

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

Alexei Bulazel @0xAlexei

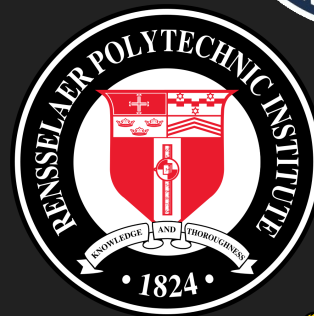
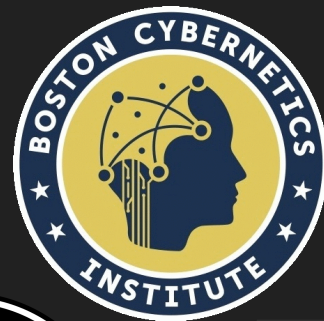


- Senior Security Researcher at River Loop Security
- Research presentations and publications:
 - Presentations at REcon (MTL & BRX), SummerCon, DEFCON, Black Hat, etc.
 - Academic publications at USENIX WOOT and ROOTS
 - Cyber policy in Lawfare, etc.
- Collaborated with Jeremy on research at RPI, MIT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

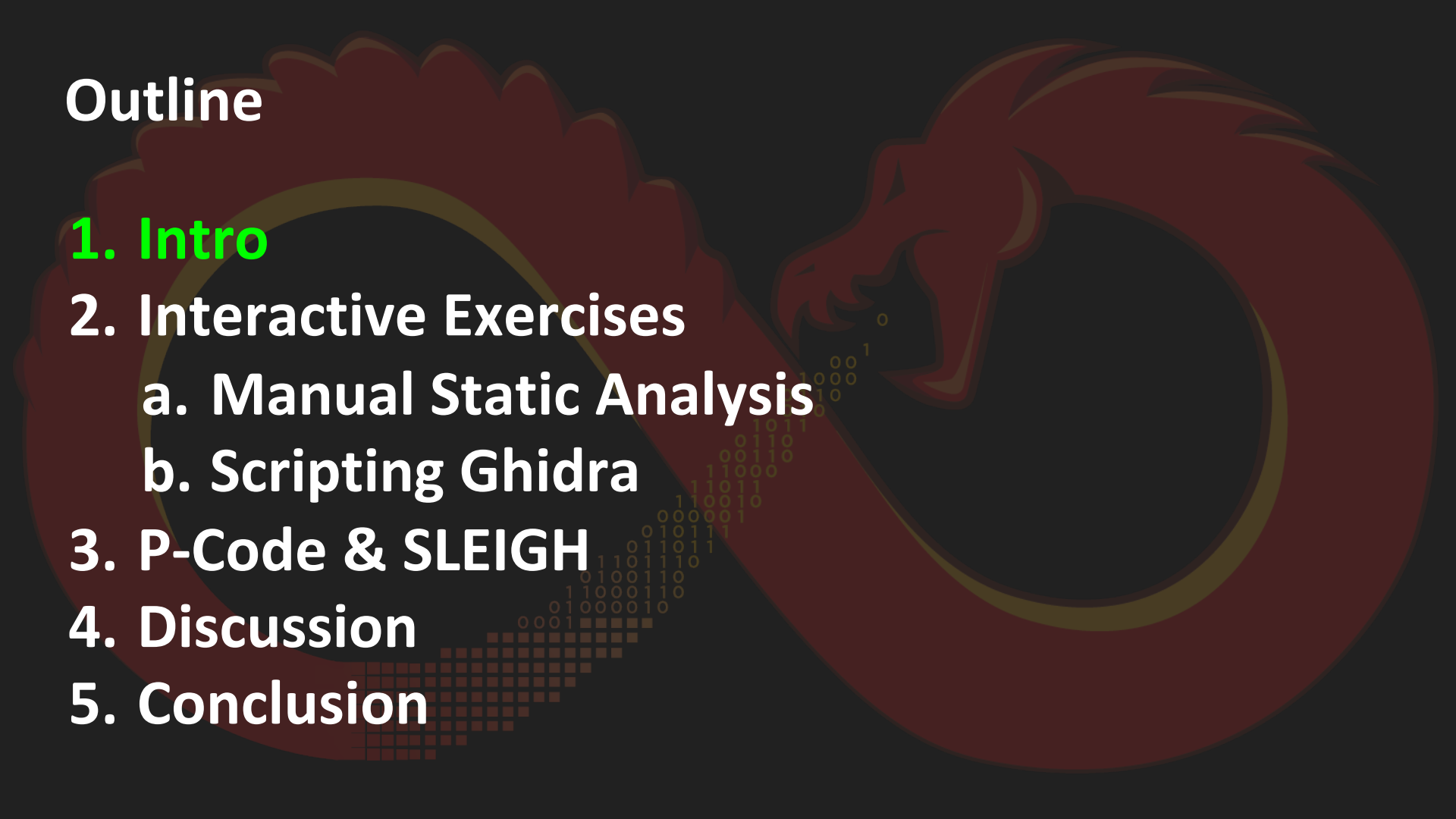
a. Manual Static Analysis

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

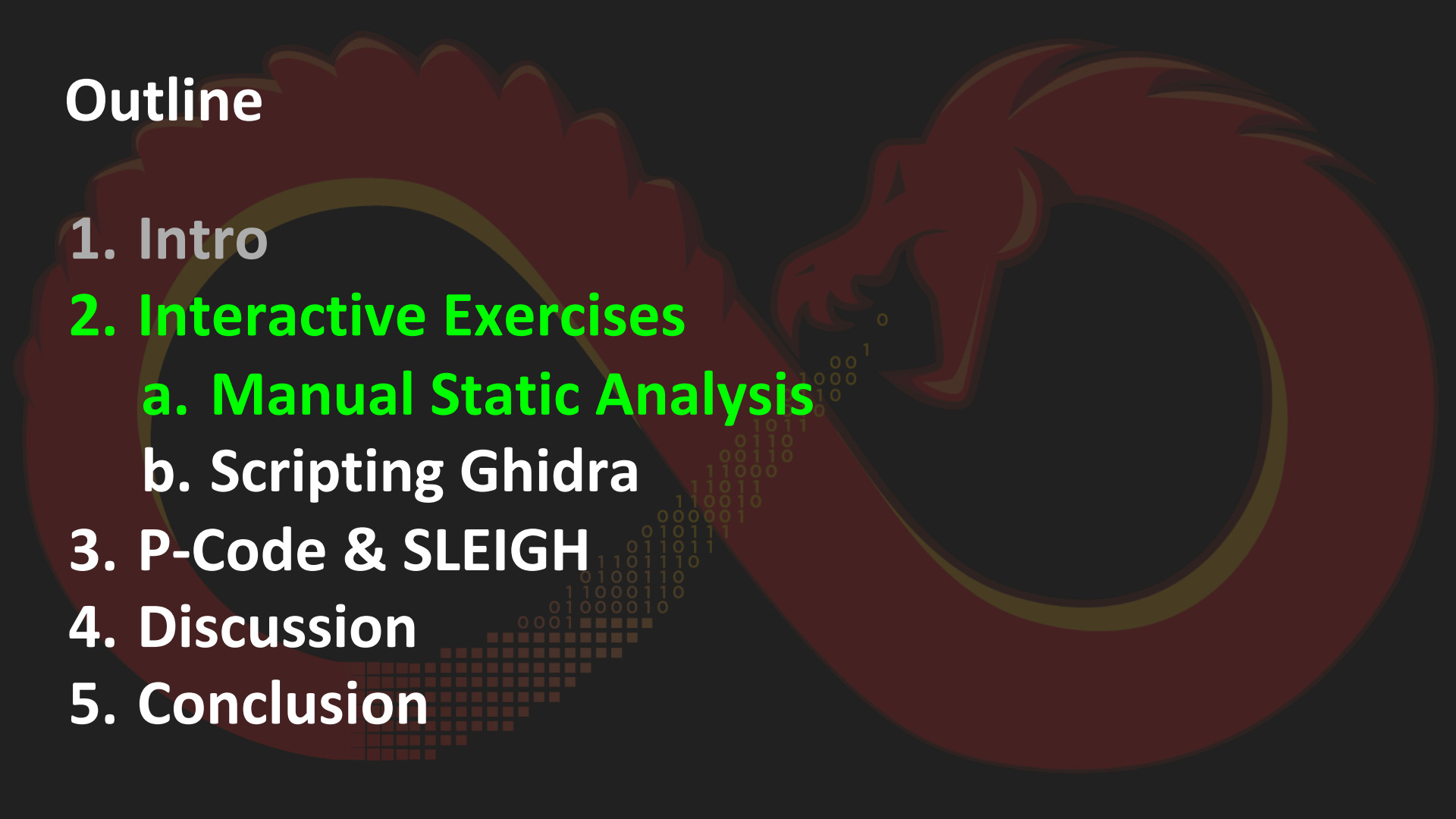
a. Manual Static Analysis

b. Scripting Ghidra

3. P-Code & SLEIGH

4. Discussion

5. Conclusion



Default UI - CodeBrowser

The screenshot displays the CodeBrowser application window titled "CodeBrowser: Infiltrate/bomb". The interface is divided into several panels:

- Program Trees:** A tree view on the left showing the file structure of the "bomb" target, including sections like .bss, .data, .got.plt, .got, .dynamic, .jcr, .fini_array, .init_array, .eh_frame, .eh_frame_hdr, .rodata, and .fini.
- Symbol Tree:** A tree view below the Program Trees showing symbols such as Imports, Exports, Functions, Labels, Classes, and Namespaces.
- Data Type Manager:** A panel below the Symbol Tree showing data types, including Built-in Types, bomb, generic_clib, generic_clib_64, and windows_vs12_32.
- Listing: bomb:** The main assembly view showing the disassembled code for the "bomb" target. The code is organized into sections (ff ff, LAB_00401499, LAB_004014a2, LAB_004014a7, LAB_004014ac, LAB_004014af, LAB_004014b1, LAB_004014b8, LAB_004014bf, LAB_004014c1, LAB_004014c6, LAB_004014cb, LAB_004014d0, LAB_004014d5) and includes instructions like ADD, RET, CALL, TEST, JNZ, MOV, CMP, JNZ, MOV, CALL, and CALL. It also shows cross-references (XREF) and a flow override (CALL_RETURN (CALL_TERMINATOR)).
- Decompile: read_line - (bomb):** A panel on the right showing the decompiled C code for the "read_line" function. The code includes variable declarations, input handling, and error checking.
- Console - Scripting:** A panel at the bottom for scripting and console output.

The bottom status bar shows the current address (004014ac), the current instruction (read_line), and the current register (TEST RAX,RAX).

The screenshot displays the Immunity Debugger interface with the following components:

- Program Trees:** Located on the left, it shows the file structure of `bomb.exe`. The `.bss` section is highlighted, and the `read_line` function is selected in the `Symbol Tree` pane.
- Decompile: read_line - (bomb):** The main window shows the decompiled C code for the `read_line` function. The code includes variable declarations, input handling, and error checking. Comments like `/* WARNING: Subroutine does not return */` are present.
- Listing: bomb:** Below the decompiled code, the assembly listing is visible, showing instructions such as `ADD RSP,0x18`, `CALL skip`, and `CALL .plt:puts`.
- Console - Scripting:** The bottom pane is empty, indicating no output has been captured yet.

The screenshot shows the Immunity Debugger interface with the following components:

- Program Trees:** Lists the loaded modules, including 'bomb'.
- Symbol Tree:** Displays the symbols for the 'bomb' module, with 'bomb' highlighted.
- Listing: bomb:** Shows the assembly code for the 'bomb' module, including the 'read_line()' function.
- Decompile: read_line - (bomb):** Displays the decompiled C code for the 'read_line()' function. The code includes a loop that reads input and checks for EOF or a specific character.
- Console - Scripting:** Shows the output of the scripting console.

Default UI - Data Type Manager

The screenshot displays the CodeBrowser application interface for analyzing a binary file named 'bomb'. The interface is divided into several panes:

- Program Trees:** Located on the top left, it shows a hierarchical view of the program's structure, including sections like .bss, .data, .got.plt, .got, .dynamic, .jcr, .fini_array, .init_array, .eh_frame, .eh_frame_hdr, .rodata, and .fini.
- Symbol Tree:** Located below the Program Trees, it shows a list of symbols and their namespaces, including Imports, Exports, Functions, Labels, Classes, and Namespaces.
- Data Type Manager:** Located at the bottom left, it is highlighted with a red rectangle. It shows a list of data types, including Built-in Types, bomb, generic_clib, generic_clib_64, and windows_vs12_32.
- Listing: bomb:** The central pane displays the assembly code for the 'bomb' function. It shows instructions such as 'ff ff', 'LAB_00401499', 'ADD RSP,0x18', 'RET', and 'undefined read_line()'. The assembly is color-coded and includes cross-references (XREF) to other parts of the program.
- Decompile: read_line - (bomb):** The right pane shows the decompiled C code for the 'read_line' function. It includes variable declarations (char cVar1, long lVar2, char *pcVar3, uint uVar4, int iVar5, int extraout_EDX, char *pcVar6, byte bVar7), function calls (fgets, puts, exit, getenv, strcmp, strcpy, printf, free), and control flow statements (if, while, do-while, break, continue, return).
- Console - Scripting:** The bottom pane is currently empty, showing the area for script execution output.

The interface also includes a menu bar (File, Edit, Analysis, Navigation, Search, Select, Tools, Window, Help) and a toolbar with various icons for file operations, search, and analysis.

Default UI - Listing (Disassembly)

The screenshot displays the CodeBrowser application interface for analyzing the 'bomb' program. The central pane shows the disassembly of the 'read_line' function, which is highlighted with a red border. The disassembly includes instructions such as 'ff ff', 'LAB_00401499', 'ADD RSP,0x18', 'RET', and a call to 'undefined read_line()'. The right-hand pane shows the decompiled C code for 'read_line', which includes variables like 'cVar1', 'lVar2', 'pcVar3', 'uVar4', 'iVar5', 'extraout_EDX', 'pcVar6', and 'bVar7'. The code includes a loop that reads input strings and checks for errors. The left-hand pane shows the 'Program Trees' and 'Symbol Tree' views, with the 'bomb' program selected. The 'Data Type Manager' and 'Console - Scripting' panes are also visible at the bottom.

Program Trees

- bomb
 - .bss
 - .data
 - .got.plt
 - .got
 - .dynamic
 - .jcr
 - .fini_array
 - .init_array
 - .eh_frame
 - .eh_frame_hdr
 - .rodata
 - .fini

Symbol Tree

- Imports
- Exports
- Functions
- Labels
- Classes
- Namespaces

Filter:

Data Type Manager

Data Types

- BuiltinTypes
- bomb
 - generic_clib
 - generic_clib_64
 - windows_vs12_32

Listing: bomb

```
ff ff
00401499 48 83 c4 18 LAB_00401499 ADD RSP,0x18 XREF[1]: 00401492(j)
0040149d c3 RET

***** FUNCTION *****
***** undefined read_line() *****
***** AL:1 <-RETURN> *****
read_line XREF[10]: Entry Point(*), main:00400e4e(c), main:00400e6a(c), main:00400e86(c), main:00400ea2(c), main:00400ebe(c), secret_phase:004004284c, 00402b5f

0040149e 48 83 ec 08 SUB RSP,0x8
004014a2 b8 00 00 MOV EAX,0x0
004014a7 e8 4d ff CALL skip undefined 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 28 00 CMP qword ptr [.bss:infile],RAX = 00000000
a9 22 28 00 JNZ LAB_004014d5
004014b1 bf d5 25 MOV EDI=>.rodata:s_Error:_Premature_EOF_on_stdin_0_ = "Error: Prem
40 00
004014c6 e8 45 f6 CALL .plt:puts int puts(char
ff ff
004014cb bf 00 00 MOV EDI,0x8
00 00
004014d0 e8 4b f7 CALL .plt:exit void exit(int
ff ff

-- Flow Override: CALL_RETURN (CALL_TERMINATOR)

LAB_004014d5 XREF[1]: 004014bf(j)
004014d5 bf f3 25 MOV EDI=>.rodata:s_GRADE_BOMB_004025f3,.rodata:s_G... = "GRADE_BOMB"
```

Decompile: read_line - (bomb)

```
1 char * read_line(void)
2
3
4 {
5     char cVar1;
6     long lVar2;
7     char *pcVar3;
8     uint uVar4;
9     int iVar5;
10    int extraout_EDX;
11    char *pcVar6;
12    byte bVar7;
13
14    bVar7 = 0;
15    lVar2 = skip();
16    if (lVar2 == 0) {
17        if (infile == stdin) {
18            puts("Error: Premature EOF on stdin");
19            /* WARNING: Subroutine does not return */
20            exit(0);
21        }
22        pcVar3 = getenv("GRADE_BOMB");
23        if (pcVar3 != (char *)0x0) {
24            /* WARNING: Subroutine does not return */
25            exit(0);
26        }
27        infile = stdin;
28        lVar2 = skip();
29        if (lVar2 == 0) {
30            puts("Error: Premature EOF on stdin");
31            /* WARNING: Subroutine does not return */
32            exit(0);
33        }
34    }
35    pcVar6 = input_strings + (long)num_input_strings * 0x50;
36    lVar2 = -1;
37    pcVar3 = pcVar6;
38    do {
39        uVar4 = (uint)lVar2;
40        if (lVar2 == 0) break;
41        lVar2 = lVar2 + -1;
42        uVar4 = (uint)lVar2;
43        cVar1 = *pcVar3;
44        pcVar3 = pcVar3 + (ulong)bVar7 * -2 + 1;
45    } while (cVar1 != 0);
46    iVar5 = uVar4 - 1;
```

Console - Scripting

004014ac read_line TEST RAX,RAX

Default UI - Decompiler

The screenshot displays the CodeBrowser application interface for decompiling a binary. The main window is titled "CodeBrowser: Infiltrate/bomb".

Left Panel:

- Program Trees:** Shows a tree structure for the "bomb" program, including sections like .bss, .data, .got.plt, .got, .dynamic, .jcr, .fini_array, .init_array, .eh_frame, .eh_frame_hdr, .rodata, and .fini.
- Symbol Tree:** Lists symbols such as Imports, Exports, Functions, Labels, Classes, and Namespaces.
- Data Type Manager:** Shows a list of data types including BuiltInTypes, bomb, generic_clib, generic_clib_64, and windows_vs12_32.

Center Panel:

Listing: bomb

```
ff ff
00401499 48 83 c4 18 LAB_00401499 RSP,0x18 XREF[1]: 00401492(j)
0040149d c3 ADD RET
***** FUNCTION *****
***** undefined read_line() *****
AL:1 read_line <-RETURN>
XREF[10]: Entry Point(*), main:00400e4e(c), main:00400e6a(c), main:00400e86(c), main:00400ea2(c), main:00400ebe(c), secret_phase:004004284c, 00402b5f

0040149e 48 83 ec 08 SUB RSP,0x8
004014a2 b8 00 00 MOV EAX,0x0
004014a7 e8 4d ff CALL skip undefined 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 28 00 CMP qword ptr [.bss:infile],RAX = 00000000
a9 22 28 00 JNZ LAB_004014d5
004014b1 bf d5 25 MOV EDI=>.rodata:s_Error:Premature_EOF_on_stdin_0... = "Error: Prem
40 00
004014c6 e8 45 f6 CALL .plt:puts int puts(char
ff ff
004014cb bf 00 00 MOV EDI,0x8
004014d0 e8 4b f7 CALL .plt:exit void exit(int
ff ff
--- Flow Override: CALL_RETURN (CALL_TERMINATOR)

LAB_004014d5 XREF[1]: 004014bf(j)
004014d5 bf f3 25 MOV EDI=>.rodata:s_GRADE_BOMB_004025f3,.rodata:s_G... = "GRADE_BOMB"
```

Right Panel:

Decompile: read_line - (bomb)

```
char * read_line(void)
{
    char cVar1;
    long lVar2;
    char *pcVar3;
    uint uVar4;
    int iVar5;
    int extraout_EDX;
    char *pcVar6;
    byte bVar7;

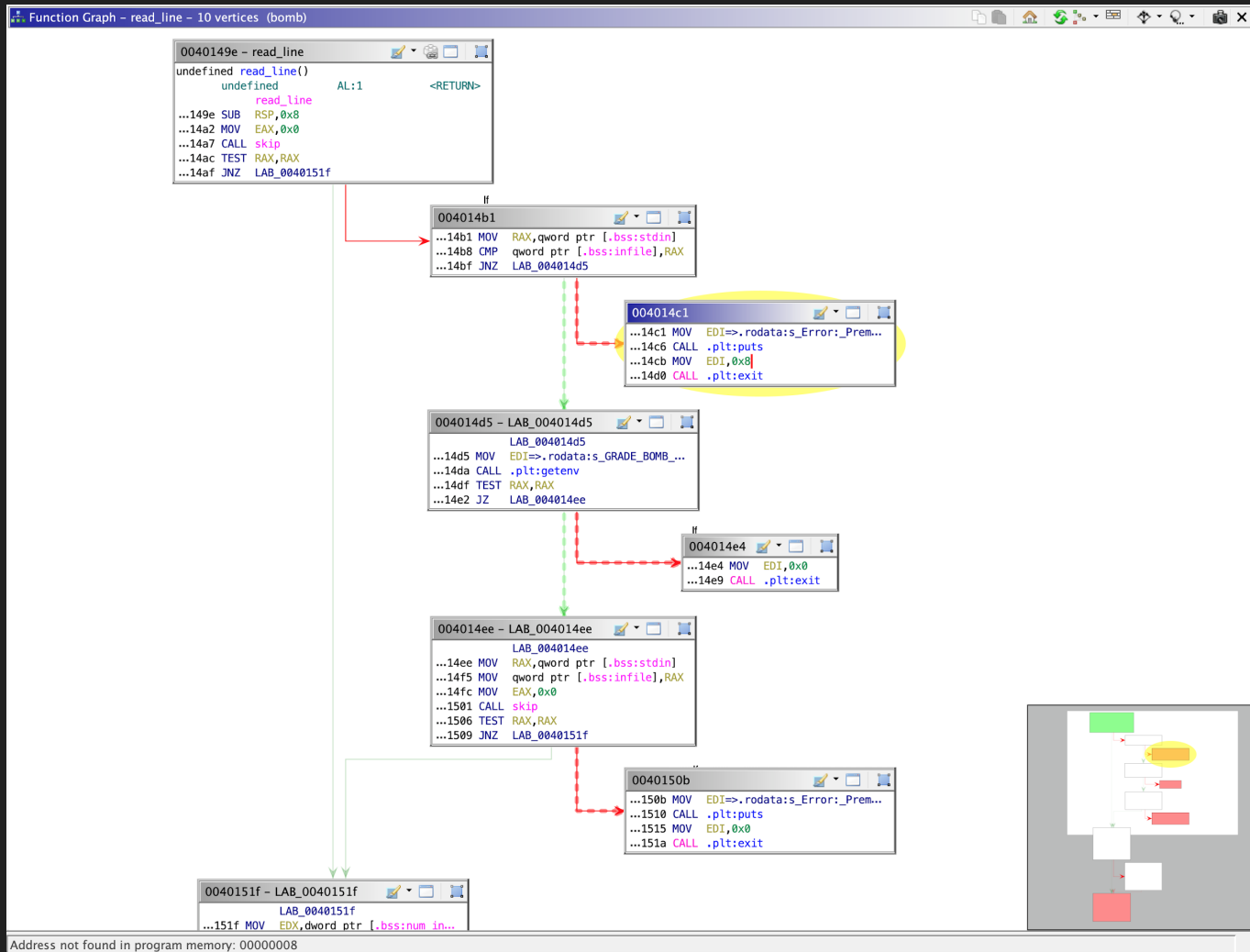
    bVar7 = 0;
    lVar2 = skip();
    if (lVar2 == 0) {
        if (infile == stdin) {
            puts("Error: Premature EOF on stdin");
            /* WARNING: Subroutine does not return */
            exit(0);
        }
        pcVar3 = getenv("GRADE_BOMB");
        if (pcVar3 != (char *)0x0) {
            /* WARNING: Subroutine does not return */
            exit(0);
        }
        infile = stdin;
        lVar2 = skip();
        if (lVar2 == 0) {
            puts("Error: Premature EOF on stdin");
            /* WARNING: Subroutine does not return */
            exit(0);
        }
    }
    pcVar6 = input_strings + (long)num_input_strings * 0x50;
    lVar2 = -1;
    pcVar3 = pcVar6;
    do {
        uVar4 = (uint)lVar2;
        if (lVar2 == 0) break;
        lVar2 = lVar2 + -1;
        uVar4 = (uint)lVar2;
        cVar1 = *pcVar3;
        pcVar3 = pcVar3 + (ulong)bVar7 * -2 + 1;
    } while (cVar1 != 0);
    iVar5 = uVar4 - 1;
```

Bottom Panel:

Console - Scripting

004014ac read_line TEST RAX,RAX

Control Flow Graph



Disassembly with P-Code

```
Listing: bomb
*bomb x

004014cb bf 08 00      MOV     EDI,0x8
00 00
    (register, 0x38, 8) = COPY (const, 0x8, 8)

004014d0 e8 4b f7      CALL    .plt:exit
ff ff
    void exit(int __status)

    (register, 0x20, 8) = INT_SUB (register, 0x20, 8), (const, 0x8, 8)
    STORE (const, 0x131, 8), (register, 0x20, 8), (const, 0x4014d5, 8)
    CALL (ram, 0x400c20, 8)
    RETURN (const, 0x0, 8)
    -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

LAB_004014d5
XREF[1]: 004014bf(j)
004014d5 bf f3 25      MOV     EDI=>.rodata:s_GRADE_BOMB_004025f3,.rodata:s_G... = "GRADE_BOMB"
40 00
    (register, 0x38, 8) = COPY (const, 0x4025f3, 8)

004014da e8 01 f6      CALL    .plt:getenv
ff ff
    char * getenv(char * __name)

    (register, 0x20, 8) = INT_SUB (register, 0x20, 8), (const, 0x8, 8)
    STORE (const, 0x131, 8), (register, 0x20, 8), (const, 0x4014df, 8)
    CALL (ram, 0x400ae0, 8)

004014df 48 85 c0      TEST    RAX,RAX
    (register, 0x200, 1) = COPY (const, 0x0, 1)
    (register, 0x20b, 1) = COPY (const, 0x0, 1)
    (unique, 0xbb10, 8) = INT_AND (register, 0x0, 8), (register, 0x0, 8)
    (register, 0x207, 1) = INT_SLESS (unique, 0xbb10, 8), (const, 0x0, 8)
    (register, 0x206, 1) = INT_EQUAL (unique, 0xbb10, 8), (const, 0x0, 8)

004014e2 74 0a        JZ      LAB_004014ee
    CBRANCH (ram, 0x4014ee, 8), (register, 0x206, 1)

004014e4 bf 00 00      MOV     EDI,0x0
00 00
    (register, 0x38, 8) = COPY (const, 0x0, 8)

004014e9 e8 32 f7      CALL    .plt:exit
ff ff
    void exit(int __status)

    (register, 0x20, 8) = INT_SUB (register, 0x20, 8), (const, 0x8, 8)
    STORE (const, 0x131, 8), (register, 0x20, 8), (const, 0x4014ee, 8)
    CALL (ram, 0x400c20, 8)
    RETURN (const, 0x0, 8)
    -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

LAB_004014ee
XREF[1]: 004014e2(j)
004014ee 48 8b 05      MOV     RAX,qword ptr [.bss:stdin]
53 22 20 00
    (register, 0x0, 8) = COPY (ram, 0x603748, 8)

004014f5 48 89 05      MOV     qword ptr [.bss:infile],RAX
6c 22 20 00
    = 00000000

    (ram, 0x603768, 8) = COPY (register, 0x0, 8)

004014fc b8 00 00      MOV     EAX,0x0
00 00
```

Decompilation Across Architectures - x64

The image displays a side-by-side comparison of assembly and decompiled C code for a function named `r_l_complete_internal` in the `elf-Linux-x64-bash` binary.

Left Panel: Assembly Listing

Listing: elf-Linux-x64-bash

*elf-Linux-x64-bash

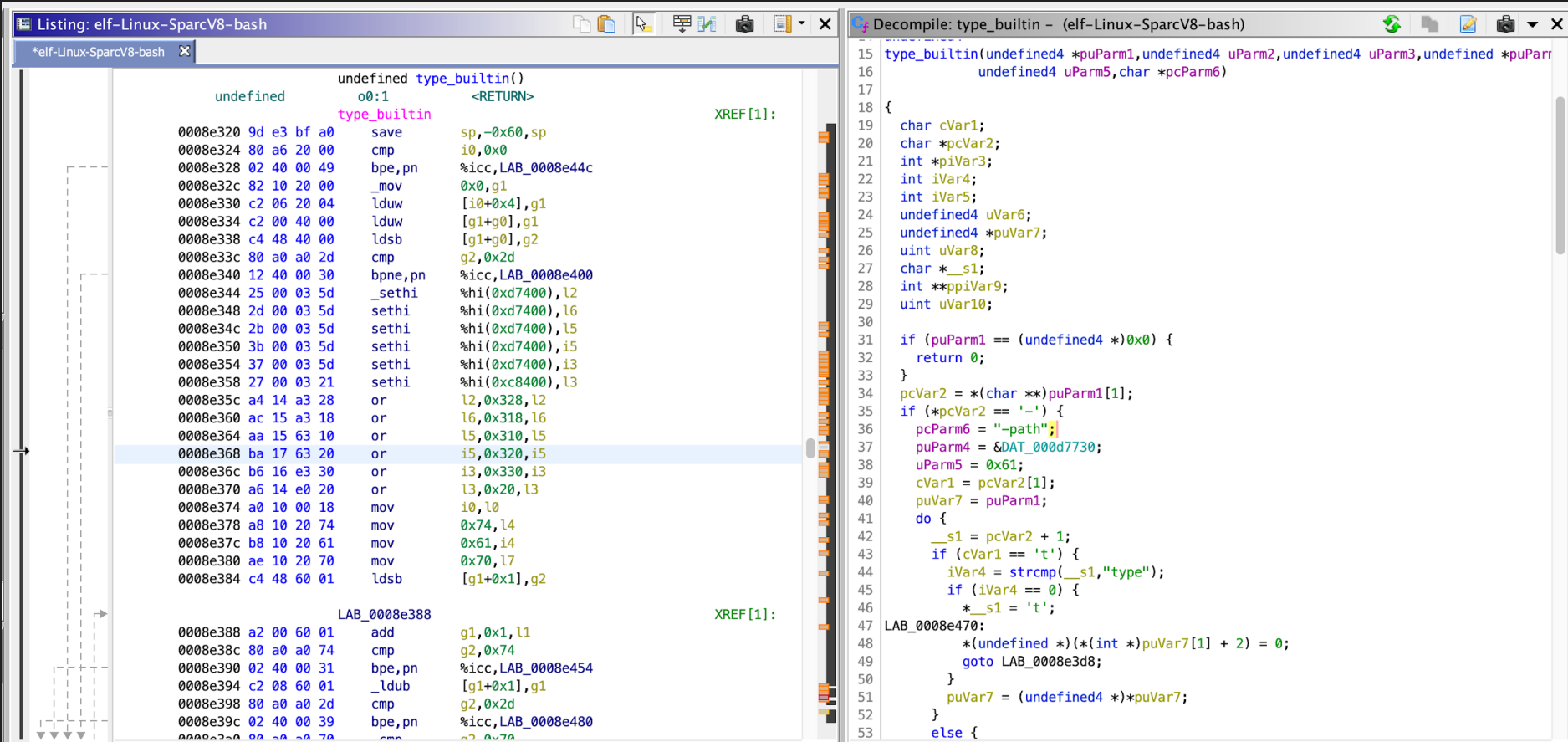
Address	Disassembly	Comment
00493a9e	75 c0	JNZ LAB_00493a60
00493aa0	e8 0b af 00 00	CALL r_l_end_undo_group
00493aa5	48 8b 7c 24 08	MOV RDI, qword ptr [RSP + local_50]
00493aaa	e9 01 ff ff ff	JMP LAB_004939b0
00493aaf	90	?? 90h
00493ab0	48 8b 7c 24 08	MOV RDI=>local_50, qword ptr [RSP + 0x8]
00493ab5	e8 86 ee ff ff	CALL FUN_00492940
00493aba	48 8b 7c 24 08	MOV RDI, qword ptr [RSP + local_50]
00493abf	e9 ec fe ff ff	JMP LAB_004939b0
00493ac4	0f	?? 0Fh
00493ac5	1f	?? 1Fh
00493ac6	40	?? 40h @
00493ac7	00	?? 00h
00493ac8	48 8b 0d d9 16 25 00	MOV RCX, qword ptr [.bss:r_l_line_buffer]
00493acf	48 63 d5	MOVSXD RDX, EBX
00493ad2	3a 44 11 ff	CMP AL, byte ptr [RCX + RDX*0x1 + -0x1]
00493ad6	0f 94 c0	SETZ AL
00493ad9	0f b6 c0	MOVZX EAX, AL
00493adc	29 c5	SUB EBX, EAX
00493ade	e9 48 ff ff ff	JMP LAB_00493a2b
00493ae3	0f	?? 0Fh
00493ae4	1f	?? 1Fh
00493ae5	44	?? 44h D
00493ae6	00	?? 00h
00493ae7	00	?? 00h

Right Panel: Decompiled C Code

Decompile: r_l_complete_internal - (elf-Linux-x64-bash)

```
41 pCVar9 = r_l_completion_entry_function;
42 if (r_l_completion_entry_function == (code *)0x0) {
43     pCVar9 = r_l_filename_completion_function;
44 }
45 uVar5 = 0;
46 if (r_l_point != 0) {
47     local_39[0] = _r_l_find_completion_word(&local_44, &local_40);
48     uVar5 = r_l_point;
49 }
50 r_l_point = uVar1;
51 __src = (char *)r_l_copy_text((ulong)uVar5, (ulong)uVar1);
52 uVar7 = 0;
53 local_50 = (char **)FUN_00492490(__src, (ulong)uVar5, (ulong)uVar1, pCVar9, (ulong)loc
54                                (ulong)(uint)(int)local_39[0]);
55 if (local_50 != (char **)0x0) {
56     iVar3 = strcmp(__src, local_50);
57     uVar7 = (ulong)(iVar3 != 0);
58 }
59 free(__src);
60 if ((local_50 == (char **)0x0) ||
61     (iVar3 = FUN_00490cb0(&local_50, (ulong)r_l_filename_completion_desired), ppcVar8
62     iVar3 == 0)) {
63     r_l_ding();
64     if (__dest != (char *)0x0) {
65         free(__dest);
66     }
67     r_l_readline_state = r_l_readline_state & 0xffffbfff;
68     DAT_006e5570 = 0;
69     r_l_completion_found_quote = 0;
70     r_l_completion_quote_character = 0;
71     return 0;
72 }
73 if (uParm1 == 0x2a) {
74     r_l_begin_undo_group();
75     if ((uVar5 != 0) && (local_39[0] != 0)) {
76         uVar5 = uVar5 - (uint)(local_39[0] == r_l_line_buffer[(long)(int)uVar5 + -1]);
77     }
78     r_l_delete_text((ulong)uVar5, (ulong)r_l_point);
79     __src = ppcVar8[1];
80     if (__src == (char *)0x0) {
81         r_l_point = uVar5;
82         __src = (char *)FUN_004916e0(*ppcVar8, 1, local_39);
83         r_l_insert_text(__src);
84         r_l_insert_text(&DAT_004ad25a);
```

Decompilation Across Architectures - SPARC



The image displays a decompilation tool interface with two main panes. The left pane shows the original SPARC assembly code, and the right pane shows the decompiled C code.

Left Pane: Listing: elf-Linux-SparcV8-bash

```
o0:1 <RETURN>
type_builtin
0008e320 9d e3 bf a0 save sp,-0x60,sp
0008e324 80 a6 20 00 cmp i0,0x0
0008e328 02 40 00 49 bpe,pn %icc,LAB_0008e44c
0008e32c 82 10 20 00 _mov 0x0,g1
0008e330 c2 06 20 04 ldub [i0+0x4],g1
0008e334 c2 00 40 00 ldub [g1+g0],g1
0008e338 c4 48 40 00 ldsb [g1+g0],g2
0008e33c 80 a0 a0 2d cmp g2,0x2d
0008e340 12 40 00 30 bpe,pn %icc,LAB_0008e400
0008e344 25 00 03 5d _sethi %hi(0xd7400),l2
0008e348 2d 00 03 5d sethi %hi(0xd7400),l6
0008e34c 2b 00 03 5d sethi %hi(0xd7400),l5
0008e350 3b 00 03 5d sethi %hi(0xd7400),i5
0008e354 37 00 03 5d sethi %hi(0xd7400),i3
0008e358 27 00 03 21 sethi %hi(0xc8400),l3
0008e35c a4 14 a3 28 or l2,0x328,l2
0008e360 ac 15 a3 18 or l6,0x318,l6
0008e364 aa 15 63 10 or l5,0x310,l5
0008e368 ba 17 63 20 or i5,0x320,i5
0008e36c b6 16 e3 30 or i3,0x330,i3
0008e370 a6 14 e0 20 or l3,0x20,l3
0008e374 a0 10 00 18 mov i0,l0
0008e378 a8 10 20 74 mov 0x74,l4
0008e37c b8 10 20 61 mov 0x61,i4
0008e380 ae 10 20 70 mov 0x70,l7
0008e384 c4 48 60 01 ldsb [g1+0x1],g2

LAB_0008e388 a2 00 60 01 add g1,0x1,l1
0008e38c 80 a0 a0 74 cmp g2,0x74
0008e390 02 40 00 31 bpe,pn %icc,LAB_0008e454
0008e394 c2 08 60 01 _ldub [g1+0x1],g1
0008e398 80 a0 a0 2d cmp g2,0x2d
0008e39c 02 40 00 39 bpe,pn %icc,LAB_0008e480
0008e3a0 80 a0 a0 70 cmp g2,0x70
```

Right Pane: Decompile: type_builtin - (elf-Linux-SparcV8-bash)

```
type_builtin(undefined4 *puParm1,undefined4 uParm2,undefined4 uParm3,undefined *puParm
undefined4 uParm5,char *pcParm6)
{
    char cVar1;
    char *pcVar2;
    int *piVar3;
    int iVar4;
    int iVar5;
    undefined4 uVar6;
    undefined4 *puVar7;
    uint uVar8;
    char *__s1;
    int **ppiVar9;
    uint uVar10;

    if (puParm1 == (undefined4 *)0x0) {
        return 0;
    }
    pcVar2 = *(char **)puParm1[1];
    if (*pcVar2 == '-') {
        pcParm6 = "-path";
        puParm4 = &DAT_000d7730;
        uParm5 = 0x61;
        cVar1 = pcVar2[1];
        puVar7 = puParm1;
        do {
            __s1 = pcVar2 + 1;
            if (cVar1 == 't') {
                iVar4 = strcmp(__s1,"type");
                if (iVar4 == 0) {
                    *__s1 = 't';
                }
            }
            LAB_0008e470:
                *(undefined *)*((int *)puVar7[1] + 2) = 0;
                goto LAB_0008e3d8;
        } while (puVar7 = (undefined4 *)*puVar7);
    }
    else {
```

Decompilation Across Architectures - PowerPC

The image displays a decompilation tool interface with two main panes. The left pane shows assembly code for a MachO file, and the right pane shows the corresponding decompiled C code.

Left Pane: Listing: MachO-OSX-ppc-openssl-1.0.1h

The assembly code is organized into several labeled blocks (LAB_000212c8, LAB_000212d0, LAB_000212d8, LAB_000212e4, LAB_000212e8, LAB_000212f4, LAB_0002130c). Each block contains instructions with their addresses, hex values, mnemonics, and operands. For example, LAB_000212c8 starts at address 000212c8 and contains instructions like `bne cr7, LAB_000212d8` and `b LAB_000212e4`.

Right Pane: Decompile: UndefinedFunction_00020aec - (MachO-OSX-ppc-openssl-1.0.1h)

The decompiled C code is a function that handles SSL/TLS operations. It includes logic for setting up SSL contexts, managing session IDs, and handling BIO callbacks. Key functions and variables include `_SSL_new`, `_SSL_callback_ctrl`, `_SSL_ctrl`, `_SSL_set_session_id_context`, `_BIO_new_socket`, `_BIO_test`, `_BIO_f_bio_test`, `_BIO_new`, `_BIO_push`, `_SSL_set_bio`, `_SSL_set_accept_state`, `_BIO_ctrl`, `_BIO_push`, `_SSL_set_debug`, `_BIO_set_callback`, `_BIO_set_callback_arg`, `_SSL_set_msg_callback`, and `_SSL_ctrl`. The code also includes error handling and logging using `_BIO_printf`.

Cross-References

The screenshot displays a debugger window with a disassembly view on the left and a code view on the right. A context menu is open over the disassembly view, showing various actions. The 'References' menu item is selected, and a sub-menu is open showing options for adding, editing, and deleting references, as well as showing references to specific addresses and call trees.

Disassembly View:

Address	Hex	Instruction	Comment
00402f80	e8 c3 fe ff ff	CALL	.text
00402f85	e8 46 e8 ff ff	CALL	_read_line
00402f8a	89 04 24	MOV	dword ptr [ebp+00000024], eax
00402f8d	e8 be e9 ff ff	CALL	_phase_defused
00402f92	e8 19 e9 ff ff	CALL	_phase_defused
00402f97	c7 04 24 44 53 40 00	MOV	dword ptr [ebp+00000024], 00000044
00402f9e	e8 a5 fe ff ff	CALL	.text
00402fa3	e8 28 e8 ff ff	CALL	_read_line
00402fa8	89 04 24	MOV	dword ptr [ebp+00000024], eax
00402fab	e8 d0 e9 ff ff	CALL	_phase_defused
00402fb0	e8 fb e8 ff ff	CALL	_phase_defused
00402fb5	c7 04 24 6d 53 40 00	MOV	dword ptr [ebp+00000024], 0000006d
00402fbc	e8 87 fe ff ff	CALL	.text
00402fc1	e8 0a e8 ff ff	CALL	_read_line
00402fc6	89 04 24	MOV	dword ptr [ebp+00000024], eax
00402fc9	e8 02 ea ff ff	CALL	_phase_defused
00402fce	e8 dd e8 ff ff	CALL	_phase_defused

Code View:

```
27 .text( which to blow yourself up.  
28 pcVar1 = _read_line();  
29 _phase_1(pcVar1);  
30 _phase_defused();  
31 .text("Phase 1 defused. How about  
32 pcVar1 = _read_line();  
33 _phase_2(pcVar1);  
34 _phase_defused();  
35 .text("That\'s number 2. Keep go  
36 pcVar1 = _read_line();  
37 _phase_3(pcVar1);  
38 _phase_defused();  
39 .text("Halfway there!");
```

Context Menu:

- Bookmark... %D
- Clear Code Bytes C
- Clear With Options...
- Clear Flow and Repair...
- Copy %C
- Copy Special... %V
- Paste
- Comments ►
- Instruction Info...
- Modify Instruction Flow...
- Patch Instruction ⇧%G
- Processor Manual...
- Processor Options...
- Create Function F
- Create Thunk Function
- Edit Function... F
- Function
- Compare Selected Functions... ⇧%F
- Edit Label... L
- Remove Label [X]
- Set Associated Label... ⌘%L
- Show Label History... H
- Clear Register Values...
- Set Register Values... %R
- Colors ►
- Fallthrough ►
- References ►
 - Add Reference from... R
 - Add/Edit... R
 - Create Memory Reference ⌘R
 - Delete Memory References [X]
 - Show References to _phase_defused ⇧%F
 - Show References to Address
 - Show Call Trees

Strings

The screenshot shows the CodeBrowser application window titled "CodeBrowser: understand:/bomb.exe". The interface includes a menu bar (File, Edit, Analysis, Navigation, Search, Select, Tools, Window, Help) and a toolbar. On the left, there is a "Program Tree" showing the file structure of "bomb.exe" (Headers, .text, .data, .rdata, .bss, .idata, .CRT, .tls, /4, /19, /31) and a "Symbol Tree" showing imports, exports, functions, labels, classes, and namespaces. A "Data Type Manager" window is also visible at the bottom left. The main window displays the "Listing: bomb.exe" for the file "*bomb.exe". It shows a list of memory addresses and their corresponding assembly instructions. A context menu is open over the listing, with "Defined Strings" highlighted. The decompiled code for the function "_main" is shown on the right, with the following content:

```
20 .text("%s: Error: Couldn't open %s\n",*_Ar
21 /* WARNING: Subroutine does not return
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 d
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();
```

The bottom of the window shows a tab for "Decompile: _main" and a "Function Graph" tab.

Type System / Structure Recovery

Structure Editor - group (busybox) [CodeBrowser(2): Infiltrate:/busybox]

Help

Structure Editor - group (busybox)

Offset	Length	Mnemonic	DataType	Name	Comment
0	4	char *	char *	gr_name	
4	4	char *	char *	gr_passwd	
8	4	__gid_t	__gid_t	gr_gid	
12	4	char **	char **	gr_mem	

Search:

Name:

Description:

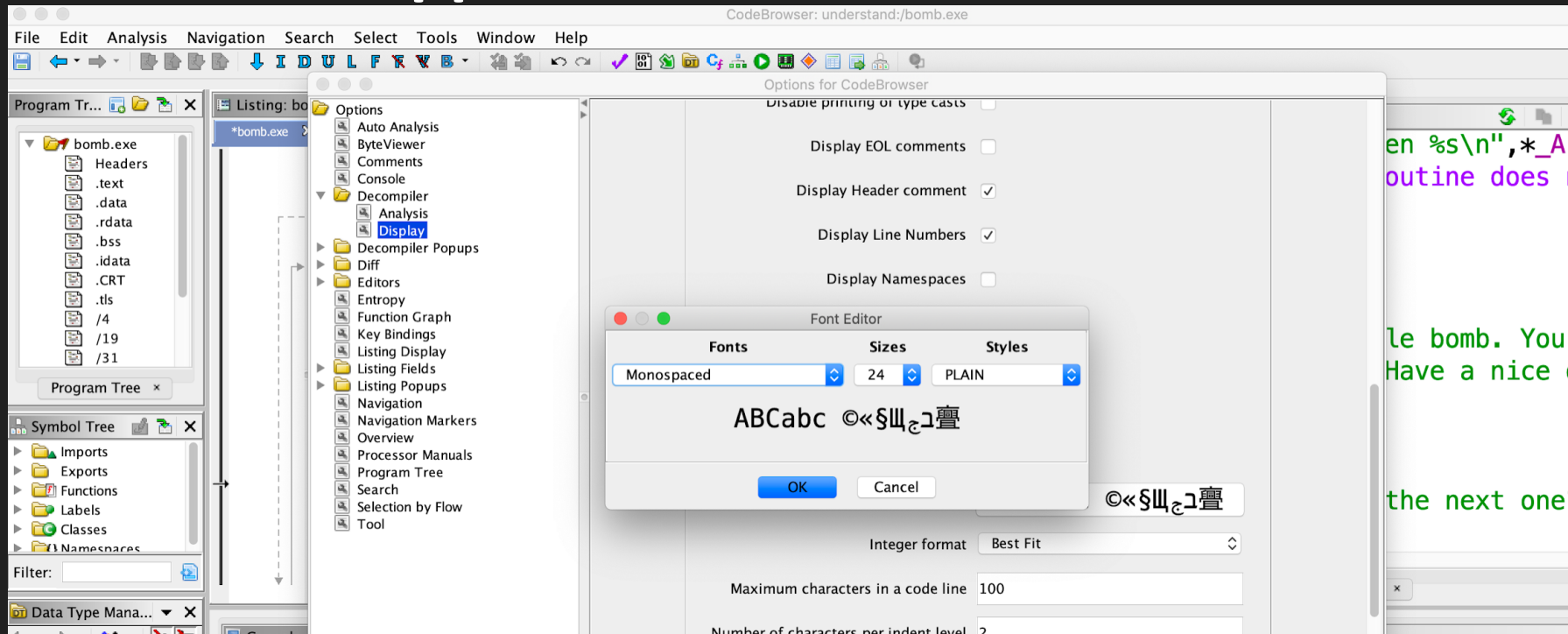
Category:

Size: Alignment: ☒ Align

align(minimum...
☒ none
☐ machine
☐

pack(maximu...
☒ none
☐

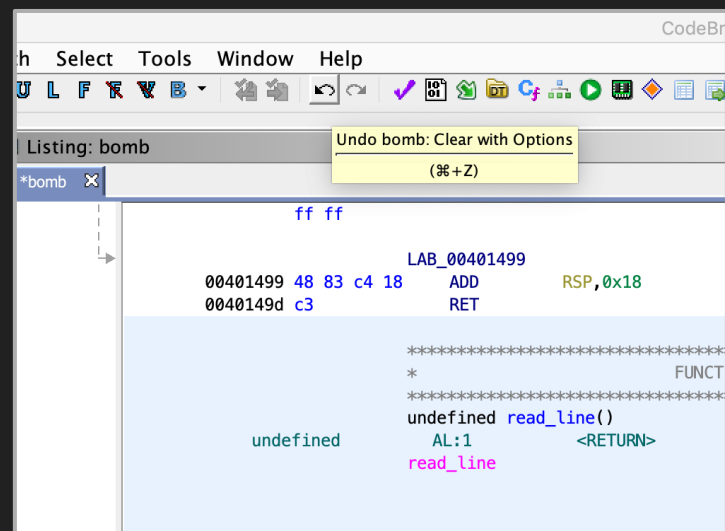
Customize Appearance



Edit > Tool Options → Filter: Fonts

The Undo Button!

0040149e	48 83 ec 08	SUB	RSP,0x8
004014a2	b8 00 00 00 00	MOV	EAX,0x0
004014a7	e8 4d ff ff ff	CALL	skip
004014ac	48 85 c0	TEST	RAX,RAX
004014af	75 6e	JNZ	LAB_0040151f
004014b1	48 8b 05 90 22 20 00	MOV	RAX,qword ptr [.bss:stdin]
004014b8	48 39 05 a9 22 20 00	CMP	qword ptr [.bss:infile],RAX
004014bf	75 14	JNZ	LAB_004014d5
004014c0	0040149e 48	??	48h H
004014c1	0040149f 83	??	83h
004014c2	004014a0 ec	??	ECh
004014c3	004014a1 08	??	08h
004014c4	004014a2 b8 00 00 00 00	MOV	EAX,0x0
004014c5	004014a7 e8 4d ff ff ff	CALL	skip
004014c6	004014ac 48 85 c0	TEST	RAX,RAX
004014c7	004014af 75 6e	JNZ	LAB_0040151f
004014c8	004014b1 48 8b 05 90 22 20 00	MOV	RAX,qword ptr [.bss:stdin]
004014c9	004014b8 48 39 05 a9 22 20 00	CMP	qword ptr [.bss:infile],RAX
004014ca	004014bf 75 14	JNZ	LAB_004014d5
004014cb	004014c1 bf d5 25 40 00	MOV	EDI=>.rodata:s_Error:_Premat
004014cc	004014c6 e8 45 f6	CALL	.plt:puts



0040149e	48 83 ec 08	SUB	RSP,0x8
004014a2	b8 00 00 00 00	MOV	EAX,0x0
004014a7	e8 4d ff ff ff	CALL	skip
004014ac	48 85 c0	TEST	RAX,RAX
004014af	75 6e	JNZ	LAB_0040151f
004014b1	48 8b 05 90 22 20 00	MOV	RAX,qword ptr [.bss:stdin]
004014b8	48 39 05 a9 22 20 00	CMP	qword ptr [.bss:infile],RAX
004014bf	75 14	JNZ	LAB_004014d5
004014c1	bf d5 25 40 00	MOV	EDI=>.rodata:s_Error:_Premat
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 Tracking: VT_WallaceSrc.exe_WallaceVersion2.exe

File Edit Window Help

Version Tracking Matches - [Session: VT_WallaceSrc.exe_WallaceVersion2.exe] - 354 matches

Tag	Sess...	St...	Type	Score	Confide...	Votes	# Co...	Mul...	Source Name...	Source Label	Source Ad...	Mul...	Dest Namesp...	Dest Label	Dest Address	Source...	Dest L...	Algorithm
3		✓	Function	1.000	1.000	1	0		Global	__FindPESection	004132d0		Global	__FindPESection	004132b0	117	117	Exact Function Instructi...
2		✓	Data	1.000	1.000	0	0			<No Symbol>	004194ca			<No Symbol>	004194ba	15	15	Exact Data Match
4		✓	Function	1.000	1.000	6	0		Global	FUN_00411da0	00411da0		Global	FUN_00411d80	00411d80	290	290	Exact Function Mnemon...
2		✓	Data	1.000	1.000	0	0			<No Symbol>	0041a028			<No Symbol>	0041a028	8	8	Exact Data Match
2		✓	Data	1.000	1.000	0	0			<No Symbol>	004193fe			<No Symbol>	004193ee	11	11	Exact Data Match
2		✓	Data	1.000	1.000	0	0		Global	s_Stack_memory_c...	00416a14		Global	s_Stack_memory_co...	00416a14	24	24	Exact Data Match
2		✓	Data	1.000	1.000	1	0		Global	u_controlfp_s(((vo...	004171a8		Global	u_controlfp_s(((vo...	004171a8	100	100	Exact Data Match
4		✓	Function	1.000	1.000	0	0		Global	_pre_cpp_init	00411e70		Global	_pre_cpp_init	00411e50	90	90	Exact Function Mnemon...
2		✓	Data	1.000	1.000	0	0		Global	s_Stack_memory_a...	00416ac0		Global	s_Stack_memory_ar...	00416ac0	44	44	Exact Data Match
2		✓	Data	1.000	1.000	1	0		Global	s_Lady_Tottington...	0041688c		Global	s_Lady_Tottington_...	0041688c	16	16	Exact Data Match
2		✓	Data	1.000	1.000	0	0			<No Symbol>	0041956c			<No Symbol>	0041955c	19	19	Exact Data Match
2		✓	Data	1.000	1.000	0	0			<No Symbol>	004194f4			<No Symbol>	004194e4	6	6	Exact Data Match
2		✓	Data	1.000	1.000	0	0			<No Symbol>	004195c2			<No Symbol>	004195b2	15	15	Exact Data Match
3		✓	Function	1.000	1.000	1	0		Global	s_Were_Rabbit_00...	00412200		Global	s_Were_Rabbit_00...	004121e0	13	13	Exact Function Instructi...
2		✓	Data	1.000	1.000	1	0		Global	<No Symbol>	0041687c		Global	<No Symbol>	0041687c	12	12	Exact Data Match
2		✓	Data	1.000	1.000	0	0			<No Symbol>	00419646			<No Symbol>	00419636	28	28	Exact Data Match
2		✓	Data	1.000	1.000	1	0		Global	s_%%s_%%_deployed...	00416830		Global	s_%%s_%%_deployed_...	00416830	22	22	Exact Data Match

Filter: [] Score Filter: 0.000 to 1.000 Confidence Filter: -9.999 to 9.999 Length Filter: 0

Version Tracking Markup Items - [Session: VT_WallaceSrc.exe_WallaceVersion2.exe] - 3 markup items

Status	Source Address	Dest Address	Markup Type	Source Value	Current Dest Value	Original Dest Value
✓✓	00412200	004121e0	Plate Comment	Library Function - Single Match Nam...	Library Function - Single Match Nam...	Library Function - Single Match Nam...
✓✓	00412200	004121e0	Function Name	_NtCurrentTeb	_NtCurrentTeb	_NtCurrentTeb
✓✓	00412200	004121e0	Function Signature	_TEB * _NtCurrentTeb(void)	_TEB * _NtCurrentTeb(void)	_TEB * _NtCurrentTeb(void)

Filter: []

Decompile View Listing View

Source: [_NtCurrentTeb\(\)](#) in /WallaceSrc.exe


```
*****
* Library Function - Single Match
* Name: _NtCurrentTeb
* Library: Visual Studio 2010 Debug
*****
[TEB * _NtCurrentTeb(void)
EAX:4 <RETURN>
_NtCurrentTeb
XREF[1]: _NtCurrentTeb:004111
_NtCurrentTeb:004111
00412200 8b ff MOV EDI,EDI
```





Destination: [_NtCurrentTeb\(\)](#) in /WallaceVersion2.exe

```
*****
* Library Function - Single Match
* Name: _NtCurrentTeb
* Library: Visual Studio 2010 Debug
*****
[TEB * _NtCurrentTeb(void)
EAX:4 <RETURN>
_NtCurrentTeb
XREF[1]: _NtCurrentTeb:004111
_NtCurrentTeb:004111
004121e0 8b ff MOV EDI,EDI
```

Version Tracker - Selecting Correlation Algorithms

Version Tracking Wizard

Select Correlation Algorithm(s) 

S...	Name	P...	Description
<input checked="" type="checkbox"/>	Exact Data Match		Compares data by iterating over all defined data meeting the minimum size requirement in the source pro...
<input checked="" type="checkbox"/>	Exact Function Bytes Match		Compares code by hashing bytes, looking for identical functions. It reports back any that have ONLY ONE...
<input checked="" type="checkbox"/>	Exact Function Instructions Match		Compares code by hashing instructions, looking for identical functions. It reports back any that have ONL...
<input checked="" type="checkbox"/>	Exact Function Mnemonics Match		Compares code by hashing instructions mnemonics, looking for identical functions. It reports back any t...
<input type="checkbox"/>	Exact Symbol Name Match		Compares symbols by iterating over all defined function and data symbols meeting the minimum size req...
<input type="checkbox"/>	Data Reference Match		Matches functions by the accepted data matches they have in common.
<input type="checkbox"/>	Combined Function and Data Reference ...		Matches functions based on the accepted data and function matches they have in common.
<input type="checkbox"/>	Function Reference Match		Matches functions by the accepted function matches they have in common.
<input type="checkbox"/>	Duplicate Data Match		Compares data by iterating over all defined data meeting the minimum size requirement in the source pro...
<input type="checkbox"/>	Duplicate Function Instructions Match		Compares code by hashing instructions (masking off operands), looking for identical functions. It report...
<input type="checkbox"/>	Duplicate Exact Symbol Name Match		Compares symbols by iterating over all defined function and data symbols meeting the minimum size req...
<input type="checkbox"/>	Similar Symbol Name Match		Compares symbols by iterating over all defined function and data symbols meeting the minimum size req...
<input type="checkbox"/>	Similar Data Match		Compares data by iterating over all defined data meeting the minimum size requirement in the source pro...

<< Back Next >> Finish Cancel

Version Tracker - Function Name Ported

Version Tracking: VT_WallaceSrc.exe_WallaceVersion2.exe

File Edit Window Help

Version Tracking Matches - [Session: VT_WallaceSrc.exe_WallaceVersion2.exe] - 1 matches (of 355)

Tag	Sess...	St...	Type	Score	Confide...	Votes	# Co...	Mul...	Source Name...	Source Label	Source Add...	Mul...	Dest Namesp...	Dest Label	Dest Address	Source...	Dest L...	Algorithm
			Function	0.000	0.000	1	0		Global	deployGadget	004118f0		Global	deployGadget	004118c0	250	261	Implied Match

Filter: deployGad Score Filter: 0.000 to 1.000 Confidence Filter: -9.999 to 9.999 Length Filter: 0

Version Tracking Markup Items - [Session: VT_WallaceSrc.exe_WallaceVersion2.exe] - 2 markup items

Status	Source Address	Dest Address	Markup Type	Source Value	Current Dest Value	Original Dest Value
✓	004118f0	004118c0	Function Name	deployGadget	deployGadget	FUN_004118c0
	004118f0	004118c0	Function Signature	void * __stdcall deployGadget(void)	undefined __stdcall deployGadget(vo...	undefined __stdcall deployGadget(vo...

Filter:

Decompile View Listing View

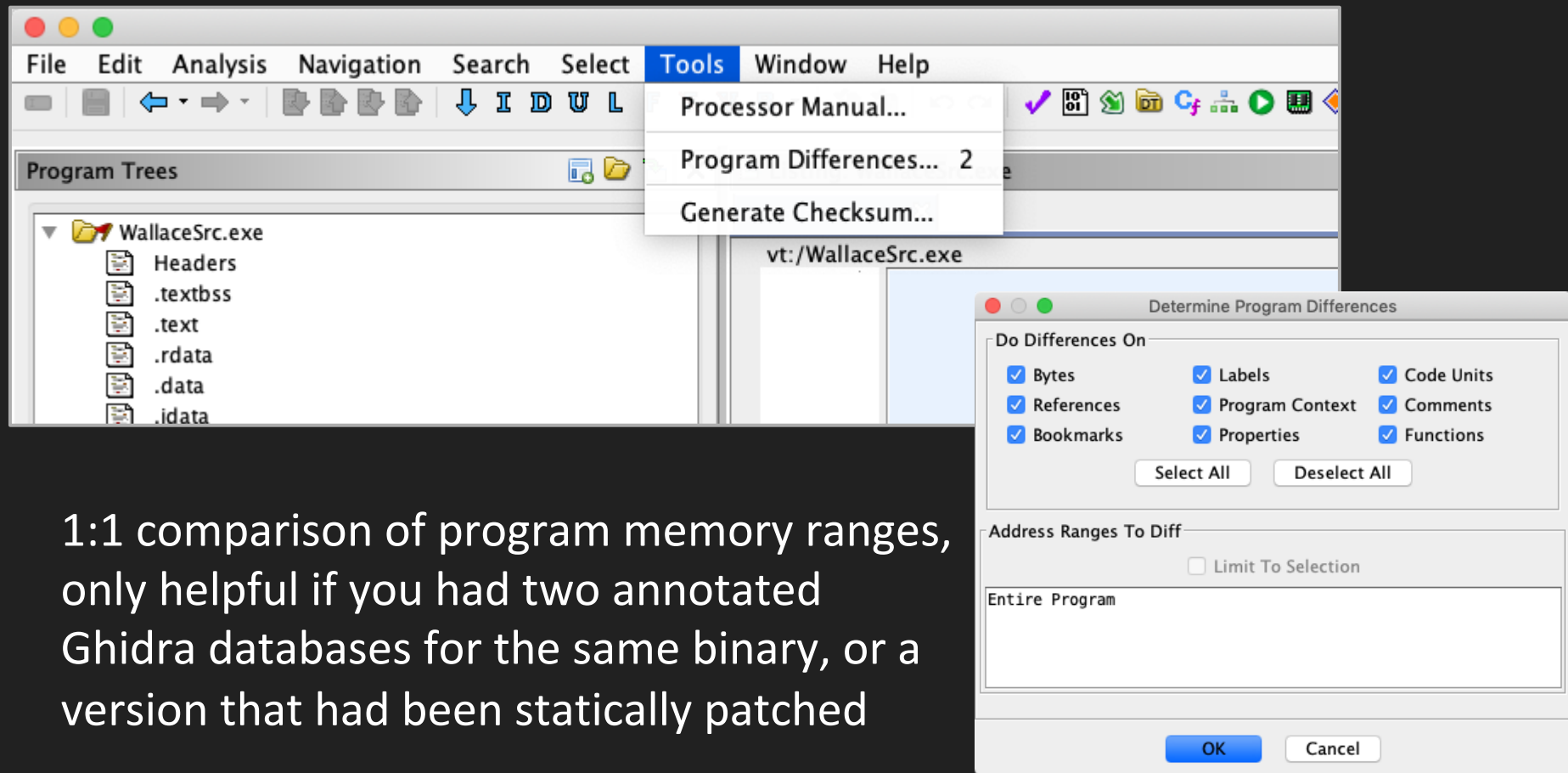
Source: deployGadget() in /WallaceSrc.exe

```
***** FUNCTION *****
void * __stdcall deployGadget(void)
EAX:4
int EAX:4 strcmp_match XREF [1]: 004119a5(w)
undefined4 Stack[-0x8]:4 local_8 XREF [2]: 0041193d(w)
undefined4 Stack[-0x10]:4 local_10 XREF [2]: 0041197b(w)
undefined4 Stack[-0x18]:4 local_18 XREF [3]: 00411924(*)
int Stack[-0x24]:4 pp XREF [6]: 004119cb(R)
void * Stack[-0xf0]:4 gadget XREF [3]: 00411988(w)
undefined4 * [2] Stack[-0x104... gadgetLocal XREF [4]: 004119b5(R)
XREF [1]: deployGadget:0041117
```

Destination: FUN_004118c0() in /WallaceVersion2.exe

```
***** FUNCTION *****
undefined __stdcall deployGadget(void)
AL:1
undefined4 Stack[-0x8]:4 local_8 XREF [2]: 0041198d(w)
undefined4 Stack[-0x10]:4 local_10 XREF [2]: 0041194b(w)
undefined4 Stack[-0x18]:4 local_18 XREF [3]: 004118f4(*)
undefined4 Stack[-0x24]:4 local_24 XREF [7]: 004119a6(R)
undefined4 Stack[-0xf0]:4 local_f0 XREF [3]: 00411958(w)
undefined4 Stack[-0x104... local_104 XREF [4]: 00411990(w)
XREF [1]: deployGadget:0041116
```


Program Differences



Program Differences

WallaceSrc.exe

vt:/WallaceSrc.exe

void * Stack[-0xf0]:4 gadget

undefined4 Stack[-0xfc]:4 local_fc

undefined4 *2] Stack[-0x104... gadgetLocal

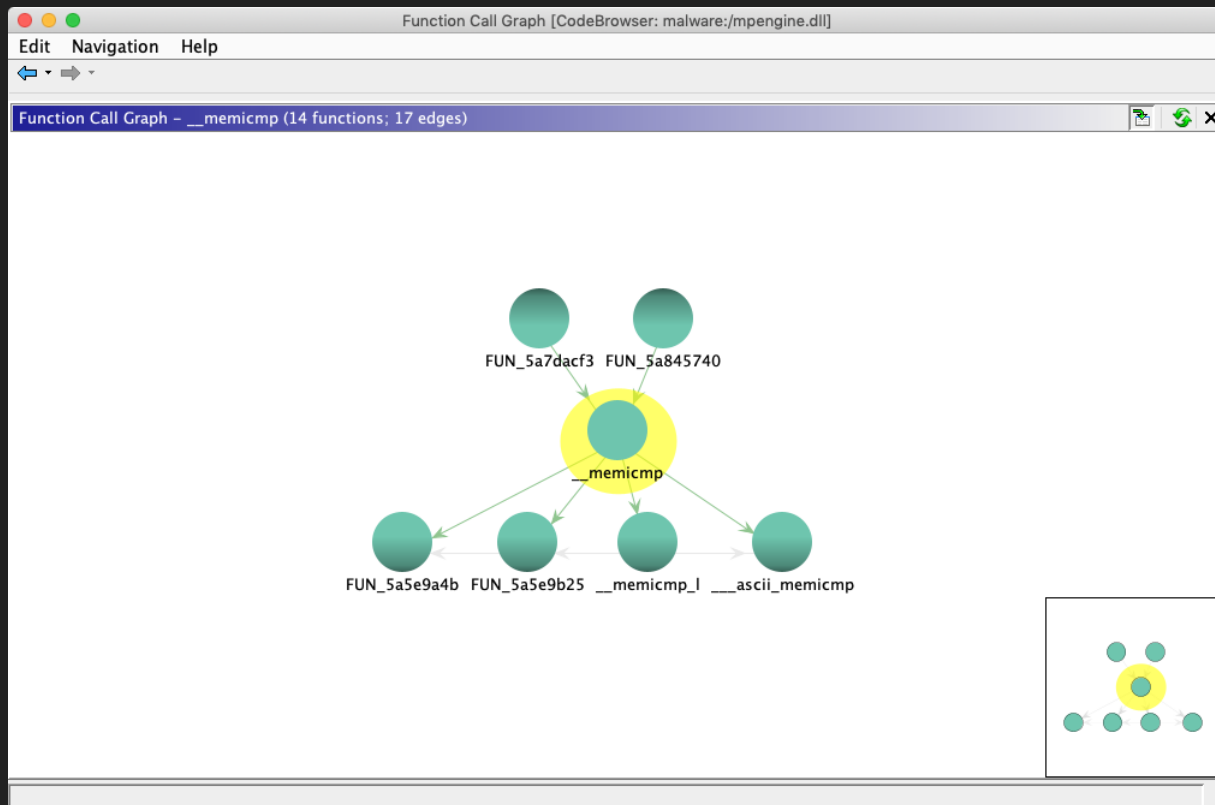
deployGadget

004118f0	55	PUSH	EBP
004118f1	8b ec	MOV	EBP,ESP
004118f3	6a ff	PUSH	-0x1
004118f5	68 2e 4b 41 00	PUSH	LAB_00414b2e
004118fa	64 a1 00 00 00 00	MOV	EAX,FS:[0x0]
00411900	50	PUSH	EAX
00411901	81 ec f4 00 00 00	SUB	ESP,0xf4
00411907	53	PUSH	EBX
00411908	56	PUSH	ESI
00411909	57	PUSH	EDI
0041190a	8d bd 00	LEA	EDI=>gadgetLocal,[0x

vt:/WallaceVersion2.exe

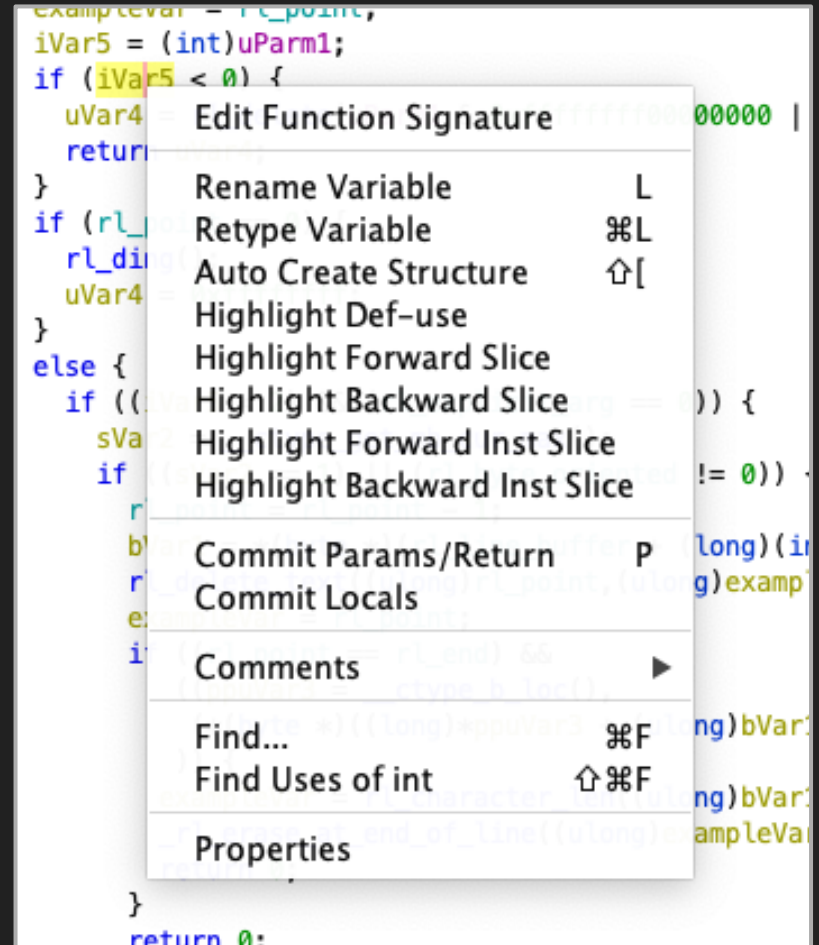
004118f1	33 c5	XOR	EAX,EBP
004118f3	50	PUSH	EAX
004118f4	8d 45 f4	LEA	EAX=>local_10,[EBP + -0xc]
004118f7	64 a3 00 00 00 00	MOV	FS:[0x0],EAX
004118fd	6a 10	PUSH	0x10
004118ff	e8 b4 f8 ff ff	CALL	operator_new
00411904	83 c4 04	ADD	ESP,0x4
00411907	89 85 14 ff ff ff	MOV	dword ptr [local_f0 + EBP],EAX

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

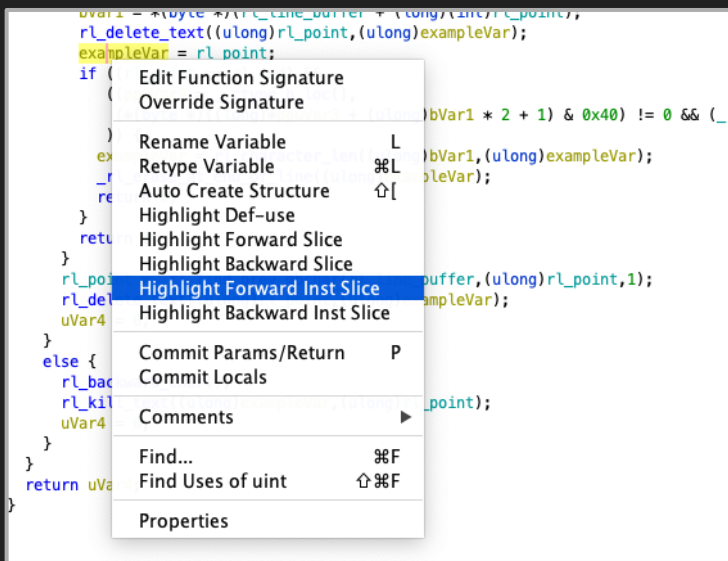


Decompiler Slicing

```
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;
```

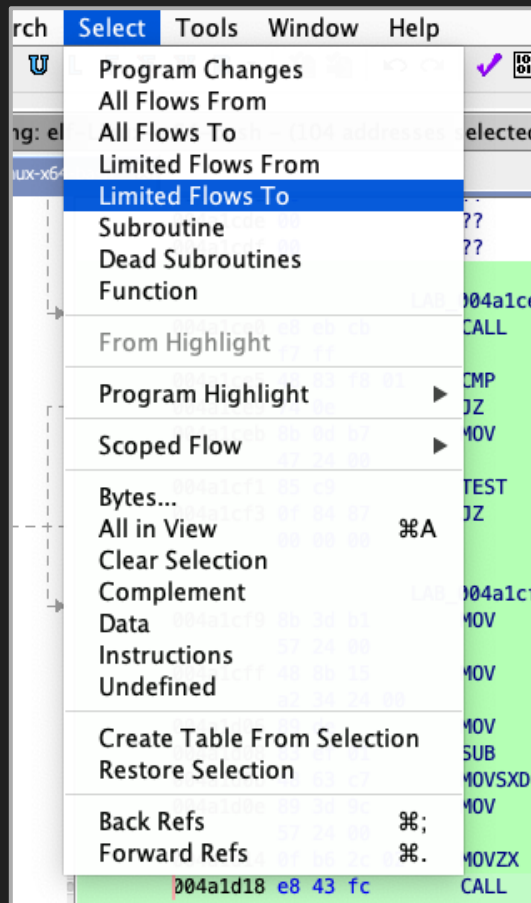
Middle Click



Highlight Forward
Inst Slice

```
uVar1 = *(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);
```

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

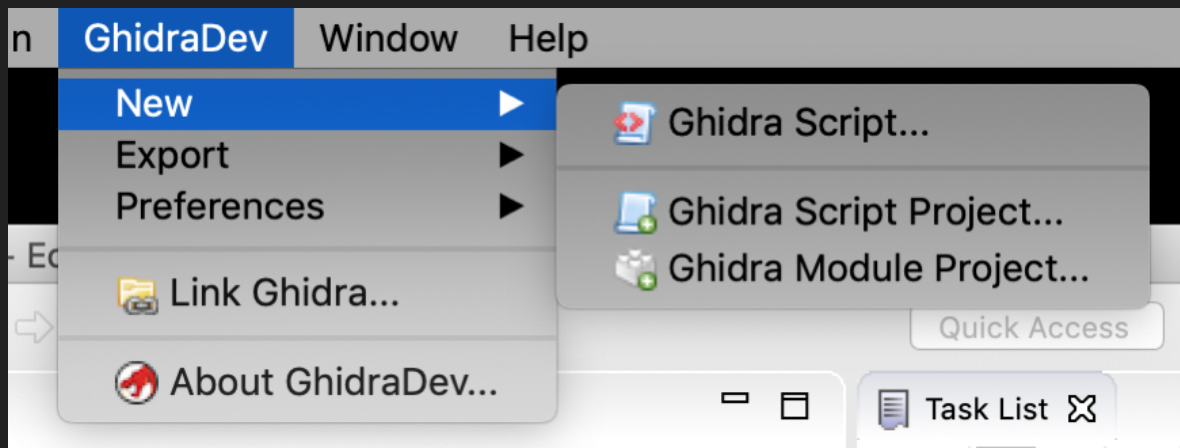
- Programmatic access to binary file formats
- P-code interaction
- Decompiler API
- C header parsing
- Interface for graphing (implementation not included)
- Cyclomatic complexity

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

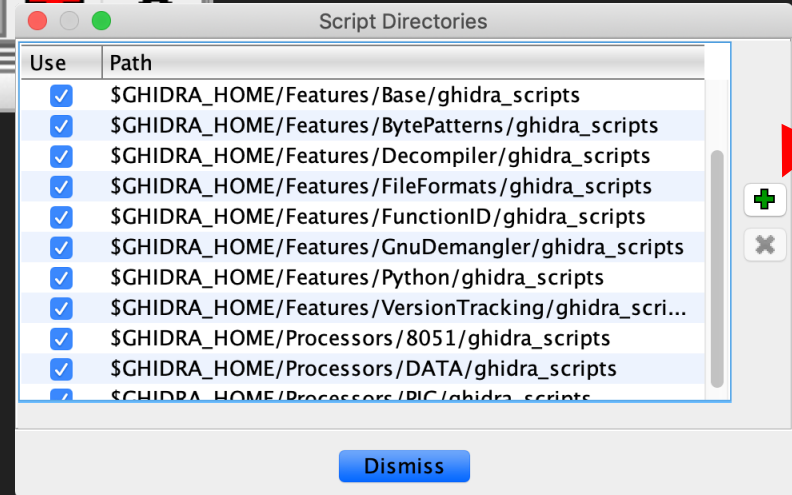


Click the “Display Script Manager” button to open the Script Manager Window



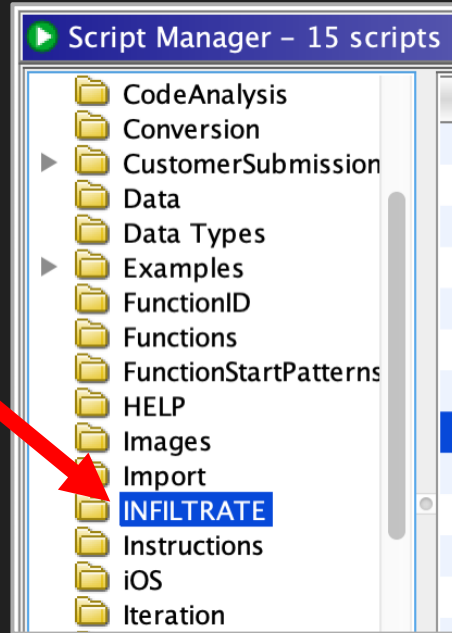
Click the “Script Directories” button on the Script Click the “Display Script Manager” button to open the Script Manager Window

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



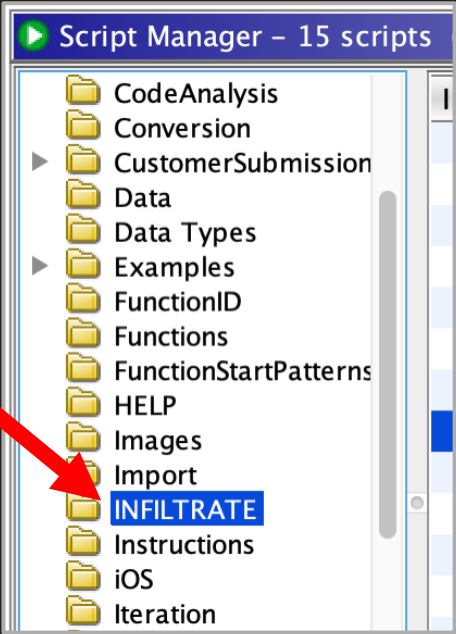
Running Script Demos

Find the
“INFILTRATE” folder
in the script
manager

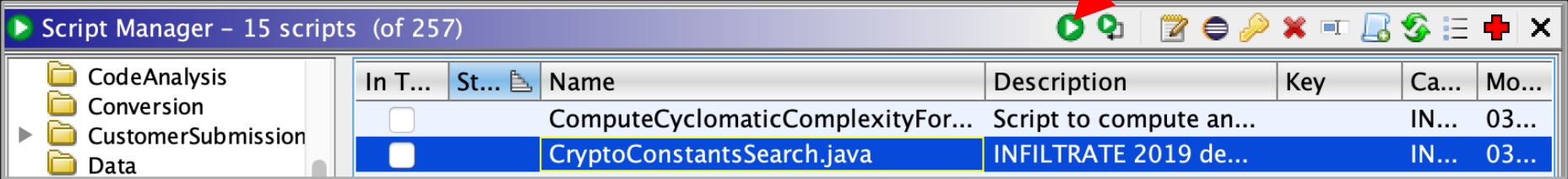


Running Script Demos

Find the
“INFILTRATE” folder
in the script
manager

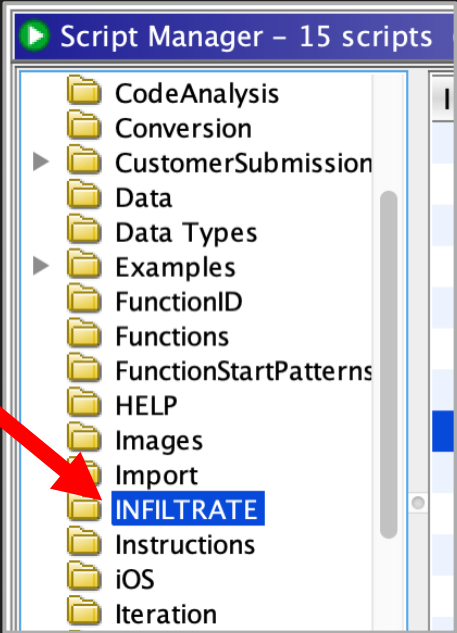


Choose a script and click “Run Script”
to run

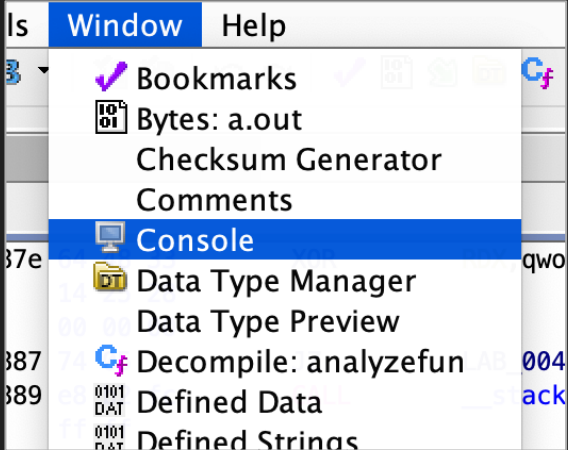


Running Script Demos

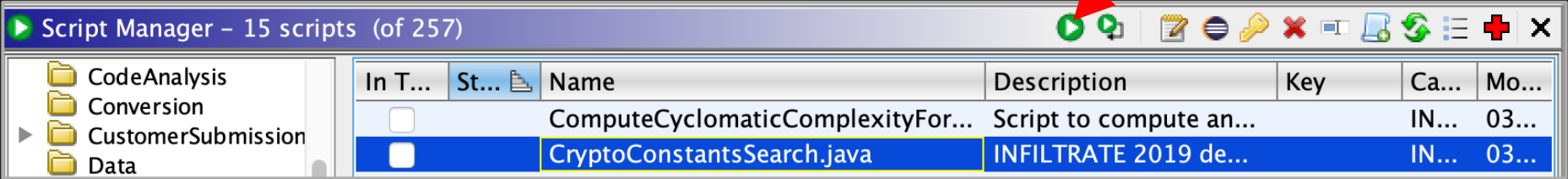
Find the
“INFILTRATE” folder
in the script
manager



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

```
Console - Scripting
HelloWorld.java> Running...
HelloWorld.java> Hello world
HelloWorld.java> _init
HelloWorld.java> FUN_00400ad0
HelloWorld.java> getenv
HelloWorld.java> __errno_location
HelloWorld.java> strcpy
HelloWorld.java> puts
HelloWorld.java> write
HelloWorld.java> __stack_chk_fail
HelloWorld.java> alarm
HelloWorld.java> close
HelloWorld.java> read
HelloWorld.java> __libc_start_main
```

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

public class HelloWorld extends GhidraScript {

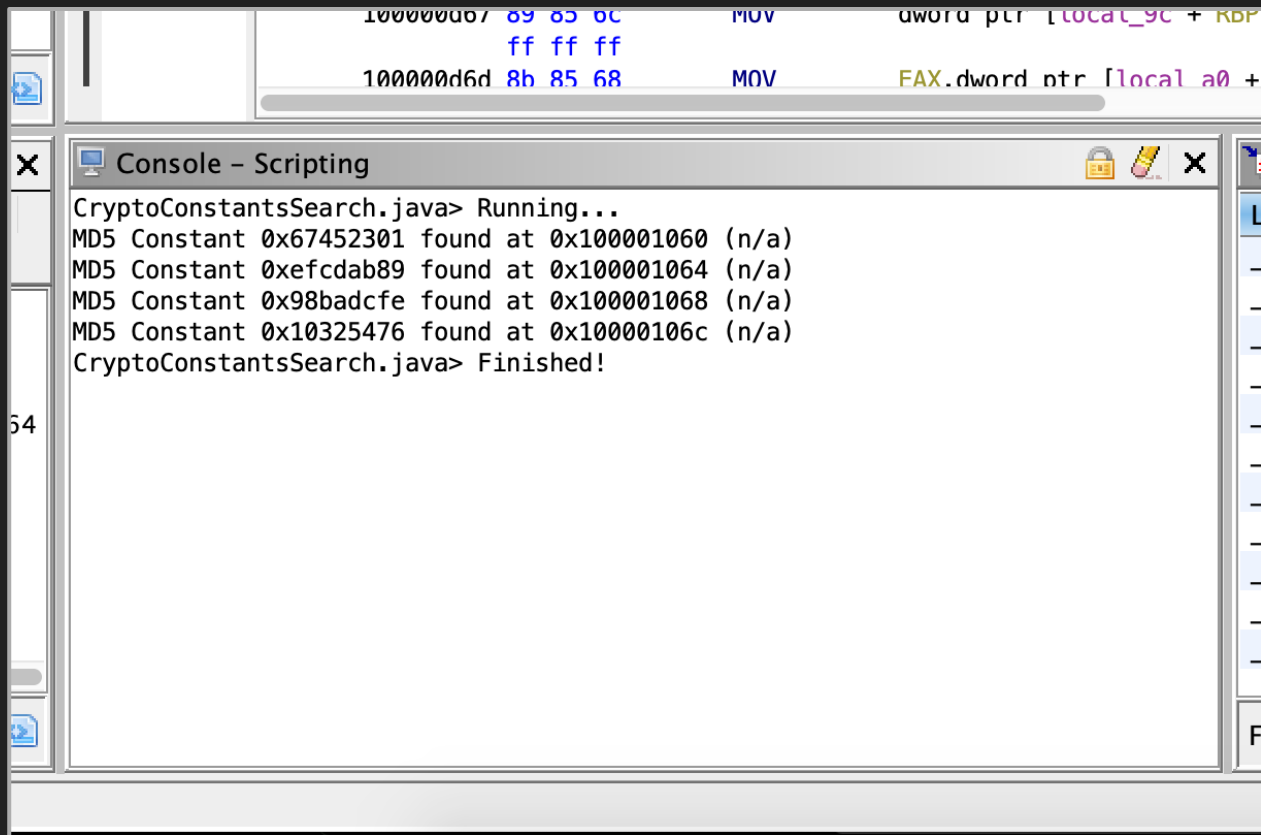
    public void run() throws Exception {
        println("Hello world");

        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



The screenshot shows a debugger window with two panes. The top pane displays assembly instructions with their addresses and operands. The bottom pane is a console window titled 'Console - Scripting' showing the output of a Java program.

Assembly instructions:

Address	Disassembly
1000000b7	MOV dword ptr [local_9c + KBP, 89 85 0c ff ff ff]
100000d6d	MOV FAX.dword ptr [local_a0 +

Console output:

```
CryptoConstantsSearch.java> Running...
MD5 Constant 0x67452301 found at 0x100001060 (n/a)
MD5 Constant 0xefcdab89 found at 0x100001064 (n/a)
MD5 Constant 0x98badcfe found at 0x100001068 (n/a)
MD5 Constant 0x10325476 found at 0x10000106c (n/a)
CryptoConstantsSearch.java> Finished!
```

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

elf-Linux-lib-x64.so - Function Cyclomatic Complexity

Location	Function Name	Cyclomatic Complexity
0012b610	cairo_device_flush	7
001c5ca0	FUN_001c5ca0	7
0019dfc0	FUN_0019dfc0	7
001614e0	cairo_region_destroy	7
001ba8b0	cairo_gl_surface_create_for_egl	7
0016ce70	FUN_0016ce70	7
001b3640	FUN_001b3640	7
001b40a0	FUN_001b40a0	7
00172ee0	FUN_00172ee0	7
001a70d0	FUN_001a70d0	7
001a3b20	FUN_001a3b20	7
0012f1c0	FUN_0012f1c0	7
0019c350	FUN_0019c350	8
001a59f0	FUN_001a59f0	8
0018e490	FUN_0018e490	8
001ce780	FUN_001ce780	8
001511f0	cairo_mesh_pattern_set_control_point	8
00123710	cairo_clip_extents	8
0016a010	FUN_0016a010	8
001dba60	FUN_001dba60	8
001d0070	FUN_001d0070	8
0015c250	FUN_0015c250	8
0013cd50	FUN_0013cd50	8
00183920	FUN_00183920	8
001c9140	FUN_001c9140	8
0015d120	FUN_0015d120	8
001ad4b0	FUN_001ad4b0	8
001cdd50	FUN_001cdd50	8
001b9b70	cairo_gl_surface_set_size	8

Filter:

Dismiss

support — -bash — 1

```
INFO FUN_001d0070 complexity: 8 (GhidraScript)
INFO FUN_001d0170 complexity: 8 (GhidraScript)
INFO FUN_001d0260 complexity: 18 (GhidraScript)
INFO FUN_001d05a0 complexity: 29 (GhidraScript)
INFO FUN_001d0b30 complexity: 2 (GhidraScript)
INFO FUN_001d0bb0 complexity: 19 (GhidraScript)
INFO FUN_001d1060 complexity: 17 (GhidraScript)
INFO FUN_001d1490 complexity: 36 (GhidraScript)
INFO FUN_001d1c50 complexity: 6 (GhidraScript)
INFO FUN_001d1dc0 complexity: 28 (GhidraScript)
INFO FUN_001d2260 complexity: 22 (GhidraScript)
INFO FUN_001d26a0 complexity: 34 (GhidraScript)
INFO FUN_001d2db0 complexity: 16 (GhidraScript)
INFO FUN_001d3040 complexity: 24 (GhidraScript)
INFO FUN_001d3440 complexity: 37 (GhidraScript)
INFO FUN_001d3930 complexity: 29 (GhidraScript)
INFO FUN_001d3c40 complexity: 23 (GhidraScript)
INFO FUN_001d3f50 complexity: 27 (GhidraScript)
INFO FUN_001d4340 complexity: 191 (GhidraScript)
INFO FUN_001d6c20 complexity: 16 (GhidraScript)
INFO cairo_pdf_surface_create_for_stream complexity: 3 (GhidraScript)
INFO cairo_pdf_surface_create complexity: 3 (GhidraScript)
INFO cairo_pdf_surface_restrict_to_version complexity: 3 (GhidraScript)
INFO cairo_pdf_get_versions complexity: 3 (GhidraScript)
INFO cairo_pdf_version_to_string complexity: 2 (GhidraScript)
INFO cairo_pdf_surface_set_size complexity: 4 (GhidraScript)
INFO FUN_001d7070 complexity: 1 (GhidraScript)
INFO FUN_001d7080 complexity: 1 (GhidraScript)
INFO FUN_001d7090 complexity: 1 (GhidraScript)
INFO FUN_001d70b0 complexity: 1 (GhidraScript)
INFO FUN_001d70d0 complexity: 2 (GhidraScript)
INFO FUN_001d71c0 complexity: 2 (GhidraScript)
INFO FUN_001d7230 complexity: 2 (GhidraScript)
INFO FUN_001d72a0 complexity: 6 (GhidraScript)
INFO FUN_001d7370 complexity: 18 (GhidraScript)
INFO FUN_001d76c0 complexity: 14 (GhidraScript)
INFO FUN_001d7820 complexity: 1 (GhidraScript)
INFO FUN_001d7870 complexity: 1 (GhidraScript)
INFO FUN_001d78d0 complexity: 14 (GhidraScript)
INFO FUN_001d7950 complexity: 1 (GhidraScript)
INFO FUN_001d7980 complexity: 5 (GhidraScript)
INFO FUN_001d79b0 complexity: 30 (GhidraScript)
INFO FUN_001d8090 complexity: 3 (GhidraScript)
INFO FUN_001d8140 complexity: 10 (GhidraScript)
INFO FUN_001d8280 complexity: 2 (GhidraScript)
```

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](https://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

00401f16 eb 03

JMP

LAB_00401f1b

BRANCH *[ram]0x401f1b:8

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;  
}
```

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 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 < 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 < 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 < 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 < 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 < 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;

0f 31

RDTSC

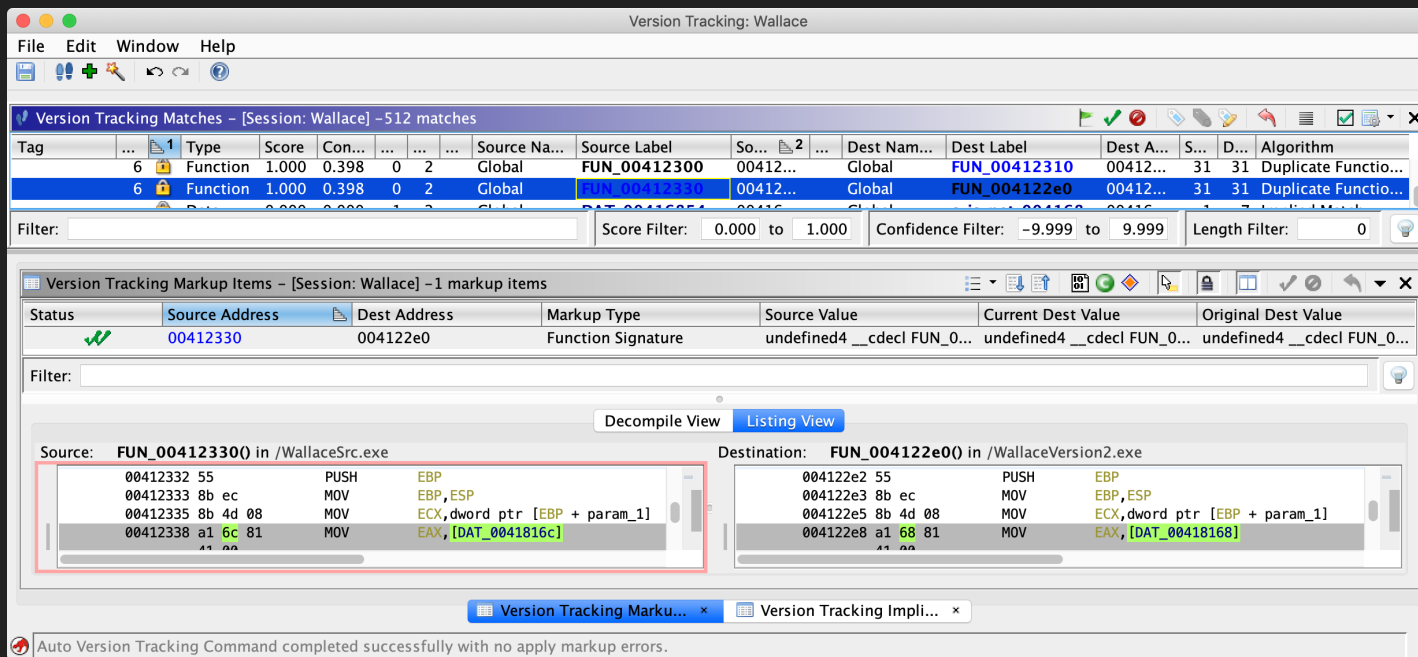
\$U9c60:8 = CALLOTHER "rdtsc"
EDX = SUBPIECE \$U9c60, 4:4
EAX = SUBPIECE \$U9c60, 0:4

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_SBBORROW 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_SBBORROW 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



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

Demo material: github.com/0xAlexei/INFILTRATE2019

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

Official NSA sites:

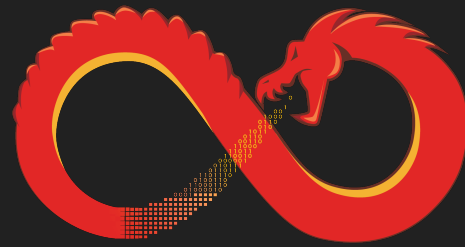
github.com/NationalSecurityAgency/ghidra

ghidra-sre.org

Ghidra training: ringzer0.training

Acknowledgements:

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

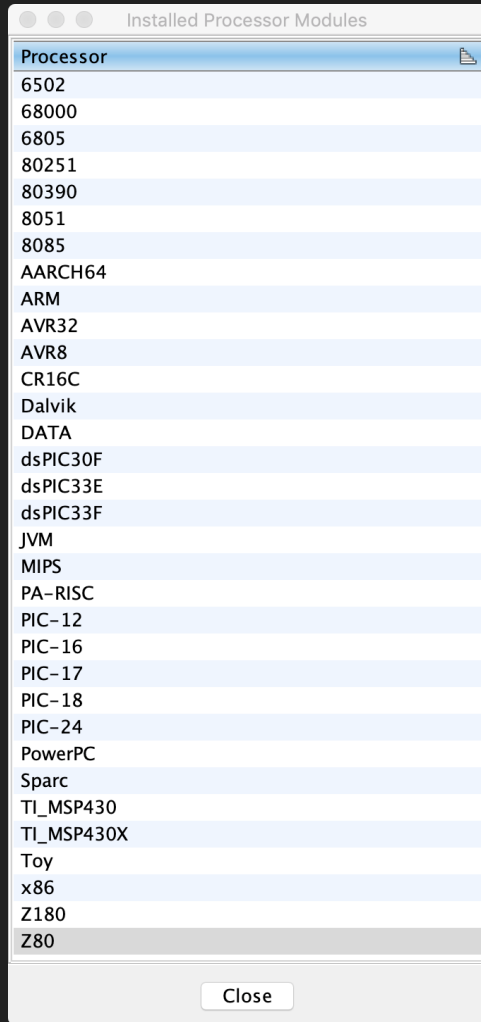
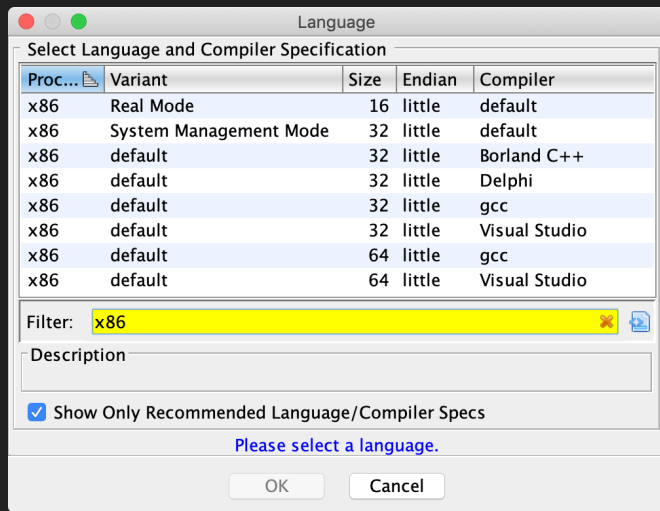


A stylized red dragon is depicted in a circular, looping pose, reminiscent of the Yin-Yang symbol. The dragon's body forms the outer ring, with its head facing right and its tail on the left. The interior of the circle is dark grey. A trail of yellow binary code (0s and 1s) starts from the bottom left and curves upwards towards the dragon's mouth. At the bottom left, there is a grid of small red squares that fades into the binary code. The word "Appendix" is written in white, bold, sans-serif font across the center of the circle.

Appendix

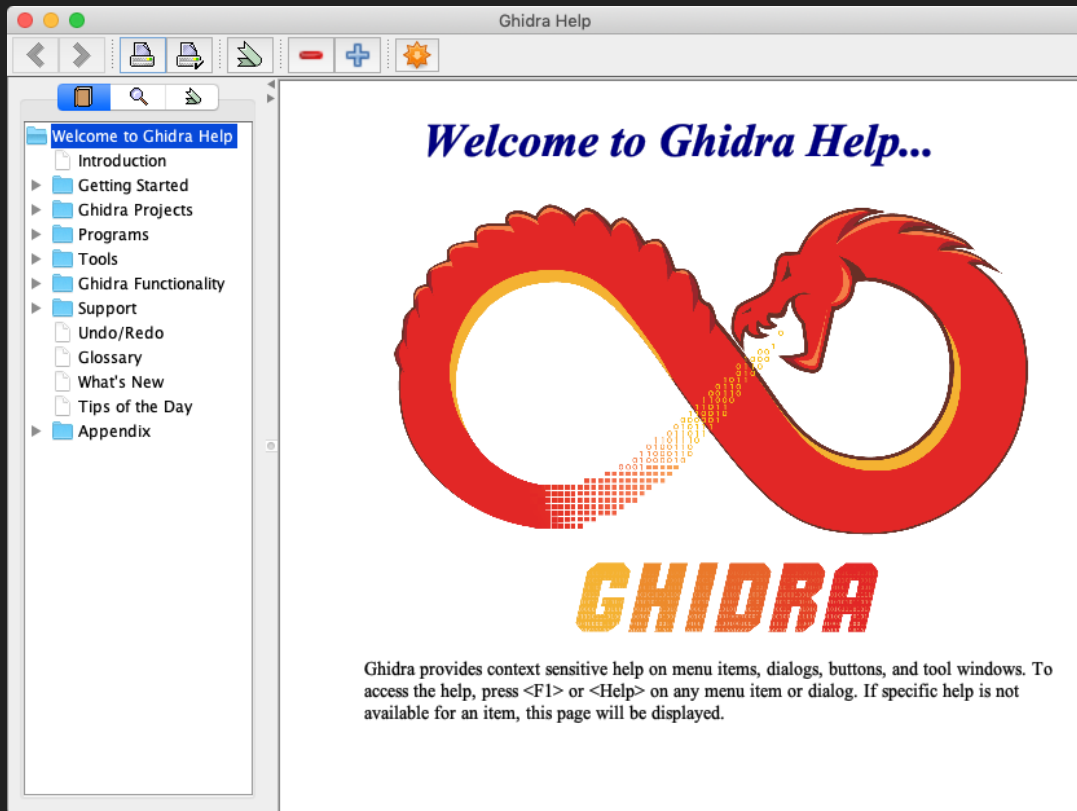
Architecture Support

- 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



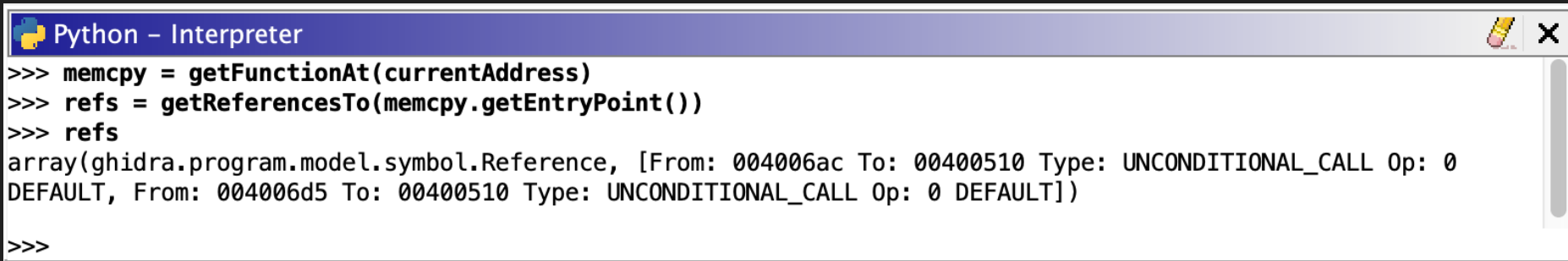
Documentation

- 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



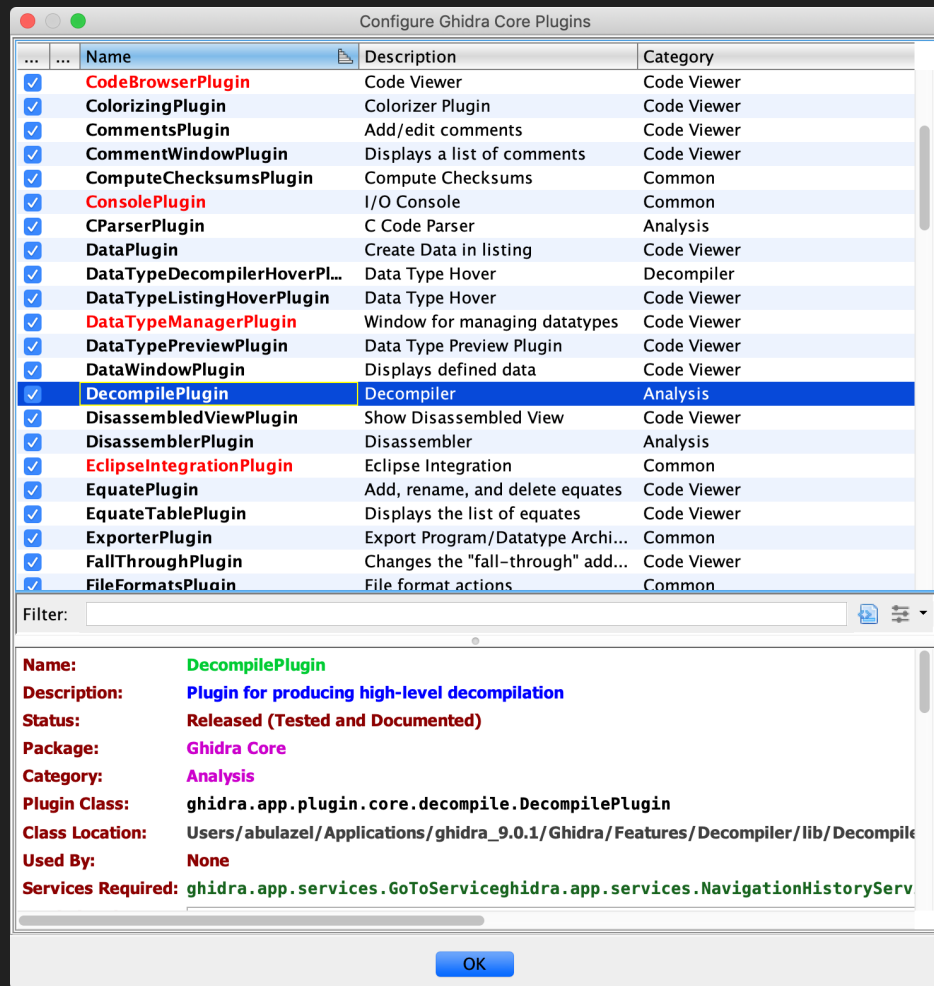
```
Python - Interpreter
>>> memcpy = getFunctionAt(currentAddress)
>>> refs = getReferencesTo(memcpy.getEntryPoint())
>>> refs
array(ghidra.program.model.symbol.Reference, [From: 004006ac To: 00400510 Type: UNCONDITIONAL_CALL Op: 0
DEFAULT, From: 004006d5 To: 00400510 Type: UNCONDITIONAL_CALL Op: 0 DEFAULT])
>>>
```

Script vs. Plugin vs. Extension

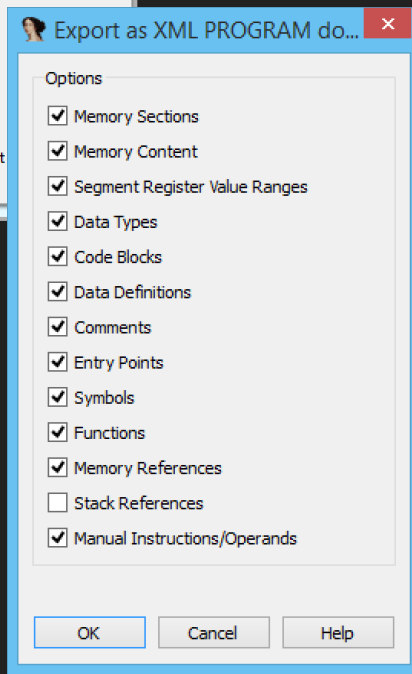
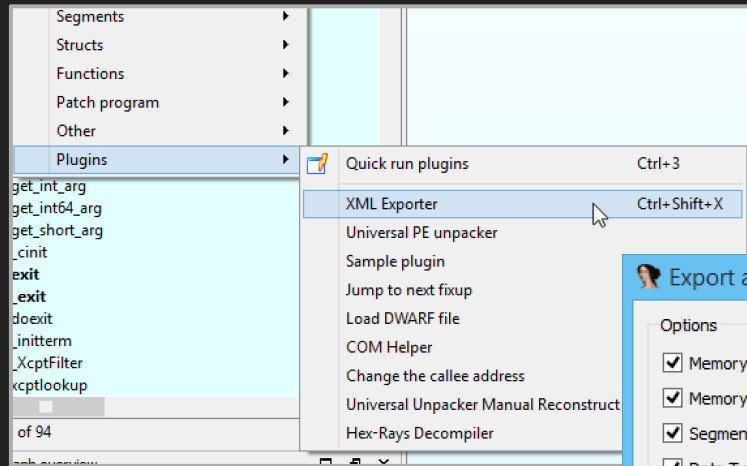
- **Scripts:** do a single thing with defined start and end point
- **Plugins:** base components for everything you interact with in Ghidra, such as UI panes
- **Extensions:** sets of plugins for extended functionality, e.g., custom binary format loaders, interfaces to external tools, libraries for use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

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`

A stylized red dragon is depicted in a circular, coiled pose against a dark grey background. The dragon's body forms a large circle, with its head facing right and its tail on the left. The dragon's scales are a vibrant red, and its eyes are yellow. A trail of yellow binary code (0s and 1s) follows the curve of the dragon's body, starting from the bottom left and moving towards the top right. The word "Links" is written in a bold, white, sans-serif font across the center of the dragon's body.

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