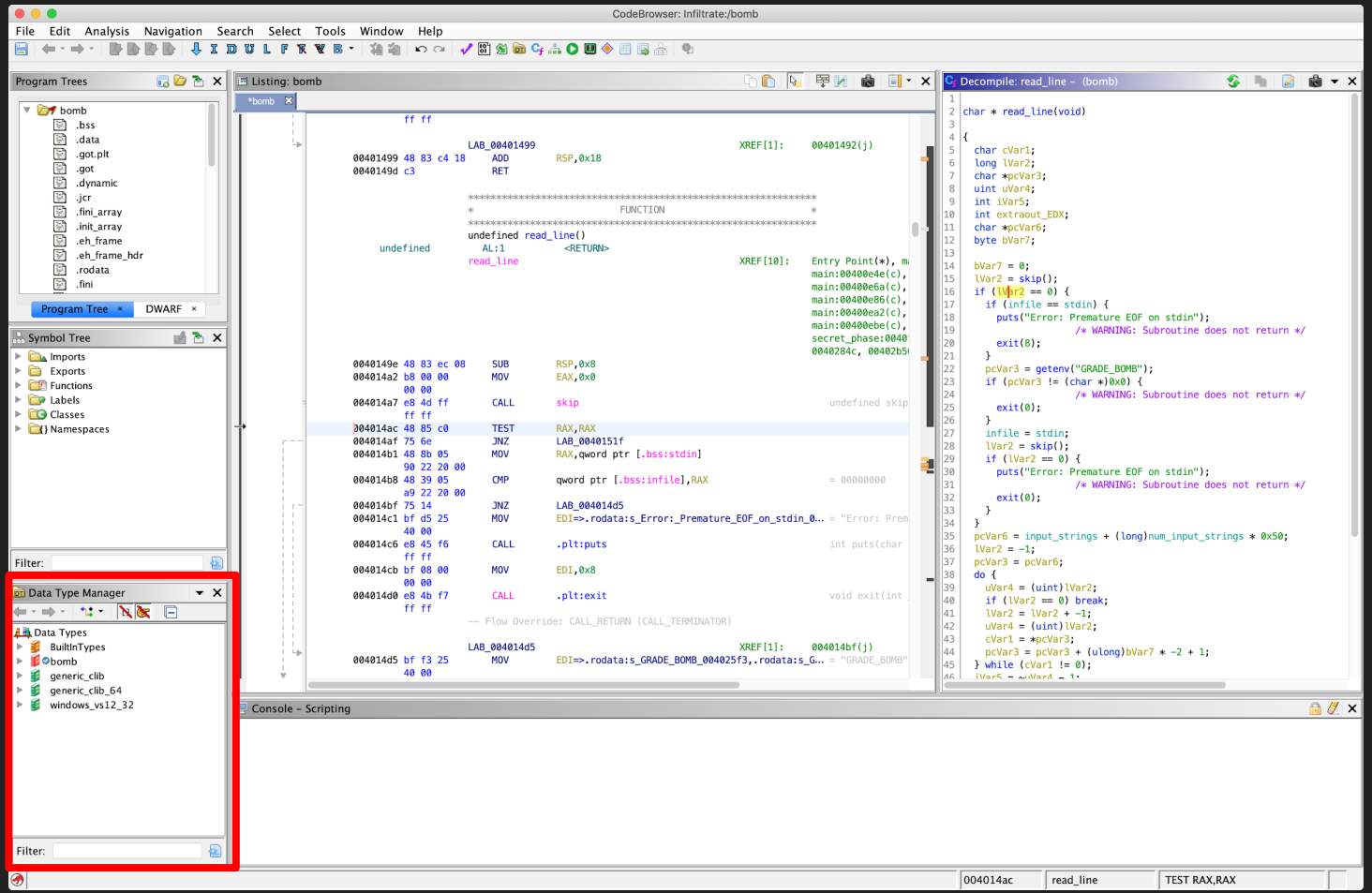
Default UI - Program Trees



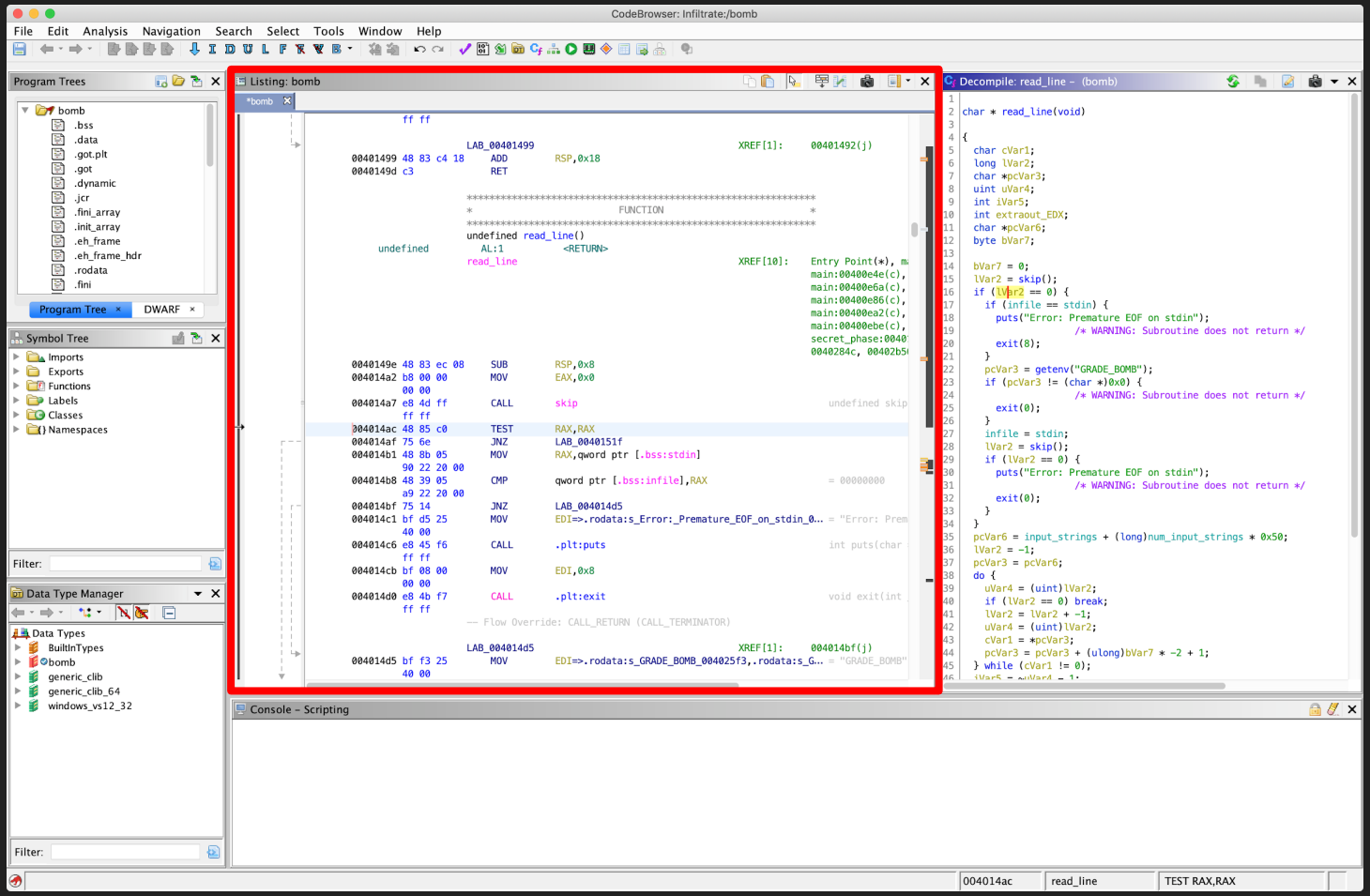
Default UI - Symbol Tree



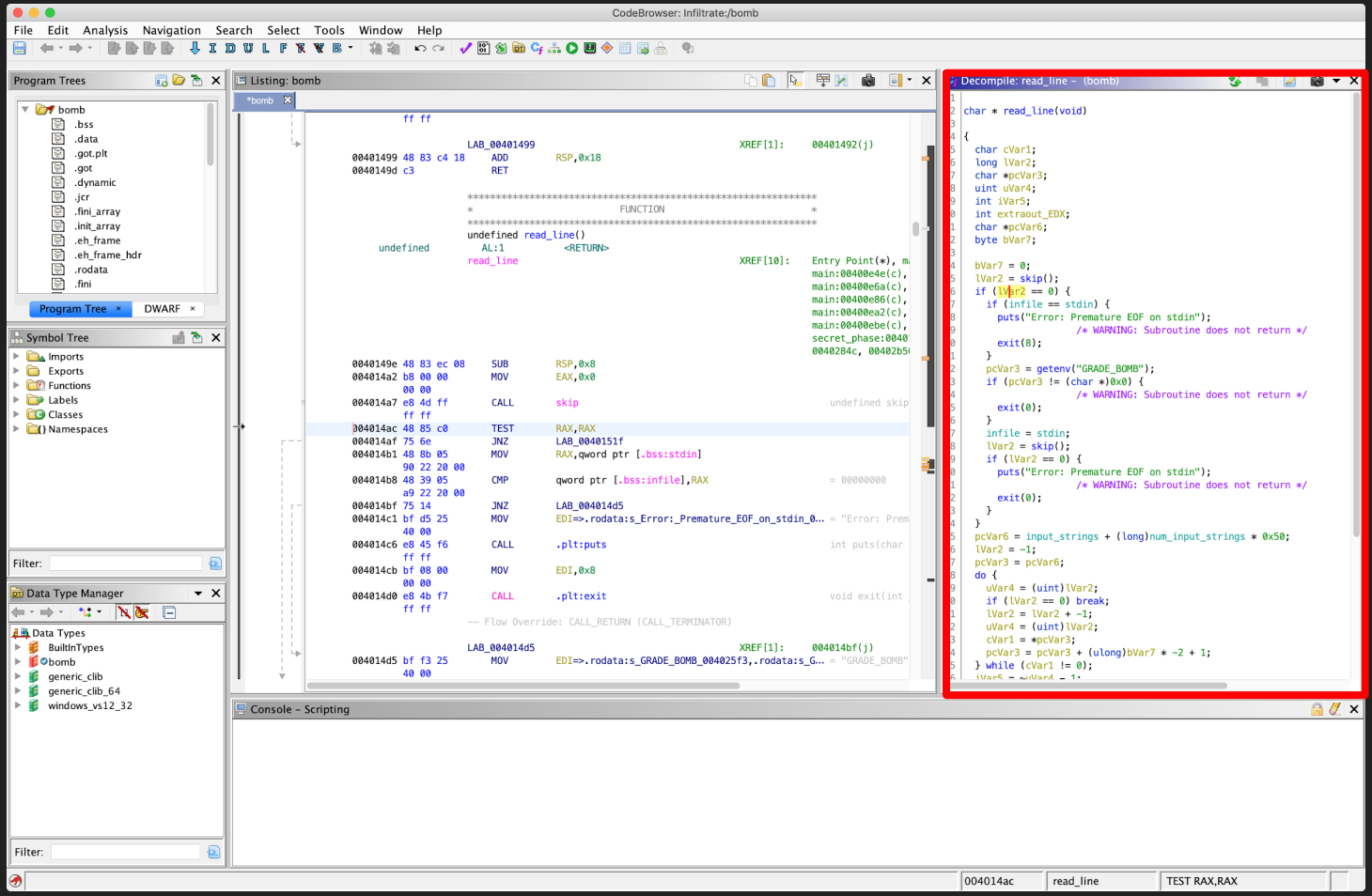
Default UI - Data Type Manager



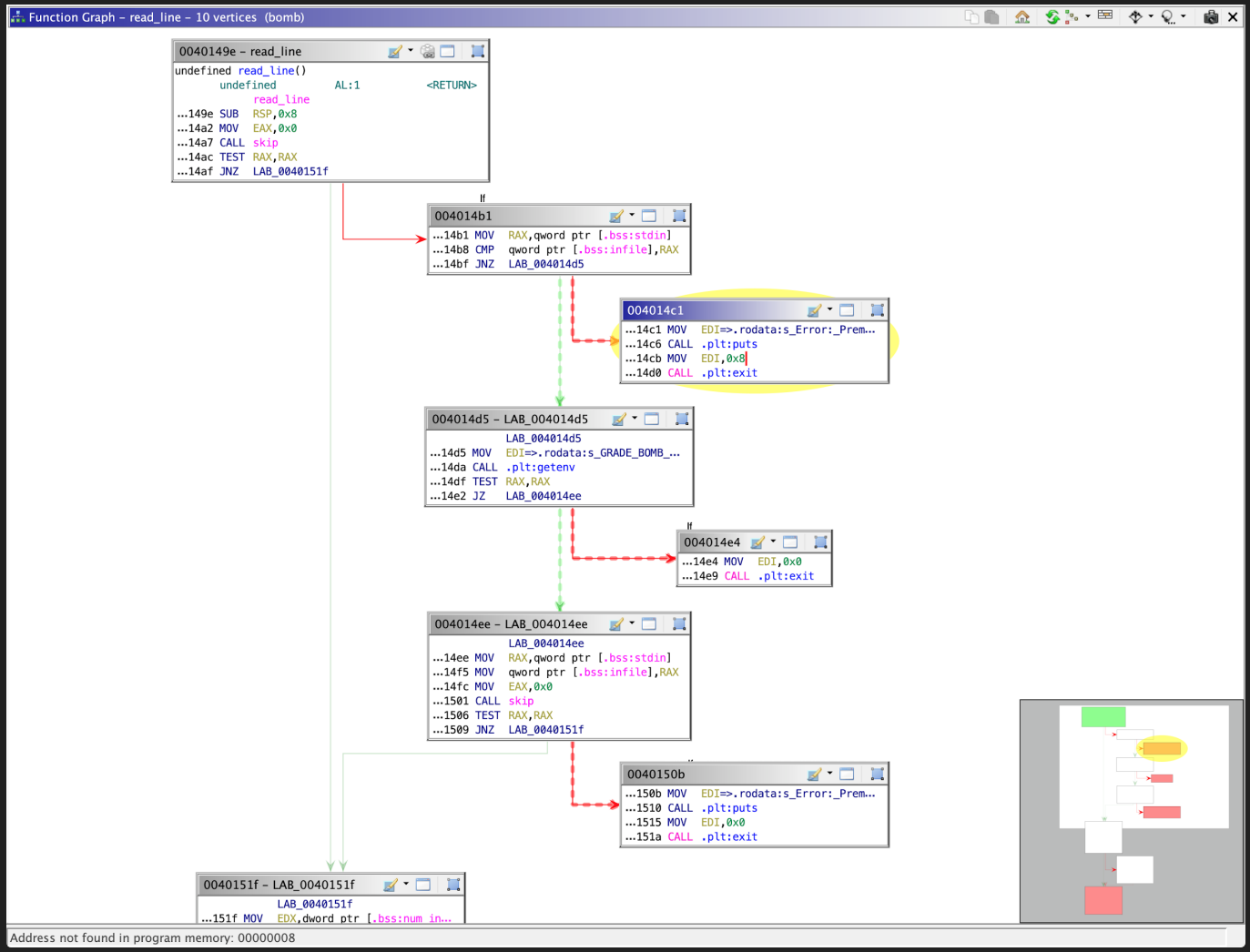
Default UI - Listing (Disassembly)



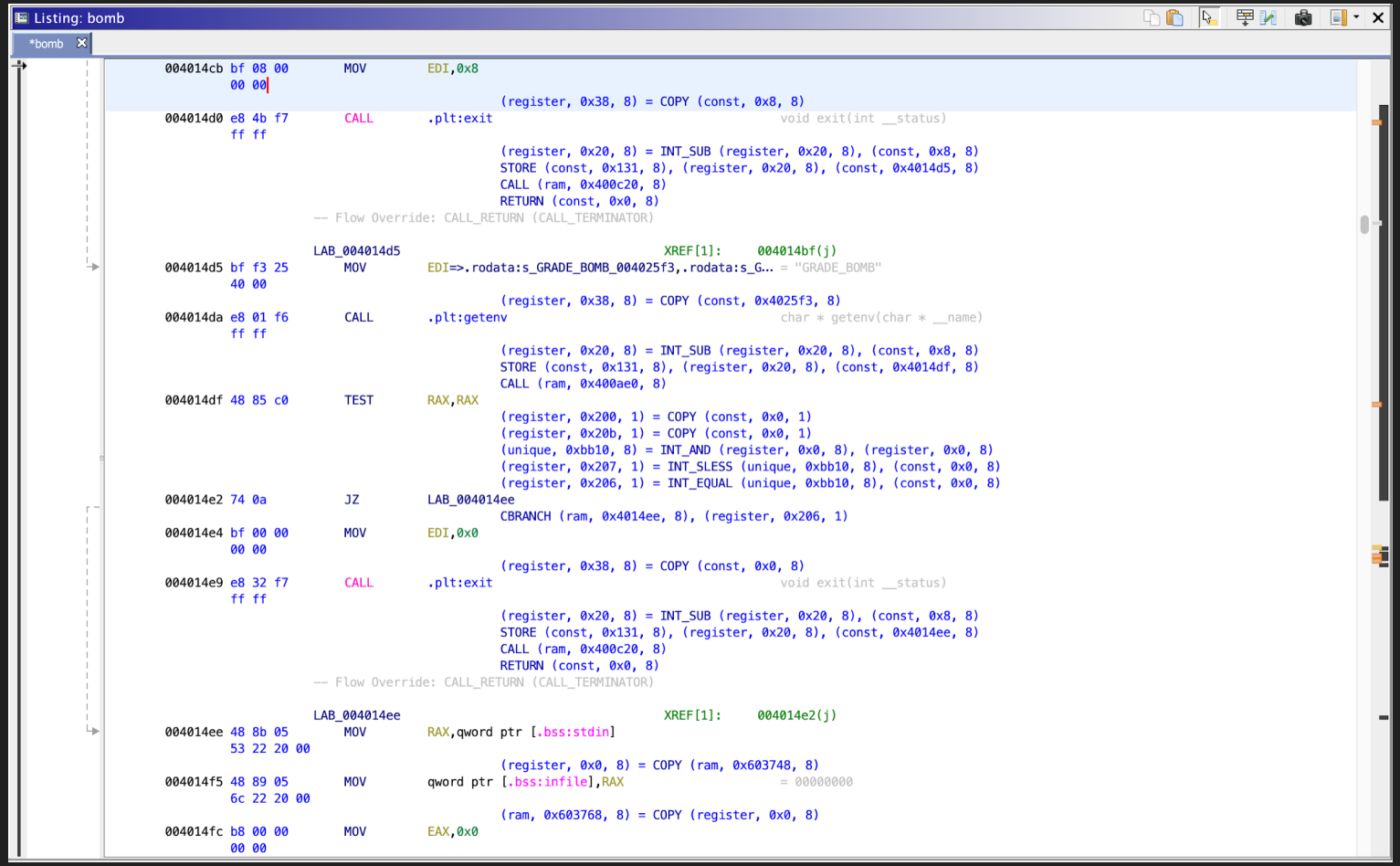
Default UI - Decompiler



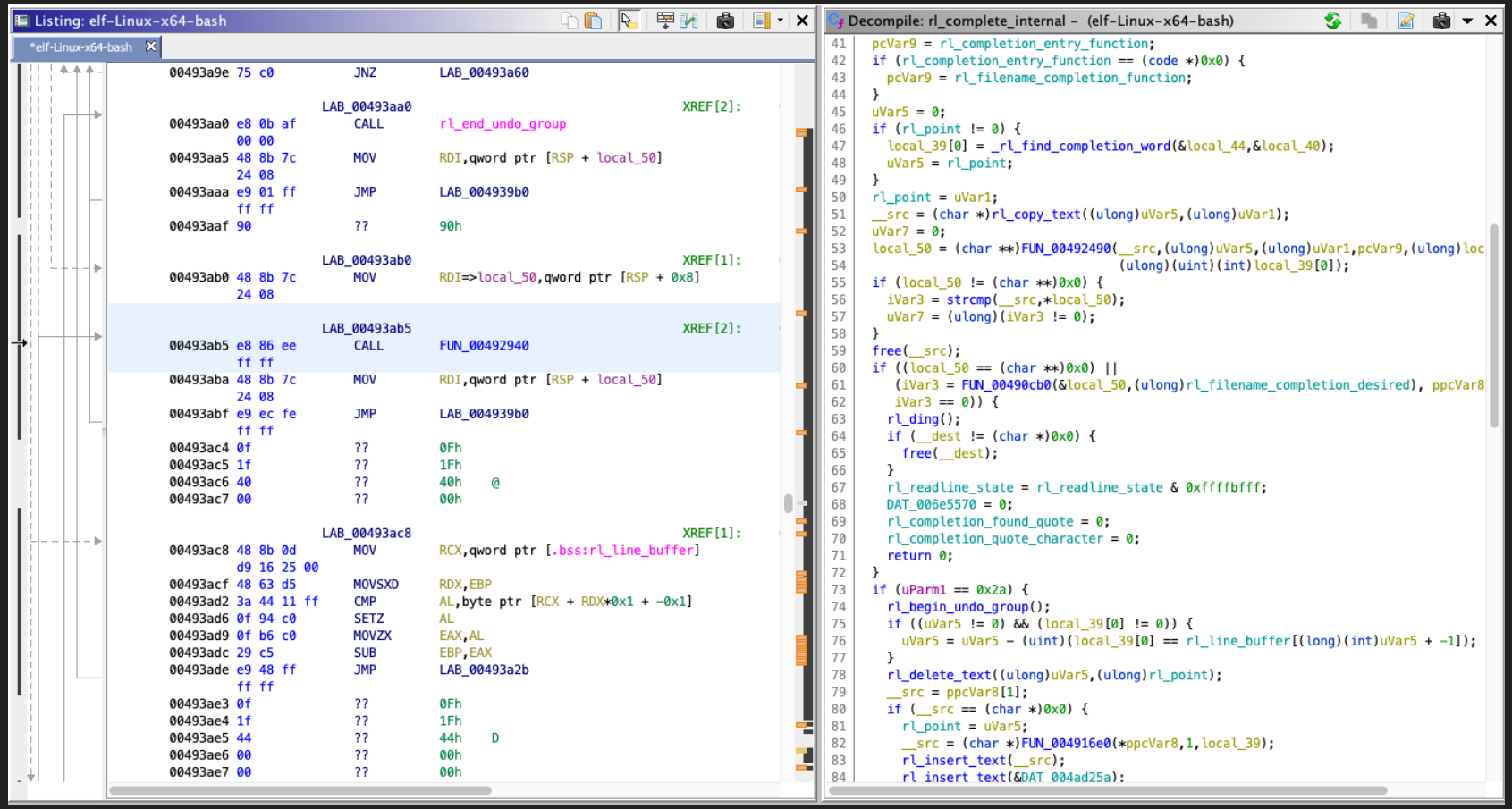
Control Flow Graph



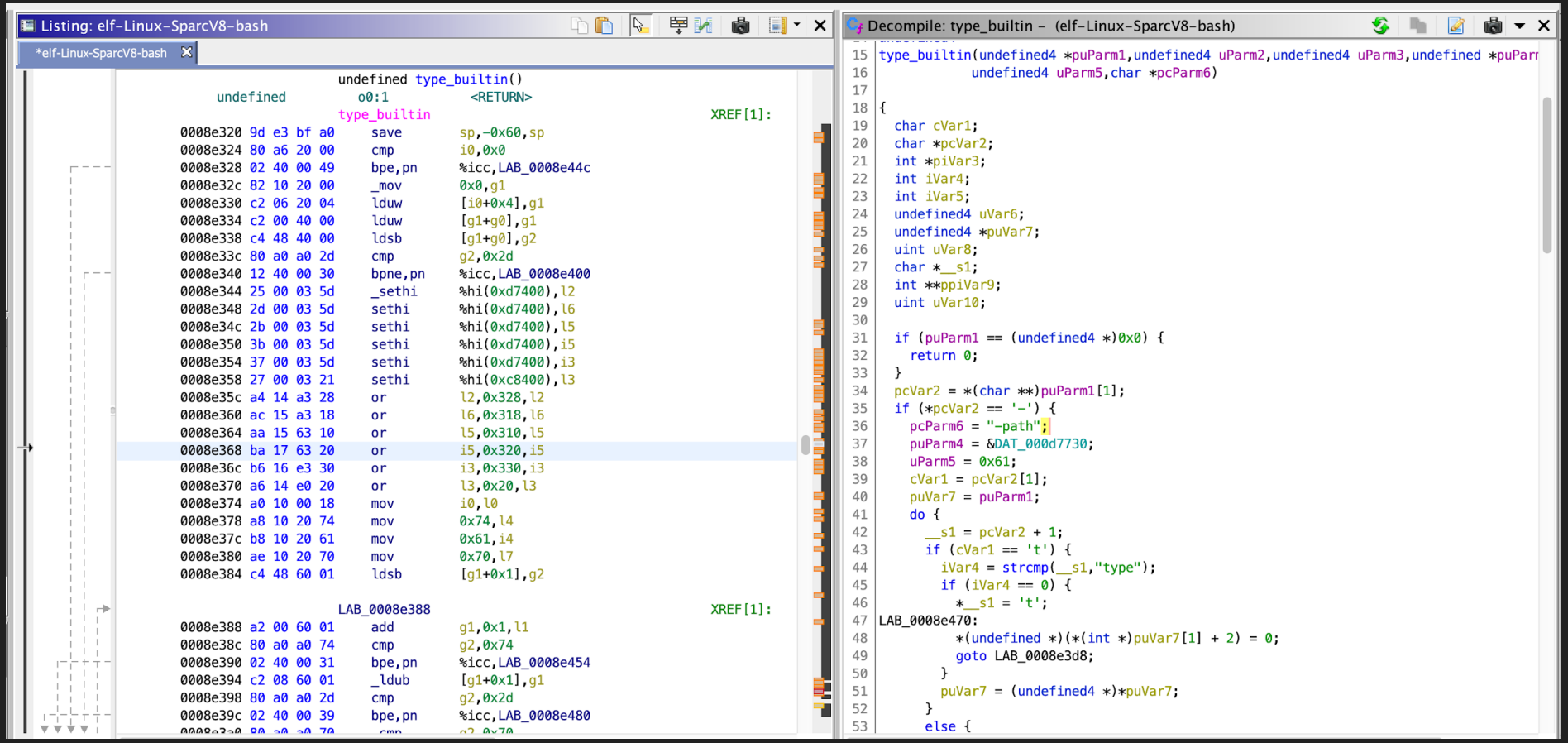
Disassembly with P-Code



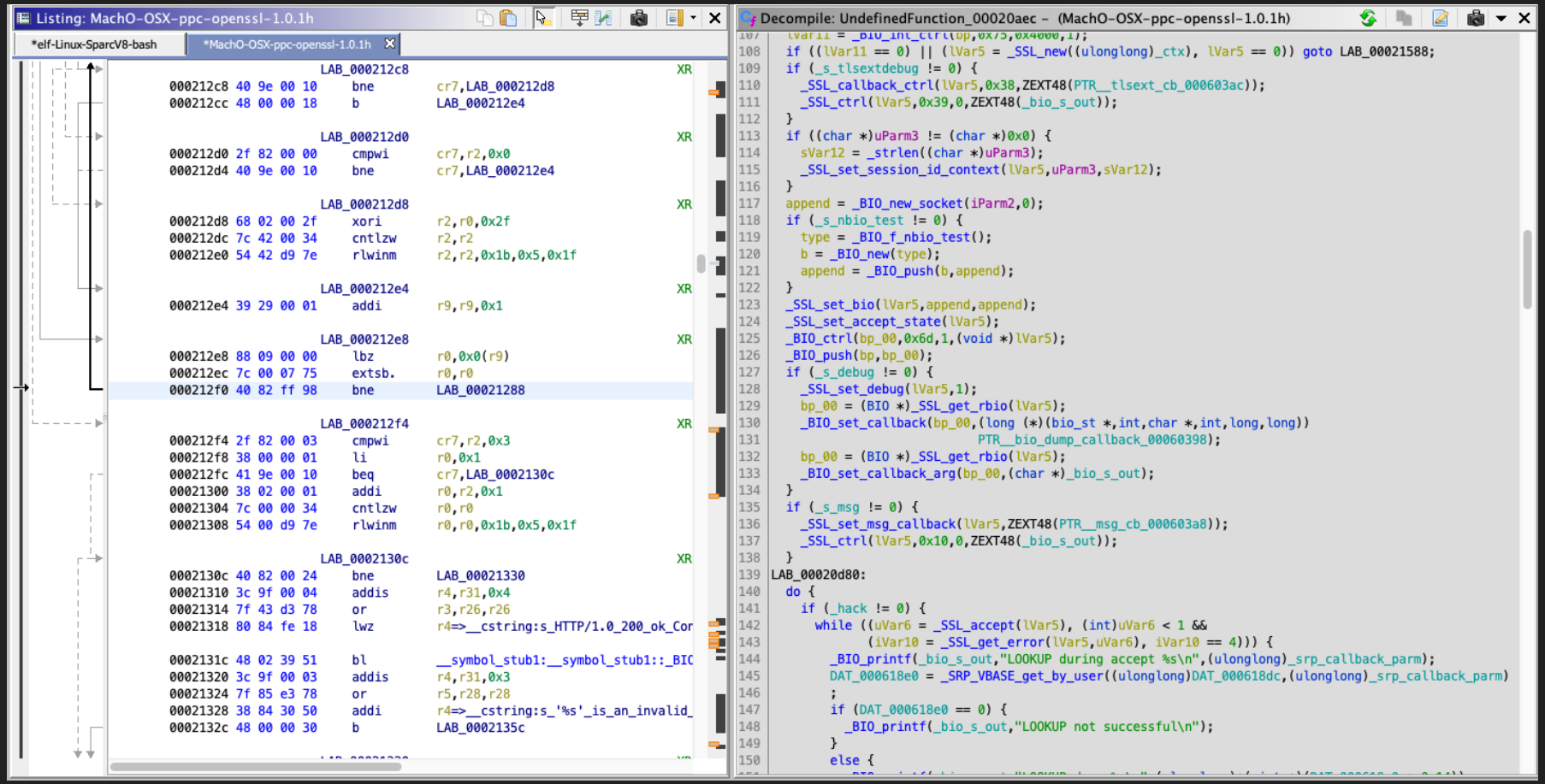
Decompilation Across Architectures - x64



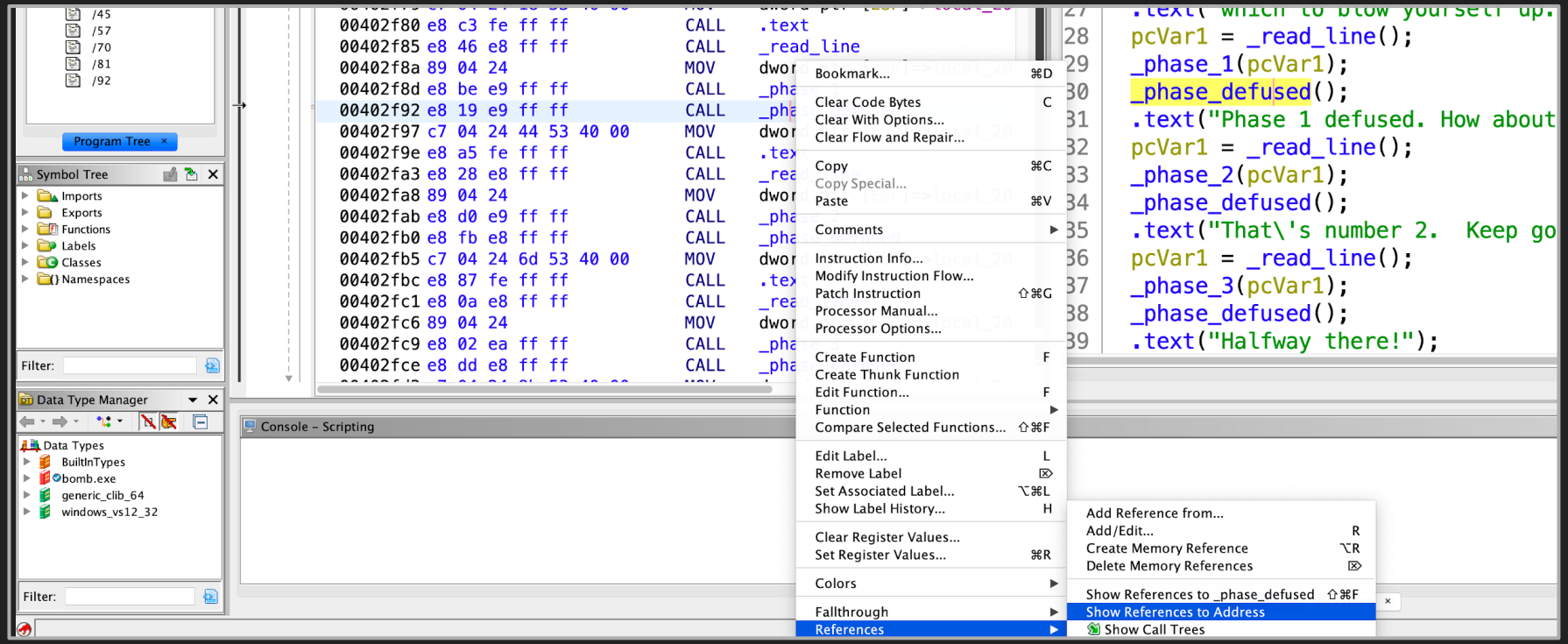
Decompilation Across Architectures - SPARC



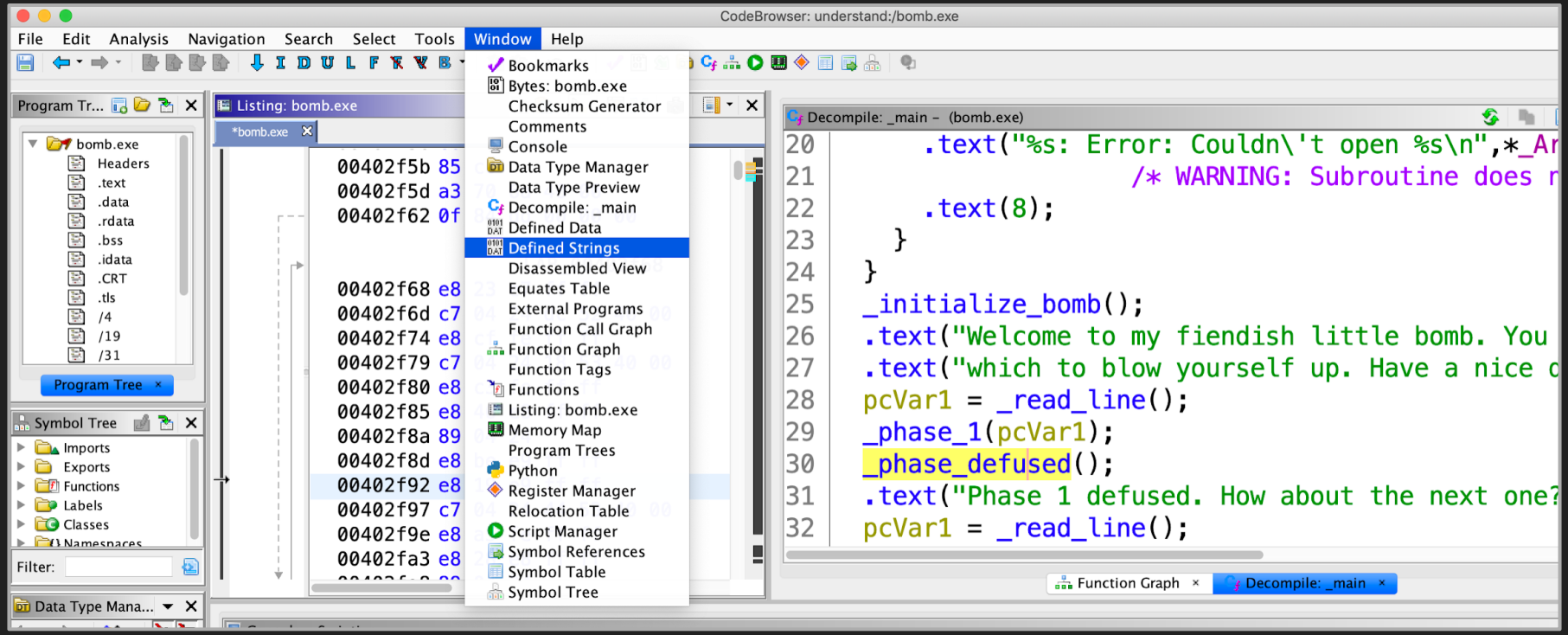
Decompilation Across Architectures - PowerPC



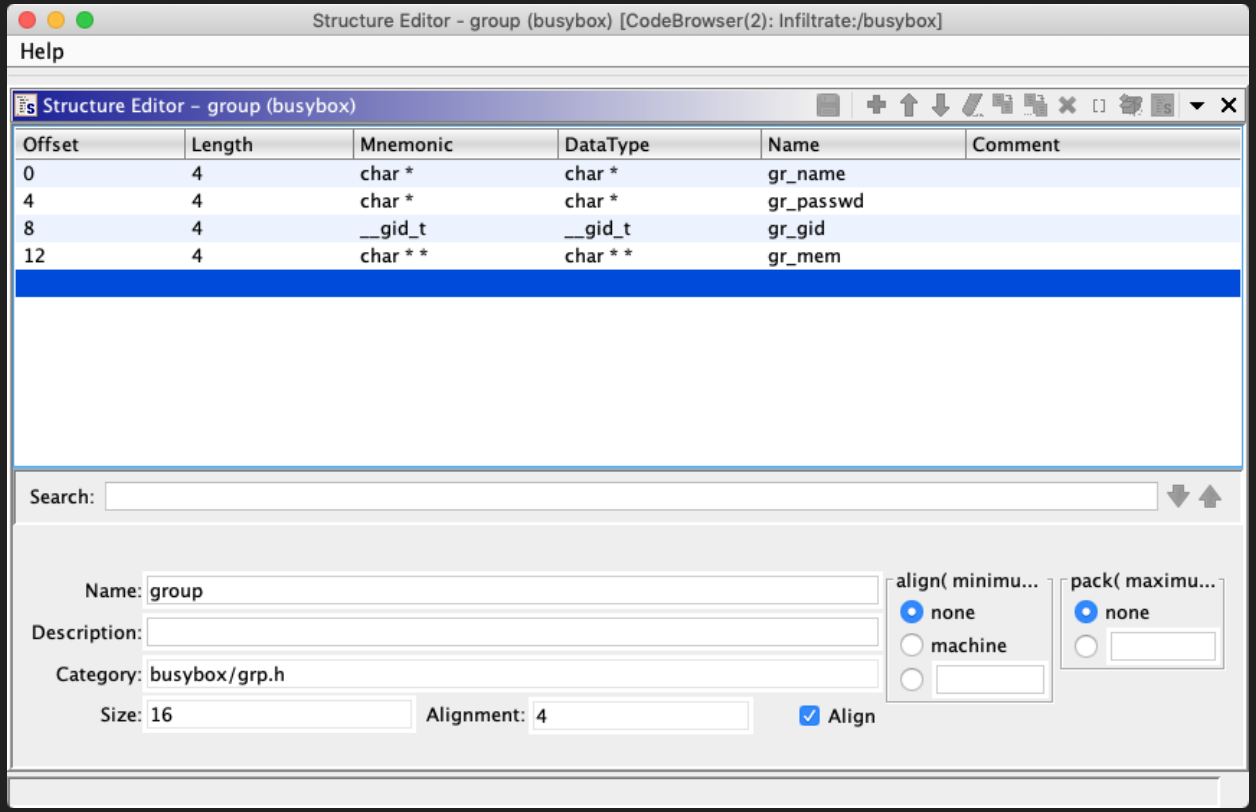
Cross-References



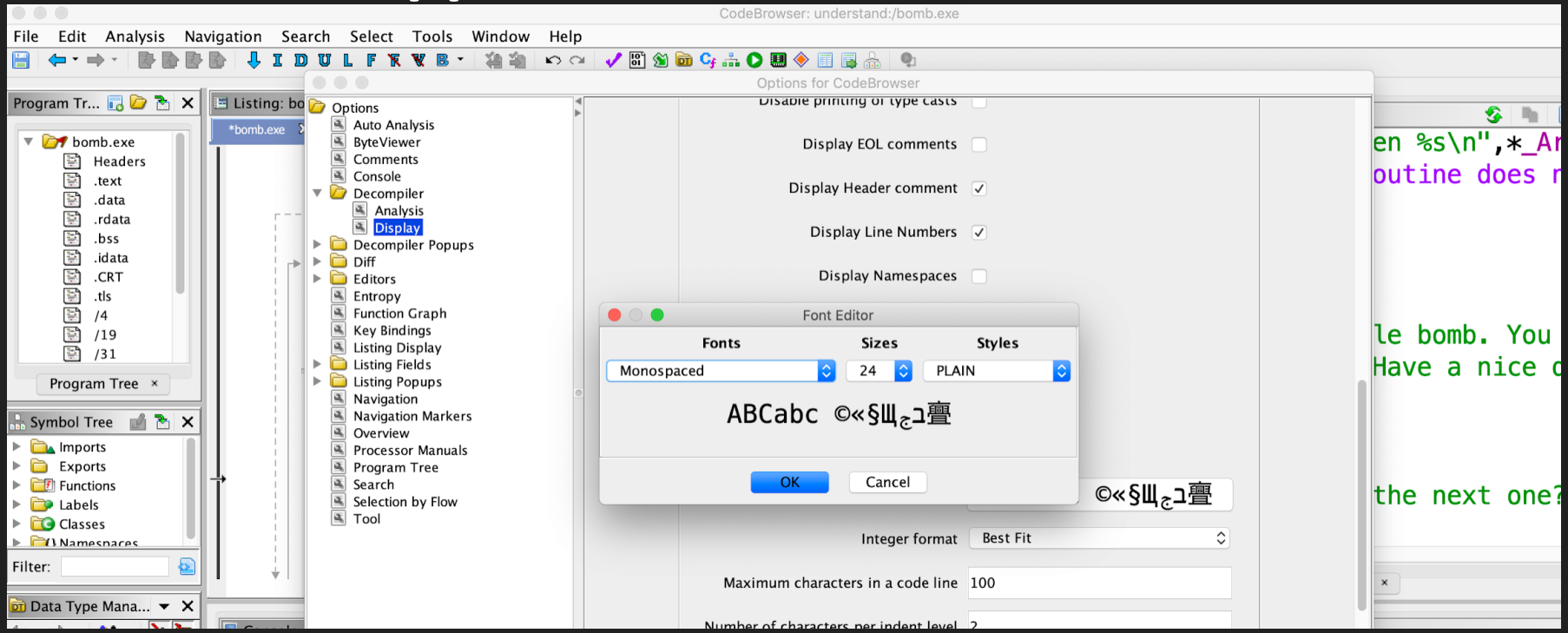
Strings


CodeBrowser: understand:/bomb.exe
File Edit Analysis Navigation Search Select Tools Window Help
Bookmarks
Bytes: bomb.exe
Checksum Generator
Comments
Console
Data Type Manager
Data Type Preview
Decompile: _main
Defined Data
Defined Strings
Disassembled View
Equates Table
External Programs
Function Call Graph
Function Graph
Function Tags
Functions
Listing: bomb.exe
Memory Map
Program Trees
Python
Register Manager
Relocation Table
Script Manager
Symbol References
Symbol Table
Symbol Tree
Program Tr...
bomb.exe
Headers
.text
.data
.rdata
.bss
.idata
.CRT
.tls
/4
/19
/31
Program Tree
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Filter:
Data Type Mana...
Listing: bomb.exe
*bomb.exe
00402f5b 85
00402f5d a3
00402f62 0f
00402f68 e8
00402f6d c7
00402f74 e8
00402f79 c7
00402f80 e8
00402f85 e8
00402f8a 89
00402f8d e8
00402f92 e8
00402f97 c7
00402f9e e8
00402fa3 e8
Decompile: _main - (bomb.exe)
20 .text("%s: Error: Couldn\'t open %s\n",*_Ar
21 /* WARNING: Subroutine does r
22 .text(8);
23 }
24 }
25 _initialize_bomb();
26 .text("Welcome to my fiendish little bomb. You
27 .text("which to blow yourself up. Have a nice c
28 pcVar1 = _read_line();
29 _phase_1(pcVar1);
30 _phase_defused();
31 .text("Phase 1 defused. How about the next one?
32 pcVar1 = _read_line();
Function Graph
Decompile: _main

Type System / Structure Recovery



Customize Appearance

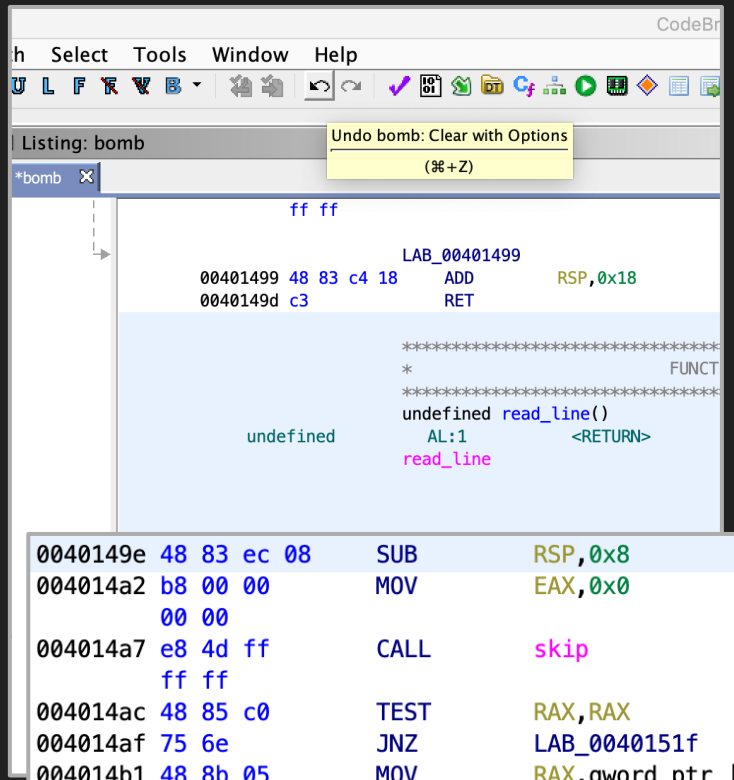


Edit >Tool Options → Filter: Fonts

The Undo Button!

```
0040149e 48 83 ec 08      SUB         RSP,0x8
004014a2 b8 00 00         MOV         EAX,0x0
         00 00
004014a7 e8 4d ff         CALL        skip
         ff ff
004014ac 48 85 c0         TEST        RAX,RAX
004014af 75 6e            JNZ         LAB_0040151f
004014b1 48 8b 05         MOV         RAX,qword ptr [.bss:stdin]
         90 22 20 00
004014b8 48 39 05         CMP         qword ptr [.bss:infile],RAX
         a9 22 20 00
004014bf 75 14            JNZ         LAB_004014d5
0040
0040
```

```
0040149e 48               ??          48h    H
0040149f 83               ??          83h
004014a0 ec               ??          ECh
004014a1 08               ??          08h
004014a2 b8 00 00         MOV         EAX,0x0
         00 00
004014a7 e8 4d ff         CALL        skip
         ff ff
004014ac 48 85 c0         TEST        RAX,RAX
004014af 75 6e            JNZ         LAB_0040151f
004014b1 48 8b 05         MOV         RAX,qword ptr [.bss:stdin]
         90 22 20 00
004014b8 48 39 05         CMP         qword ptr [.bss:infile],RAX
         a9 22 20 00
004014bf 75 14            JNZ         LAB_004014d5
004014c1 bf d5 25         MOV         EDI=>.rodata:s_Error:_Premat
         40 00
004014c6 e8 45 f6         CALL        .plt:puts
```



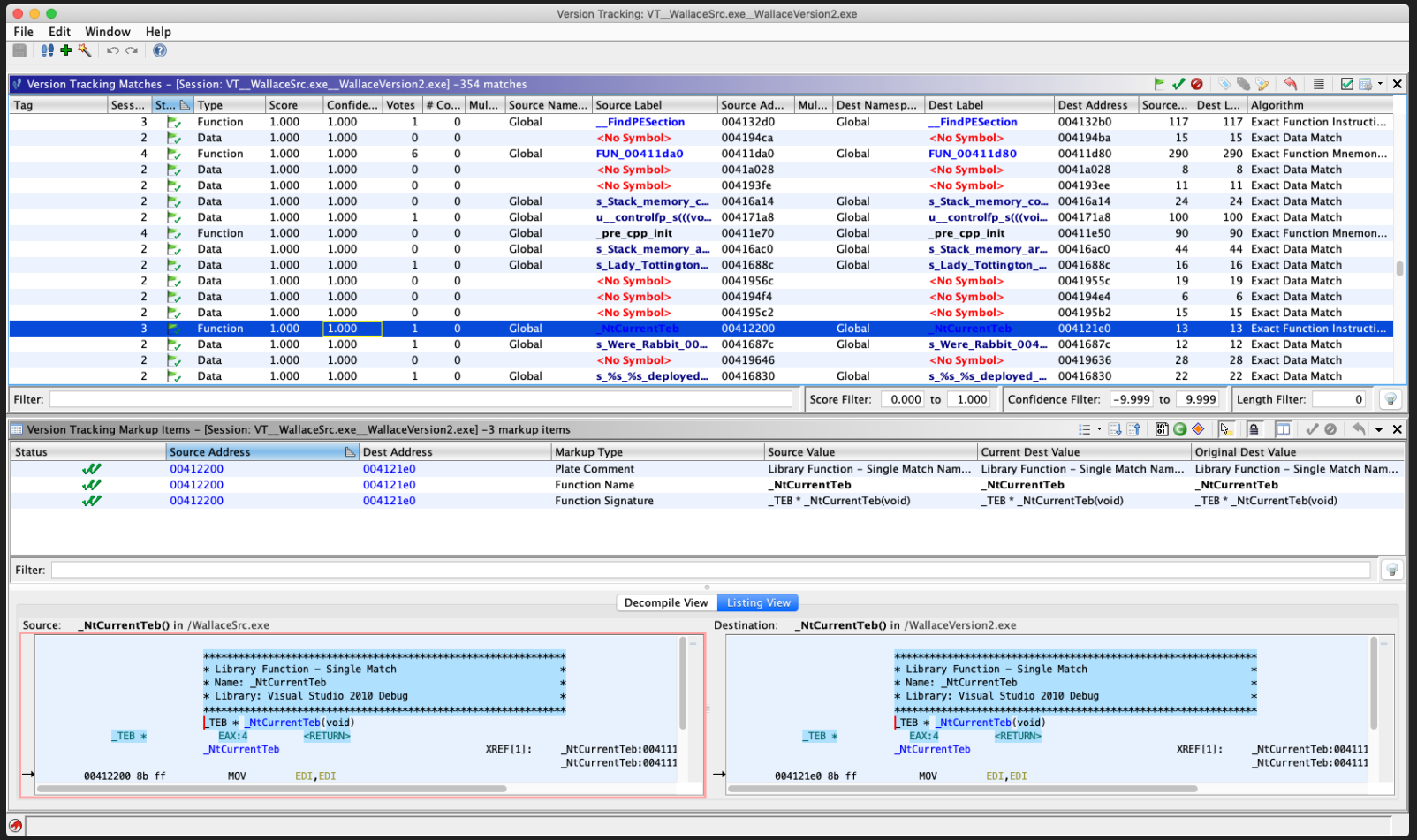

```
0040149e 48 83 ec 08      SUB         RSP,0x8
004014a2 b8 00 00         MOV         EAX,0x0
         00 00
004014a7 e8 4d ff         CALL        skip
         ff ff
004014ac 48 85 c0         TEST        RAX,RAX
004014af 75 6e            JNZ         LAB_0040151f
004014b1 48 8b 05         MOV         RAX,qword ptr [.bss:stdin]
         90 22 20 00
004014b8 48 39 05         CMP         qword ptr [.bss:infile],RAX
         a9 22 20 00
004014bf 75 14            JNZ         LAB_004014d5
004014c1 bf d5 25         MOV         EDI=>.rodata:s_Error:_Premat
         40 00
004014c6 e8 45 f6         CALL        .plt:puts
```

Version Tracker

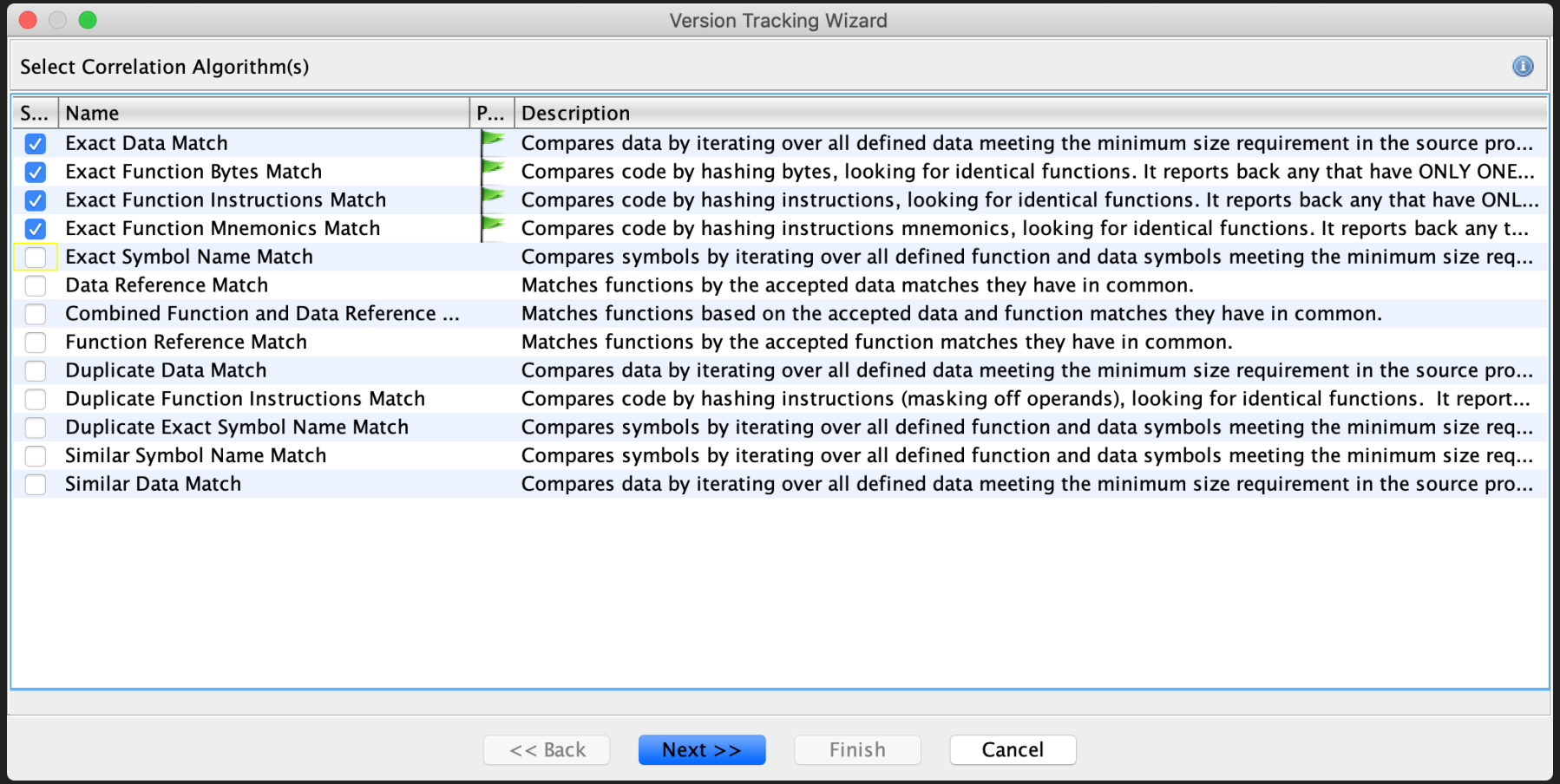
- Feature for porting RE symbols, annotations, etc. between incrementally updated versions of the same program

- In our experience, not well suited for quick 1-day discovery in patch analysis
  - Use Diaphora or BinDiff for this purpose

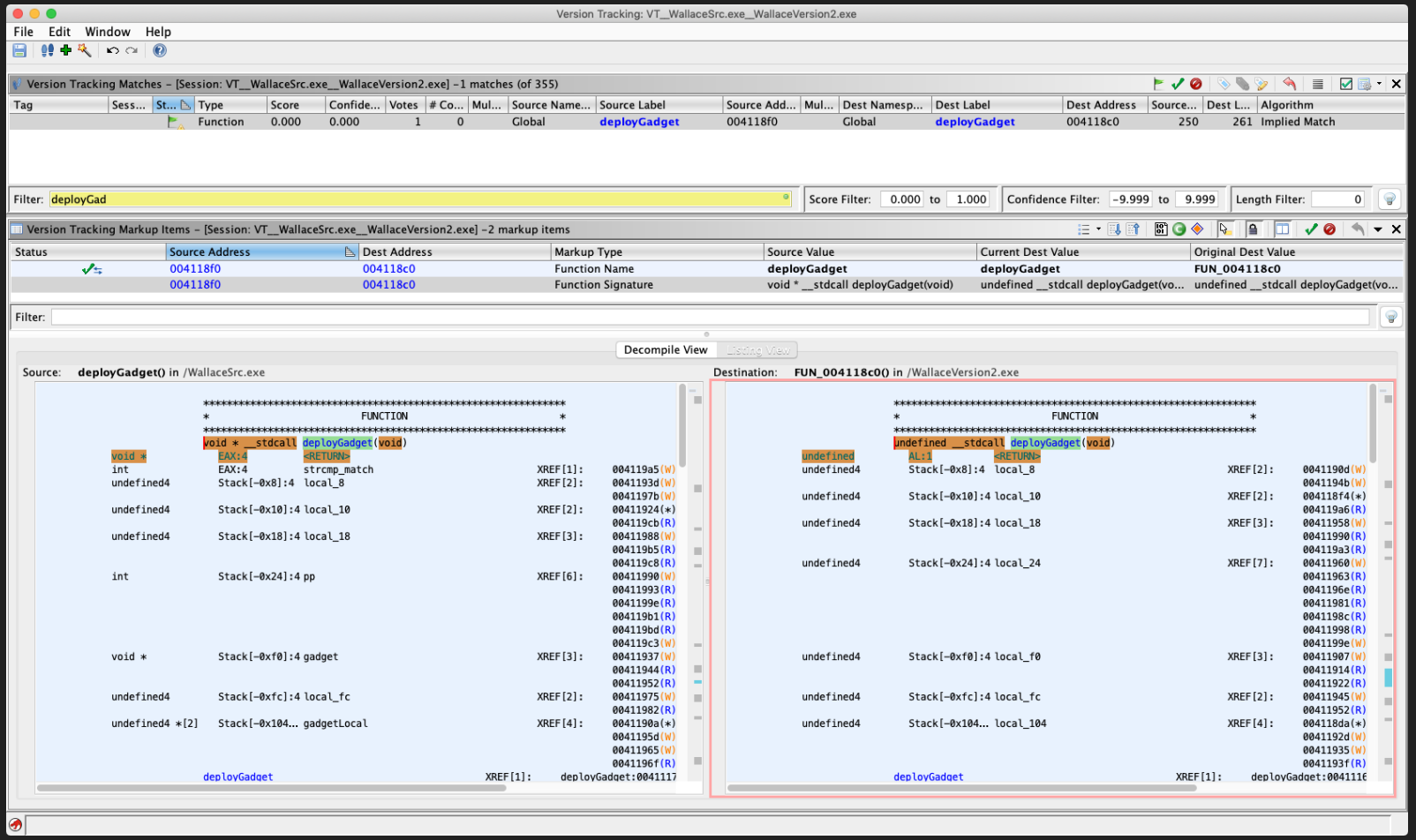
Version Tracker - Overview



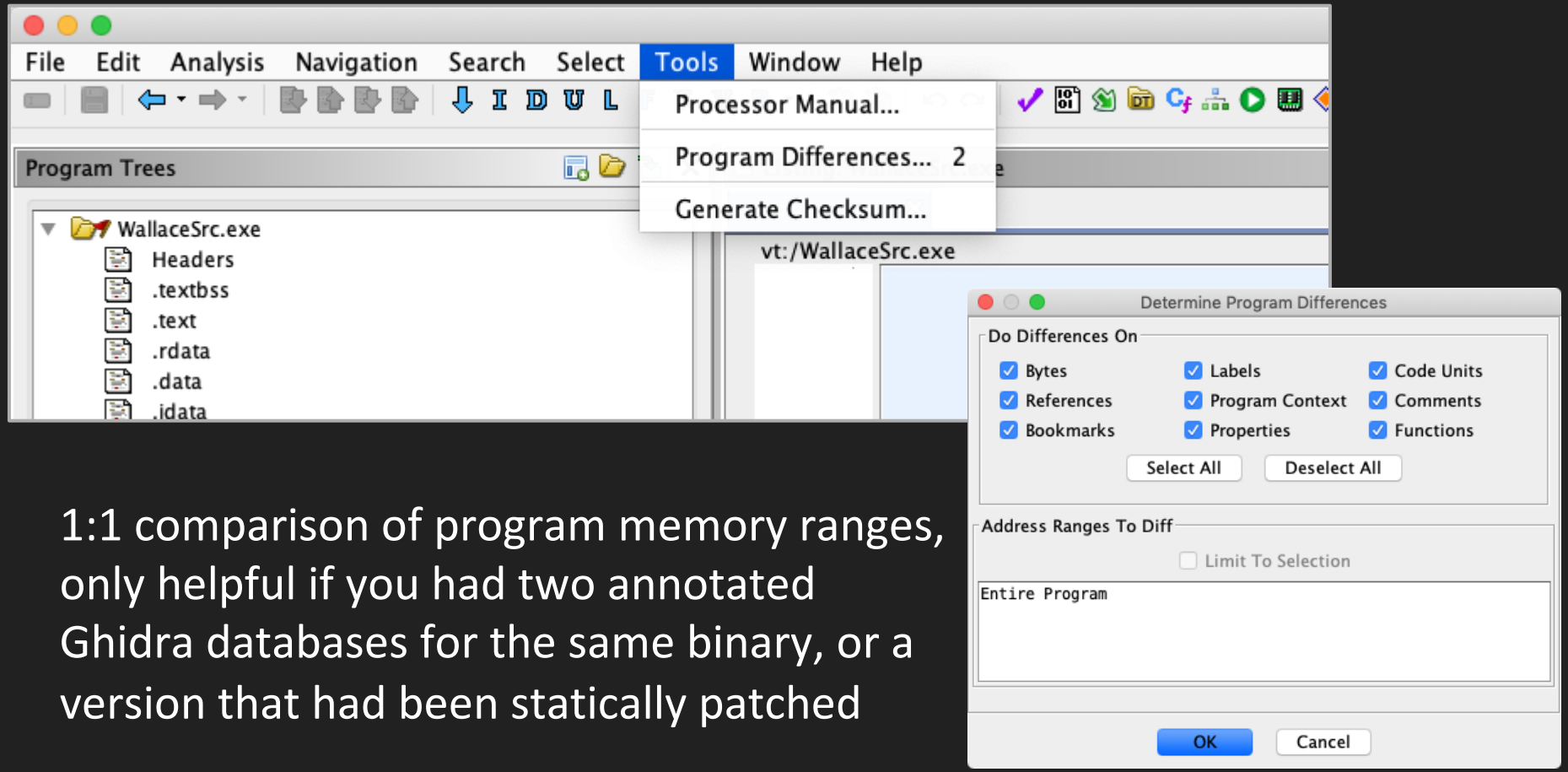
Version Tracker - Selecting Correlation Algorithms



Version Tracker - Function Name Ported

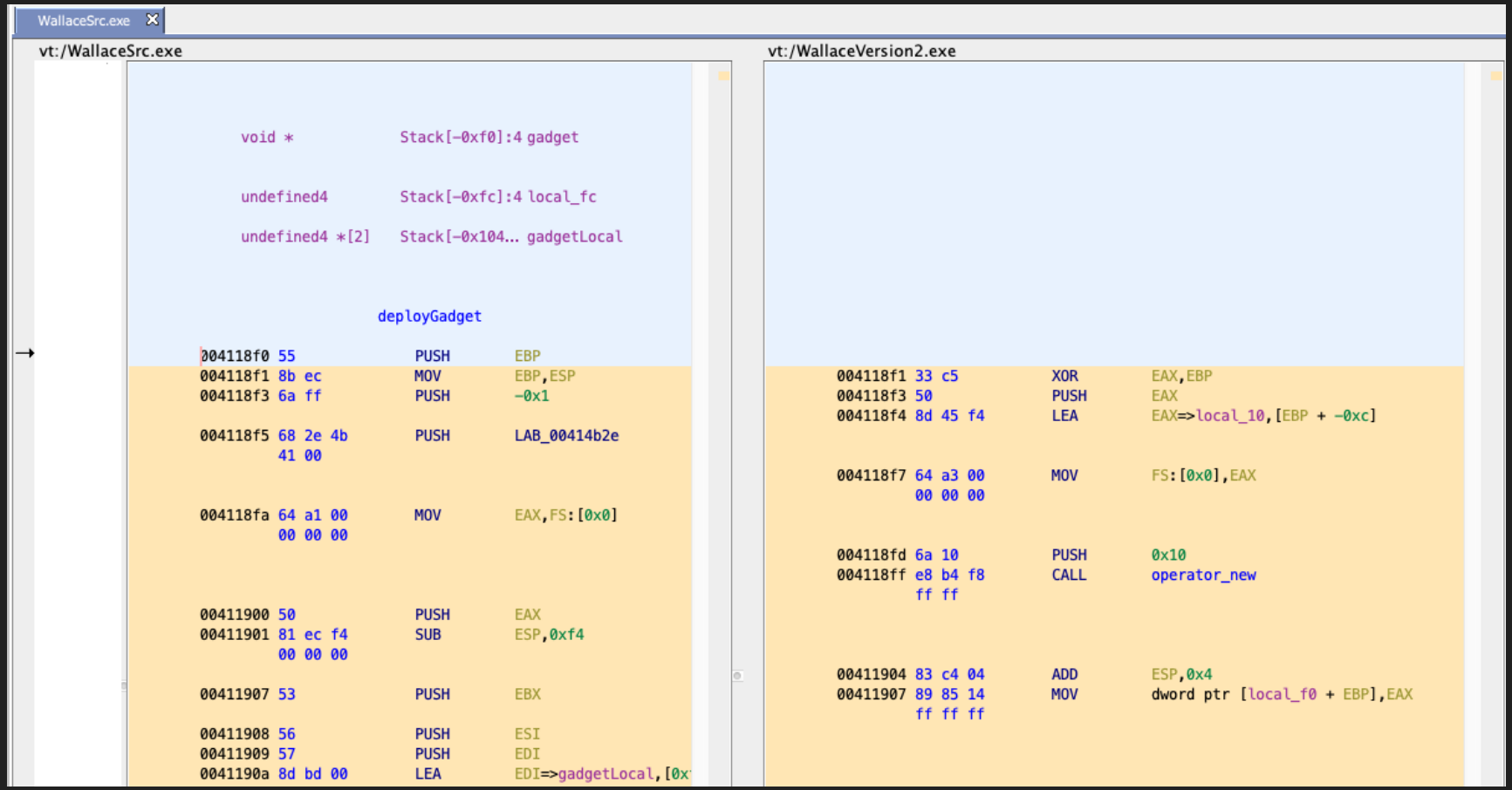


Program Differences

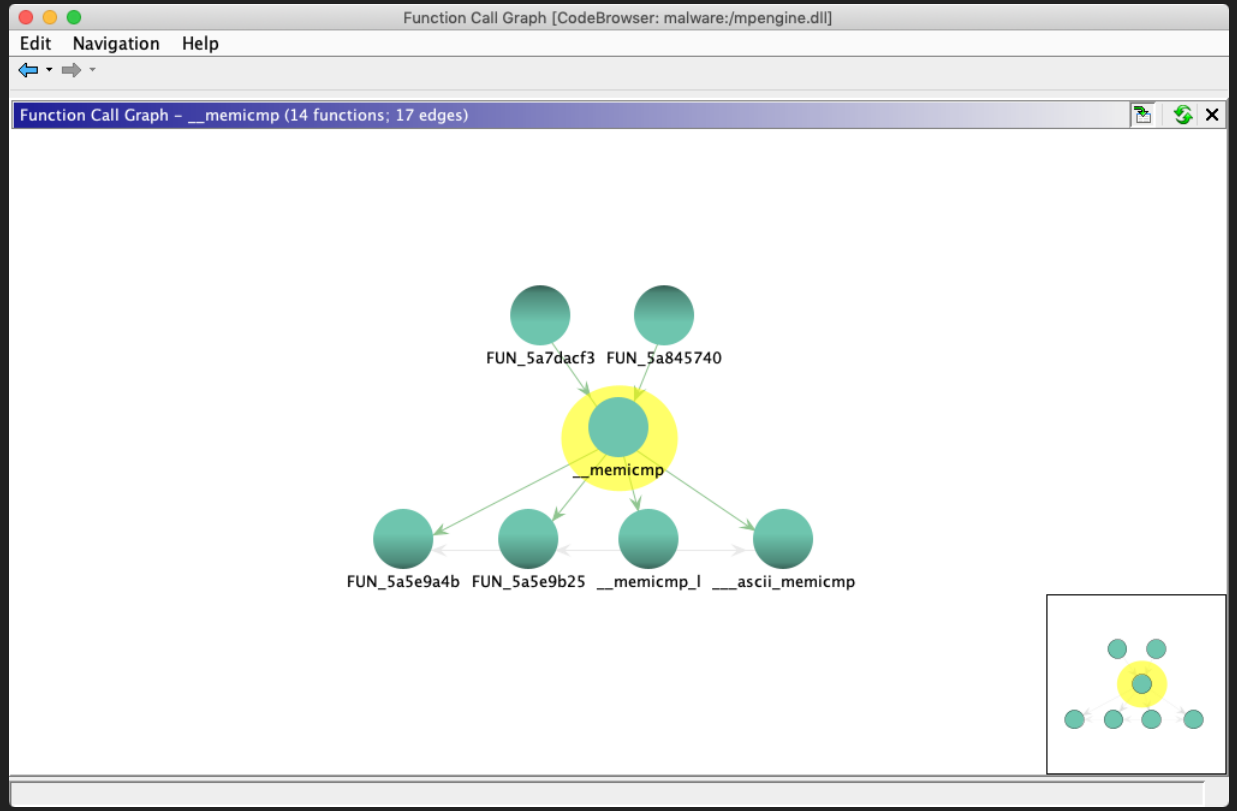


1:1 comparison of program memory ranges, only helpful if you had two annotated Ghidra databases for the same binary, or a version that had been statically patched

Program Differences

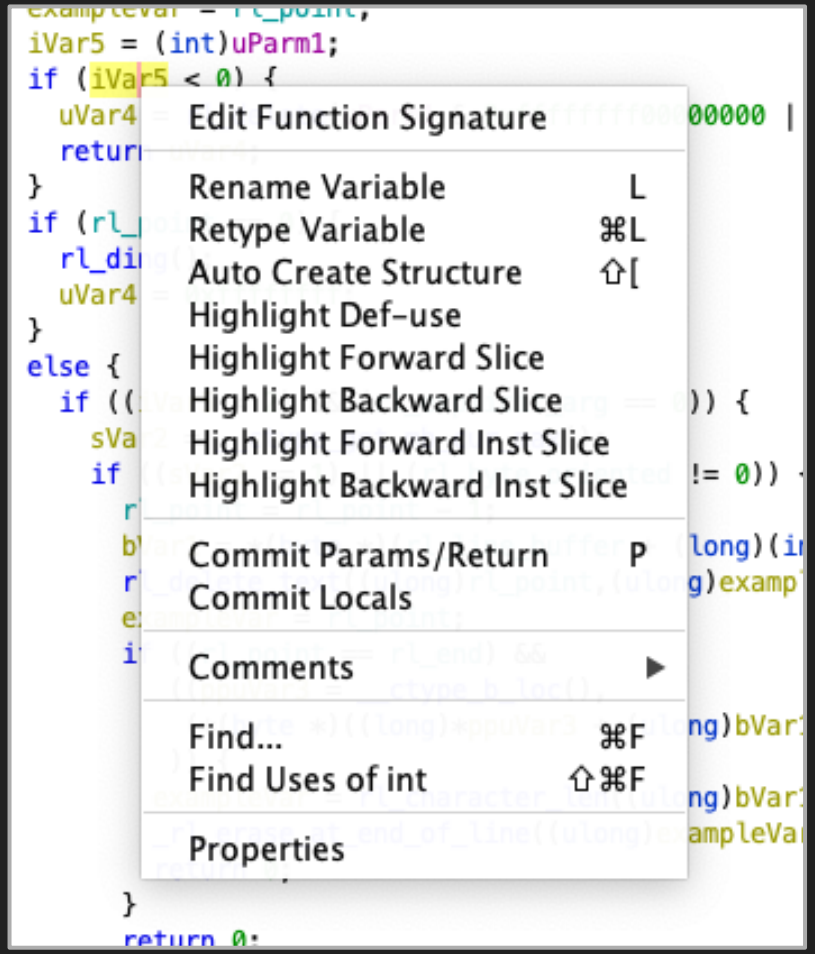


Function Call Graph



Decompiler Slicing

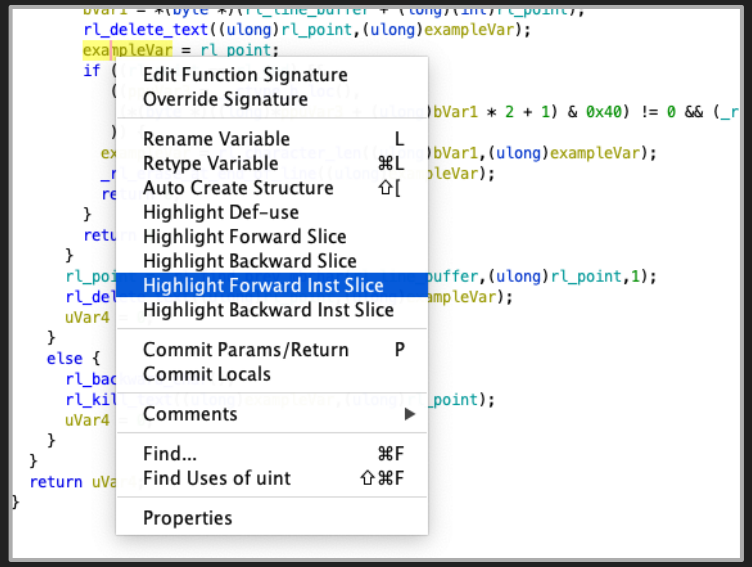
- Decompiler calculates data flow during auto-analysis
- Users can right-click on variables to view def-use chain and forward / backward slices
- Menu bar “Select” options allow users to trace flows to / from given points



Decompiler Slicing

```
undefined8 _rl_rubout_char(ulong uParm1)

{
  byte bVar1;
  size_t sVar2;
  ushort **ppuVar3;
  undefined8 uVar4;
  int iVar5;
  uint exampleVar;

  exampleVar = rl_point;
  iVar5 = (int)uParm1;
  if (iVar5 < 0) {
    uVar4 = rl_delete(uParm1 & 0xffffffff00000000 | (ulong)(uint)-iVar5);
    return uVar4;
  }
  if (rl_point == 0) {
    rl_ding();
    uVar4 = 0xffffffff;
  }
  else {
    if ((iVar5 < 2) && (rl_explicit_arg == 0)) {
      sVar2 = __ctype_get_mb_cur_max();
      if ((sVar2 == 1) || (rl_byte_oriented != 0)) {
        rl_point = rl_point - 1;
        bVar1 = *(byte *)(rl_line_buffer + (long)(int)rl_point);
        rl_delete_text((ulong)rl_point,(ulong)exampleVar);
        exampleVar = rl_point;
        if ((rl_point == rl_end) &&
           ((ppuVar3 = __ctype_b_loc(),
            (*(byte *)((long)*ppuVar3 + (ulong)bVar1 * 2 + 1) & 0x40) != 0 &&
           )) {
          exampleVar = rl_character_len((ulong)bVar1,(ulong)exampleVar);
          _rl_erase_at_end_of_line((ulong)exampleVar);
          return 0;
        }
        return 0;
      }
      rl_point = _rl_find_prev_mbchar(rl_line_buffer,(ulong)rl_point,1);
      rl_delete_text((ulong)rl_point,(ulong)exampleVar);
      uVar4 = 0;
    }
    else {
      rl_backward_char();
      rl_kill_text((ulong)exampleVar,(ulong)rl_point);
      uVar4 = 0;
    }
  }
  return uVar4;
}
```

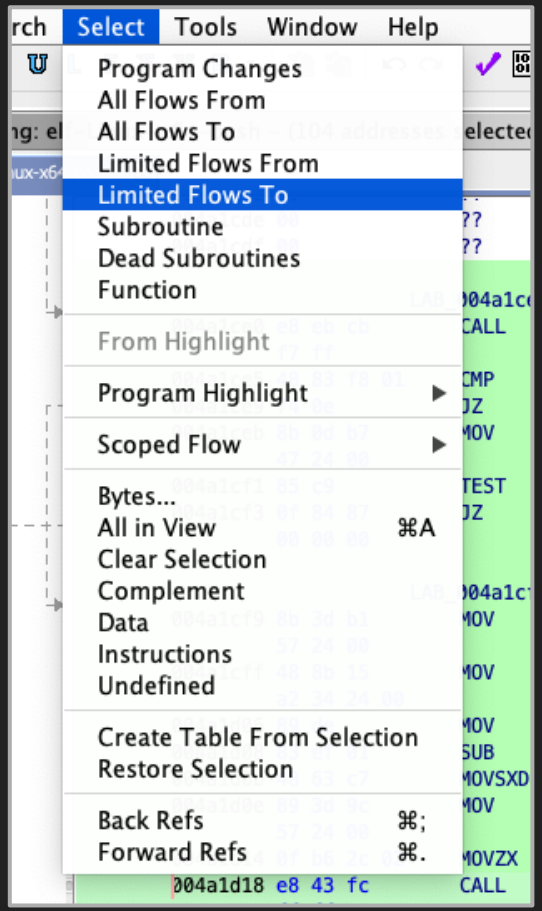
Middle Click



Highlight Forward Inst Slice

```
  rl_delete_text((ulong)rl_point,(ulong)exampleVar);
  exampleVar = rl_point;
  if ((rl_point == rl_end) &&
     ((ppuVar3 = __ctype_b_loc(),
      (*(byte *)((long)*ppuVar3 + (ulong)bVar1 * 2 + 1) & 0x40) != 0 && (_rl
     )) {
    exampleVar = rl_character_len((ulong)bVar1,(ulong)exampleVar);
    _rl_erase_at_end_of_line((ulong)exampleVar);
    return 0;
  }
  return 0;
}
```

Decompiler Slicing




```
undefined8 _rl_rubout_char(ulong uParm1)

{
  byte bVar1;
  size_t sVar2;
  ushort **ppuVar3;
  undefined8 uVar4;
  int iVar5;
  uint exampleVar;

  exampleVar = rl_point;
  iVar5 = (int)uParm1;
  if (iVar5 < 0) {
    uVar4 = rl_delete(uParm1 & 0xffffffff00000000 | (ulong)(uint)-iVar5);
    return uVar4;
  }
  if (rl_point == 0) {
    rl_ding();
    uVar4 = 0xffffffff;
  }
  else {
    if ((iVar5 < 2) && (rl_explicit_arg == 0)) {
      sVar2 = __ctype_get_mb_cur_max();
      if ((sVar2 == 1) || (rl_byte_oriented != 0)) {
        rl_point = rl_point - 1;
        bVar1 = *(byte *)(rl_line_buffer + (long)(int)rl_point);
        rl_delete_text((ulong)rl_point,(ulong)exampleVar);
        exampleVar = rl_point;
        if ((rl_point == rl_end) &&
           ((ppuVar3 = __ctype_b_loc(),
            (*(byte *)((long)*ppuVar3 + (ulong)bVar1 * 2 + 1) & 0x40) != 0 && (_rl_last_c_pos != 0))
           )) {
          exampleVar = rl_character_len((ulong)bVar1,(ulong)exampleVar);
          _rl_erase_at_end_of_line((ulong)exampleVar);
          return 0;
        }
        return 0;
      }
      rl_point = _rl_find_prev_mbchar(rl_line_buffer,(ulong)rl_point,1);
      rl_delete_text((ulong)rl_point,(ulong)exampleVar);
      uVar4 = 0;
    }
    else {
      rl_backward_char();
      rl_kill_text((ulong)exampleVar,(ulong)rl_point);
      uVar4 = 0;
    }
  }
  return uVar4;
}
```

Outline

1. Intro
2. Interactive Exercises
    a. Manual Static Analysis
    b. Scripting Ghidra
3. P-Code & SLEIGH
4. Discussion
5. Conclusion

Scripting With Ghidra

- Available in Java (natively) and Python (via Jython)
- Can be run with interactive GUI or in headless mode
- Ghidra comes with 230+ scripts pre-installed
  - Educational examples
  - Code patching
  - Import / export
  - Analysis enhancements
    - Windows, Mac, Linux, VXWorks
    - PE, ELF, Mach-O, COFF
    - x86, MIPS, ARM/THUMB, 8051, etc…

Ghidra APIs

FlatProgramAPI

- Simple “flattened” API for Ghidra scripting
- Programmatic access to common tasks
  - query / modify / iterate / create / delete - functions / data / instructions / comments
- Mostly doesn’t require the use of Java objects
- Stable

Ghidra Program API

- More complex rich API for deeper scripting
- Object-oriented (Program, Memory, Function, Instruction, etc…)
- Utility functions help with common scripting tasks
- UI scripting / interactivity
- Prone to change between versions

API Highlights

Rich Scripting Interface

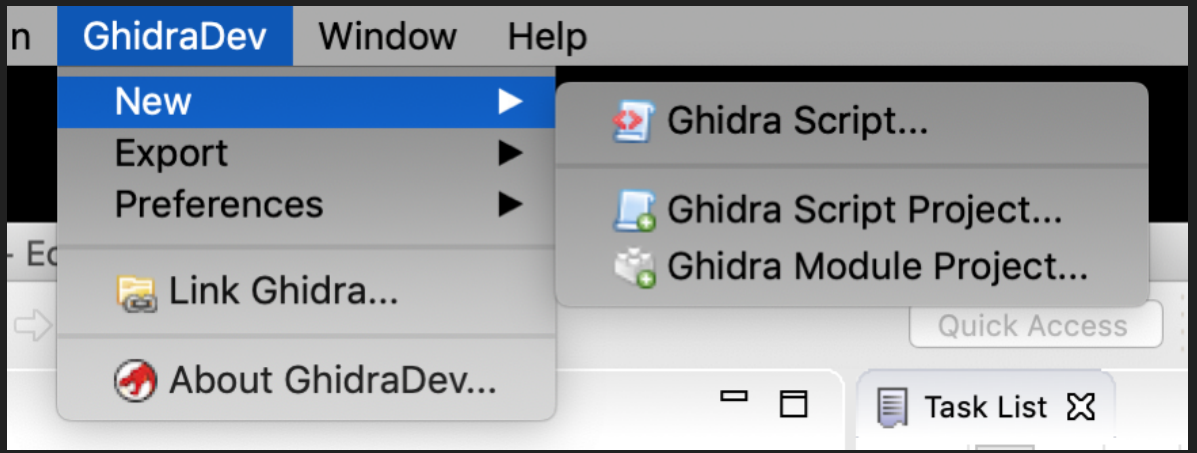
- Programmatic access to binary file formats
- P-code interaction
- Decompiler API
- C header parsing
- Interface for graphing (implementation not included)
- Cyclomatic complexity

Common Utilities Included

- UI windows
- Assembly
- Data serialization
- String manipulation
- Hashing
- Search / byte matching
- XML utilities

Eclipse Integration

- Ghidra has built-in Eclipse integration, via its “GhidraDev” Eclipse plugin
- **NOTE:** For these exercises, we’ll be using Ghidra’s built-in basic editor - don’t waste time trying to get Eclipse set up during this workshop



Scripting Demos

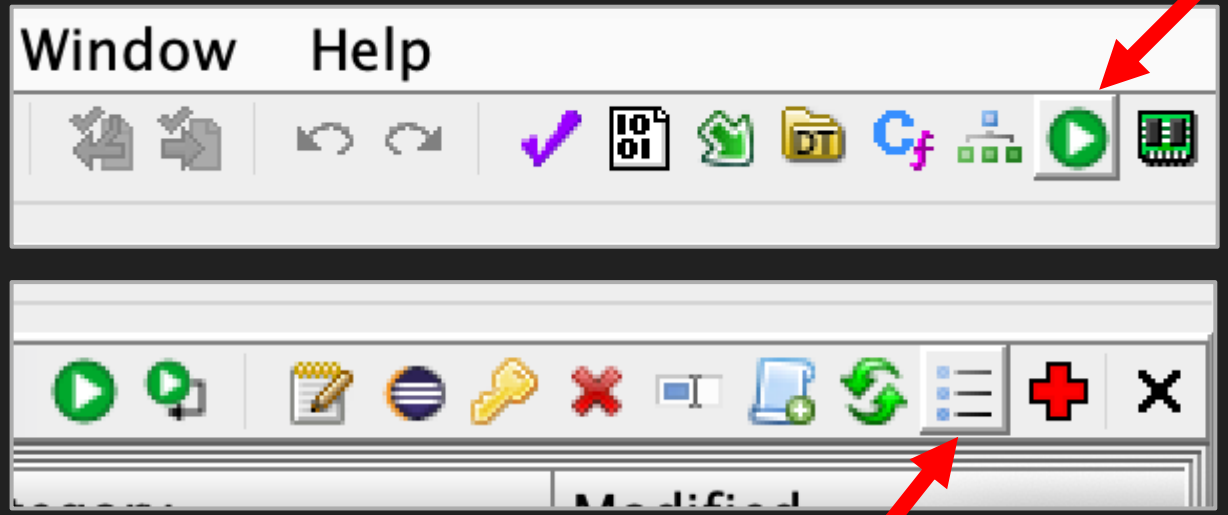
1. Hello World
    a. Java
    b. Python
2. Crypto Constant Search
3. Cyclomatic Complexity
4. Xor with Python

Importing Demo Scripts

Click the “Display Script Manager” button to open the Script Manager Window

Importing Demo Scripts

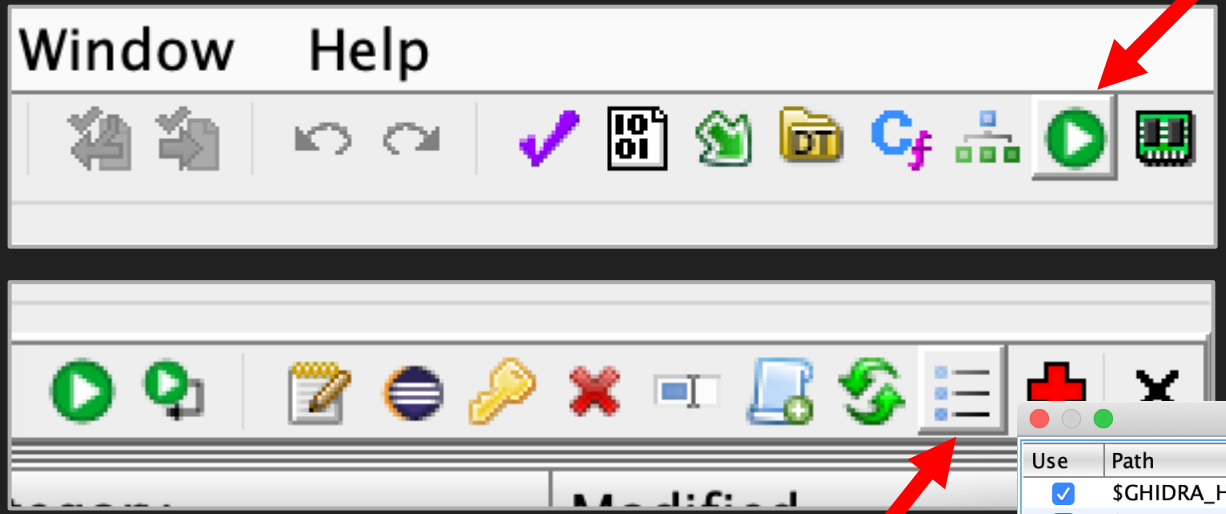
Click the “Display Script Manager” button to open the Script Manager Window



Click the “Script Directories” button on the Script Click the “Display Script Manager” button to open the Script Manager Window

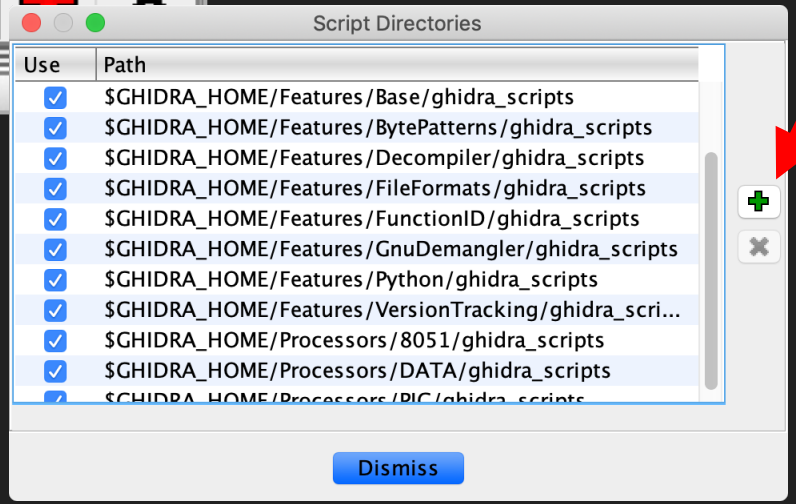
Importing Demo Scripts

Click the “Display Script Manager” button to open the Script Manager Window



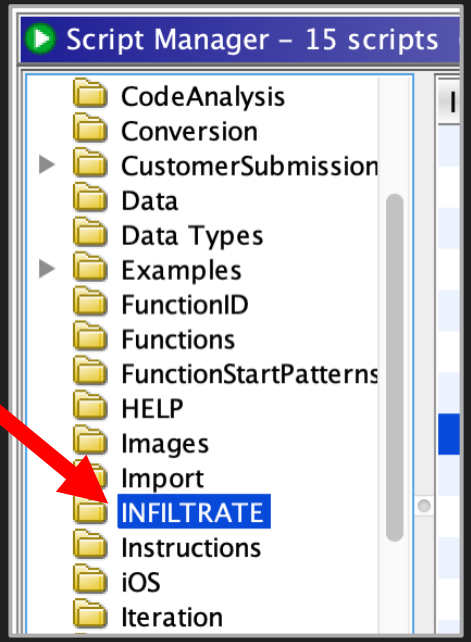
Click the green plus to open the file chooser, choose the script directory

Click the “Script Directories” button on the Script Click the “Display Script Manager” button to open the Script Manager Window



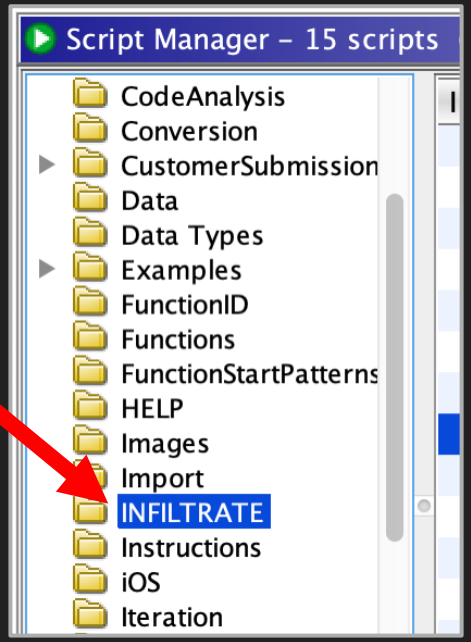
Running Script Demos

Find the “INFILTRATE” folder in the script manager

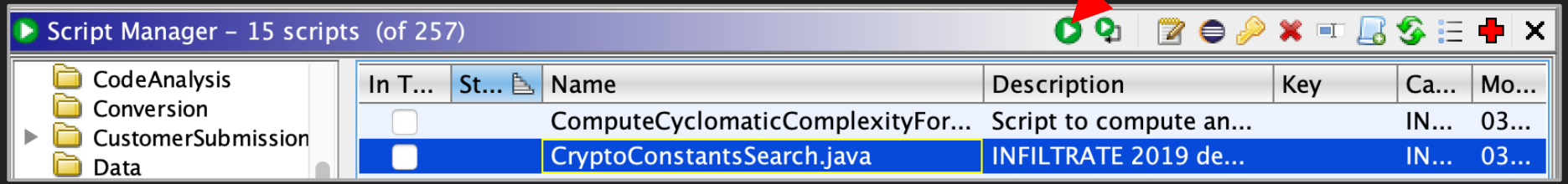


Running Script Demos

Find the “INFILTRATE” folder in the script manager

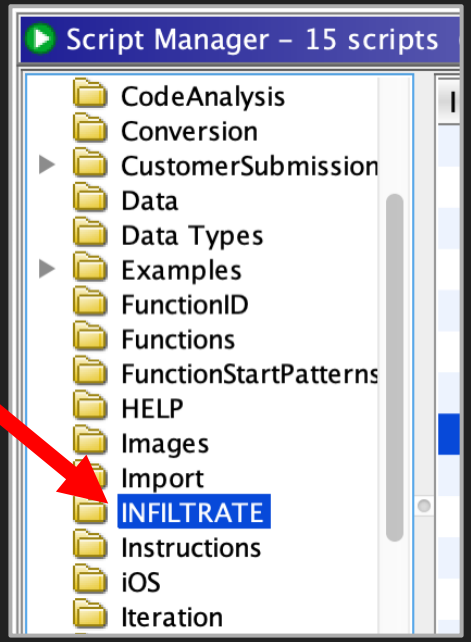


Choose a script and click “Run Script” to run

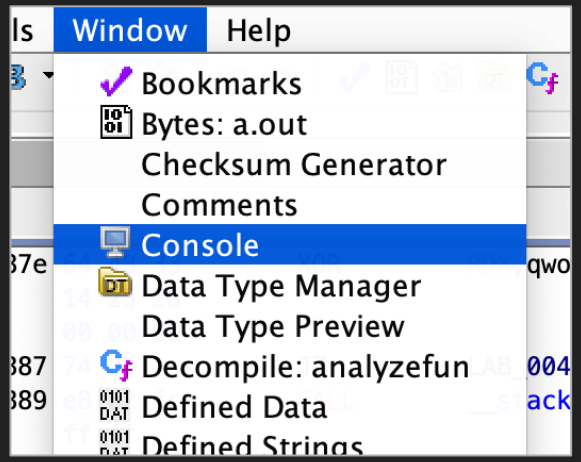


Running Script Demos

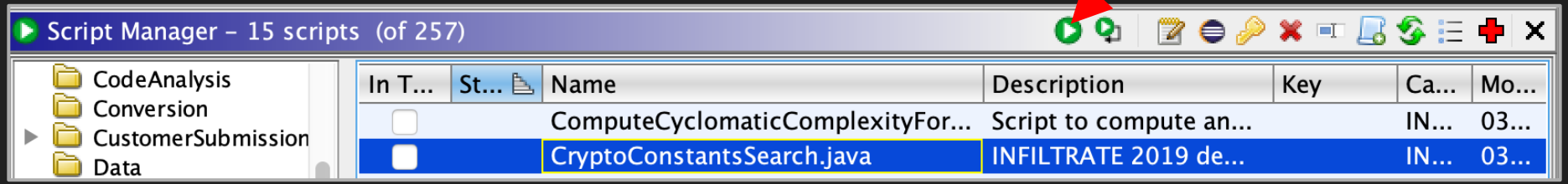
Find the “INFILTRATE” folder in the script manager



Make sure the “Console” window is open if you want to see output



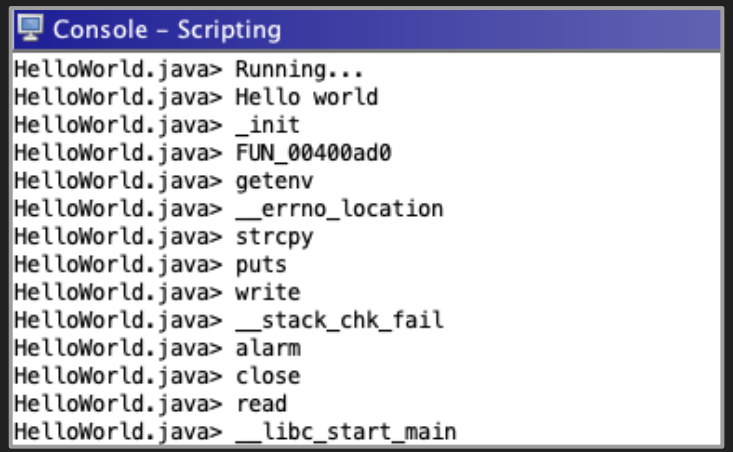
Choose a script and click “Run Script” to run



DEMO: Hello World

- A simple script to print “Hello World” and then iterate over all functions in the program, printing out their names

- `HelloWorld.java`
- `HelloWorld.py`

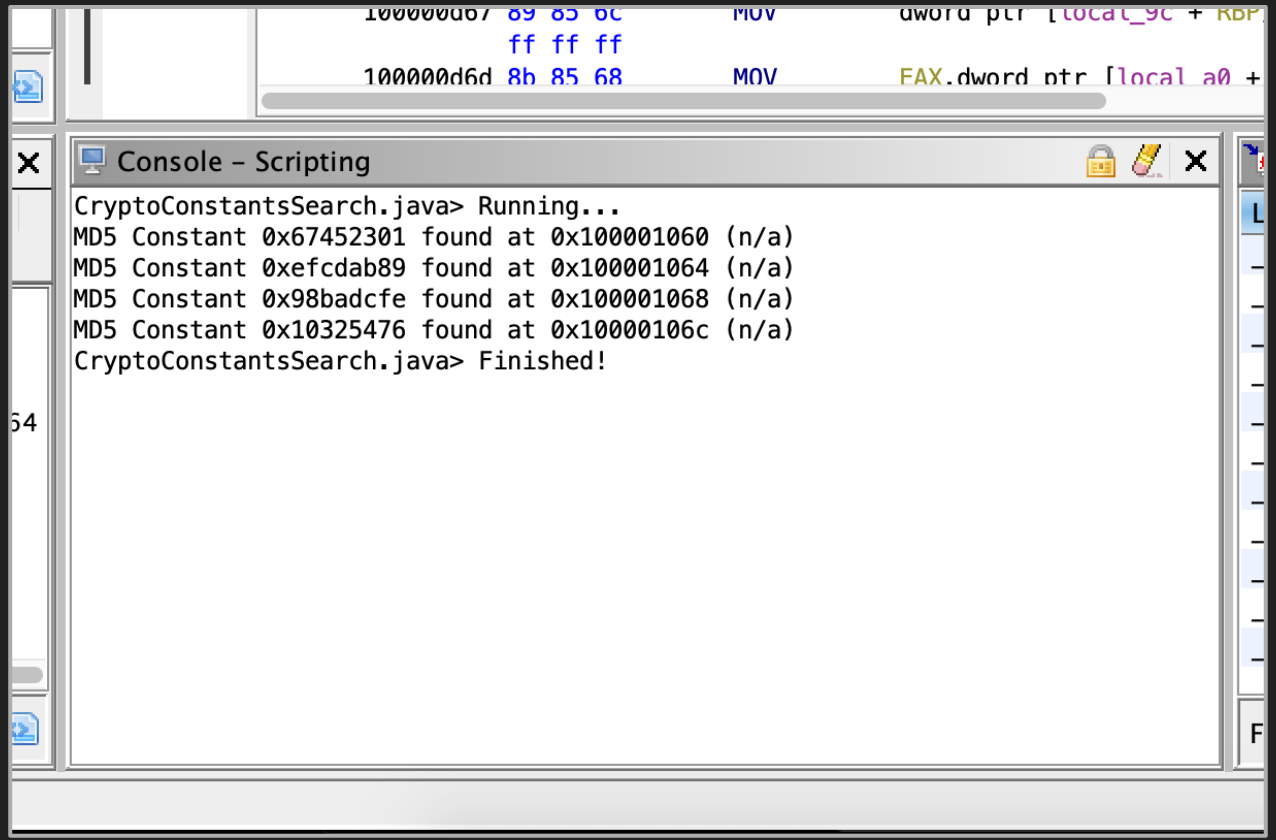



```java
import ghidra.app.script.GhidraScript;

public class HelloWorld extends GhidraScript {

    public void run() throws Exception {
    println("Hello world");

    Function currentFunc = getFirstFunction();
    while (currentFunc != null){
        println(currentFunc.getName());
        currentFunc = getFunctionAfter(currentFunc);
    }
    }
}
```

DEMO: Crypto Search

- Find MD5 constants present in a binary, report offset and function name
- Take binary endianness into account automatically without user specification
- `CryptoConstantsSearch.java`

DEMO: Crypto Search

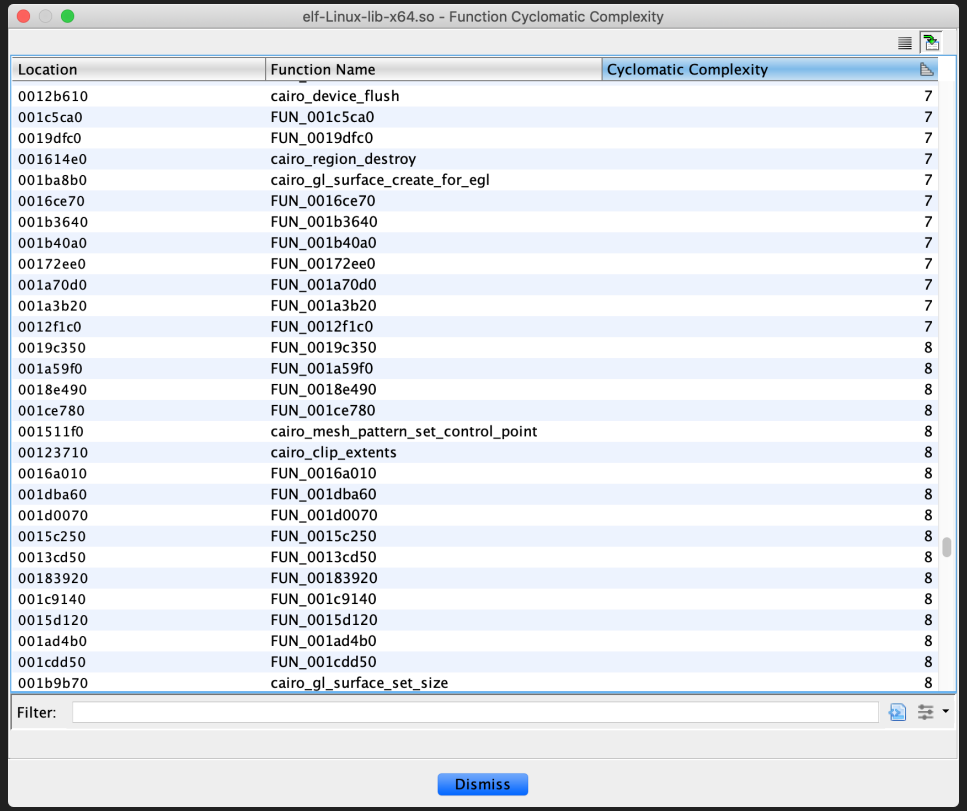


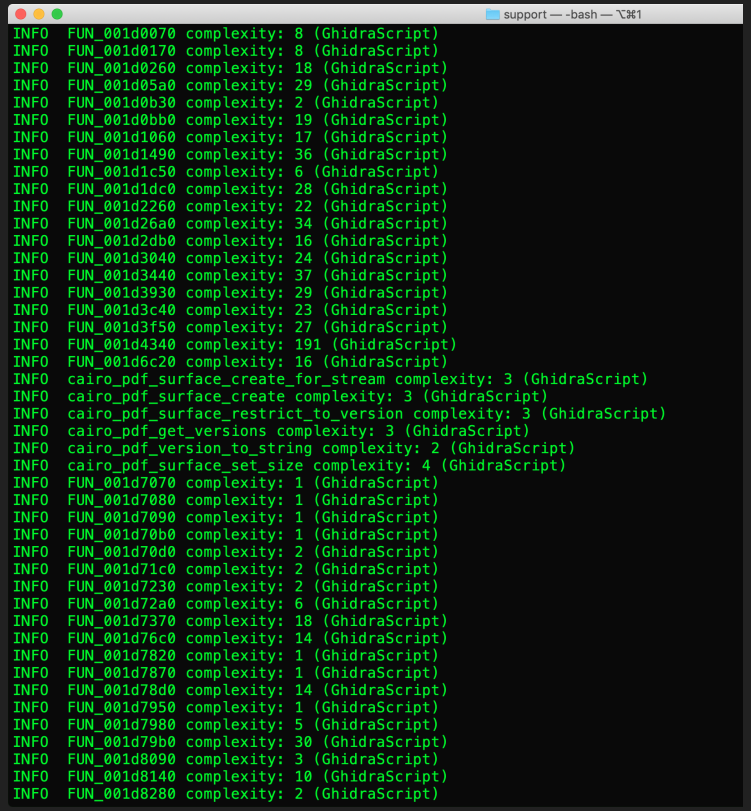
DEMO: Calculating Cyclomatic Complexity

- Leverage Ghidra's API for calculating cyclomatic complexity* to easily analyze a whole program
- Pop a GUI window if running interactively, else print to the terminal in headless mode
- `ComputeCyclomaticComplexityForAllFunctions.java`

* cyclomatic complexity is a measure of the number of unique paths which may be taken through a given function. It can be helpful in finding complex functions likely to have vulnerabilities, e.g, complex parsing routines or state machines.

DEMO: Calculating Cyclomatic Complexity





DEMO: Python Scripting

- Ghidra can run Python in a Jython environment, the Ghidra Java API is exposed to Python
- This script takes a user address selection and XORs the bytes in place with `0x41`
- `XorMemoryScript.py`

DEMO: Python Scripting

7 addresses selected)

100001ce8 00	??	00h	
100001ce9 00	??	00h	
100001cea 24	??	24h	$
100001ceb 2f	??	2Fh	/
100001cec 22	??	22h	"
100001ced 33	??	33h	3
100001cee 38	??	38h	8
100001cef 31	??	31h	1
100001cf0 35	??	35h	5
100001cf1 24	??	24h	$
100001cf2 25	??	25h	%
100001cf3 61	??	61h	a
100001cf4 2c	??	2Ch	,
100001cf5 24	??	24h	$
100001cf6 32	??	32h	2
100001cf7 32	??	32h	2
100001cf8 20	??	20h	
100001cf9 26	??	26h	&
100001cfa 24	??	24h	$
100001cfb 00	??	00h	
100001cfc 00	??	00h	

7 addresses selected)

100001ce8 00	??	00h	
100001ce9 00	??	00h	
100001cea 65	??	65h	e
100001ceb 6e	??	6Eh	n
100001cec 63	??	63h	c
100001ced 72	??	72h	r
100001cee 79	??	79h	y
100001cef 70	??	70h	p
100001cf0 74	??	74h	t
100001cf1 65	??	65h	e
100001cf2 64	??	64h	d
100001cf3 20	??	20h	
100001cf4 6d	??	6Dh	m
100001cf5 65	??	65h	e
100001cf6 73	??	73h	s
100001cf7 73	??	73h	s
100001cf8 61	??	61h	a
100001cf9 67	??	67h	g
100001cfa 65	??	65h	e
100001cfb 00	??	00h	
100001cfc 00	??	00h	

Outline

1. Intro
2. Interactive Exercises
    a. Manual Static Analysis
    b. Scripting Ghidra
3. P-Code & SLEIGH
4. Discussion
5. Conclusion

P-Code

- Ghidra's intermediate language
  - Dates back to at least 2005 according to documentation
- Code for different processors can be lifted into p-code, data-flow analysis and decompilation can then run over the p-code
- Pseudo-assembly, represents lifted instructions as small atomic operations without side-effects
- Built-in floating point support

```
MOVSXD      RAX,EDX
                    (register, 0x0, 8) = INT_SEXT (register, 0x10, 4)
LEA         RAX,[RAX + RAX*0x4]
                    (unique, 0x660, 8) = INT_MULT (register, 0x0, 8), (const, 0x4, 8)
                    (unique, 0x680, 8) = INT_ADD (register, 0x0, 8), (unique, 0x660, 8)
                    (register, 0x0, 8) = COPY (unique, 0x680, 8)
```

P-Code Design

- The language is machine independent.
- The language is designed to model general purpose processors.
- Instructions operate on user defined registers and address spaces.
- All data is manipulated explicitly. Instructions have no indirect effects.
- Individual p-code operations mirror typical processor tasks and concepts.

Quoted from `docs/languages/html/sleigh.html`

Processor to p-code modeling:

- RAM → *address space*
- Register → *varnode*
- Instruction → *operation*

```
SCASB.REPNE   RDI
                        $U22d0:1 = INT_EQUAL RCX, 0:8
                        CBRANCH *[ram]0x401548:8, $U22d0
                        RCX = INT_SUB RCX, 1:8
                        $U1d90:8 = COPY RDI
                        $U1da0:8 = INT_ADD RDI, 1:8
                        $U1db0:8 = INT_ZEXT DF
                        $U1dc0:8 = INT_MULT 2:8, $U1db0
                        RDI = INT_SUB $U1da0, $U1dc0
                        $U1de0:1 = LOAD ram($U1d90)
                        CF = INT_LESS AL, $U1de0
                        $U1de0:1 = LOAD ram($U1d90)
                        OF = INT_SBORROW AL, $U1de0
                        $U1de0:1 = LOAD ram($U1d90)
                        $Uac60:1 = INT_SUB AL, $U1de0
                        SF = INT_SLESS $Uac60, 0:1
                        ZF = INT_EQUAL $Uac60, 0:1
                        $U22f0:1 = BOOL_NEGATE ZF
                        CBRANCH *[ram]0x401546:8, $U22f0
NOT           RCX
                        RCX = INT_NEGATE RCX
```

Category	P-Code Operations
Data Moving	`COPY, LOAD, STORE`
Arithmetic	`INT_ADD, INT_SUB, INT_CARRY, INT_SCARRY, INT_SBORROW, INT_2COMP, INT_MULT, INT_DIV, INT_SDIV, INT_REM, INT_SREM`
Logical	`INT_NEGATE, INT_XOR, INT_AND, INT_OR, INT_LEFT, INT_RIGHT, INT_SRIGHT`
Int Comparison	`INT_EQUAL, INT_NOTEQUAL, INT_SLESS, INT_SLESSEQUAL, INT_LESS, INT_LESSEQUAL`
Boolean	`BOOL_NEGATE, BOOL_XOR, BOOL_AND, BOOL_OR`
Floating Point	`FLOAT_ADD, FLOAT_SUB, FLOAT_MULT, FLOAT_DIV, FLOAT_NEG, FLOAT_ABS, FLOAT_SQRT, FLOAT_NAN`
FP Compare	`FLOAT_EQUAL, FLOAT_NOTEQUAL, FLOAT_LESS, FLOAT_LESSEQUAL`
FP Conversion	`INT2FLOAT, FLOAT2FLOAT, TRUNC, CEIL, FLOOR, ROUND`
Branching	`BRANCH, CBRANCH, BRANCHIND, CALL, CALLIND, RETURN`
Extension / Truncation	`INT_ZEXT, INT_SEXT, PIECE, SUBPIECE`

DEMO: Source-Sink Analysis

- Use Ghidra p-code and the decompiler's analysis to identify the sources for values passed to function calls of interest (`malloc`), particularly function calls accepting user input
- *Solving* for the actual arguments requires a *solver*, this is a much simpler analysis that can empower a human analyst to hone in on interesting calls

- Start at the varnode for each argument to `malloc`, then trace back to the p-code operation that it's derived from
  - From there, recursively trace back the p-code operation(s) defining the varnode(s) that define that the inputs to those operations
- At function call sites, trace in, and find how the returned values are derived
- When a parameter is used, trace back to call sites which set the parameter

SLEIGH

- Ghidra’s language for describing instruction sets to facilitate RE
- **Disassembly:** translate bit-encoded machine instructions into human-readable assembly language statements
- **Semantics:** translate machine instructions into p-code instructions (one-to-many) for decompilation, analysis, and emulation
- Based off of SLED (Specification Language for Encoding and Decoding), a 1997 academic IL

SLEIGH Example - x86 `JMP rel8`

Raw bytes: `0xEB 0x03`

x86 instruction: `JMP $+5`

SLEIGH Example - x86 `JMP rel8`

Raw bytes: `0xEB 0x03`

x86 instruction: `JMP $+5`

SLEIGH Example - x86 JMP rel8

Raw bytes: `0xEB 0x03`
x86 instruction: `JMP $+5`

SLEIGH:

```
:JMP rel8 is vexMode=0 & byte=0xeb; rel8 {
   goto rel8;
}
```

```
00401f16 eb 03          JMP         LAB_00401f1b
                                    BRANCH *[ram]0x401f1b:8
```

SLEIGH Example - x86 JMP rel8

Raw bytes: 0xEB 0x03

x86 instruction: JMP $+5

```
rel8: reloc is simm8 [ reloc=inst_next+simm8; ] {
    export *[ram]:$(SIZE) reloc;
}
```

SLEIGH:

```
:JMP rel8 is vexMode=0 & byte=0xeb; rel8 {
   goto rel8;
}
```

```
00401f16 eb 03          JMP         LAB_00401f1b
                                    BRANCH *[ram]0x401f1b:8
```

SLEIGH Example - x86 JMP rel8

Raw bytes: 0xEB 0x03

x86 instruction: JMP $+5

```
rel8: reloc is simm8 [ reloc=inst_next+simm8; ] {
    export *[ram]:$(SIZE) reloc;
}
```

SLEIGH:

```
:JMP rel8 is vexMode=0 & byte=0xeb; rel8 {
   goto rel8;
}
```

SLEIGH Example - x86 JMP rel8

Raw bytes: 0xEB 0x03

x86 instruction: JMP $+5

```
rel8: reloc is simm8 [ reloc=inst_next+simm8; ] {
   export *[ram]:$(SIZE) reloc;
}
```

SLEIGH:

```
:JMP rel8 is vexMode=0 & byte=0xeb; rel8 {
   goto rel8;
}
```

SLEIGH Example - x86 `XOR AL, imm8`

Raw bytes: `0x34 0x57`

x86 instruction: `XOR AL, 0x57`

SLEIGH Example - x86 `XOR AL, imm8`

Raw bytes: `0x34 0x57`

x86 instruction: `XOR AL, 0x57`

```
34 57       XOR     AL,0x57
                      CF = COPY 0:1
                      OF = COPY 0:1
                      AL = INT_XOR AL, 0x57:1
                      SF = INT_SLESS AL, 0:1
                      ZF = INT_EQUAL AL, 0:1
```

SLEIGH Example - x86 `XOR AL, imm8`

Raw bytes: `0x34 0x57`

x86 instruction: `XOR AL, 0x57`

SLEIGH:

```
:XOR AL,imm8 is vexMode=0 & byte=0x34; AL & imm8 {
   logicalflags();
   AL = AL ^ imm8;
   resultflags( AL );
}
```

```
34 57       XOR     AL,0x57
                      CF = COPY 0:1
                      OF = COPY 0:1
                      AL = INT_XOR AL, 0x57:1
                      SF = INT_SLESS AL, 0:1
                      ZF = INT_EQUAL AL, 0:1
```

SLEIGH Example - x86 XOR AL, imm8

Raw bytes: 0x34 0x57

x86 instruction: XOR AL, 0x57

```
macro logicalflags() {
 CF = 0;
 OF = 0;
}
```

SLEIGH:

```
:XOR AL,imm8 is vexMode=0 & byte=0x34; AL & imm8 {
   logicalflags();
   AL = AL ^ imm8;
   resultflags( AL );
}

 macro resultflags(result) {
  SF = result s < 0;
  ZF = result == 0;
  # PF, AF not implemented
 }
```

```
34 57      XOR     AL,0x57
                     CF = COPY 0:1
                     OF = COPY 0:1
                     AL = INT_XOR AL, 0x57:1
                     SF = INT_SLESS AL, 0:1
                     ZF = INT_EQUAL AL, 0:1
```

SLEIGH Example - x86 XOR AL, imm8

Raw bytes: 0x34 0x57

x86 instruction: XOR AL, 0x57

SLEIGH:

```
:XOR AL,imm8 is vexMode=0 & byte=0x34; AL & imm8 {
   logicalflags();
   AL = AL ^ imm8;
   resultflags( AL );
}
```

```
macro logicalflags() {
 CF = 0;
 OF = 0;
}
```

```
macro resultflags(result) {
 SF = result s < 0;
 ZF = result == 0;
 # PF, AF not implemented
}
```

```
34 57      XOR     AL,0x57
                   CF = COPY 0:1
                   OF = COPY 0:1
                   AL = INT_XOR AL, 0x57:1
                   SF = INT_SLESS AL, 0:1
                   ZF = INT_EQUAL AL, 0:1
```

SLEIGH Example - x86 XOR AL, imm8

Raw bytes: 0x34 0x57

x86 instruction: XOR AL, 0x57

SLEIGH:

```
:XOR AL,imm8 is vexMode=0 & byte=0x34; AL & imm8 {
   logicalflags();
   AL = AL ^ imm8;
   resultflags( AL );
}
```

```
macro logicalflags() {
 CF = 0;
 OF = 0;
}
```

```
macro resultflags(result) {
 SF = result s < 0;
 ZF = result == 0;
 # PF, AF not implemented
}
```

```
34 57      XOR     AL,0x57
                   CF = COPY 0:1
                   OF = COPY 0:1
                   AL = INT_XOR AL, 0x57:1
                   SF = INT_SLESS AL, 0:1
                   ZF = INT_EQUAL AL, 0:1
```

SLEIGH Example - x86 XOR AL, imm8

Raw bytes: 0x34 0x57

x86 instruction: XOR AL, 0x57

```
macro logicalflags() {
 CF = 0;
 OF = 0;
}
```

SLEIGH:

```
:XOR AL,imm8 is vexMode=0 & byte=0x34; AL & imm8 {
   logicalflags();
   AL = AL ^ imm8;
   resultflags( AL );
}
```

```
 macro resultflags(result) {
  SF = result s < 0;
  ZF = result == 0;
  # PF, AF not implemented
 }
```

```
34 57      XOR     AL,0x57
                   CF = COPY 0:1
                   OF = COPY 0:1
                   AL = INT_XOR AL, 0x57:1
                   SF = INT_SLESS AL, 0:1
                   ZF = INT_EQUAL AL, 0:1
```

SLEIGH Example - x86 XOR AL, imm8

Raw bytes: 0x34 0x57

x86 instruction: XOR AL, 0x57

SLEIGH:

```
:XOR AL,imm8 is vexMode=0 & byte=0x34; AL & imm8 {
   logicalflags();
   AL = AL ^ imm8;
   resultflags( AL );
}
```

```
macro logicalflags() {
 CF = 0;
 OF = 0;
}
```

```
macro resultflags(result) {
 SF = result s < 0;
 ZF = result == 0;
 # PF, AF not implemented
}
```

```
34 57      XOR     AL,0x57
```

```
CF = COPY 0:1
OF = COPY 0:1
AL = INT_XOR AL, 0x57:1
SF = INT_SLESS AL, 0:1
ZF = INT_EQUAL AL, 0:1
```

SLEIGH Example - x86 `RDTSC`

Raw bytes: `0x0F 0x31`

x86 instruction: `RDTSC`

SLEIGH Example - x86 RDTSC

Raw bytes: `0x0F 0x31`

x86 instruction: `RDTSC`

```
0f 31     RDTSC
```

```
$U9c60:8 = CALLOTHER "rdtsc"
EDX = SUBPIECE $U9c60, 4:4
EAX = SUBPIECE $U9c60, 0:4
```

SLEIGH Example - x86 RDTSC

Raw bytes: `0x0F 0x31`

x86 instruction: `RDTSC`

SLEIGH:

```
:RDTSC is vexMode=0 & byte=0xf; byte=0x31 {
    tmp:8 = rdtsc();
    EDX = tmp(4);
    EAX = tmp(0);
}
```

```
0f 31     RDTSC
```

```
$U9c60:8 = CALLOTHER "rdtsc"
EDX = SUBPIECE $U9c60, 4:4
EAX = SUBPIECE $U9c60, 0:4
```

SLEIGH Example - x86 RDTSC

Raw bytes: `0x0F 0x31`

x86 instruction: `RDTSC`

SLEIGH:

```
:RDTSC is vexMode=0 & byte=0xf; byte=0x31 {
    tmp:8 = rdtsc();
    EDX = tmp(4);
    EAX = tmp(0);
}
```

```
define pcodeop rdtsc;
```

```
0f 31     RDTSC
```

```
$U9c60:8 = CALLOTHER "rdtsc"
EDX = SUBPIECE $U9c60, 4:4
EAX = SUBPIECE $U9c60, 0:4
```

SLEIGH Example - x86 RDTSC

Raw bytes: `0x0F 0x31`

x86 instruction: `RDTSC`

SLEIGH:

```
:RDTSC is vexMode=0 & byte=0xf; byte=0x31 {
    tmp:8 = rdtsc();
    EDX = tmp(4);
    EAX = tmp(0);
}
```

```
define pcodeop rdtsc;
```

```
0f 31     RDTSC
```

```
$U9c60:8 = CALLOTHER "rdtsc"
EDX = SUBPIECE $U9c60, 4:4
EAX = SUBPIECE $U9c60, 0:4
```

SLEIGH Example - x86 RDTSC

Raw bytes: `0x0F 0x31`

x86 instruction: `RDTSC`

SLEIGH:

```
:RDTSC is vexMode=0 & byte=0xf; byte=0x31 {
    tmp:8 = rdtsc();
    EDX = tmp(4);
    EAX = tmp(0);
}
```

```
define pcodeop rdtsc;
```

```
0f 31     RDTSC
```

```
$U9c60:8 = CALLOTHER "rdtsc"
EDX = SUBPIECE $U9c60, 4:4
EAX = SUBPIECE $U9c60, 0:4
```

SLEIGH Example - x86 RDTSC

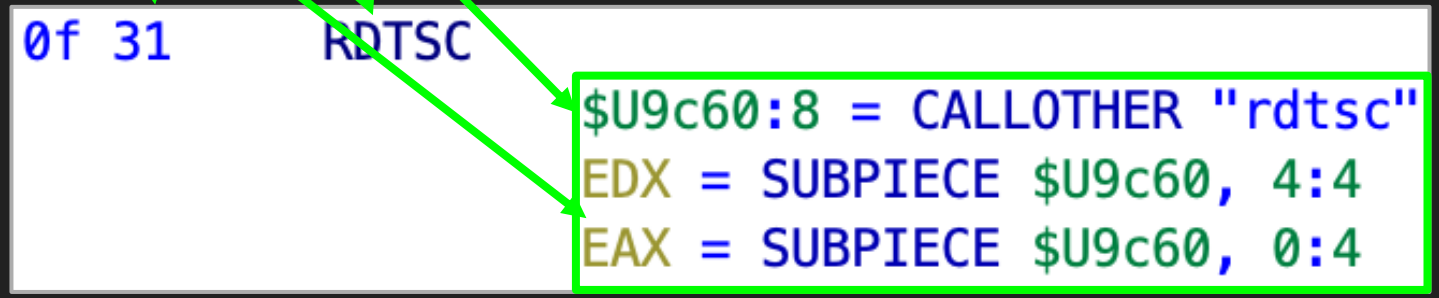
Raw bytes: `0x0F 0x31`

x86 instruction: `RDTSC`

SLEIGH:

```
:RDTSC is vexMode=0 & byte=0xf; byte=0x31 {
    tmp:8 = rdtsc();
    EDX = tmp(4);
    EAX = tmp(0);
}
```

```
define pcodeop rdtsc;
```

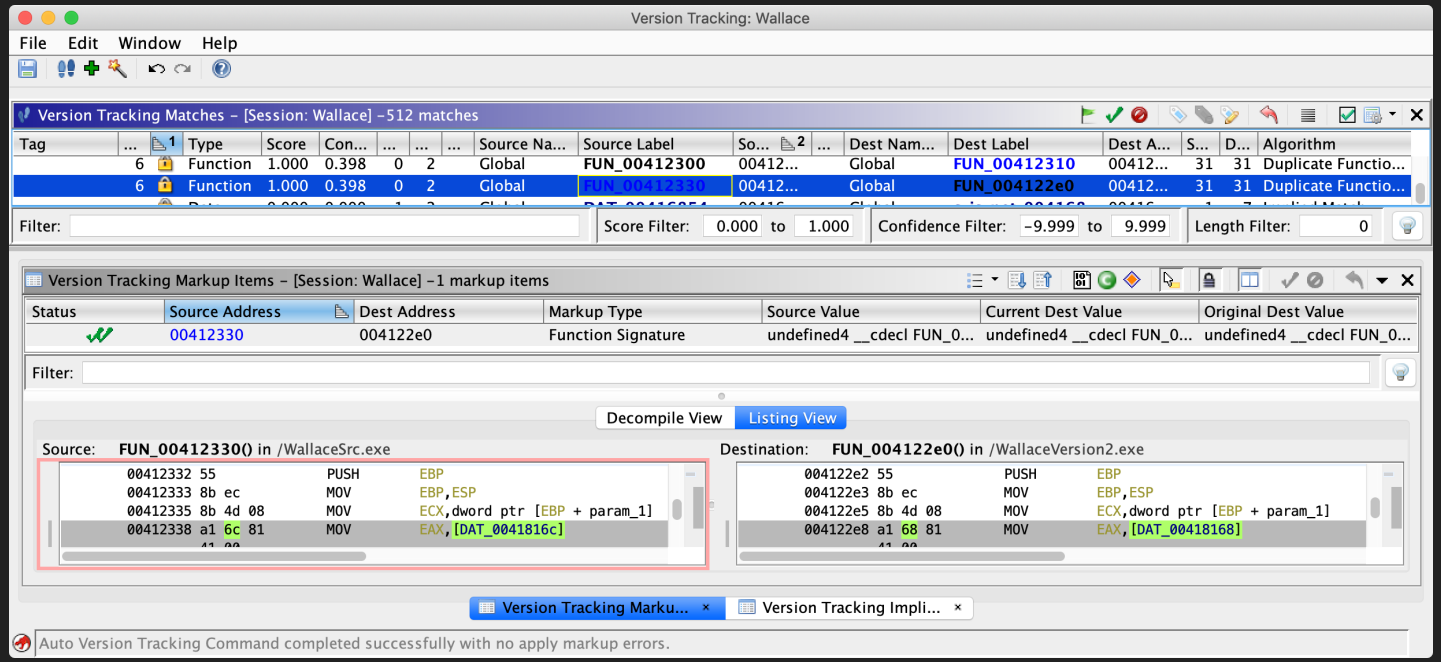


Outline

1. Intro
2. Interactive Exercises
    a. Manual Static Analysis
    b. Scripting Ghidra
3. P-Code & SLEIGH
4. Discussion
5. Conclusion

Other Major Features

- Multi-user collaboration
- Version tracking
- Extensibility for new binary loaders and architectures
- Headless mode
- GhidraDev Eclipse plugin
- Debugger promised at RSA
- Undocumented p-code emulator



IDA vs Binary Ninja vs Ghidra

IDA

- Maturity
- Windows support
- Decompiler
- Existing corpus of powerful plugins
- Debugger
- Support for paid customers
- Well tested
- Industry standard

Binary Ninja

- Innovation and modern design
- Program analysis features (SSA)
- Multi-level IL
- Rich API
- Embeddable
- Python-native scripting
- Clean modern UI
- Community

Ghidra

- Maturity
- Embedded support
- Decompiler
- Massive API
- Documentation
- Breath of features
- Collaboration
- Version tracking
- Price and open source extensibility

Decompiler - IDA Hex-Rays vs Ghidra

IDA Hex-Rays

- Optional add-on for IDA for IDA
- Microcode-based
- Supports limited architectures
- Better built-in support for Windows
- Variables, data, and functions can be xrefed from decompiler
- Variables can be mapped
- Variable representation can be changed in the decompiler (decimal, hex, char immediate, etc)
- Click to highlight

Ghidra Decompiler Decompiler

- Deeply integrated with Ghidra
- P-code based
- Supports all architectures
- No way to xref from decompiler
- Produces fewer `goto` statements and seemingly more idiomatic C
- Built in program analysis features, e.g., slicing and data flow
- Variables cannot be mapped
- Variable representation cannot be changed in the decompiler
- *Middle click* to highlight

ILs - Binary Ninja vs Ghidra

Binary Ninja

- Multi-level: LLIL, MLIL, forthcoming HLIL
- Machine consumable and human readable
- SSA form
- Designed in light of years of program analysis research
- Feels nicer to work with
- Deferred flag calculations

Ghidra

- Single level p-code, but can be enhanced by decompiler analysis
- Designed for machine consumption first, not human readability
- Uses SSA during decompilation, but raw p-code is not SSA
- Design origins based off of program analysis research from 20+ years ago

ILs - Binary Ninja vs Ghidra

```
phase_4:
rsp = rsp - 0x18
rcx = rsp + 0xc {var_c}
rdx = rsp + 8 {var_10}
esi = 0x4025cf  {"%d %d"}
eax = 0
call(__isoc99_sscanf)
if (eax != 2) then 7 @ 0x401035 else 9 @ 0x401033
```

LLIL

```
phase_4:
int32_t* rcx = &var_c
int32_t* rdx = &var_10
rax = __isoc99_sscanf(arg1, 0x4025cf, rdx, rcx)  {"%d %d"}
if (rax.eax != 2) then 4 @ 0x401035 else 6 @ 0x401033
```

MLIL

```
0040100c - phase_4
undefined phase_4()
        undefined           AL:1           <RETURN>
        undefined4          Stack[-0xc]:4  local_c
        undefined4          Stack[-0x10]:4 local_10
        CF = INT_LESS RSP, 24:8
        OF = INT_SBORROW RSP, 24:8
        RSP = INT_SUB RSP, 24:8
        SF = INT_SLESS RSP, 0:8
        ZF = INT_EQUAL RSP, 0:8
        $U770:8 = INT_ADD 12:8, RSP
        RCX = COPY $U770
        $U770:8 = INT_ADD 8:8, RSP
        RDX = COPY $U770
        RSI = COPY 0x4025cf:8
        RAX = COPY 0:8
        RSP = INT_SUB RSP, 8:8
        STORE ram(RSP), 0x401029:8
        CALL *[ram]0x400bf0:8
        CF = INT_LESS EAX, 2:4
        OF = INT_SBORROW EAX, 2:4
        $U58c0:4 = INT_SUB EAX, 2:4
        SF = INT_SLESS $U58c0, 0:4
        ZF = INT_EQUAL $U58c0, 0:4
        $U2080:1 = BOOL_NEGATE ZF
        CBRANCH *[ram]0x401035:8, $U2080
```

We Like Ghidra For...

- Scripting reverse engineering
- Firmware / embedded systems analysis
- Analysis of software that Hex-Rays can’t decompile
- Collaborative long-term professional RE
- Professional reversing at a computer workstation with multiple monitors, full keyboard with function keys, mouse with middle click and scroll wheel, etc...

Scripting - Java vs Python

- Java will catch errors at compile time, Ghidra's API is highly object-oriented and benefits from this
- Complex Python scripts feel like binding together Java API calls with Python control flow and syntax
- **Recommended workflow:** prototype and experiment with APIs / objects in the Python interpreter, write final code in Java

Jython

For Reverse Engineers, By Reverse Engineers

- Built for multi-monitor use
- “Moving ants” highlight on control flow graphs
- Configurable “tool” views
- Hotkeys mappable to actions and scripts
- Right click > “extract and import”
- Processor manual integration
- Undo button
- Import directly from zip file
- Snapshot views
- Configurable listings
- Version tracker
- Project-based multi-binary RE
- `F1` to open help on whatever the mouse is pointing at
- File System browser
- Highly configurable assembly code listing
- Data flow analysis built into UI
- Embedded image detection
- Search for matching instructions
- Unique windows
  - Checksum Generator
  - Disassembled View
  - Data Type Preview
  - Function Tags
  - Symbol tree

Contributing to Ghidra

- Ghidra code is available on Github
  - Apache License 2.0

- NSA has been responsive to community questions and bug reports posted on Github
  - The agency has already published two minor-version updates

Official site: ghidra-sre.org
Open source: github.com/NationalSecurityAgency/ghidra
github.com/NationalSecurityAgency/ghidra-data

Outline

1. Intro
2. Interactive Exercises
   a. Manual Static Analysis
   b. Scripting Ghidra
3. P-Code & SLEIGH
4. Discussion
5. Conclusion

Conclusion

@0xAlexei / @0xJeremy

Ghidra is a powerful binary reverse engineering tool built by the US National Security Agency

- For reverse engineers, by reverse engineers
- Interactive and headless scripting
- Built for program analysis
- We have yet to see what the community will do with Ghidra, this is just the beginning

Acknowledgements:

- NSA's Ghidra team
- Rob Joyce
- Rolf Rolles
- Evan Jensen
- Sophia d'Antoine
- Dave Aitel and the INFILTRATE team

Demo material: github.com/0xAlexei/INFILTRATE2019

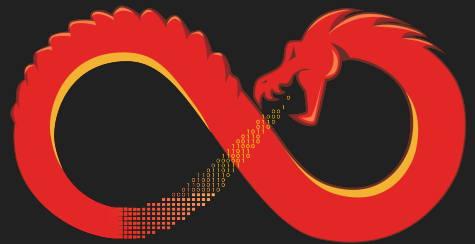
IDA keybindings: github.com/JeremyBlackthorne/Ghidra-Keybindings

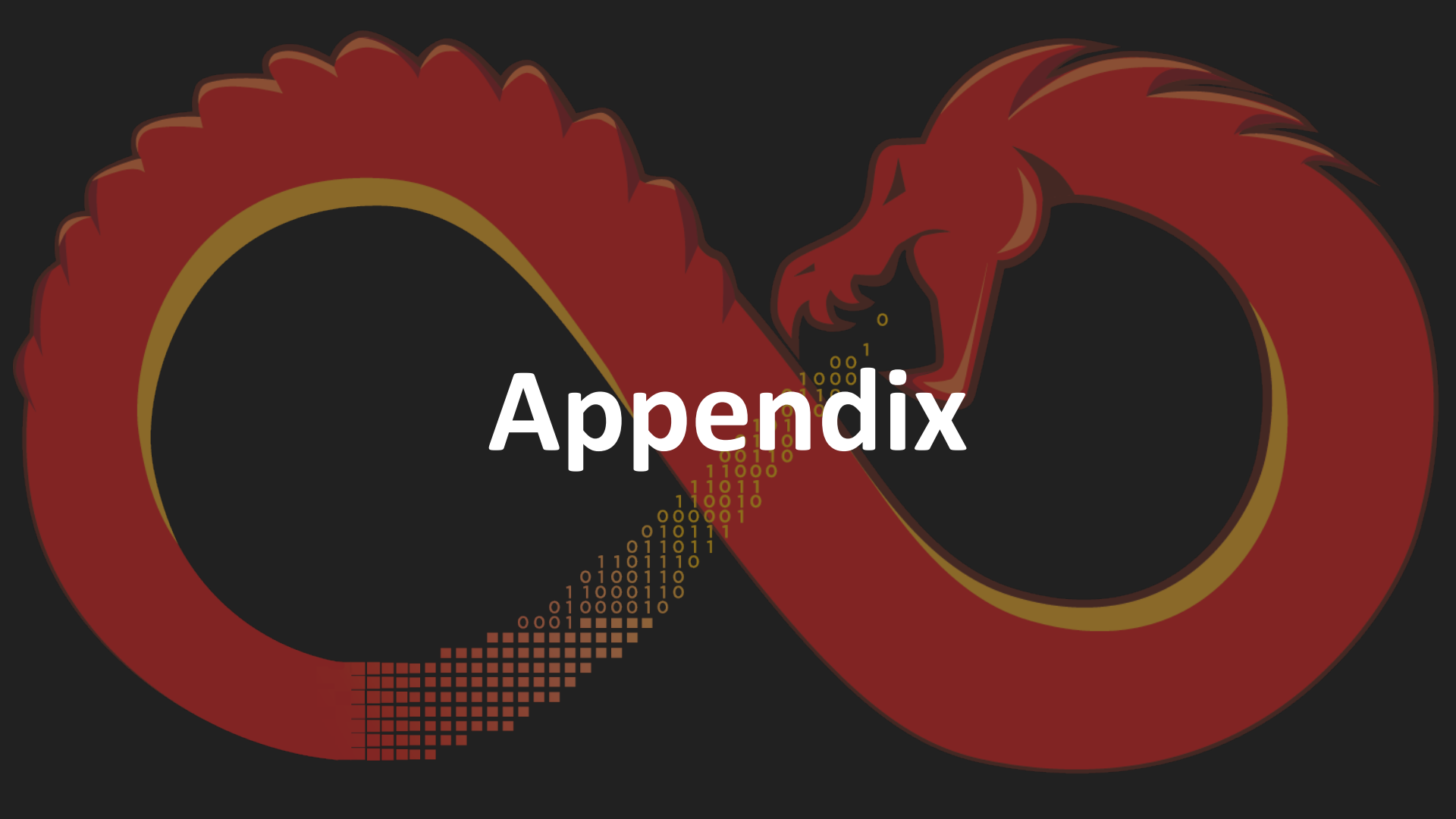
Official NSA sites:

github.com/NationalSecurityAgency/ghidra

ghidra-sre.org

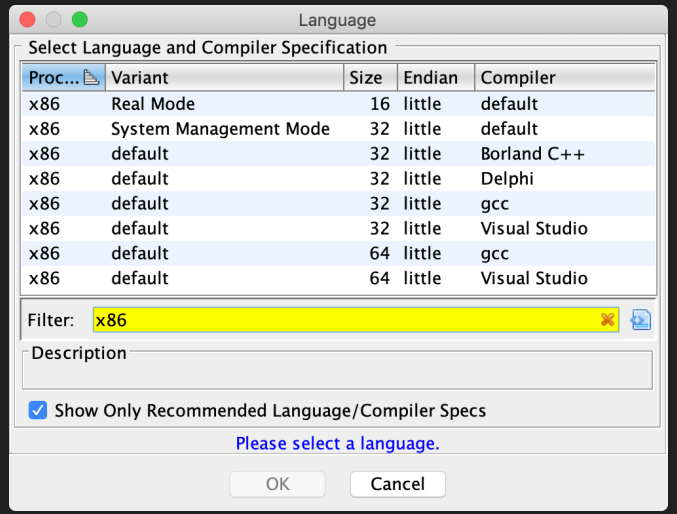
Ghidra training: ringzer0.training

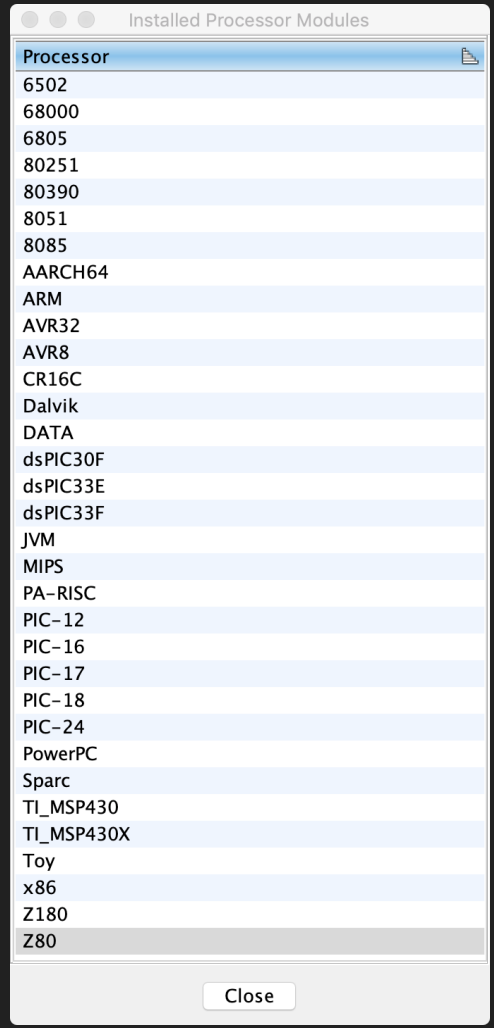



Appendix

Architecture Support

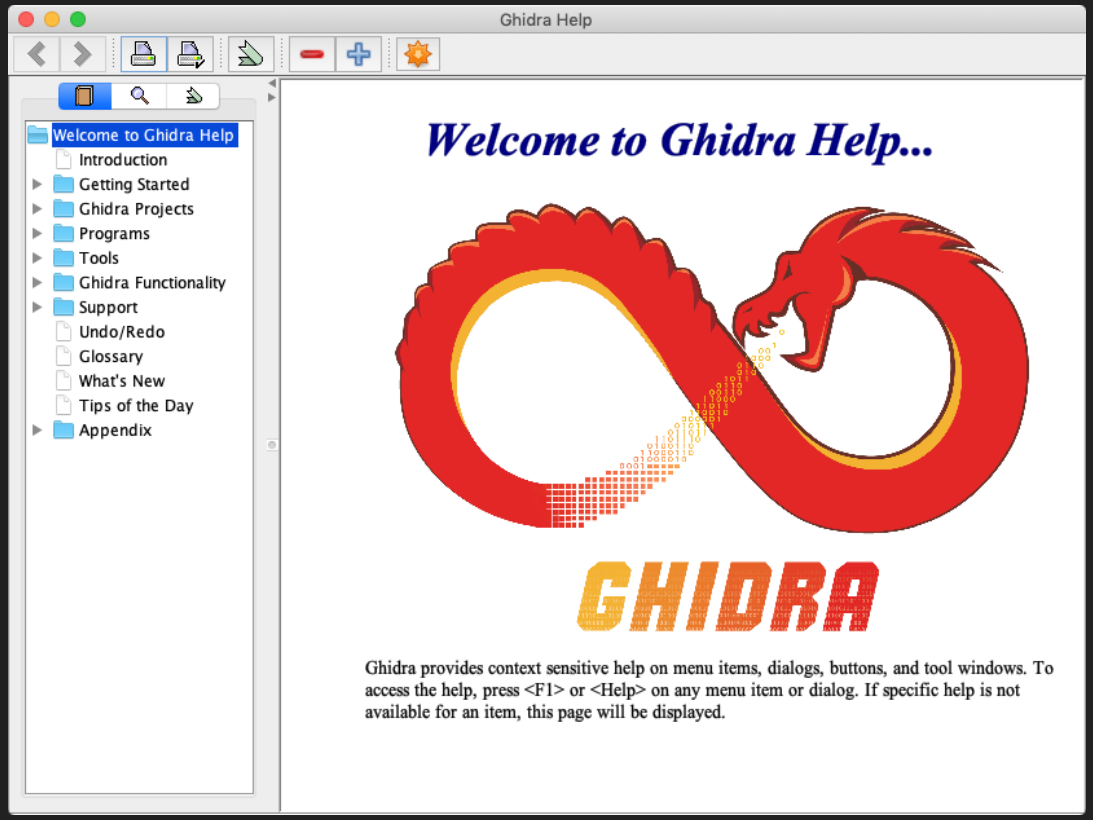
- Supports a variety of common desktop, embedded, and VM architectures
- Can handle unique compiler idioms and CPU modes
- Users can extend Ghidra with their own custom processor modules
- Ghidra can *decompile* anything it can disassemble





Documentation

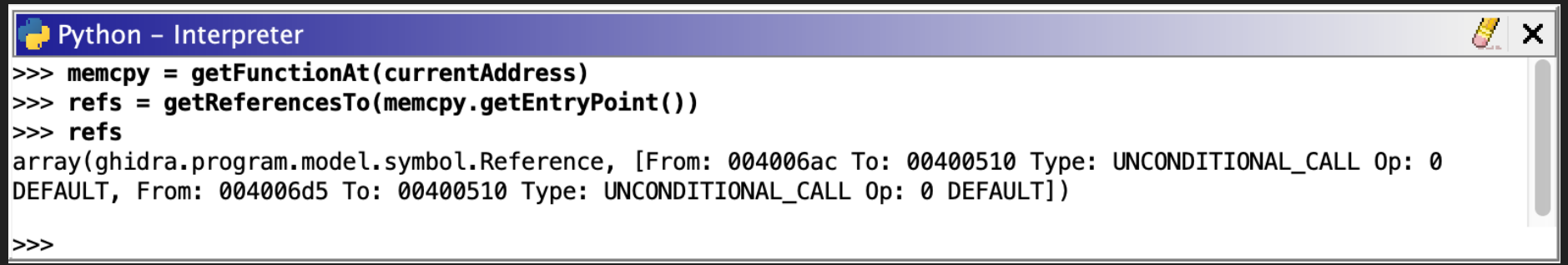
- Help > Contents
- `F1` or `Help` key while pointing at any field or menu option
- `docs` directory
  - JavaDoc
  - Several classes
  - P-code and SLEIGH
- Doxygen in source files



Python Interpreter Window

- Unlike Java, which must be compiled in order to run, Python can be run inside Ghidra in an interactive REPL shell

- The shell can be helpful for exploring unfamiliar objects - Ghidra has great Python object `__str__` implementations



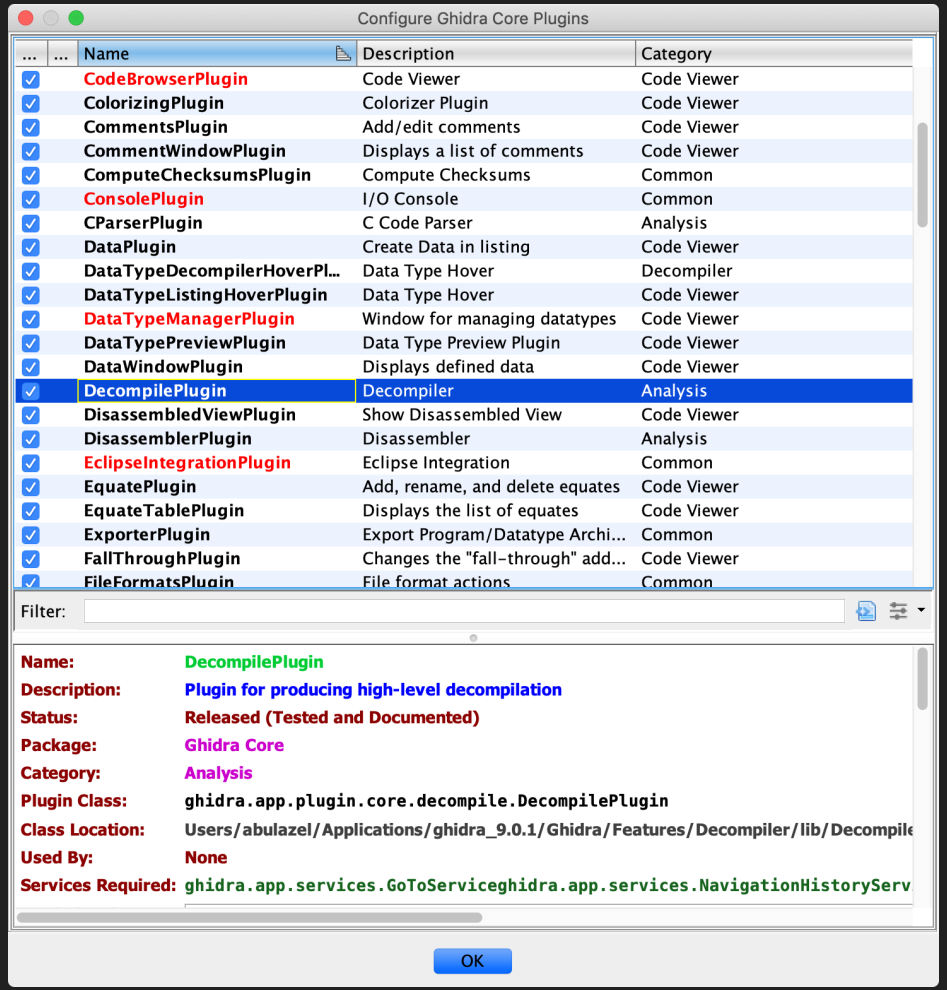
Script vs. Plugin vs. Extension

- **Scripts:** do a single thing with defined start and end point
- **Plugins:** base components for everything you interact with in Ghidra, such as UI panes
- **Extensions:** sets of plugins for extended functionality, e.g., custom binary format loaders, interfaces to external tools, libraries for use by other scripts
  - **Example:** Rolf Rolles' GhidraPAL `github.com/RolfRolles/GhidraPAL/releases`

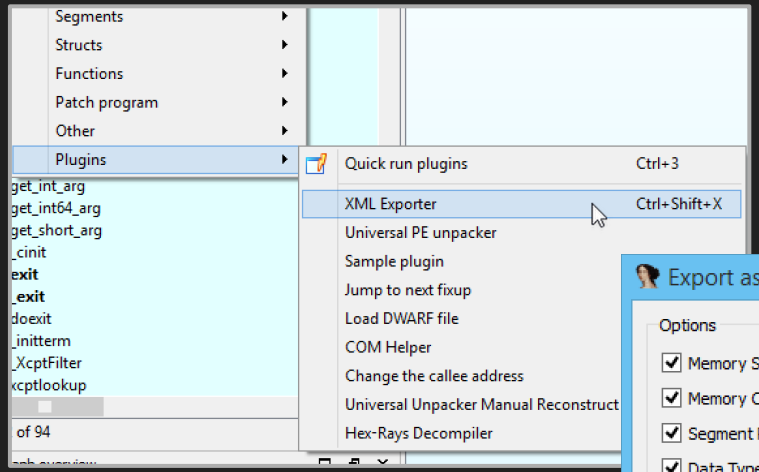
https://github.com/NationalSecurityAgency/ghidra/issues/243#issuecomment-475640292

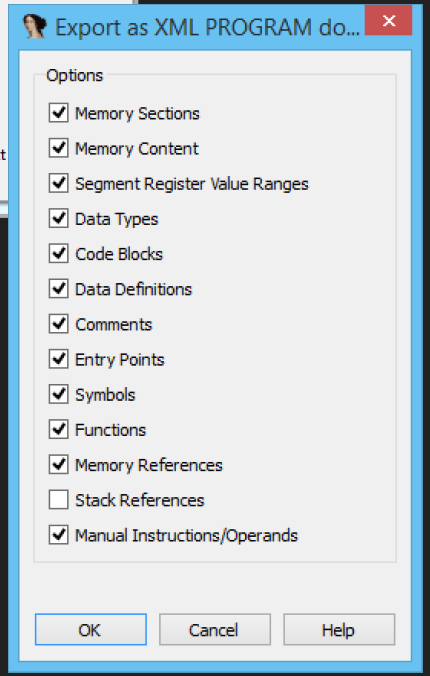
Tools

- Ghidra tools are assemblies of plugins
- Ghidra comes with two tools: CodeBrowser and VersionTracker
- Tools can be configured and customized to suit unique RE needs - though currently there doesn't seem to be much point



IDA Interoperability





Ghidra comes with importers and exporters to enable Ghidra / IDA interoperability

Decompiler Windows Structs - Hex-Rays vs Ghidra

```
20  v12 = GetSystemMetrics(0);
21  cy = GetSystemMetrics(1);
22  hWnd = GetDesktopWindow();
23  hDC = GetDC(hWnd);
24  hdc = CreateCompatibleDC(hDC);
25  h = CreateCompatibleBitmap(hDC, v12, cy);
26  SelectObject(hdc, h);
27  BitBlt(hdc, 0, 0, v12, cy, hDC, 0, 0, 0xCC0020u);
28  GetObjectA(h, 24, &pv);
29  bmi.bmiHeader.biSize = 40;
30  bmi.bmiHeader.biWidth = v6;
31  bmi.bmiHeader.biHeight = cLines;
32  bmi.bmiHeader.biPlanes = 1;
33  bmi.bmiHeader.biBitCount = 32;
34  bmi.bmiHeader.biCompression = 0;
35  bmi.bmiHeader.biSizeImage = 0;
36  bmi.bmiHeader.biXPelsPerMeter = 0;
37  bmi.bmiHeader.biYPelsPerMeter = 0;
38  bmi.bmiHeader.biClrUsed = 0;
39  bmi.bmiHeader.biClrImportant = 0;
40  dwBytes = cLines * 4 * ((32 * v6 + 31) / 32);
41  hMem = GlobalAlloc(0x42u, cLines * 4 * ((32 * v6 + 31) / 32));
42  bmi.bmiColors[0] = (RGBQUAD)GlobalLock(hMem);
43  GetDIBits(hDC, (HBITMAP)h, 0, cLines, (LPVOID)bmi.bmiColors[0], &bmi, 0);
44  v16 = 54;
45  v15 = dwBytes + 54;
46  v14 = 19778;
47  v8 = GlobalAlloc(0x42u, dwBytes + 54);
48  v9 = GlobalLock(v8);
49  memcpy(v9, &v14, 0xEu);
50  memcpy((char *)v9 + 14, &bmi, 0x28u);
51  memcpy((char *)v9 + 54, (const void *)bmi.bmiColors[0], dwBytes);
52  GlobalUnlock(hMem);
53  GlobalFree(hMem);
54  ReleaseDC(hWnd, hDC);
55  DeleteDC(hdc);
56  DeleteObject(h);
57  result = v9;
58  *a1 = v9;
59  *a2 = dwBytes + 54;
60  return result;
```

```
35  local_50 = (HDC)0x0;
36  local_20 = GetSystemMetrics(0);
37  local_8 = GetSystemMetrics(1);
38  DAT_004078cc = GetDesktopWindow();
39  DAT_004078c8 = GetDC(DAT_004078cc);
40  local_50 = CreateCompatibleDC(DAT_004078c8);
41  local_1c = CreateCompatibleBitmap(DAT_004078c8,local_20,local_8);
42  SelectObject(local_50,local_1c);
43  BitBlt(local_50,0,0,local_20,local_8,DAT_004078c8,0,0,0xcc0020);
44  GetObjectA(local_1c,0x18,local_70);
45  local_4c = 0x28;
46  local_48 = local_6c;
47  local_44 = local_68;
48  local_40 = 1;
49  local_3e = 0x20;
50  uStack60 = 0;
51  uStack58 = 0;
52  local_38 = 0;
53  local_34 = 0;
54  local_30 = 0;
55  local_2c = 0;
56  local_28 = 0;
57  iVar1 = local_6c * 0x20 + 0x1f;
58  dwBytes = ((int)(iVar1 + (iVar1 >> 0x1f & 0x1fU)) >> 5) * 4 * local_68;
59  hMem = GlobalAlloc(0x42,dwBytes);
60  local_24 = (undefined4 *)GlobalLock(hMem);
61  GetDIBits(DAT_004078c8,local_1c,0,local_68,local_24,(LPBITMAPINFO)&local_4c,0);
62  dwBytes_00 = dwBytes + 0x36;
63  local_e = 0x36;
64  local_18 = 0x4d42;
65  local_16 = dwBytes_00;
66  local_58 = GlobalAlloc(0x42,dwBytes_00);
67  local_54 = (undefined4 *)GlobalLock(local_58);
68  FUN_00401930(local_54,(undefined4 *)&local_18,0xe);
69  FUN_00401930((undefined4 *)((int)local_54 + 0xe),(undefined4 *)&local_4c,0x28);
70  FUN_00401930((undefined4 *)((int)local_54 + 0x36),local_24,dwBytes);
71  GlobalUnlock(hMem);
72  GlobalFree(hMem);
73  ReleaseDC(DAT_004078cc,DAT_004078c8);
74  DeleteDC(local_50);
75  DeleteObject(local_1c);
76  *(undefined4 **)param_1 = local_54;
77  *param_2 = dwBytes_00;
78  return;
```

P-Code Decompilation / Analysis

- “Raw p-code” = direct translation of one CPU instructions to p-code ops
- During decompilation, p-code is analyzed, and may be modified
  - Insertion of `MULTIEQUAL` instructions (SSA phi-nodes)
  - Association of parameters with `CALL` ops and return values with `RETURN` ops
  - Construction of abstract syntax tree
  - etc… - see linked documents
- The Decompiler is a C++ binary that runs on the host system

- When writing scripts interacting with p-code expect to experiment, read source code, and glean usage from example included scripts

`docs/languages/html/additionalpcode.html`
`Ghidra/Features/Decompiler/src/decompile/cpp/docmain.hh`


Links

Recommended Readings

Elias Bachaalany's quick overview: 0xeb.net/2019/03/ghidra-a-quick-overview

Danny Quist on getting started with Ghidra:
github.com/dannyquist/re/blob/master/ghidra/ghidra-getting-started.md

Rolf Rolles' GhidraPAL program analysis library:
github.com/RolfRolles/GhidraPAL
msreverseengineering.com/blog/2019/4/17/an-abstract-interpretation-based-deobfuscation-plugin-for-ghidra

Travis Goodspeed on reversing MD380 firmware with Ghidra:
github.com/travisgoodspeed/md380tools/wiki/GHIDRA

Additional Recommended Readings

Links to many loaders and processor modules:
groups.google.com/forum/?utm_medium=email&utm_source=footer#!msg/sleigh/pD12wcoKUQM/rG8esJAVBAAJ

Writing a WASM Loader: habr.com/en/post/443318/

Sega Genesis loader: zznop.github.io/romhacking/2019/03/14/sega-genesis-rom-hacking-with-ghidra.html

Because Security's first impressions, with many Ghidra video links:
blog.because-security.com/t/ghidra-wiki/431#Firstimpress283

Elias Bachaalany's "Daenerys" IDA Pro / Ghidra interoperability framework:
0xeb.net/2019/03/daenerys-ida-pro-and-ghidra-interoperability-framework

Academic Work Related to SLEIGH / P-code

Norman Ramsey and Mary F. Fernández. 1997. Specifying representations of machine instructions. ACM Transactions on Programming Languages and Systems (TOPLAS) Vol. 19, Issue 3 (1997)
citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.360&rep=rep1&type=pdf

Cristina Cifuentes and Mike Van Emmerik. "UQBT: Adaptable binary translation at low cost." Computer 33.3 (2000)
personales.ac.upc.edu/vmoya/docs/00825697.pdf

The New Jersey Machine-Code Toolkit (1990s)
www.cs.tufts.edu/~nr/toolkit/

See `docs/languages/html/sleigh.html`