

Reverse Engineering with Ghidra

Paolo Falcarin

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics
paoletto.falcarin@unive.it

CM0626 – Software Security
CM0631-2 – Software Security

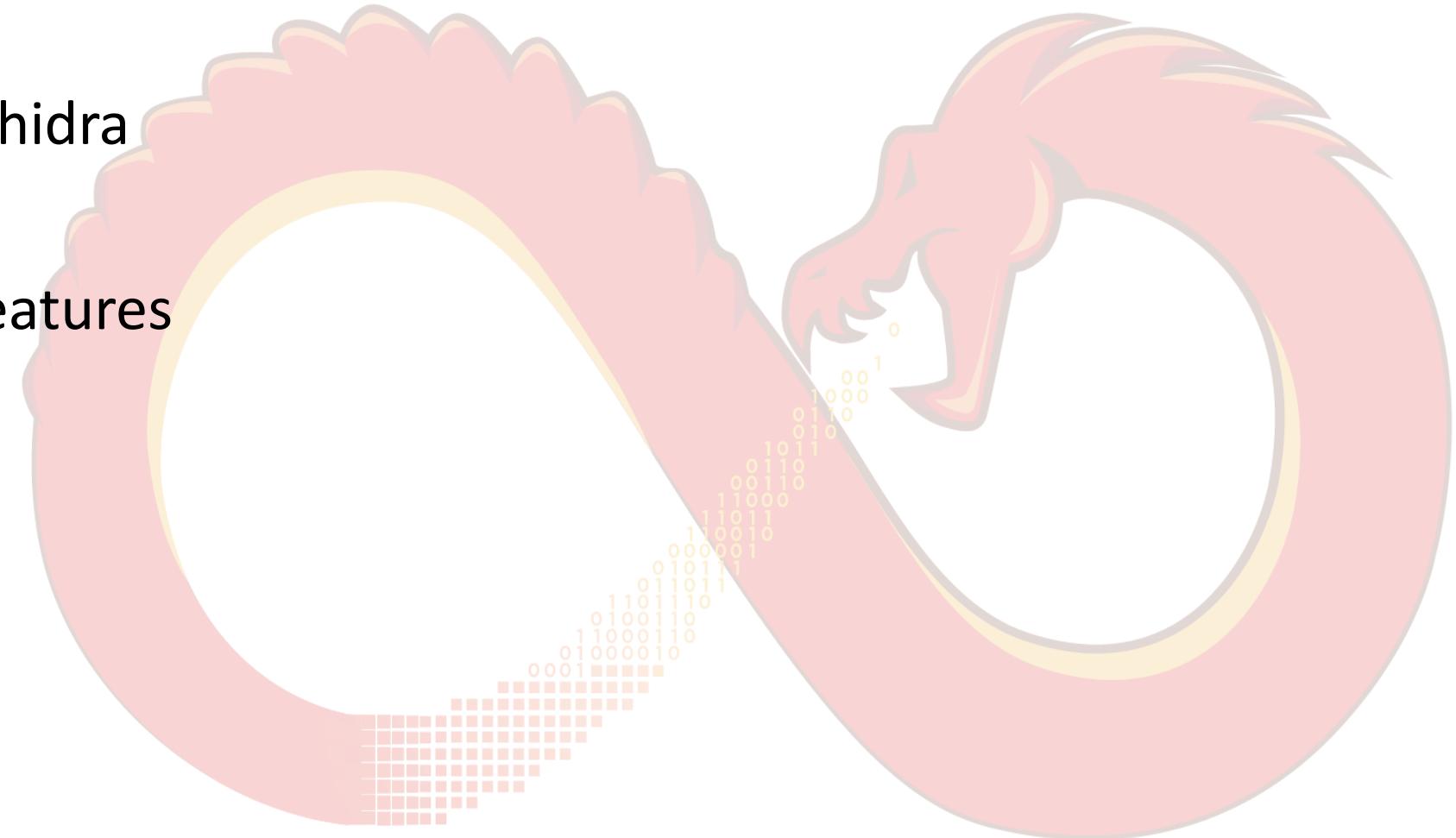
25 February 2025



Outline



- Ghidra Overview
- Static Attack with Ghidra
- Ghidra Scripting
- Ghidra Advanced Features

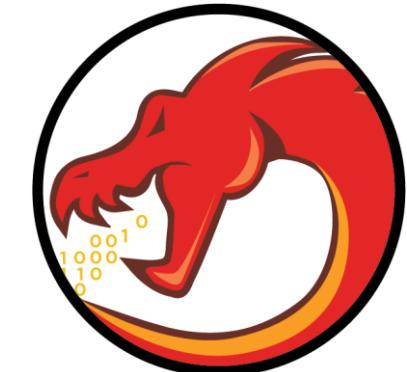
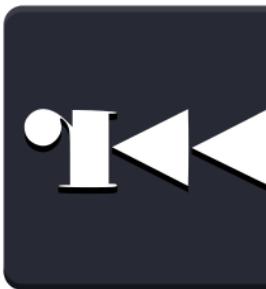


Reverse Engineering Tools



Ca' Foscari
University
of Venice

- IDA Pro from Hex Rays
 - Historical commercial leader
 - <https://hex-rays.com/ida-pro/>
- BinaryNinja
 - Emerging commercial leader
 - <https://binary.ninja/>
- Radare2
 - Kali-Linux Open-Source Tool
 - <https://rada.re/n/radare2.html>
- Ghidra
 - Emerging Multi-platform Open-Source Tool
 - Published in early 2019



Other Reverse Engineering Tools



Ca' Foscari
University
of Venice

- ODA online disassembler
 - Many Hardware architectures supported
 - <https://syscall7.com/oda/>
- Hopper
 - For Mac and Linux
 - <https://www.hopperapp.com>
- Angr
 - open-source binary analysis platform for Python.
 - It combines both static and dynamic symbolic ("concolic") analysis, providing tools to solve a variety of tasks.
 - <https://angr.io/>



Ghidra



Ca' Foscari
University
of Venice

- 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

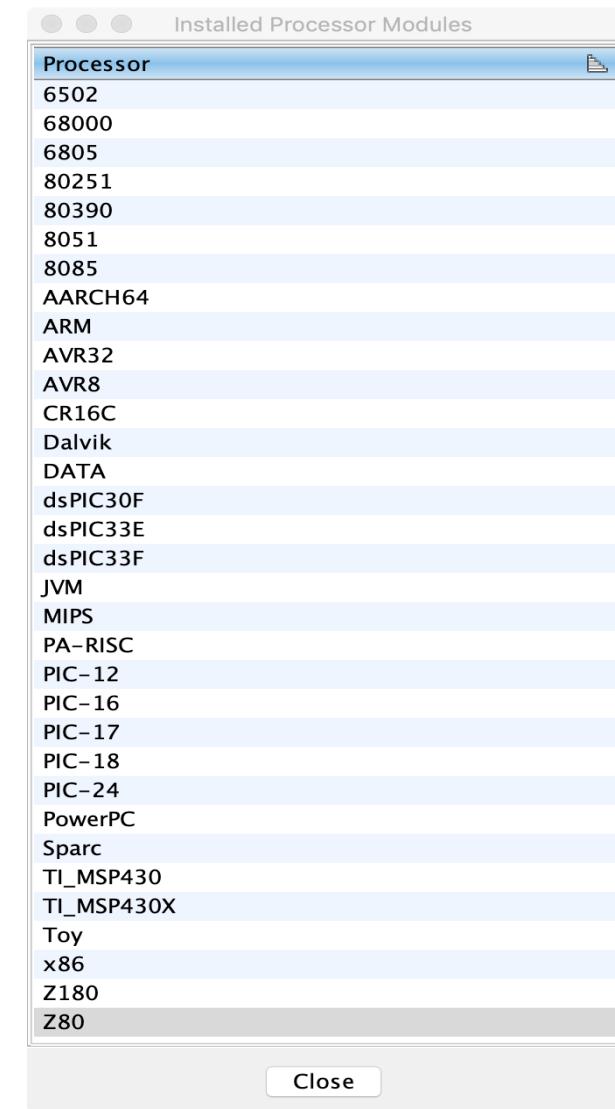
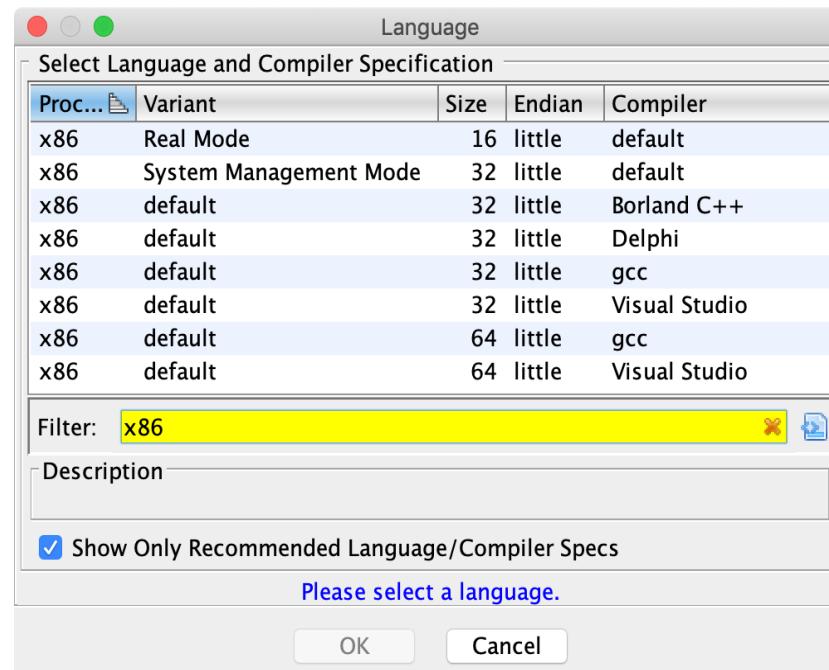


```
$ grep -r "1999" * --include *.java
src/Generic-src/ghidra/util/exception/NotYetImplementedException.java: * @version 1999/02/05
src/SoftwareModeling-src/ghidra/program/model/address/AddressOutOfBoundsException.java: * @version 1999-03-31
src/SoftwareModeling-src/ghidra/program/model/scalar/ScalarOverflowException.java: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

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

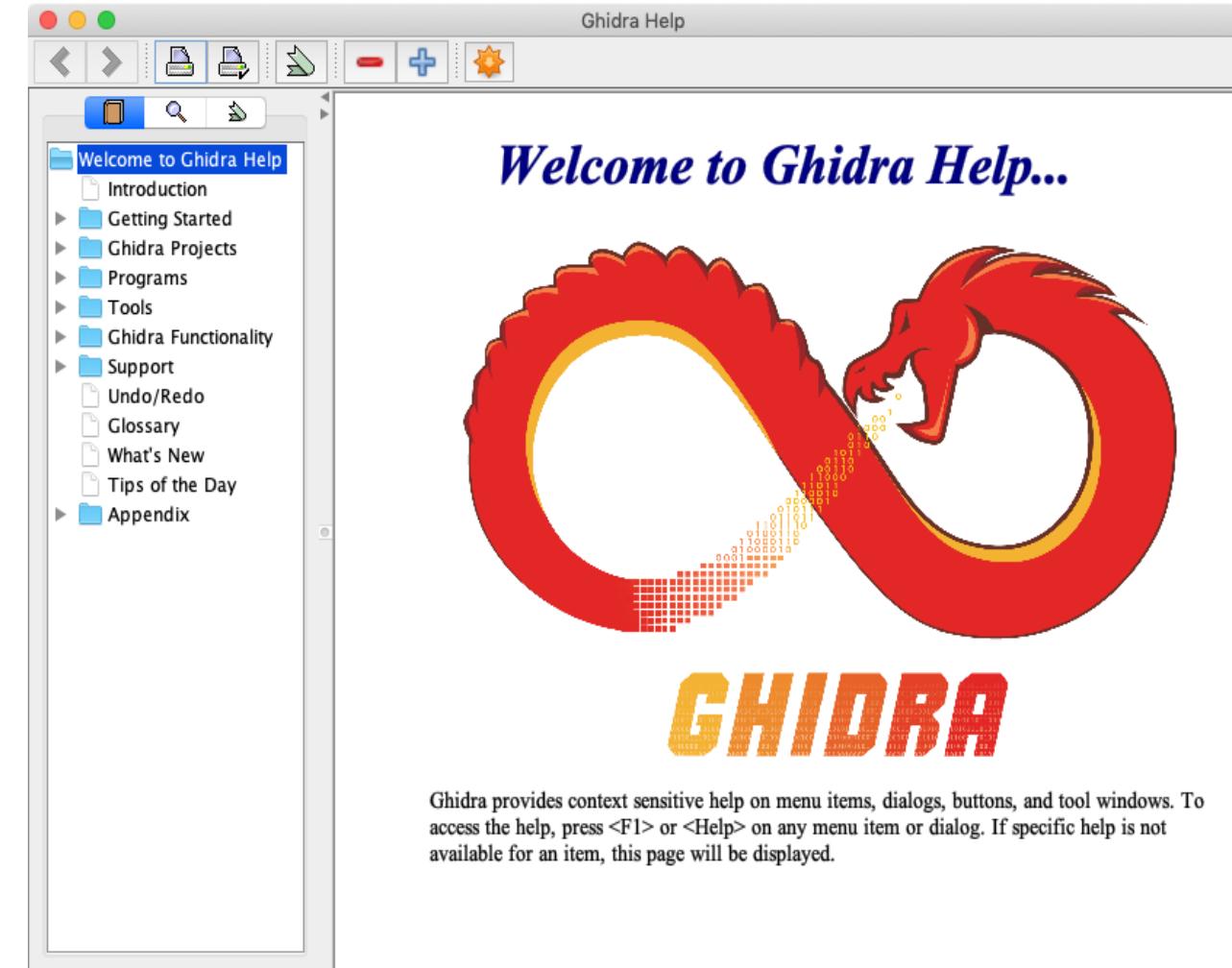


Documentation



Ca' Foscari
University
of Venice

- Help > Contents
- F1 or Help key while pointing at any field or menu option
- docs directory
- JavaDoc
- Several classes
- P-code



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

Configure Ghidra Core Plugins

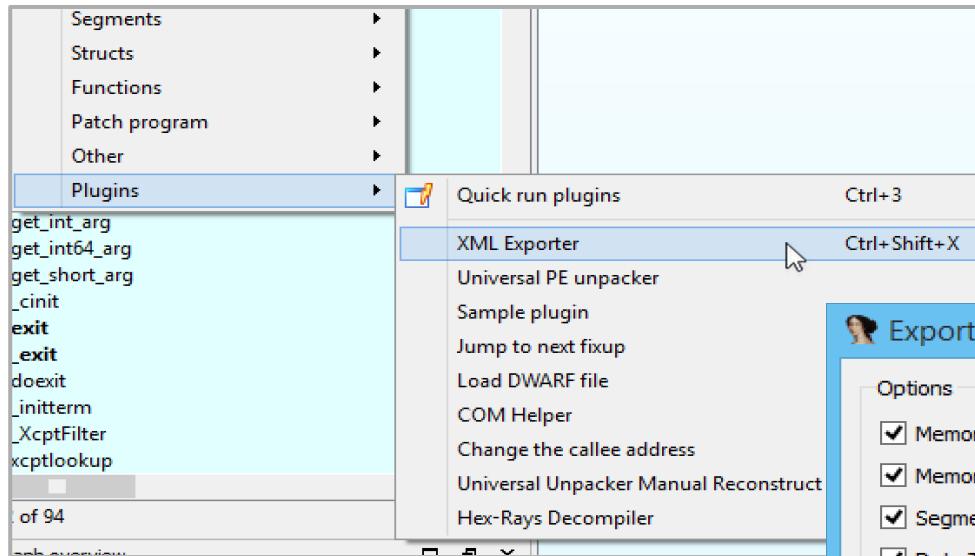
...	Name	Description	Category
<input checked="" type="checkbox"/>	CodeBrowserPlugin	Code Viewer	Code Viewer
<input checked="" type="checkbox"/>	ColorizingPlugin	Colorizer Plugin	Code Viewer
<input checked="" type="checkbox"/>	CommentsPlugin	Add/edit comments	Code Viewer
<input checked="" type="checkbox"/>	CommentWindowPlugin	Displays a list of comments	Code Viewer
<input checked="" type="checkbox"/>	ComputeChecksumsPlugin	Compute Checksums	Common
<input checked="" type="checkbox"/>	ConsolePlugin	I/O Console	Common
<input checked="" type="checkbox"/>	CParserPlugin	C Code Parser	Analysis
<input checked="" type="checkbox"/>	DataPlugin	Create Data in listing	Code Viewer
<input checked="" type="checkbox"/>	DataTypeDecompilerHoverPl...	Data Type Hover	Decompiler
<input checked="" type="checkbox"/>	DataTypeListingHoverPlugin	Data Type Hover	Code Viewer
<input checked="" type="checkbox"/>	DataTypeManagerPlugin	Window for managing datatypes	Code Viewer
<input checked="" type="checkbox"/>	DataTypePreviewPlugin	Data Type Preview Plugin	Code Viewer
<input checked="" type="checkbox"/>	DataWindowPlugin	Displays defined data	Code Viewer
<input checked="" type="checkbox"/>	DecompilePlugin	Decompiler	Analysis
<input checked="" type="checkbox"/>	DisassembledViewPlugin	Show Disassembled View	Code Viewer
<input checked="" type="checkbox"/>	DisassemblerPlugin	Disassembler	Analysis
<input checked="" type="checkbox"/>	EclipseIntegrationPlugin	Eclipse Integration	Common
<input checked="" type="checkbox"/>	EquatePlugin	Add, rename, and delete equates	Code Viewer
<input checked="" type="checkbox"/>	EquateTablePlugin	Displays the list of equates	Code Viewer
<input checked="" type="checkbox"/>	ExporterPlugin	Export Program/Datatype Archi...	Common
<input checked="" type="checkbox"/>	FallThroughPlugin	Changes the "fall-through" add...	Code Viewer
<input checked="" type="checkbox"/>	FileFormatsPlugin	File format actions	Common

Filter:

Name: **DecompilePlugin**
Description: **Plugin for producing high-level decompilation**
Status: **Released (Tested and Documented)**
Package: **Ghidra Core**
Category: **Analysis**
Plugin Class: **ghidra.app.plugin.core.decompile.DecompilePlugin**
Class Location: **Users/abulazel/Applications/ghidra_9.0.1/Ghidra/Features/Decompiler/lib/Decompile**
Used By: **None**
Services Required: **ghidra.app.services.GoToServiceghidra.app.services.NavigationHistoryServ**

OK

IDA Interoperability



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

```
C:\Decompile: FUN_00401070 - (13-2)
22 undefined *h;
word local_18;
dword local_16;
dword local_e;
dword cy;
27
28 uVar1 = (*(code **)(undefined *)0xffffffff)(0);
29 uVar2 = (*(code **)(undefined *)0xffffffff)(1);
30 hWnd = (*(code **)(undefined *)0xffffffff)();
31 hDC = (*(code **)(undefined *)0xffffffff)(hWnd);
32 uVar3 = (*(code **)(undefined *)0xffffffff)(hDC);
33 uVar4 = (*(code **)(undefined *)0xffffffff)(hDC,uVar1,uVar2);
34 (*(code **)(undefined *)0xffffffff)(uVar3,uVar4);
35 (*(code **)(undefined *)0xffffffff)(uVar3,0,0,uVar1,uVar2,hDC,0,0,0xcc0020);
36 (*(code **)(undefined *)0xffffffff)(uVar4,0x18,&pv);
37 bmi.bmiHeader.biSize = 0x28;
38 bmi.bmiHeader.biPlanes = 1;
39 bmi.bmiHeader.biBitCount = 0x20;
40 bmi.bmiHeader.biCompression._0_2_ = 0;
41 bmi.bmiHeader.biCompression._2_2_ = 0;
42 bmi.bmiHeader.biSizeImage = 0;
43 bmi.bmiHeader.biXPelsPerMeter = 0;
44 bmi.bmiHeader.biYPelsPerMeter = 0;
45 bmi.bmiHeader.biClrUsed = 0;
46 bmi.bmiHeader.biClrImportant = 0;
47 iVar6 = local_6c * 0x20 + 0x1f;
48 iVar6 = ((int)(iVar6 + (iVar6 >> 0x1f & 0x1fU)) >> 5) * 4 * cLines;
49 uVar1 = (*(code **)(undefined *)0xffffffff)(0x42,iVar6);
50 bmi.bmiColors = (*(code **)(undefined *)0xffffffff)(uVar1);
51 (*(code **)(undefined *)0xffffffff)(hDC,uVar4,0,cLines,bmi.bmiColors,&bmi,0);
52 local_18 = 0x4d42;
53 uVar2 = (*(code **)(undefined *)0xffffffff)(0x42,iVar6 + 0x36);
54 iVar5 = (*(code **)(undefined *)0xffffffff)(uVar2);
55 _memcpy(iVar5,&local_18,0xe);
56 _memcpy(iVar5 + 0xe,&bmi,0x28);
57 _memcpy(iVar5 + 0x36,bmi.bmiColors,iVar6);
58 (*(code **)(undefined *)0xffffffff)(uVar1);
59 (*(code **)(undefined *)0xffffffff)(uVar1);
60 (*(code **)(undefined *)0xffffffff)(hWnd,hDC);
61 (*(code **)(undefined *)0xffffffff)(uVar3);
62 (*(code **)(undefined *)0xffffffff)(uVar4);
63 *(int *)param_1 = iVar5;
64 *(int *)param_2 = iVar6 + 0x36;
65 return;
```

Default UI - CodeBrowser



Ca' Foscari
University
of Venice

The screenshot displays the CodeBrowser interface with the following windows:

- Program Trees**: Shows the file structure of the target binary, including sections like .bss, .data, .got.plt, .got, .dynamic, .jcr, .fini_array, .init_array, .eh_frame, .eh_frame_hdr, .rodata, and .fini.
- Listing: bomb**: Displays the assembly code for the `bomb` section. A specific instruction at address `00401499` is highlighted: `ADD RSP,0x18`. The assembly code is as follows:

```
ff ff
LAB_00401499    ADD    RSP,0x18
0040149d c3      RET
```
- Decompile: read_line - (bomb)**: Shows the decompiled C-like pseudocode for the `read_line` function. It includes error handling for EOF on standard input.

```
1 2 char * read_line(void)
3 {
4     char cVar1;
5     long lVar2;
6     char *pcVar3;
7     uint uVar4;
8     int iVar5;
9     int extraout_EDX;
10    char *pcVar6;
11    byte bVar7;
12
13    bVar7 = 0;
14    lVar2 = skip();
15    if (lVar2 == 0) {
16        if (infile == stdin) {
17            puts("Error: Premature EOF on stdin");
18            /* WARNING: Subroutine does not return */
19            exit(8);
20        }
21        pcVar3 = getenv("GRADE_BOMB");
22        if (pcVar3 != (char *)0x0) {
23            /* WARNING: Subroutine does not return */
24            exit(0);
25        }
26        infile = stdin;
27        lVar2 = skip();
28        if (lVar2 == 0) {
29            puts("Error: Premature EOF on stdin");
30            /* WARNING: Subroutine does not return */
31            exit(0);
32        }
33    }
34    pcVar6 = input_strings + (long)num_input_strings * 0x50;
35    lVar2 = -1;
36    pcVar3 = pcVar6;
37    do {
38        uVar4 = (uint)lVar2;
39        if (lVar2 == 0) break;
40        lVar2 = lVar2 + -1;
41        uVar4 = (uint)lVar2;
42        cVar1 = *pcVar3;
43        pcVar3 = pcVar3 + (ulong)bVar7 * -2 + 1;
44        } while (cVar1 != 0);
45        iVar5 = ~lVar2 - 1.
```
- Symbol Tree**: Shows the symbol tree with categories like Imports, Exports, Functions, Labels, Classes, and Namespaces.
- Data Type Manager**: Shows the data type manager with entries for BuiltInTypes, `bomb`, generic_clib, generic_clib_64, and windows_vs12_32.
- Console - Scripting**: An empty console window.

Default UI - Program Trees

CodeBrowser: Infiltrate:/bomb

Program Trees

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

Listing: bomb

```

00401499 48 3 c4 18 ADD RSP,0x18
0040149d c3 RET

***** FUNCTION *****
undefined read_line()
AL:1 <RETURN>
read_line

XREF[1]: 00401492(j)

0040149e 48 83 ec 08 SUB RSP,0x8
004014a2 b8 00 00 MOV EAX,0x0
004014a7 e8 4d ff CALL skip
004014ac 48 85 c0 TEST RAX,RAX
004014a2 75 6e JNZ LAB_0040151f
004014b1 48 8b 05 MOV RAX,qword ptr [.bss:stdin]
004014b8 48 39 05 CMP qword ptr [.bss:infile],RAX = 00000000
004014bf 75 14 JNZ LAB_004014d5
004014c1 bf d5 25 MOV EDI=>.rodata:s_Error:_Premature_EOF_on_stdin_0... = "Error: Prem
40 00
004014c6 e8 45 f6 CALL .plt:puts
004014cb bf 08 00 MOV EDI,0x8
004014d0 e8 4b f7 CALL .plt:exit
ff ff

Flow Override: CALL_RETURN (CALL_TERMINATOR)

LAB_004014d5 MOV EDI=>.rodata:s_Grade_BOMB_004025f3,.rodata:s_G... = "GRADE_BOMB"
40 00

```

XREF[1]: 004014bf(j)

Decompile: read_line - (bomb)

```

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

```

Symbol Tree

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

Data Type Manager

- Data Types
 - BuiltinTypes
 - bomb
 - generic_clib
 - generic_clib_64
 - windows_vs12_32

Console - Scripting

Filter:

004014ac | read_line | TEST RAX,RAX

Default UI - Symbol Tree

CodeBrowser: Infiltrate:/bomb

File Edit Analysis Navigation Search Select Tools Window Help

Program Trees Listing: bomb Decompile: read_line - (bomb)

Symbol Tree

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

Filter:

Data Type Manager

- Data Types
 - BuiltinTypes
 - bomb
 - generic_clib
 - generic_clib_64
 - windows_vs12_32

Filter:

ff ff

00401499 48 83 c4 18 ADD RSP,0x18
0040149d c3 RET

***** FUNCTION *****

undefined undefined read_line()
read_line

XREF[1]: 00401492(j)
XREF[10]: Entry Point(*), m_main:00400e4e(c), main:00400e6a(c), main:00400e86(c), main:00400ea2(c), main:00400eb(c), secret_phase:004004284c, 00402b51

0040149e 48 83 ec 08 SUB RSP,0x8
004014a2 b8 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 MOV RAX,qword ptr [.bss:stdin]

90 22 20 00

004014b8 48 39 05 CMP qword ptr [.bss:infile],RAX = 00000000
a9 22 20 00

004014bf 75 14 JNZ LAB_004014d5
004014c1 bf d5 25 MOV EDI=>.rodata:s_Error:_Premature_EOF_on_stdin_0... = "Error: Premature EOF on stdin"

40 00

004014c6 e8 45 f6 CALL .plt:puts
ff ff

004014cb bf 08 00 MOV EDI,0x8
00 00

004014d0 e8 4b f7 CALL .plt:exit
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"

40 00

Decompile: read_line - (bomb)

```

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

```

Console - Scripting

004014ac read_line TEST RAX,RAX

Default UI - Data Type Manager

CodeBrowser: Infiltrate:/bomb

File Edit Analysis Navigation Search Select Tools Window Help

Program Trees Listing: bomb Decompile: read_line - (bomb)

Symbol Tree Console - Scripting

Data Type Manager

Filter:

- BuiltInTypes
- bomb
- generic_clib
- generic_clib_64
- windows_vs12_32

The screenshot shows the CodeBrowser interface with several windows open. The 'Data Type Manager' window is highlighted with a red border. The 'Listing: bomb' window displays assembly code for the 'read_line' function, showing instructions like ADD, RET, SUB, MOV, CALL, TEST, JNZ, and CMP. The 'Decompile: read_line - (bomb)' window shows the corresponding C-like pseudocode. The 'Symbol Tree' and 'Console - Scripting' windows are also visible.

Default UI - Listing (Disassembly)

CodeBrowser: Infiltrate:/bomb

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
LAB_00401499 48 83 c4 18 ADD    RSP,0x18
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:00400eb(c),
secret_phase:0040
0040284c, 00402b51

0040149e 48 83 ec 08 SUB   RSP,0x8
004014a2 b8 00 00 MOV    EAX,0x0
004014a7 e8 4d ff CALL   skip
                                undefined skip
004014ac 48 85 c0 TEST  RAX,RAX
004014af 75 6e JNZ   LAB_0040151f
004014b1 48 8b 05 MOV    RAX,qword ptr [.bss:stdin]
004014b2 90 22 20 00 CMP   qword ptr [.bss:infile],RAX = 00000000
004014b8 48 39 05 CMP   qword ptr [.bss:infile],RAX = 00000000
004014bf 75 14 JNZ   LAB_004014d5
004014c1 bf d5 25 MOV    EDI=>.rodata:s_Error:_Premature_EOF_on_stdin_0...
004014c6 40 00
004014cb e8 45 f6 CALL   .plt:puts
                                int puts(char
004014cb bf 08 00 MOV    EDI,0x8
004014d0 00 00
004014d0 e8 4b f7 CALL   .plt:exit
                                void exit(int
004014d5 bf f3 25 MOV    EDI=>.rodata:s_GRADE_BOMB_004025f3,.rodata:s_G...
                                = "GRADE_BOMB"
004014d5 40 00

-- Flow Override: CALL_RETURN (CALL_TERMINATOR)

LAB_004014d5 40 00 XREF[1]: 004014bf(j)
EDI=>.rodata:s_GRADE_BOMB_004025f3,.rodata:s_G... = "GRADE_BOMB"

```

Decompile: read_line - (bomb)

```

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

```

Console - Scripting

004014ac read_line TEST RAX,RAX

Default UI - Decompiler

CodeBrowser: Infiltrate:/bomb

File Edit Analysis Navigation Search Select Tools Window Help

Program Trees X

Listing: bomb

ff ff

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(*), m:

main:00400e4e(c), main:00400e6a(c), main:00400e86(c), main:00400ea2(c), main:00400ebe(c), secret_phase:0040' 0040284c, 00402b50

0040149e 48 83 ec 08 SUB RSP,0x8

004014a2 b8 00 00 MOV EAX,0x0

004014a7 e8 4d ff CALL skip undefined skip

004014ac 48 85 c0 TEST RAX,RAX

004014af 75 6e JNZ LAB_0040151f

004014b1 48 b8 05 MOV RAX,qword ptr [.bss:stdin]

98 22 20 00

004014b8 48 39 05 CMP qword ptr [.bss:infile],RAX = 00000000

a9 22 20 00

004014bf 75 14 JNZ LAB_004014d5

004014c1 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 08 00 MOV EDI,0x8

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

ff ff

— Flow Override: CALL_RETURN (CALL_TERMINATOR)

LAB_004014d5 bf f3 25 MOV EDI=>.rodata:s_GRADE_BOMB_004025f3,.rodata:s_G... = "GRADE_BOMB"

XREF[1]: 004014bf(j)

40 00

Decompiled code:

```

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

Console - Scripting

Filter: []

Data Type Manager

Data Types

- BuiltinTypes
- bomb
- generic_clib
- generic_clib_64
- windows_vs12_32

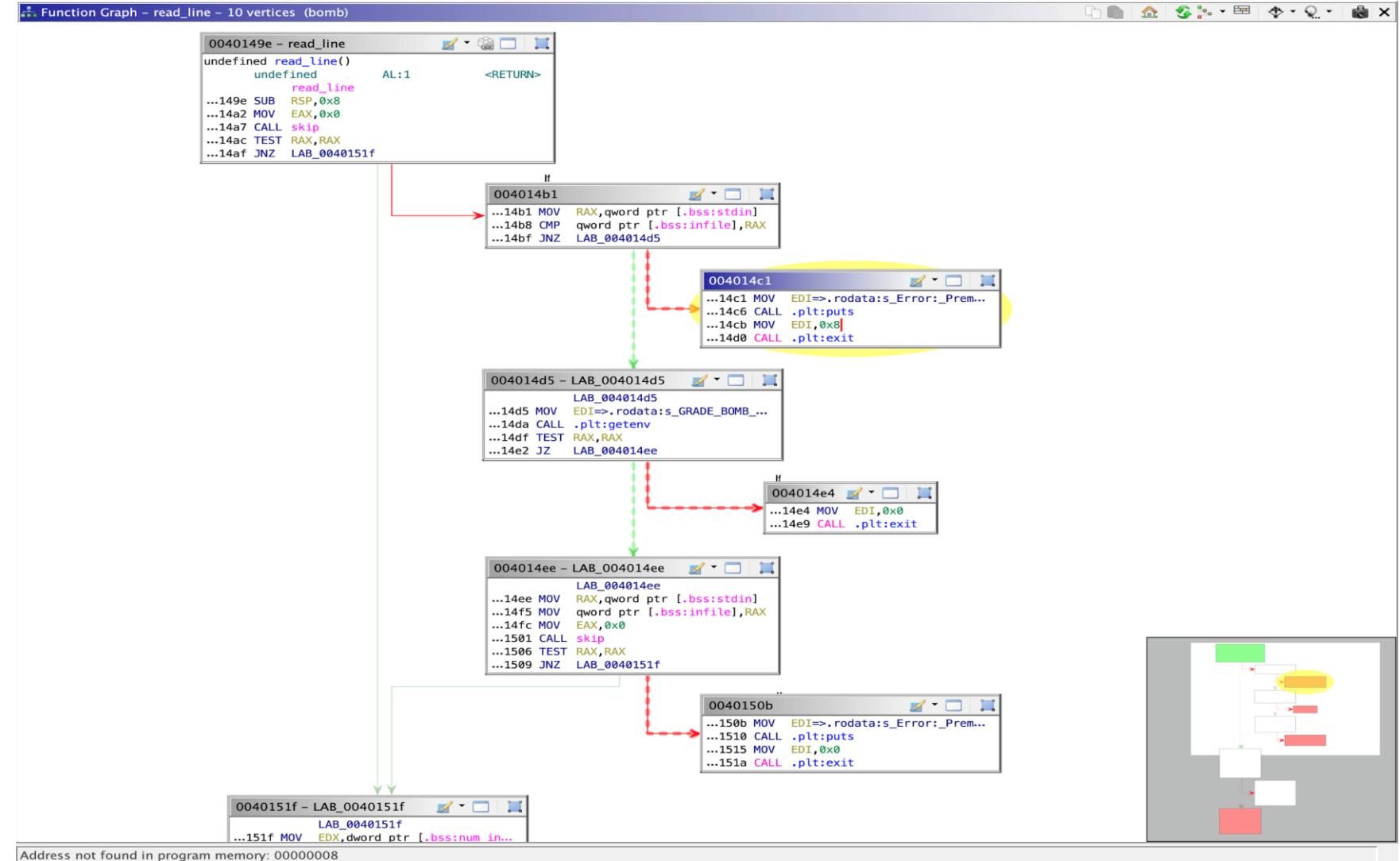
Filter: []

004014ac read_line TEST RAX,RAX

Control Flow Graph



Ca' Foscari
University
of Venice



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)

004014d0 e8 4b f7      CALL   .plt:exit
ff ff |                  (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)

004014d5 bf f3 25      MOV    EDI,>.rodata:s_GRADE_BOMB_004025f3,.rodata:s_G...
40 00 |                  = "GRADE_BOMB"
004014d5 bf f3 25      MOV    EDI,>.rodata:s_GRADE_BOMB_004025f3,.rodata:s_G...
40 00 |                  XREF[1]: 004014bf(j)
004014da e8 01 f6      CALL   .plt:getenv
ff ff |                  char * getenv(char * __name)

004014da e8 01 f6      CALL   .plt:getenv
ff ff |                  (register, 0x38, 8) = COPY (const, 0x4025f3, 8)
004014df 48 85 c0      TEST   RAX,RAX
004014e2 74 0a          JZ    LAB_004014ee
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)

004014e9 e8 32 f7      CALL   .plt:exit
ff ff |                  (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)

004014ee 48 8b 05      MOV    RAX,qword ptr [.bss:stdin]
53 22 20 00 |
004014ee 48 8b 05      MOV    RAX,qword ptr [.bss:stdin]
53 22 20 00 |                  XREF[1]: 004014e2(j)
004014f5 48 89 05      MOV    qword ptr [.bss:infile],RAX
6c 22 20 00 |                  = 00000000
004014f5 48 89 05      MOV    qword ptr [.bss:infile],RAX
6c 22 20 00 |                  (register, 0x0, 8) = COPY (ram, 0x603748, 8)
004014fc b8 00 00      MOV    EAX,0x0
00 00 |                  (ram, 0x603768, 8) = COPY (register, 0x0, 8)

```

Decompilation across Architectures - x64



The screenshot shows two windows from the Immunity Debugger interface. The left window, titled 'Listing: elf-Linux-x64-bash', displays assembly code for the x64 architecture. The right window, titled 'Decompile: rl_complete_internal - (elf-Linux-x64-bash)', shows the corresponding C-like pseudocode generated by the decompiler. The assembly code includes various instructions like JNZ, CALL, MOV, and JMP, along with labels and memory addresses. The decompiled code is a function named 'rl_complete_internal' that interacts with global variables and functions like 'rl_completion_entry_function' and 'FUN_00492490'. The two panes are synchronized, allowing the user to see the original machine code and its high-level representation simultaneously.

```
41 pcVar9 = rl_completion_entry_function;
42 if (rl_completion_entry_function == (code *)0x0) {
43     pcVar9 = rl_filename_completion_function;
44 }
45 uVar5 = 0;
46 if (rl_point != 0) {
47     local_39[0] = _rl_find_completion_word(&local_44,&local_40);
48     uVars = rl_point;
49 }
50 rl_point = uVar1;
51 __src = (char *)rl_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)rl_filename_completion_desired), ppcVar8
62     iVar3 == 0)) {
63     rl_ding();
64     if (__dest != (char *)0x0) {
65         free(__dest);
66     }
67     rl readline_state = rl readline_state & 0xfffffbff;
68     DAT_006e5570 = 0;
69     rl_completion_found_quote = 0;
70     rl_completion_quote_character = 0;
71     return 0;
72 }
73 if (uParm1 == 0x2a) {
74     rl_begin_undo_group();
75     if ((uVar5 != 0) && (local_39[0] != 0)) {
76         uVar5 = uVar5 - (uint)(local_39[0] == rl_line_buffer[(long)(int)uVar5 + -1]);
77     }
78     rl_delete_text((ulong)uVar5,(ulong)rl_point);
79     __src = ppcVar8[1];
80     if (__src == (char *)0x0) {
81         rl_point = uVar5;
82         __src = (char *)FUN_004916e0(*ppcVar8,1,local_39);
83         rl_insert_text(__src);
84         rl insert text(&DAT_004ad25a);
```

Decompilation across Architectures - SPARC



Listing: elf-Linux-SparcV8-bash

```
*elf-Linux-SparcV8-bash
        undefined type_builtin()
        undefined      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    lduw   [i0+0x4],g1
0008e334 c2 00 40 00    lduw   [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    bpne,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
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 90 70 70 70    cmp    ? av7a
```

Decompile: type_builtin – (elf-Linux-SparcV8-bash)

```
15 type_builtin(undefined4 *puParm1,undefined4 uParm2,undefined4 uParm3,undefined4 *puPar
16                                         undefined4 uParm5,char *pcParm6)
17 {
18     char cVar1;
19     char *pcVar2;
20     int *piVar3;
21     int iVar4;
22     int iVar5;
23     undefined4 uVar6;
24     undefined4 *puVar7;
25     uint uVar8;
26     char *_s1;
27     int **ppiVar9;
28     uint uVar10;
29
30     if (puParm1 == (undefined4 *)0x0) {
31         return 0;
32     }
33     pcVar2 = *(char **)puParm1[1];
34     if (*pcVar2 == '-') {
35         pcParm6 = "-path";
36         puParm4 = &DAT_000d7730;
37         uParm5 = 0x61;
38         cVar1 = pcVar2[1];
39         puVar7 = puParm1;
40         do {
41             _s1 = pcVar2 + 1;
42             if (cVar1 == 't') {
43                 iVar4 = strcmp(_s1,"type");
44                 if (iVar4 == 0) {
45                     *_s1 = 't';
46 LAB_0008e470:
47                     *(undefined4)(*(int *)puVar7[1] + 2) = 0;
48                     goto LAB_0008e3d8;
49                 }
50                 puVar7 = (undefined4 *)*puVar7;
51             }
52         } else {
```

Decompilation Across Architectures - PowerPC

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

```
*elf-Linux-SparcV8-bash *MachO-OSX-ppc-openssl-1.0.1h
      LAB_000212c8
000212c8 40 9e 00 10    bne    cr7,LAB_000212d8
000212cc 48 00 00 18    b      LAB_000212e4

      LAB_000212d0
000212d0 2f 82 00 00    cmpwi   cr7,r2,0x0
000212d4 40 9e 00 10    bne    cr7,LAB_000212e4

      LAB_000212d8
000212d8 68 02 00 2f    xori   r2,r0,0x2f
000212dc 7c 42 00 34    cntlzw r2,r2
000212e0 54 42 d9 7e    rlwinm r2,r2,0x1b,0x5,0x1f

      LAB_000212e4
000212e4 39 29 00 01    addi   r9,r9,0x1

      LAB_000212e8
000212e8 88 09 00 00    lbz    r0,0x0(r9)
000212ec 7c 00 07 75    extsb. r0,r0
000212f0 40 82 ff 98    bne    LAB_00021288

      LAB_000212f4
000212f4 2f 82 00 03    cmpwi   cr7,r2,0x3
000212f8 38 00 00 01    li     r0,0x1
000212fc 41 9e 00 10    beq    cr7,LAB_0002130c
00021300 38 02 00 01    addi   r0,r2,0x1
00021304 7c 00 00 34    cntlzw r0,r0
00021308 54 00 d9 7e    rlwinm r0,r0,0x1b,0x5,0x1f

      LAB_0002130c
0002130c 40 82 00 24    bne    LAB_00021330
00021310 3c 9f 00 04    addis   r4,r31,0x4
00021314 7f 43 d3 78    or     r3,r26,r26
00021318 80 84 fe 18    lwz    r4=>__cstring:s_HTTP/1.0_200_ok_Cor
0002131c 48 02 39 51    bl     __symbol_stub1:__symbol_stub1::__BIC
00021320 3c 9f 00 03    addis   r4,r31,0x3
00021324 7f 85 e3 78    or     r5,r28,r28
00021328 38 84 30 50    addi   r4=>__cstring:s_%s'_is_an_invalid_
0002132c 48 00 00 30    b      LAB_0002135c
```

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

```
107  lVar11 = _BIO_int_ctrl(bp,0x1c,0x40000,1);
108  if ((lVar11 == 0) || (lVar5 = _SSL_new((ulonglong)_ctx), lVar5 == 0)) goto LAB_00021588;
109  if (_s_tlsextdebug != 0) {
110      _SSL_callback_ctrl(lVar5,0x38,ZEXT48(PTR__tlsext_cb_000603ac));
111      _SSL_ctrl(lVar5,0x39,0,ZEXT48(_bio_s_out));
112  }
113  if ((char *)uParm3 != (char *)0x0) {
114      sVar12 = strlen((char *)uParm3);
115      _SSL_set_session_id_context(lVar5,uParm3,sVar12);
116  }
117  append = _BIO_new_socket(iParm2,0);
118  if (_s_nbio_test != 0) {
119      type = _BIO_f_nbio_test();
120      b = _BIO_new(type);
121      append = _BIO_push(b,append);
122  }
123  _SSL_set_bio(lVar5,append,append);
124  _SSL_set_accept_state(lVar5);
125  _BIO_ctrl(bp_00,0x6d,1,(void *)lVar5);
126  _BIO_push(bp,bp_00);
127  if (_s_debug != 0) {
128      _SSL_set_debug(lVar5,1);
129      bp_00 = (BIO *)_SSL_get_rbio(lVar5);
130      _BIO_set_callback(bp_00,(long (*)(bio_st *,int,char *,int,long,long))
131                        PTR__bio_dump_callback_00060398);
132      bp_00 = (BIO *)_SSL_get_rbio(lVar5);
133      _BIO_set_callback_arg(bp_00,(char *)_bio_s_out);
134  }
135  if (_s_msg != 0) {
136      _SSL_set_msg_callback(lVar5,ZEXT48(PTR__msg_cb_000603a8));
137      _SSL_ctrl(lVar5,0x10,0,ZEXT48(_bio_s_out));
138  }
139  LAB_00020d80:
140  do {
141      if (_hack != 0) {
142          while ((uVar6 = _SSL_accept(lVar5), (int)uVar6 < 1 &&
143                  (iVar10 = _SSL_get_error(lVar5,uVar6), iVar10 == 4))) {
144              _BIO_printf(_bio_s_out,"LOOKUP during accept %s\n",(ulonglong)_srp_callback_parm);
145              DAT_000618e0 = _SRP_VBASE_get_by_user((ulonglong)DAT_000618dc,(ulonglong)_srp_callback_parm);
146          ;
147          if (DAT_000618e0 == 0) {
148              _BIO_printf(_bio_s_out,"LOOKUP not successful\n");
149          }
150      }
151  }
```

Cross-References



The screenshot shows a debugger interface with several windows:

- Program Tree**: Shows file navigation.
- Symbol Tree**: Shows imports, exports, functions, labels, classes, and namespaces.
- Data Type Manager**: Shows data types like `BuiltInTypes`, `bomb.exe`, `generic_clib_64`, and `windows_vs12_32`.
- Console - Scripting**: An empty console window.
- Assembly View**: Displays assembly code with addresses like `00402f80` and instructions like `CALL .text _read_line`.
- Context Menu**: A right-clicked menu for the assembly code, listing options such as `Bookmark...`, `Clear Code Bytes`, `Copy`, `Comments`, `Create Function`, `Edit Label...`, `Clear Register Values...`, `Colors`, `Fallthrough`, `References`, and others.
- Code View**: Shows C-like pseudocode with syntax highlighting. Lines 29 and 30 are highlighted in yellow. Lines 32 and 33 are green. Lines 36 and 37 are blue. Lines 39 and 40 are red.

```
.text WHICH TO DOW YOURSELF UP.
pcVar1 = _read_line();
_phase_1(pcVar1);
_phase_defused();
.text("Phase 1 defused. How about
pcVar1 = _read_line();
_phase_2(pcVar1);
_phase_defused();
.text("That's number 2. Keep go
pcVar1 = _read_line();
_phase_3(pcVar1);
_phase_defused();
.text("Halfway there!");
```

Strings



CodeBrowser: understand:/bomb.exe

File Edit Analysis Navigation Search Select Tools Window Help

Program Tree X

Symbol Tree X

Data Type Mana... X

Listing: bomb.exe *bomb.exe X

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
00402fa6 00

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

Cf Decompile: _main - (bomb.exe)

```
20 .text("%s: Error: Couldn't open %s\n", *_Ar
21 /* WARNING: Subroutine does n
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 o
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 X Decompile: _main X

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: **group**

Description:

Category: busybox/grp.h

Size: 16 Alignment: 4 Align

align(minimum...) pack(maximum...)

- none
- machine
-

- none
-

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
004014c1	bf d5 25	MOV	EDI=>.rodata:s_Error:_Premat
	40 00		
004014c6	e8 45 f6	CALL	.plt:puts
	40 00		

The screenshot shows the Immunity Debugger interface. A context menu is open over the instruction at address 00401499, which is highlighted in yellow. The menu item "Undo bomb: Clear with Options" is selected. The assembly listing window shows the following code:

00401499	48 83 c4 18	ADD	RSP, 0x18
0040149d	c3	RET	

	*	FUNCT	

	undefined	read_line()	
	AL:1	<RETURN>	
	read_line		
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
	40 00		

Static Attack with Ghidra



Ghidra set-up

1. Install JDK (version ≥ 11), add its bin directory to your PATH

jdk.java.net/11

2. Download Ghidra

ghidra-sre.org

github.com/NationalSecurityAgency/ghidra/releases

3. The installation instructions are here:

<https://ghidra-sre.org/InstallationGuide.html#Platforms>

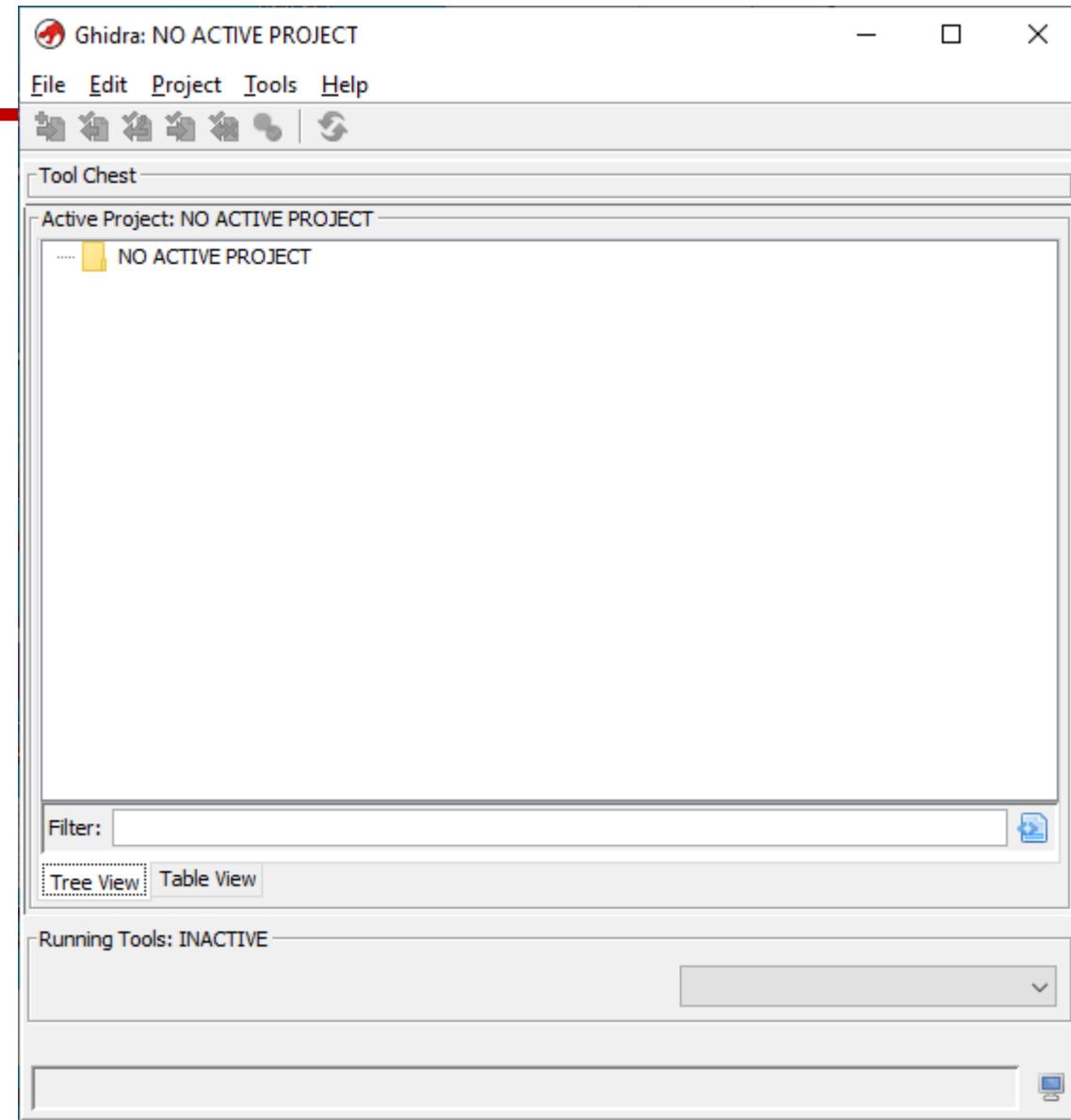
4. Run Ghidra

`ghidraRun.bat` (or `ghidra.sh`)

MacOS: Make sure you start ghidra from bash shell, not zsh shell (which is now default on MacOS)

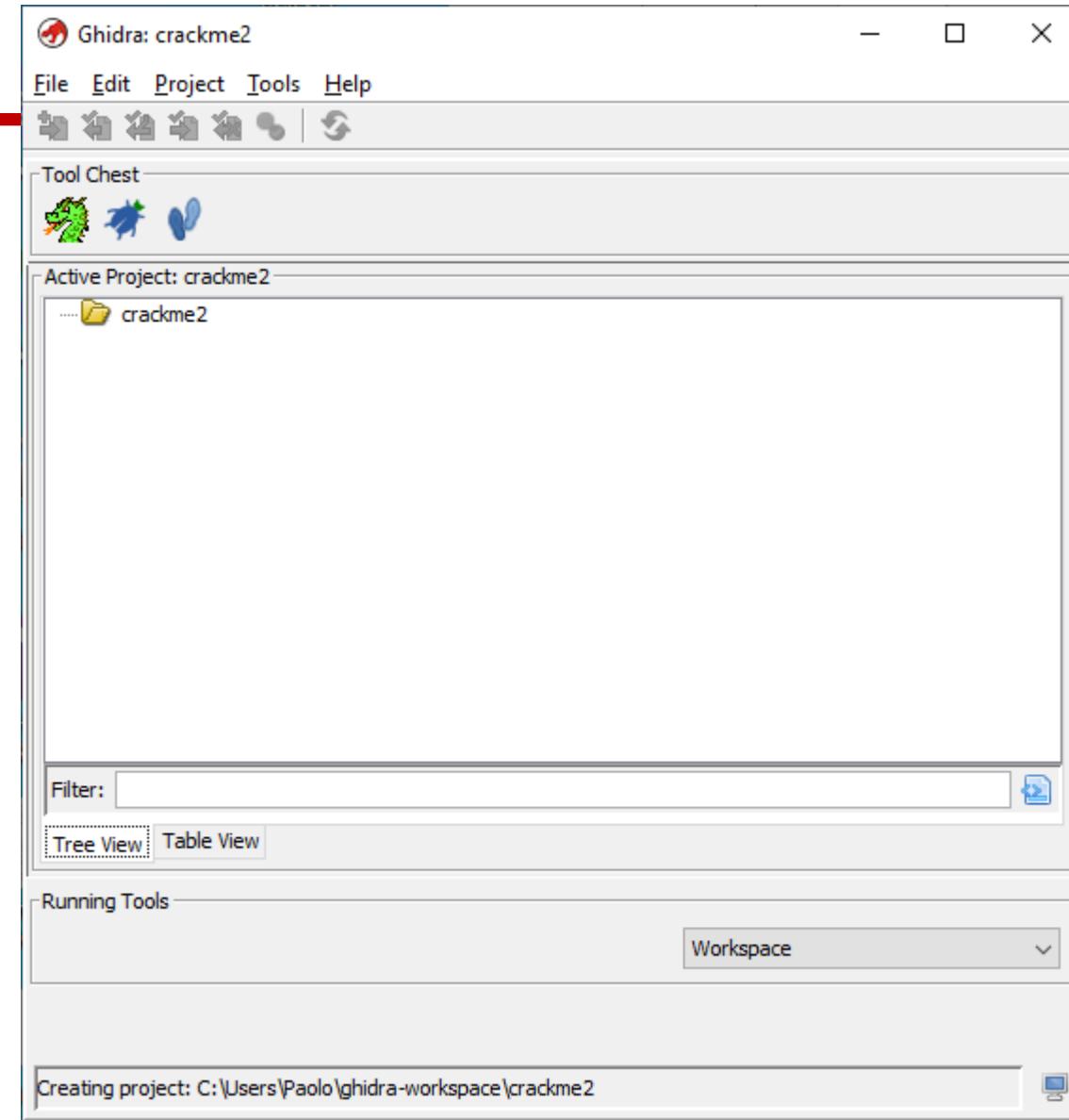
Ghidra

- Open Ghidra



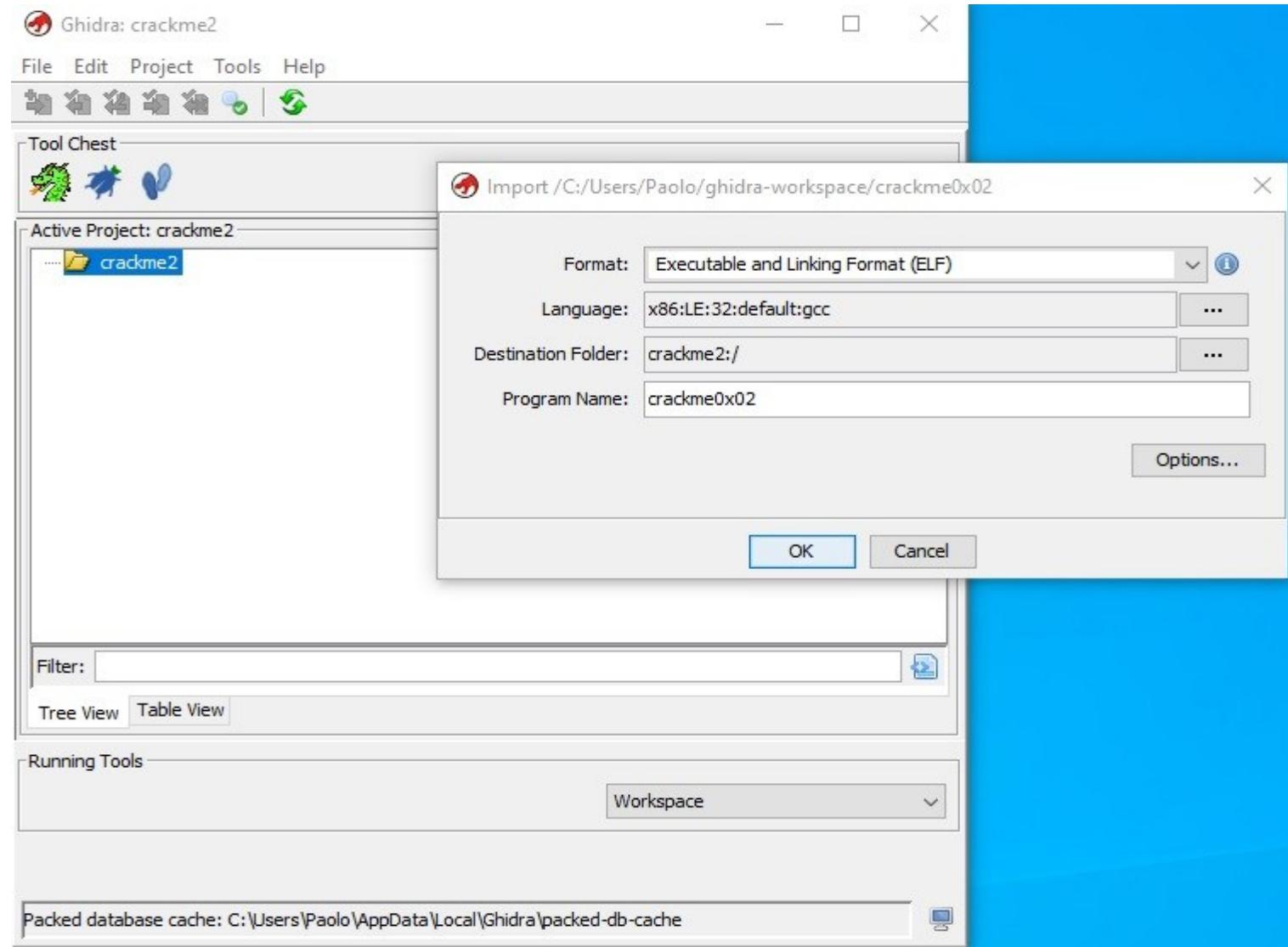
New Project

- Not-shared
- Name it “Crackme2”



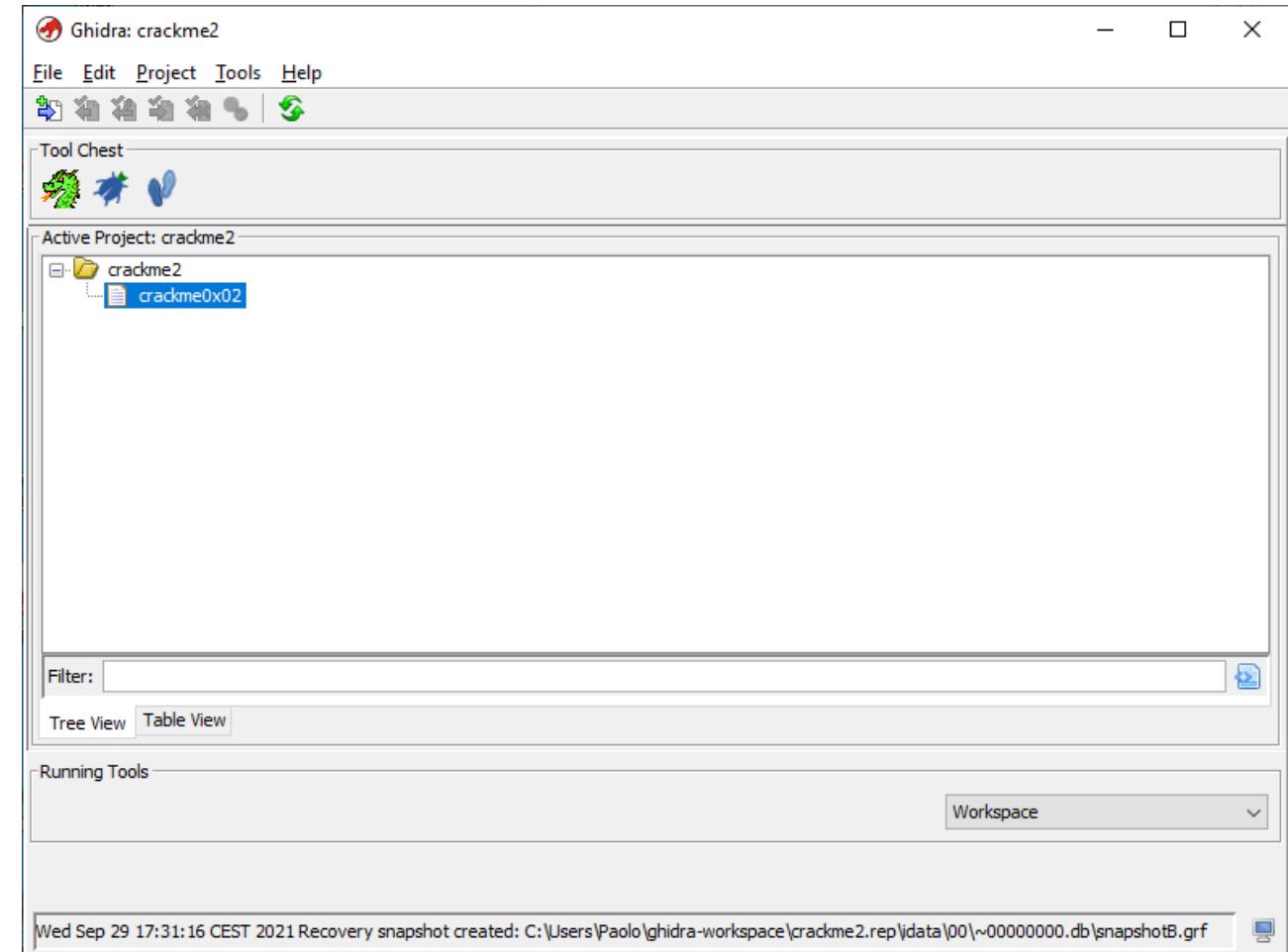
Import Program in project

- Menu File
- Import File
- Browse and select Crackme0x02



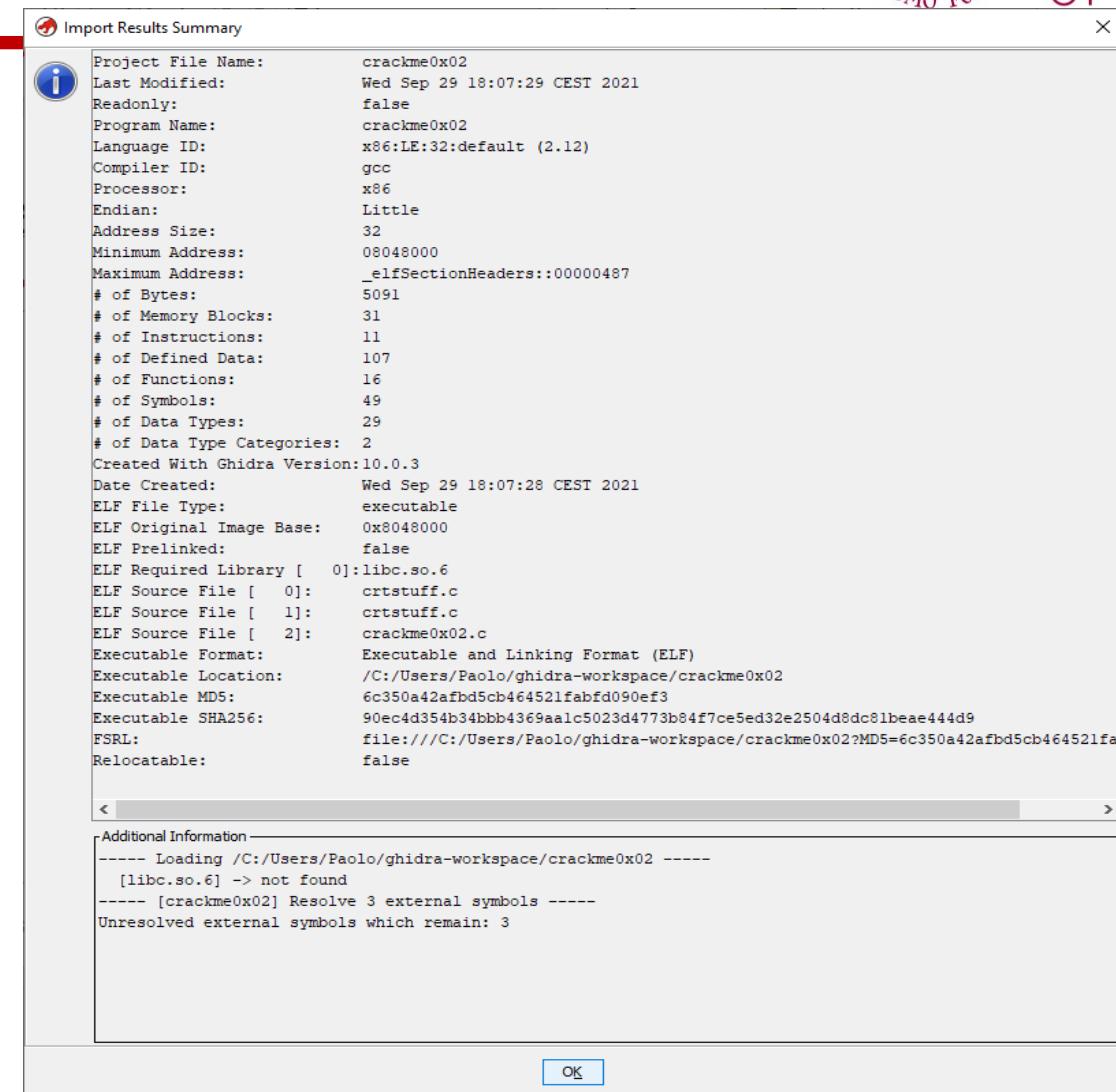
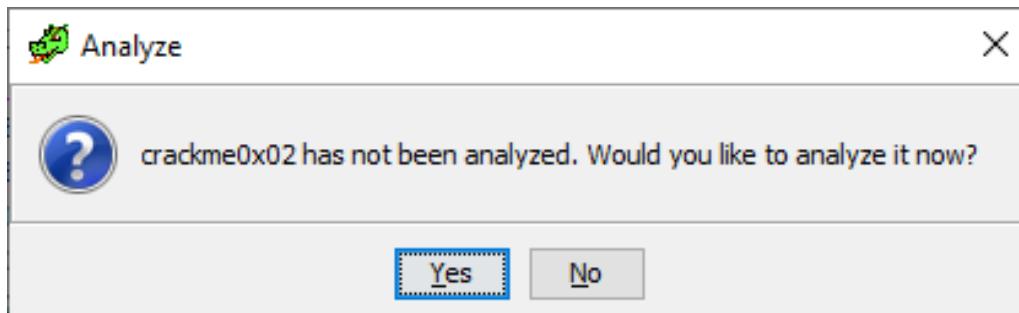
Launch CodeBrowser

- Click Ghidra
(the green dragon icon)
to open the CodeBrowser



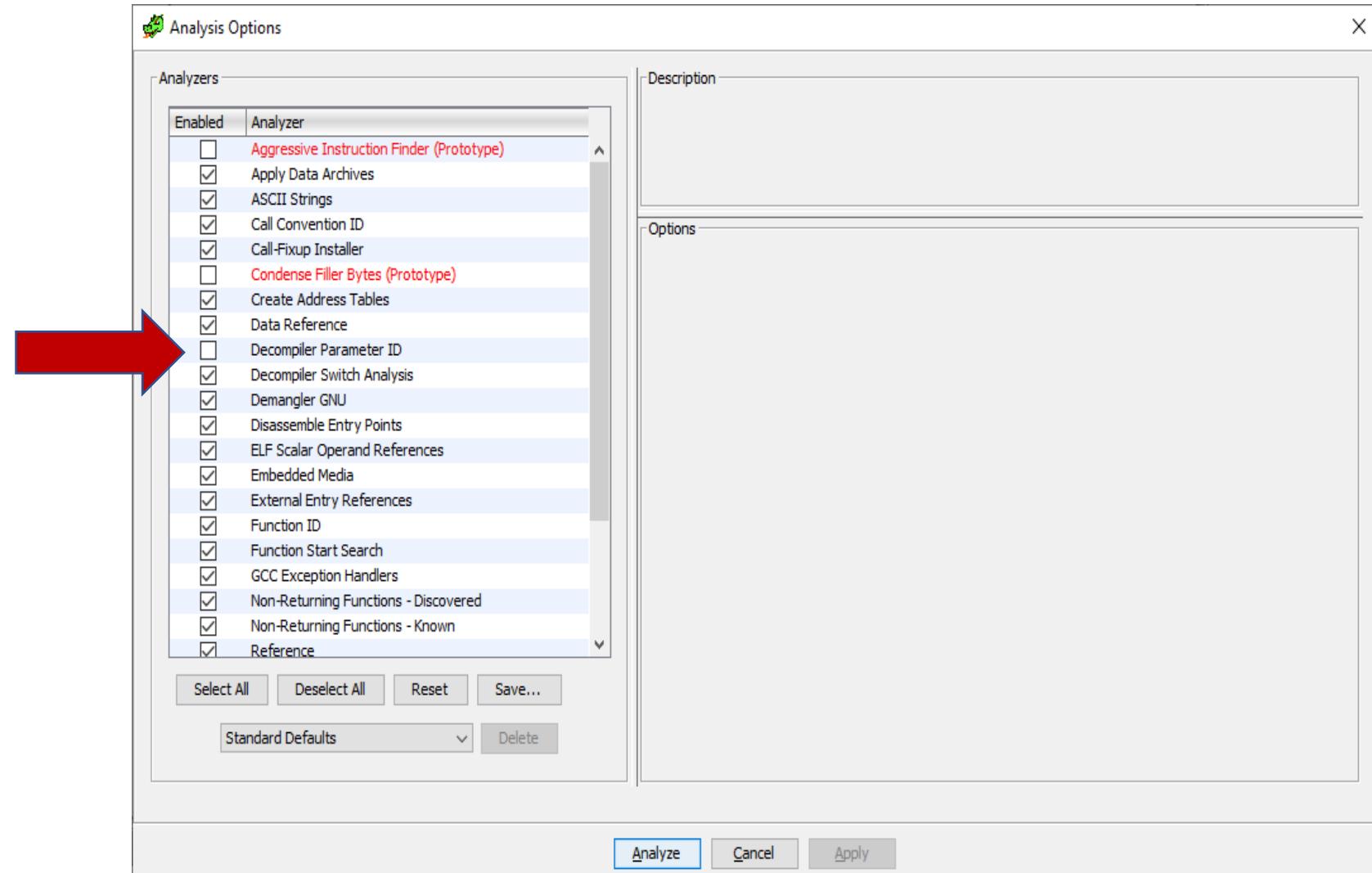
About Program

- Info on the file
- Then Prompt to analyse it

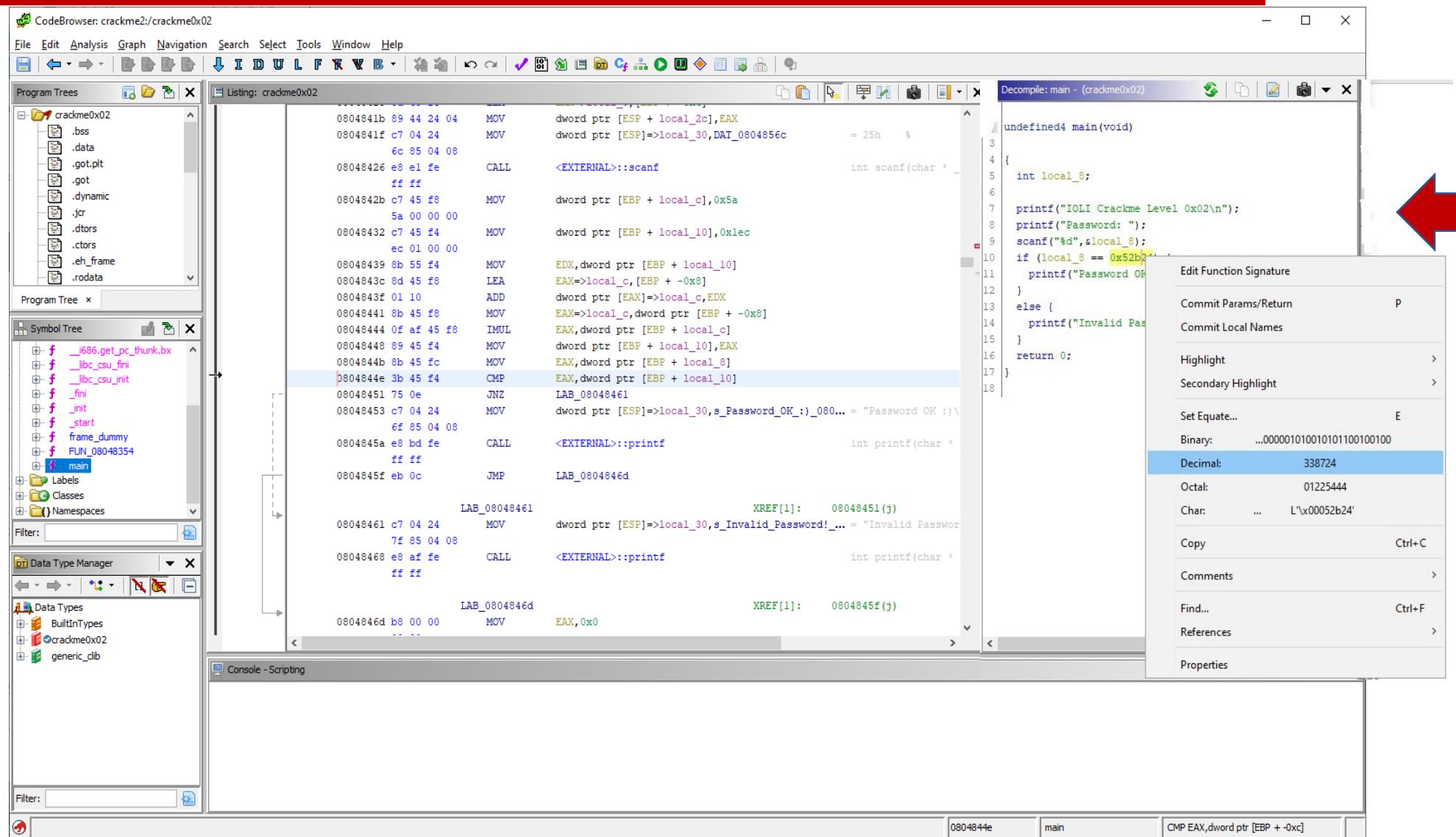


Analysis Options: Decompiler

- Select Decompiler Parameter ID
- Then in the Functions View, select the “main” function



CodeBrowser: Crackme0x02



The screenshot shows the CodeBrowser interface for the crackme0x02 challenge. The main window has several panes:

- Program Trees**: Shows the file structure of the crackme0x02 executable.
- Symbol Tree**: Lists symbols such as `_i686.get_pc_thunk.bx`, `__libc_csu_fini`, `__libc_csu_init`, `_fini`, `_init`, `_start`, `frame_dummy`, `FUN_08048354`, and `main`.
- Listing**: Displays the assembly code for the `main` function. The assembly code includes instructions like MOV, CALL, and CMP, along with their corresponding memory addresses and opcodes.
- Decompile**: Shows the decompiled C code for the `main` function. The code reads a password from standard input, compares it to a hardcoded value (0x52b24), and prints "Password OK" or "Invalid Password".
- Console - Scripting**: A text input field for running scripts.

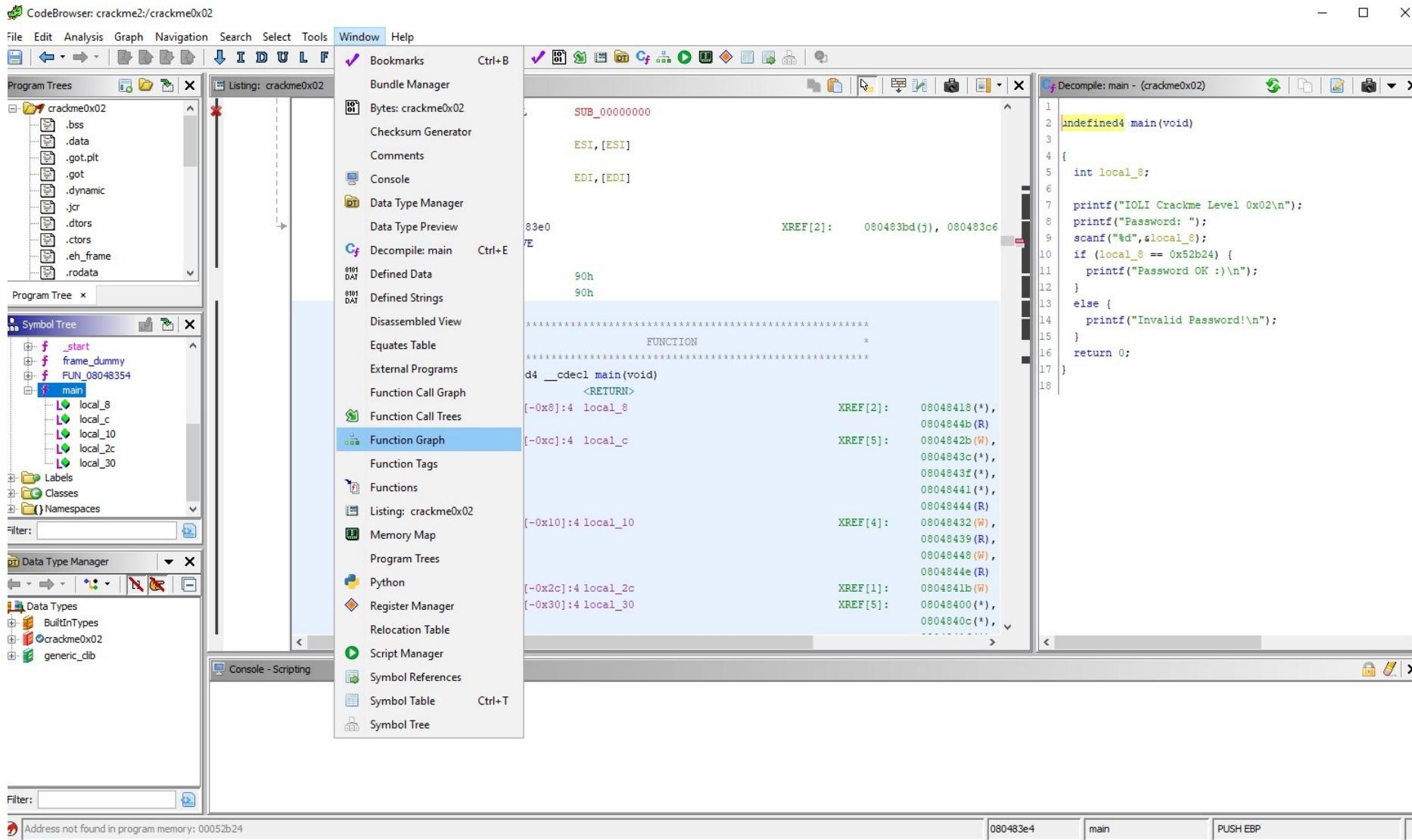
A red arrow points from the text "Decompiled C code here" to the decompiled C code pane. A context menu is open over a constant value in the decompiled code, specifically the decimal value 338724. The menu options include:

- Edit Function Signature
- Commit Params/Return
- Commit Local Names
- Highlight
- Secondary Highlight
- Set Equate...
- Binary: ...000001010010101100100100
- Decimal: 338724 (selected)
- Octal: 01225444
- Char: ... L'\x00052b24'
- Copy (Ctrl+C)
- Comments
- Find... (Ctrl+F)
- References
- Properties

Decompiled C
code here

Right-Click on
constants to
see the values
in various
bases or char
encoding

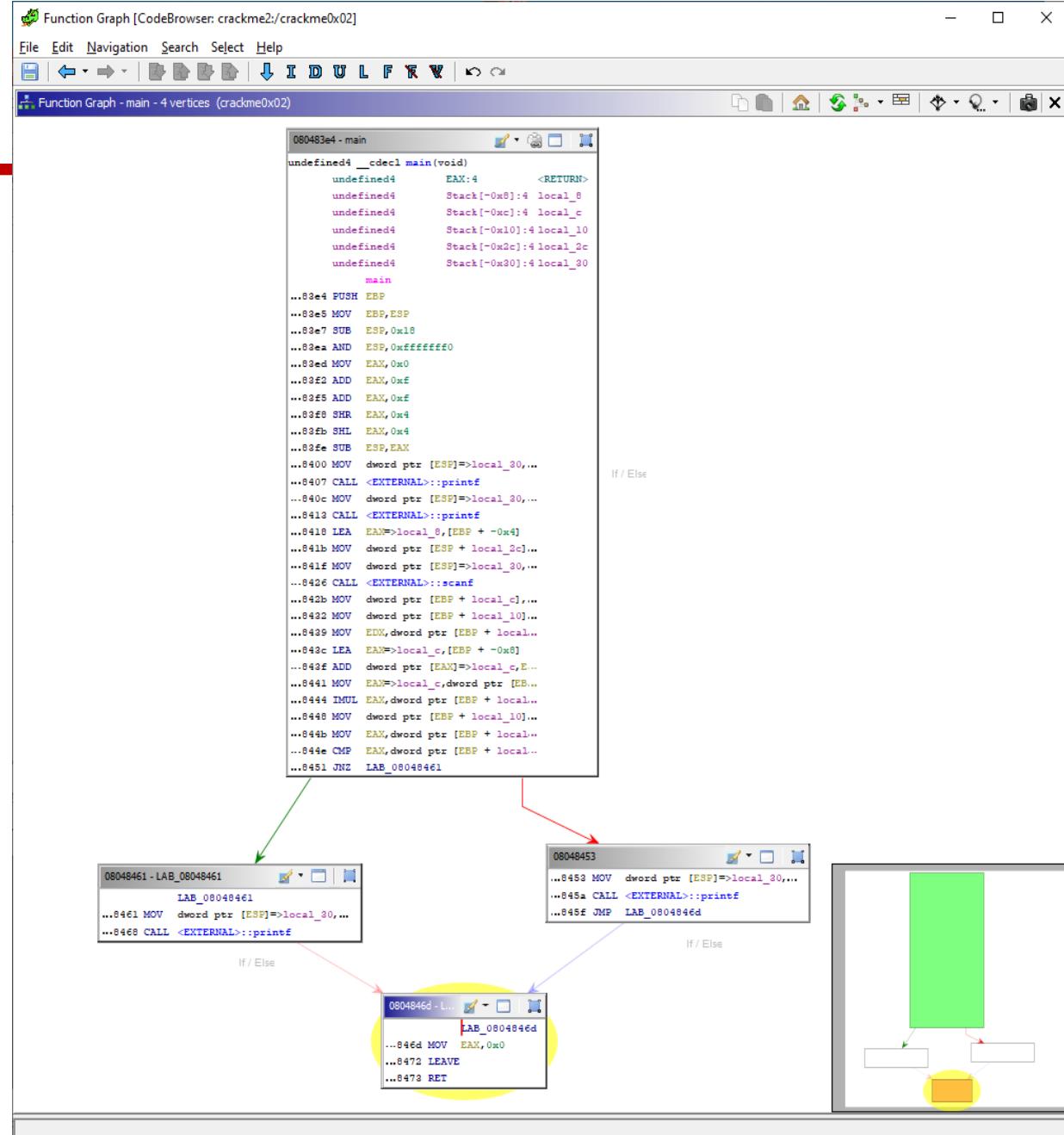
Select a function and create its CFG



1. Select function
2. Menu Window
3. Function Graph

CFG

- If (green) else (red)
- View Outline
- Resizable Layout



Highlight Paths

- When selecting a jump instruction

Function Graph [CodeBrowser: crackme2:/crackme0x02]

File Edit Navigation Search Select Help

I D U L F R V

Function Graph - main - 4 vertices (crackme0x02)

080483e4 - main

```
undefined4 __cdecl main(void)
{
    undefined4    EAX:4           <RETURN>
    undefined4    Stack[-0x8]:4 local_8
    undefined4    Stack[-0xc]:4 local_c
    undefined4    Stack[-0x10]:4 local_10
    undefined4    Stack[-0x2c]:4 local_2c
    undefined4    Stack[-0x30]:4 local_30

    main
...83e4 PUSH EBP
...83e5 MOV EBP,ESP
...83e7 SUB ESP,0x18
...83e8 AND EBP,0xffffffff
...83ed MOV EAX,0x0
...83f2 ADD EAX,0xf
...83f5 ADD EAX,0xf
...83f8 SHR EAX,0x4
...83fb SHL EAX,0x4
...83fe SUB EBP,EAX
...8400 MOV dword ptr [EBP]>local_30,...
...8407 CALL <EXTERNAL>::printf
...840c MOV dword ptr [EBP]>local_30,...
...8413 CALL <EXTERNAL>::printf
...8418 LEA EAX=>local_8,[EBP + -0x4]
...841b MOV dword ptr [EBP + local_2c],...
...841f MOV dword ptr [EBP]>local_30,...
...8426 CALL <EXTERNAL>::scanf
...842b MOV dword ptr [EBP + local_c],...
...8432 MOV dword ptr [EBP + local_10],...
...8439 MOV EDX,dword ptr [EBP + local_...
...843c LEA EAX=>local_c,[EBP + -0x8]
...843f ADD dword ptr [EAX]=>local_c,E...
...8441 MOV EAX=>local_c,dword ptr [EB...
...8444 IMUL EAX,dword ptr [EBP + local_...
...8448 MOV dword ptr [EBP + local_10],...
...844b MOV EAX,dword ptr [EBP + local_...
...844e CMP EAX,dword ptr [EBP + local_...
...8451 JNC LAB_08048461
```

If / Else If / Else

08048461 - LAB_08048461

```
LAB_08048461
...8461 MOV dword ptr [ESP]>local_30,...
...8468 CALL <EXTERNAL>::printf
```

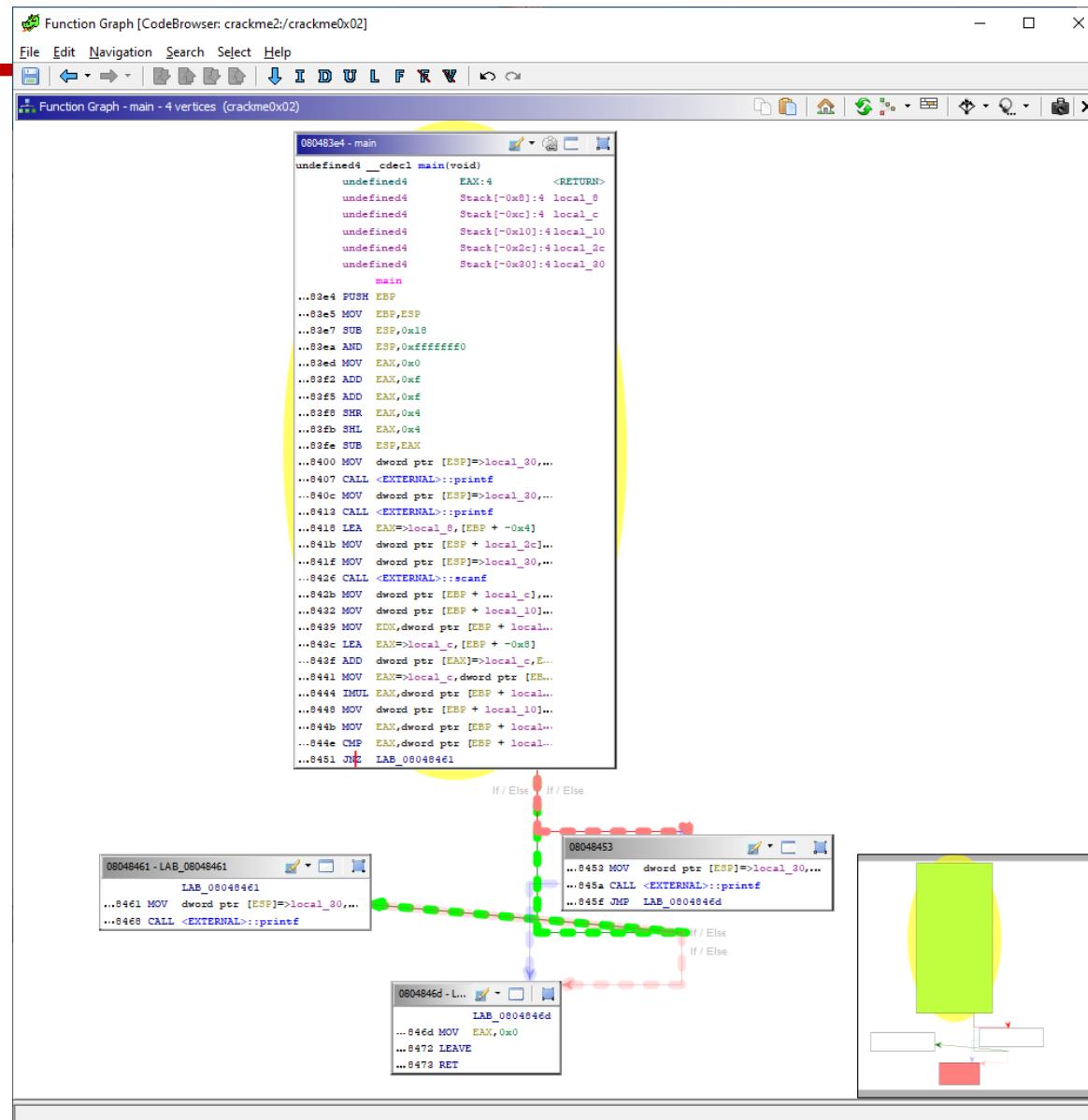
08048453

```
...8453 MOV dword ptr [ESP]>local_30,...
...845a CALL <EXTERNAL>::printf
...845f JMP LAB_0804846d
```

0804846d - LAB_0804846d

```
LAB_0804846d
...846d MOV EAX,0x0
...8472 LEAVE
...8473 RET
```

If / Else If / Else



Jumps Recap



CCCC	Name	Means
0000	O	overflow
0001	NO	Not overflow
0010	C/B/NAE	Carry, below, not above nor equal
0011	NC/AE/NB	Not carry, above or equal, not below
0100	E/Z	Equal, zero
0101	NE/NZ	Not equal, not zero
0110	BE/NA	Below or equal, not above
0111	A/NBE	Above, not below nor equal
1000	S	Sign (negative)
1001	NS	Not sign
1010	P/PE	Parity, parity even
1011	NP/PO	Not parity, parity odd
1100	L/NGE	Less, not greater nor equal
1101	GE/NL	Greater or equal, not less
1110	LE/NG	Less or equal, not greater
1111	G/NLE	Greater, not less nor equal

Patch Instruction

CodeBrowser: crackme2/crackme0x02

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees Listing: crackme0x02 Decompile: main - (crackme0x02)

```

1 undefined4 main(void)
2 {
3     int local_8;
4
5     printf("IOLI Crackme Level 0x02\n");
6     printf("Password: ");
7     scanf("%d",&local_8);
8     if (local_8 == 0x52b24) {
9         printf("Password OK :)\n");
10    }
11    else {
12        printf("Invalid Password!\n");
13    }
14    return 0;
15 }
16
17 }
```

Symbol Tree

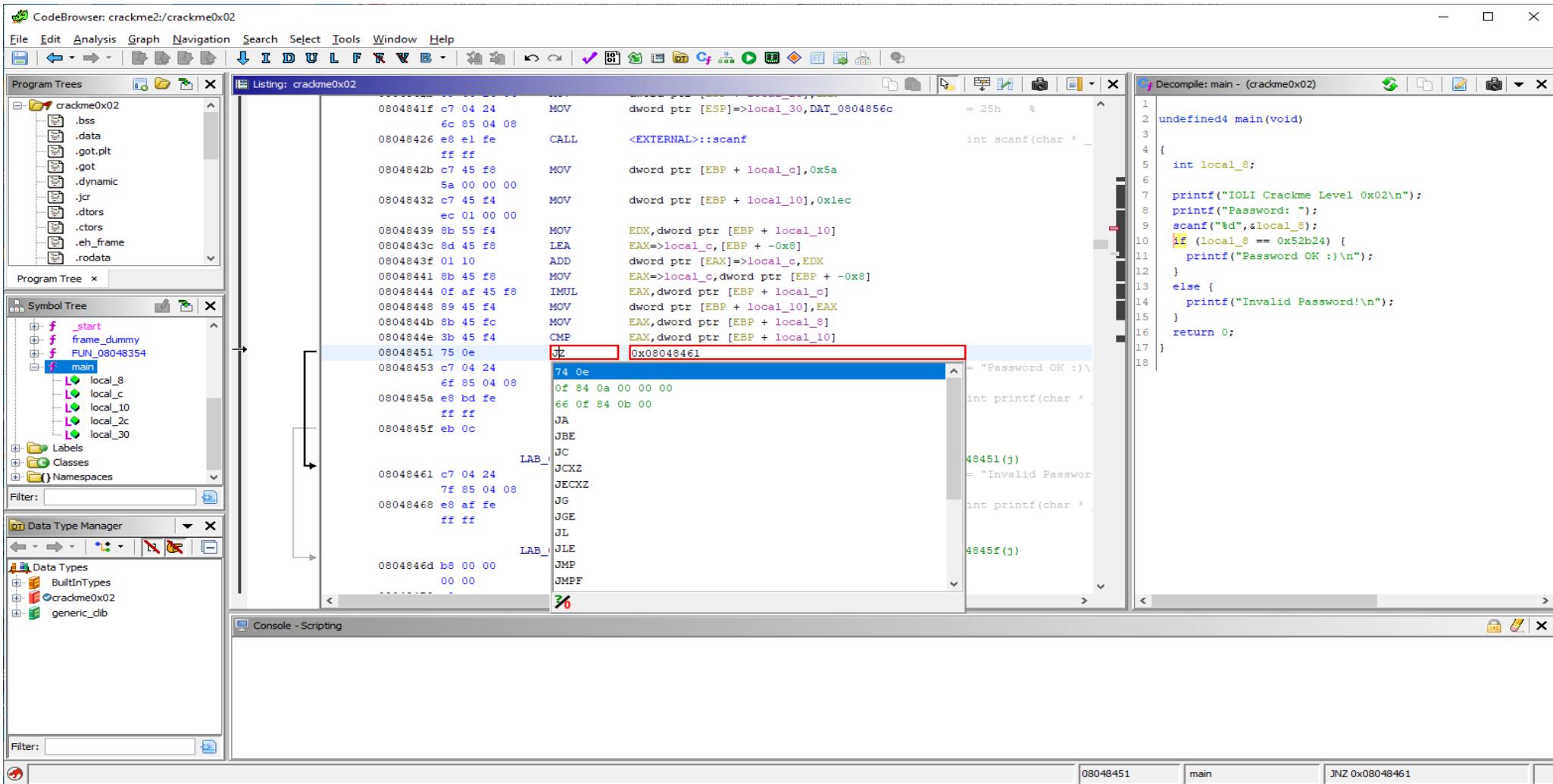
Data Type Manager

Console - Scripting

0804841f c7 04 24 MOV dword ptr [ESP] => local_30, DAT_0804856c = 25h %
08048426 e8 e1 fe CALL <EXTERNAL>::scanf
0804842b c7 45 f8 MOV dword ptr [EBP + local_c], 0x5a
08048432 c7 45 f4 MOV dword ptr [EBP + local_10], 0xlec
08048439 8b 55 f4 MOV EDX, dword ptr [EBP + local_10]
0804843c 8d 45 f8 LEA EAX=>local_c, [EBP + -0x8]
0804843f 01 10 ADD dword ptr [EAX]=>local_c, EDX
08048441 8b 45 f8 MOV EAX=>local_c, dword ptr [EBP + -0x8]
08048444 0f af 45 f8 IMUL EAX, dword ptr [EBP + local_c]
08048448 89 45 f4 MOV dword ptr [EBP + local_10], EAX
0804844b 8b 45 fc MOV EAX, dword ptr [EBP + local_8]
0804844e 3b 45 f4 CMP EAX, dword ptr [EBP + local_10]
08048451 75 0e JNE 0x08048461 = "Password OK :)\n"
08048453 c7 04 24 0f 85 0a 00 00 00
0804845a e8 bd fe 66 0f 85 0b 00
0804845f eb 0c JNC
08048461 c7 04 24 JNS 48451(j)
08048468 e8 af fe JNO = "Invalid Passwor
0804846d b8 00 00 LAB_4845f(j)
00 00 LAB_4845f(j)

Edit Assembly Code

- Hit Enter to save the change

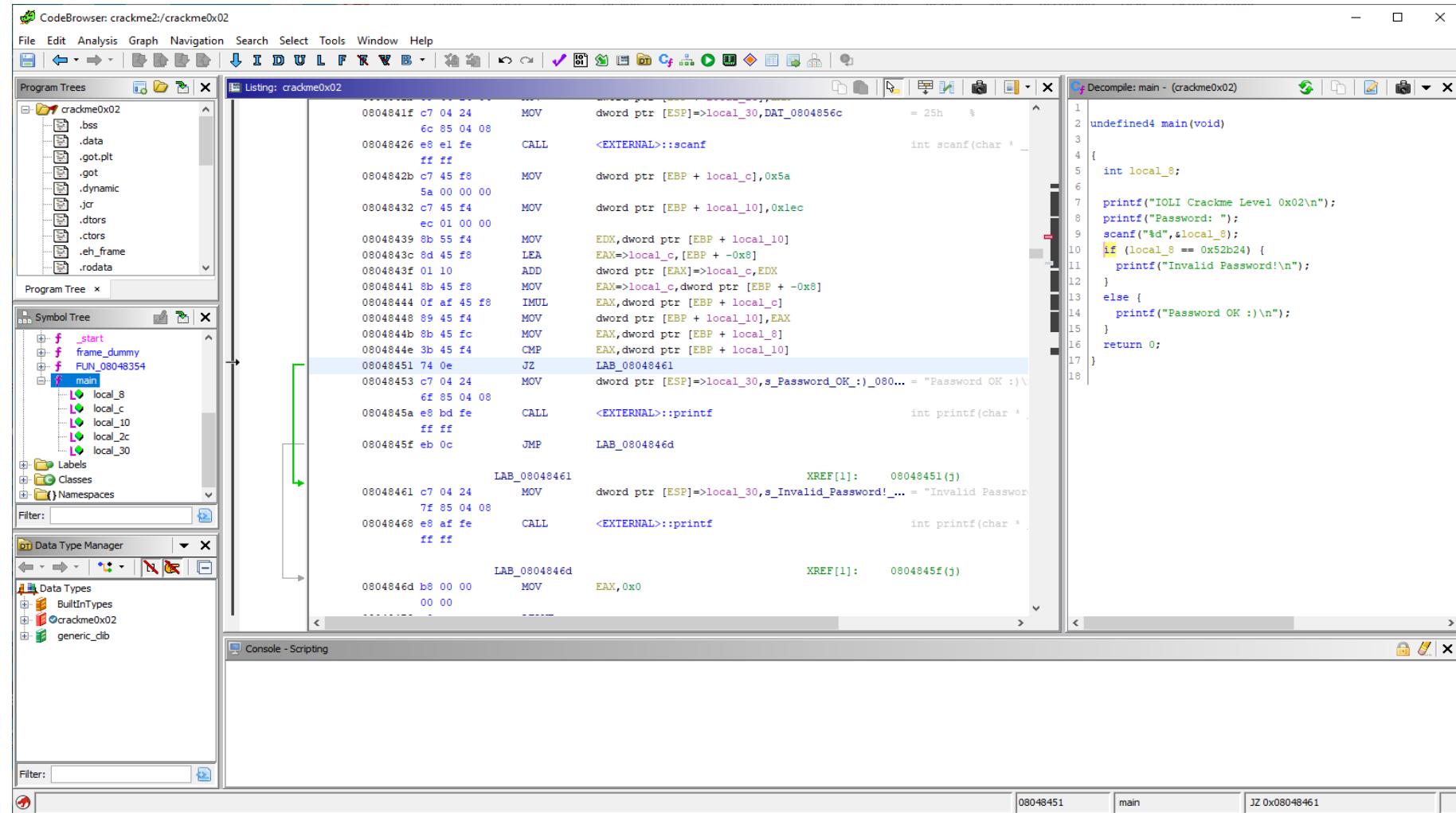


The screenshot shows the CodeBrowser interface for a binary file named 'crackme0x02'. The interface is divided into several panes:

- Program Trees**: Shows the file structure with sections like .bss, .data, .got.plt, .got, .dynamic, .jcr, .dtors, .ctors, .eh_frame, and .rodata.
- Symbol Tree**: Shows symbols including _start, frame_dummy, FUN_08048354, and main, which contains local variables local_8, local_c, local_10, local_2c, and local_30.
- Listing**: Displays assembly code with memory addresses, opcodes, and comments. A specific instruction at address 0x08048451 is highlighted with a red box and the value 'je 0x08048461' is shown in the status bar.
- Decompile**: Shows the corresponding C code for the assembly listing. It includes a printf statement for the password and an if-else block to check if the password is '52b24'. The assembly address 0x08048451 is also visible in the decompiled code.
- Console - Scripting**: An empty console window.
- Status Bar**: Shows the assembly address 08048451, the function name main, and the jump address JNZ 0x08048461.

If else condition changed!

- Our first static attack on Ghidra!
- Opcode 75 (JNZ) is now 74 (JZ)



The screenshot shows the Ghidra IDE interface with the following panes:

- Program Trees**: Shows the file structure of the binary: crackme0x02, containing sections like bss, .data, .got.plt, .got, .dynamic, .jcr, .dtors, .ctors, .eh_frame, and .rodata.
- Symbol Tree**: Shows symbols for main, local_8, local_c, local_10, local_2c, and local_30.
- Listing**: Displays the assembly code. A specific instruction at address 0x08048451 (opcode 74) is highlighted in blue, indicating it has been modified from JNZ to JZ.
- Decompile**: Shows the corresponding C decompilation of the main function. The original code had an if-else block where the password was checked against 0x52b24. The modified version checks against 0x0.
- Data Type Manager**: Shows the available data types: Data Types, BuiltInTypes, crackme0x02, and generic_dib.
- Console - Scripting**: An empty console window.

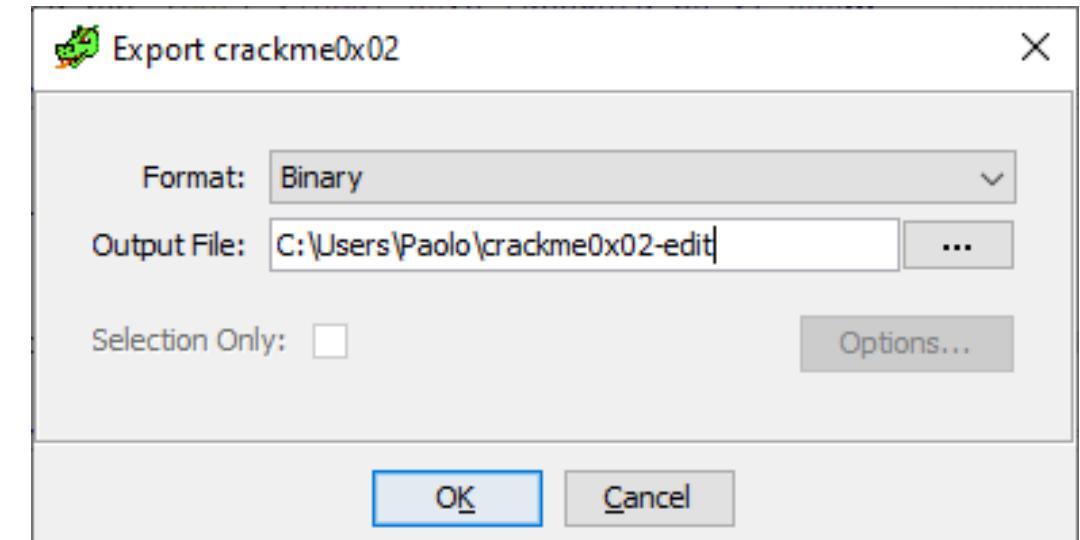
```

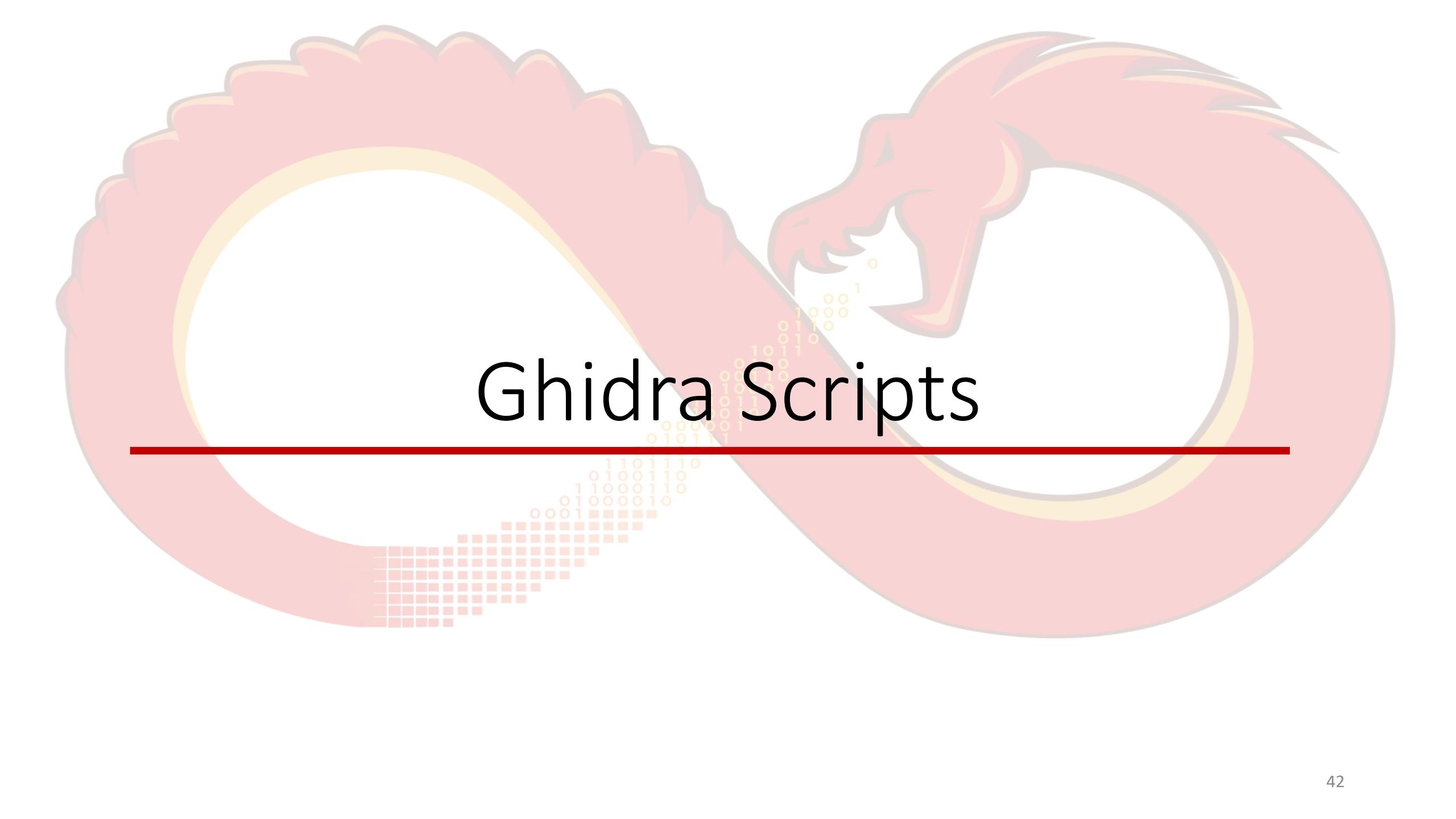
1 undefined4 main(void)
2 {
3     int local_8;
4
5     printf("IOLI Crackme Level 0x02\n");
6     printf("Password: ");
7     scanf("%d",&local_8);
8     if (local_8 == 0x52b24) {
9         printf("Invalid Password!\n");
10    }
11    else {
12        printf("Password OK :)\n");
13    }
14
15    return 0;
16 }
17
18

```

Export the edited binary to a new file

- Choose the Binary format
- In Linux make it executable
 - chmod a+rx crackme0x02-edit.bin
- Then run it and see the different behaviour!





Ghidra Scripts

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

Scripting With Ghidra



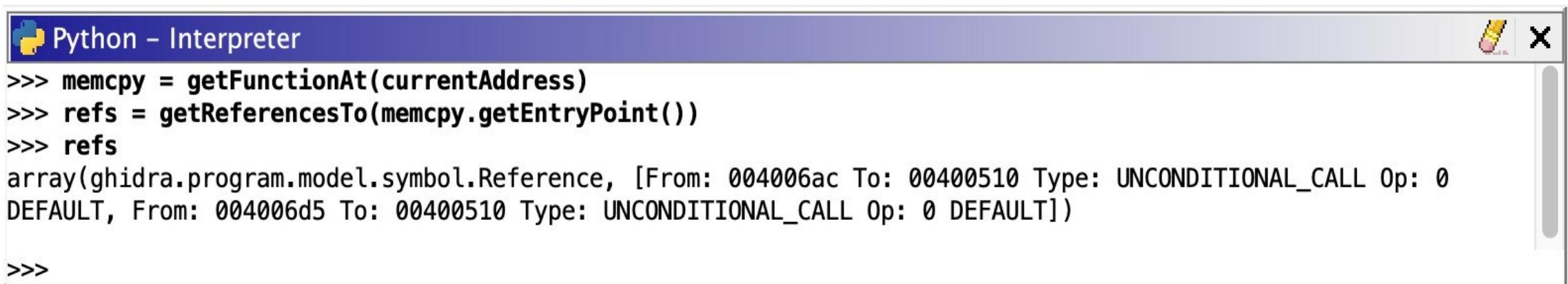
Ca' Foscari
University
of Venice

- 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
- Scripts for Analysis enhancements
 - Windows, Mac, Linux, VXWorks
 - PE, ELF, Mach-O, COFF
 - x86, MIPS, ARM/THUMB, 8051, etc...



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

A screenshot of the Python - Interpreter window in Ghidra. The window title is "Python - Interpreter". The content area shows Python code being executed:

```
>>> 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])
>>>
```

The code uses the Ghidra API to find a function at the current address, get its entry point, and then get all references to that entry point. The output shows two references: one from address 004006ac to 00400510 (Type: UNCONDITIONAL_CALL) and another from 004006d5 to 00400510 (Type: UNCONDITIONAL_CALL).

Ghidra APIs



Ca' Foscari
University
of Venice

- 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



Ca' Foscari
University
of Venice

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: We’ll be using Ghidra’s built-in basic editor – no need for it now



Scripting - Java vs Python

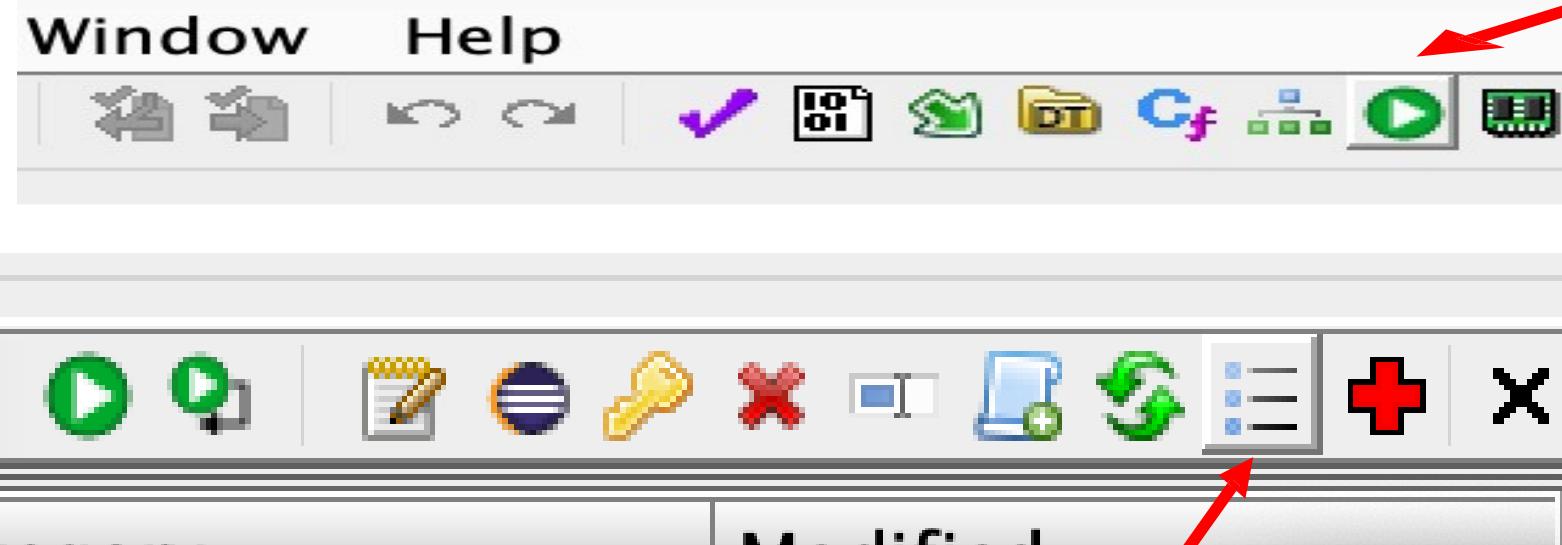
- Java will catch errors at compile time, Ghidra's API is highly object-oriented and benefits from this
- Complex Python scripts feel like binding together Java API calls with Python control flow and syntax
- Recommended workflow: prototype and experiment with APIs / objects in the Python interpreter,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



Importing Demo Scripts



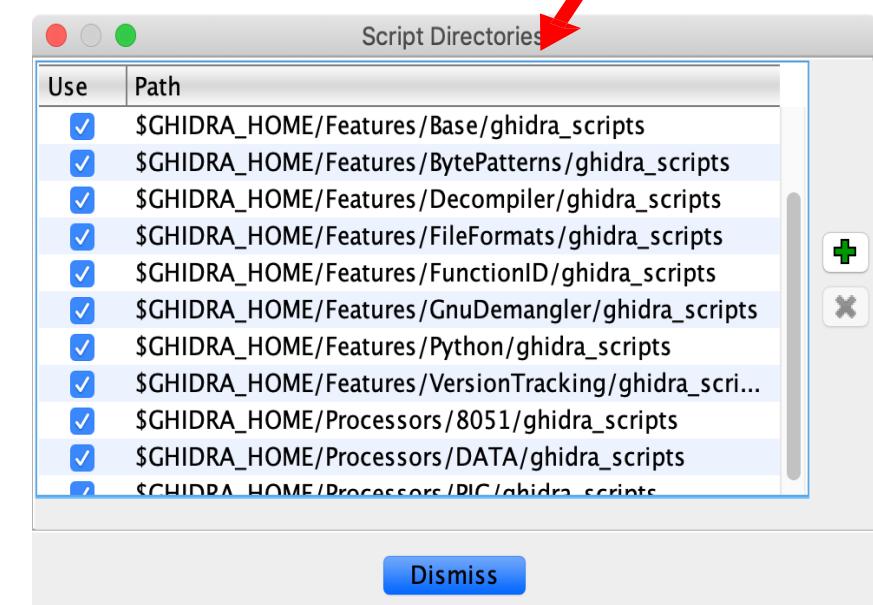
Ca' Foscari
University
of Venice



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

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



HelloWorld Script

Script Manager [CodeBrowser]

Edit Help

Script Manager - 256 scripts

In Tool	Status	Name	Description	Key	Category	Modified
		FixArrayListReferencesScript.java	When an array of structures is created at a...		Data Types	09/23/2021
		FixSwitchStatementsWithDecompiler.java	Fix any unknown switch instructions with the...		Analysis	09/23/2021
		FixupCompositeDataTypesScript.java	Fixes up all composite datatypes within the ...		Data Types	09/23/2021
		FixupELFExternalSymbolsScript.java	Fixes up any unresolved external symbols (f...		Symbol	09/23/2021
		FixupNoReturnFunctionsNoRepairScript.java	Attempt to detect defined functions in a pro...		Functions	09/23/2021
		FixupNoReturnFunctionsScript.java	Attempt to detect defined functions in a pro...		Functions	09/23/2021
		FixUpRttiAnalysisScript.java	Script to fix up Windows RTTI vtables and st...		C++	09/23/2021
		FormatExampleScript.java	<html>An example using the <code>pr...		Examples	09/23/2021
		FormatExampleScriptPy.py	An example using the Python string formatti...		Examples->Pyt...	09/23/2021
		FunctionIDHeadlessPostscript.java	Fixes switch statements and checks number/...		FunctionID	09/23/2021
		FunctionIDHeadlessPrescript.java	Turns off Function ID and Library Identificati...		FunctionID	09/23/2021
		GccRttiAnalysisScript.java	Script to create gcc RTTI vtables and struct...		C++	09/23/2021
		GenerateLotsOfProgramsScript.java				09/23/2021
		GenerateMaskedBitStringScript.java				09/23/2021
		GetAndSetAnalysisOptionsScript.java	Shows examples of how to get, set, and res...		Examples	09/23/2021
		GetSymbolForDynamicAddress.java			iOS	09/23/2021
		ghidra_basics.py	Examples of basic Ghidra scripting in Python		Examples->Pyt...	09/23/2021
		GraphAST.java	Decompile the function at the cursor, then b...		PCode	09/23/2021
		GraphASTAndFlow.java	Decompile the function at the cursor, then b...		PCode	09/23/2021
		GraphClassesScript.java	Script to graph class hierarchies given metad...		C++	09/23/2021
		GraphSelectedAST.java	Decompile the function at the cursor, then b...		PCode	09/23/2021
		HelloWorldPopupScript.java	Writes "Hello World" in a popup dialog.		Examples	09/23/2021
		HelloWorldScript.java	Writes "Hello World" to console. Ctrl-Shift-C...		Examples	09/23/2021
		IdPeRttiScript.java	Script to fix up Windows RTTI vtables and st...		C++	09/23/2021
		ImportAllProgramsFromADirectoryScript.java	Imports all programs from a selected directo...		Import	09/23/2021
		ImportMSLibs.java	Massive recursive import for a MS Visual Stu...		FunctionID	09/23/2021
		ImportProgramScript.java	Imports a program and opens it in the curre...		Import	09/23/2021
		ImportSymbolsScript.py	Imports a file with lines in the form "symbolN...		Data	09/23/2021

Filter: Filter:

HelloWorldScript.java

Console – Scripting Output

HelloWorldScript.java> Running...

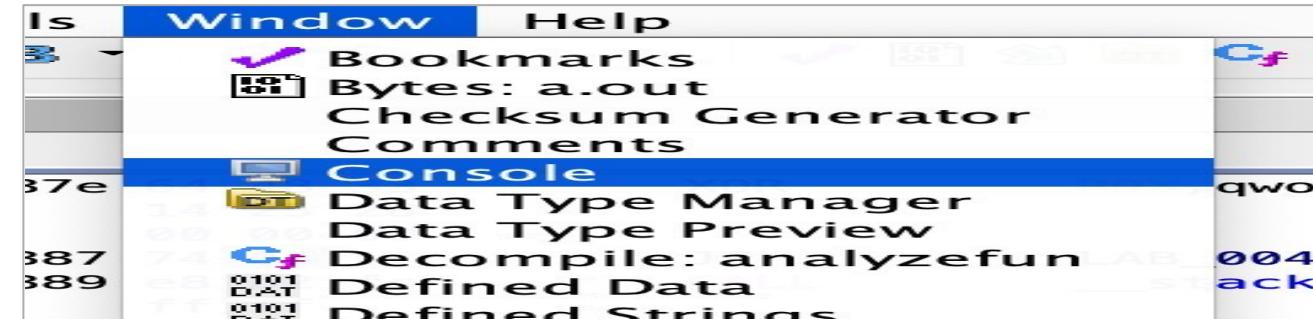
HelloWorldScript.java> Hello
World

HelloWorldScript.java> Finished!

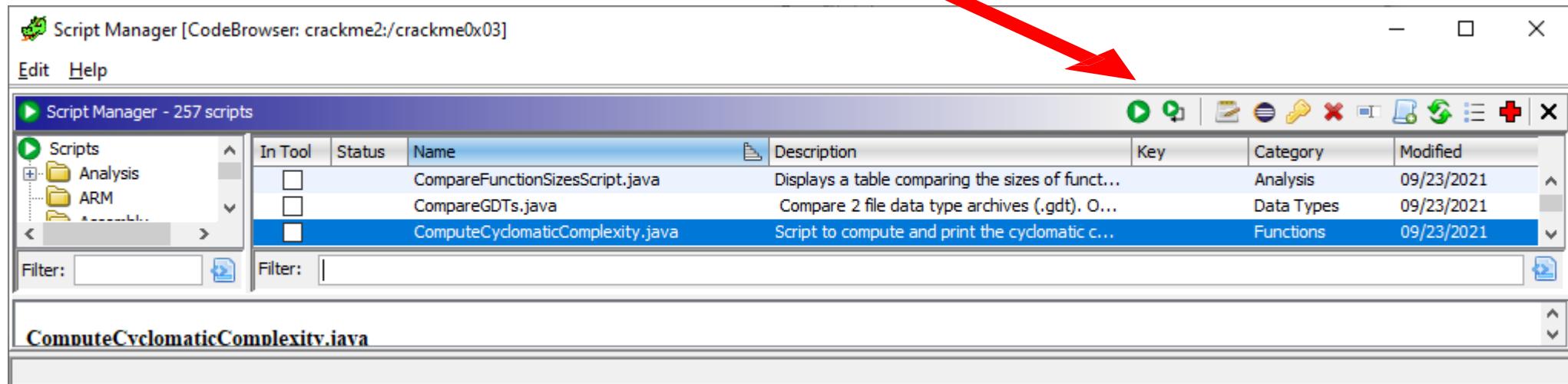
Running Script Demos



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



Choose a script ComputeCyclomaticComplexity and click “Run Script” to run





Calculating Cyclomatic Complexity

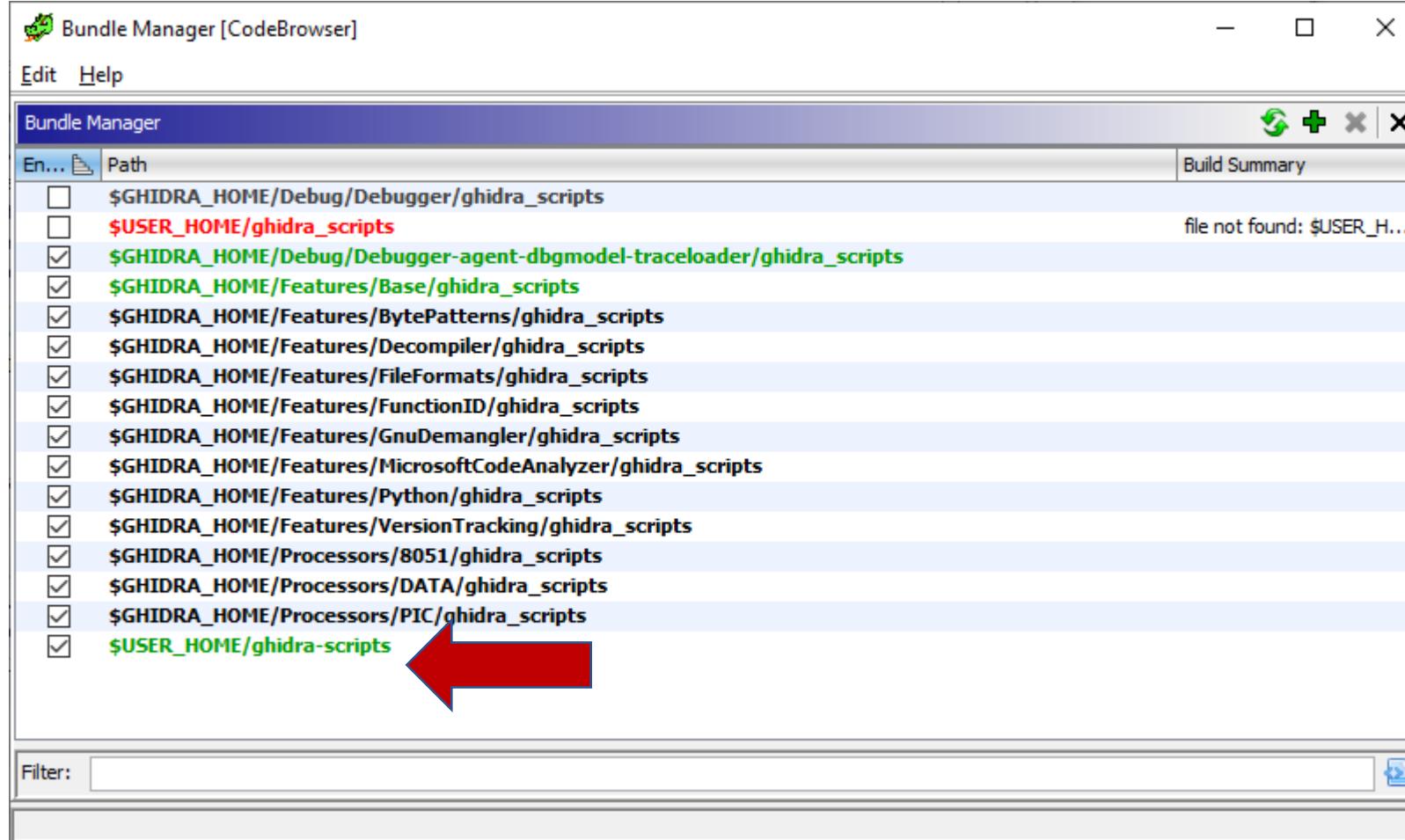
- Cyclomatic complexity is a measure of the number of unique paths which may be taken through a given function.
- The cyclomatic complexity M is then defined as

$$M = E - N + 2P$$

where

- E = the number of edges of the graph.
- N = the number of nodes of the graph.
- P = the number of connected components in the Control Flow Graph
- Leverage Ghidra's API for calculating cyclomatic complexity to easily analyze a whole program
- It can be helpful in finding complex functions likely to have vulnerabilities, e.g. complex parsing routines or state machines.

Add Your Script Folder

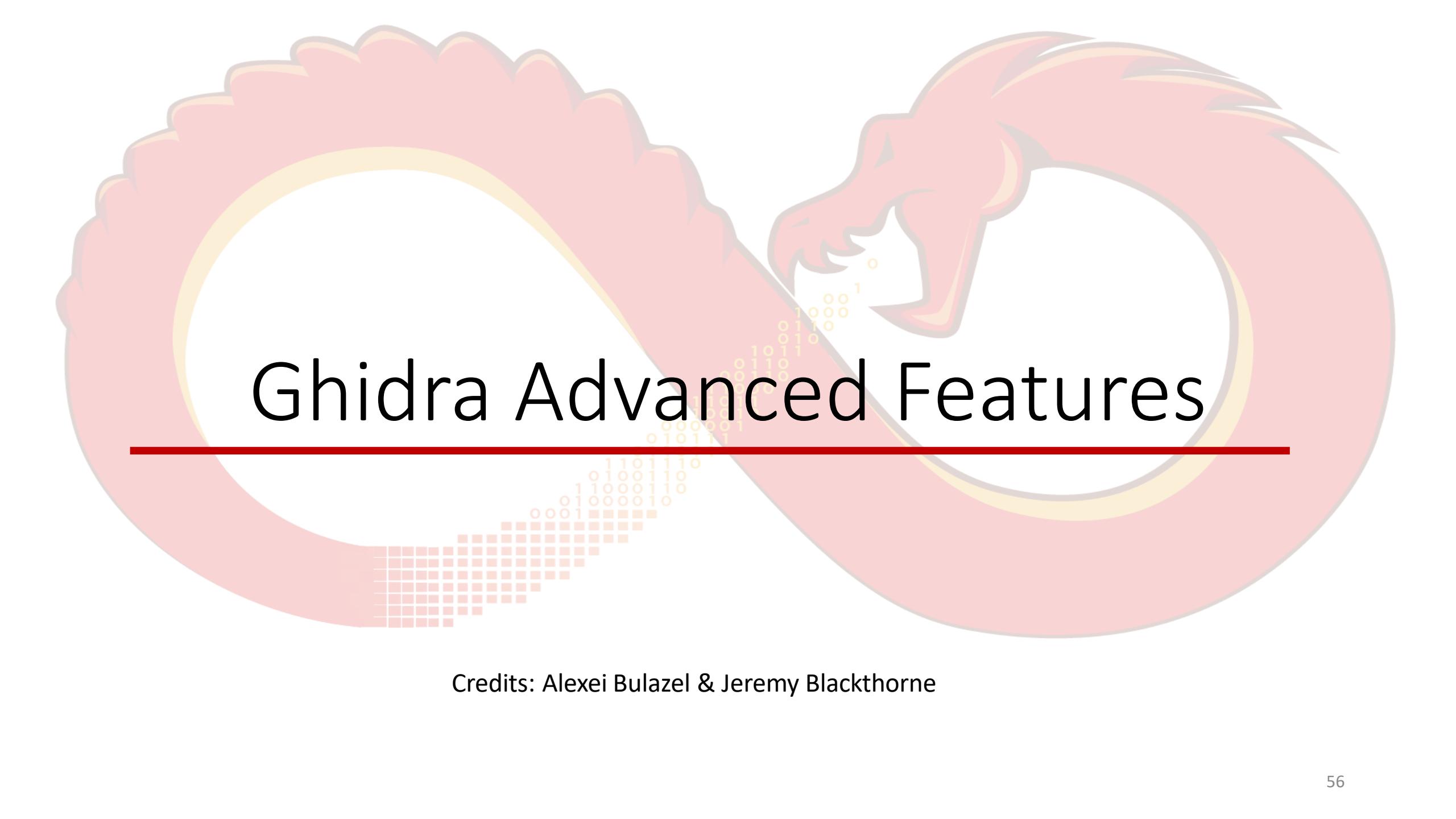


Browse to your script folder to add it to the other bundles

Task



- Now look at the other available scripts and try them on the crackme files or any other files you want.
- Explain your findings to the class!



Ghidra Advanced Features

Credits: Alexei Bulazel & Jeremy Blackthorne

Version Tracker - Overview



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

- Not well suited for quick 1-day discovery in patch analysis
 - Use Diaphora or BinDiff for this purpose

The screenshot shows the Version Tracker interface with three main windows:

- Version Tracking Matches**: A table showing 354 matches between two versions of the WallaceSrc.exe program. The columns include Tag, Session, Status, Type, Score, Confidence, Votes, # Co..., Multi..., Source Name..., Source Label, Source Address, Multi..., Dest Name..., Dest Label, Dest Address, Source..., Dest L..., and Algorithm. Many entries show "No Symbol" for both source and destination labels.
- Version Tracking Markup Items**: A table showing 3 markup items. It includes columns for Status, Source Address, Dest Address, Markup Type, Source Value, Current Dest Value, and Original Dest Value. The items are:
 - Source Address: 00412200, Dest Address: 004121e0, Type: Plate Comment, Value: Library Function - Single Match Nam...
 - Source Address: 00412200, Dest Address: 004121e0, Type: Function Name, Value: _NtCurrentTeb
 - Source Address: 00412200, Dest Address: 004121e0, Type: Function Signature, Value: _TEB * _NtCurrentTeb(void)
- Decompile View**: Compares assembly code for the function _NtCurrentTeb between the two versions. The source code is in WallaceSrc.exe and the destination is in WallaceVersion2.exe. The assembly shows a MOV instruction from EDI to EDI at address 00412200.

Version Tracker - Selecting Correlation Algorithms



Ca' Foscari
University
of Venice

Version Tracking Wizard

Select Correlation Algorithm(s)

S...	Name	P...	Description
<input checked="" type="checkbox"/>	Exact Data Match		C.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		C.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		C.compares code by hashing instructions, looking for identical functions. It reports back any that have ONLY...
<input checked="" type="checkbox"/>	Exact Function Mnemonics Match		C.compares code by hashing instructions mnemonics, looking for identical functions. It reports back any t...
<input type="checkbox"/>	Exact Symbol Name Match		C.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		C.compares data by iterating over all defined data meeting the minimum size requirement in the source pro...
<input type="checkbox"/>	Duplicate Function Instructions Match		C.compares code by hashing instructions (masking off operands), looking for identical functions. It report...
<input type="checkbox"/>	Duplicate Exact Symbol Name Match		C.compares symbols by iterating over all defined function and data symbols meeting the minimum size req...
<input type="checkbox"/>	Similar Symbol Name Match		C.compares symbols by iterating over all defined function and data symbols meeting the minimum size req...
<input type="checkbox"/>	Similar Data Match		C.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 <RETURN>  
int undefined4  
Stack[-0x8]:4 local_8  
Stack[-0x10]:4 local_10  
Stack[-0x18]:4 local_18  
int Stack[-0x24]:4 pp  
Stack[-0xf0]:4 gadget  
Stack[-0xfc]:4 local_fc  
undefined4 * [2] Stack[-0x104... gadgetLocal  
  
deployGadget
```

Destination: FUN_004118c0() in /WallaceVersion2.exe

```
*****  
*          FUNCTION          *  
*****  
undefined AL:1 __stdcall deployGadget(void)  
undefined undefined4  
Stack[-0x8]:4 local_8  
Stack[-0x10]:4 local_10  
Stack[-0x18]:4 local_18  
undefined4  
Stack[-0x24]:4 local_24  
undefined4  
Stack[-0xf0]:4 local_f0  
Stack[-0xfc]:4 local_fc  
undefined4  
Stack[-0x104... local_104  
  
deployGadget
```

XREF[1]: 004119a5(W) XREF[2]: 0041193d(W) XREF[3]: 0041197b(W) XREF[4]: 00411924(*) XREF[5]: 004119cb(R) XREF[6]: 00411988(W) XREF[7]: 00411990(W) XREF[8]: 00411993(R) XREF[9]: 0041199e(R) XREF[10]: 004119b1(R) XREF[11]: 004119bd(R) XREF[12]: 004119c3(W) XREF[13]: 00411937(W) XREF[14]: 00411944(R) XREF[15]: 00411952(R) XREF[16]: 00411975(W) XREF[17]: 0041199a(*) XREF[18]: 0041195d(W) XREF[19]: 00411965(W) XREF[20]: 0041196f(R) XREF[21]: 0041190d(W) XREF[22]: 0041194b(W) XREF[23]: 004118f4(*) XREF[24]: 004119a6(R) XREF[25]: 00411958(W) XREF[26]: 00411990(R) XREF[27]: 004119a3(R) XREF[28]: 00411960(W) XREF[29]: 00411963(R) XREF[30]: 0041196e(R) XREF[31]: 00411981(R) XREF[32]: 0041198c(R) XREF[33]: 00411998(R) XREF[34]: 0041199e(W) XREF[35]: 00411907(W) XREF[36]: 00411914(R) XREF[37]: 00411922(R) XREF[38]: 00411945(W) XREF[39]: 00411952(R) XREF[40]: 004118da(*) XREF[41]: 0041192d(W) XREF[42]: 00411935(W) XREF[43]: 0041193f(R)

XREF[1]: deployGadget:0041116 XREF[2]: deployGadget:0041116 XREF[3]: deployGadget:0041116 XREF[4]: deployGadget:0041116 XREF[5]: deployGadget:0041116 XREF[6]: deployGadget:0041116 XREF[7]: deployGadget:0041116 XREF[8]: deployGadget:0041116 XREF[9]: deployGadget:0041116 XREF[10]: deployGadget:0041116 XREF[11]: deployGadget:0041116 XREF[12]: deployGadget:0041116 XREF[13]: deployGadget:0041116 XREF[14]: deployGadget:0041116 XREF[15]: deployGadget:0041116 XREF[16]: deployGadget:0041116 XREF[17]: deployGadget:0041116 XREF[18]: deployGadget:0041116 XREF[19]: deployGadget:0041116 XREF[20]: deployGadget:0041116 XREF[21]: deployGadget:0041116 XREF[22]: deployGadget:0041116 XREF[23]: deployGadget:0041116 XREF[24]: deployGadget:0041116 XREF[25]: deployGadget:0041116 XREF[26]: deployGadget:0041116 XREF[27]: deployGadget:0041116 XREF[28]: deployGadget:0041116 XREF[29]: deployGadget:0041116 XREF[30]: deployGadget:0041116 XREF[31]: deployGadget:0041116 XREF[32]: deployGadget:0041116 XREF[33]: deployGadget:0041116 XREF[34]: deployGadget:0041116 XREF[35]: deployGadget:0041116 XREF[36]: deployGadget:0041116 XREF[37]: deployGadget:0041116 XREF[38]: deployGadget:0041116 XREF[39]: deployGadget:0041116 XREF[40]: deployGadget:0041116 XREF[41]: deployGadget:0041116 XREF[42]: deployGadget:0041116 XREF[43]: deployGadget:0041116

Program Differences



Ca' Foscari
University
of Venice

The screenshot shows a software interface for comparing binary files. The main window has a menu bar with File, Edit, Analysis, Navigation, Search, Select, Tools, Window, and Help. The Tools menu is open, showing options: Processor Manual..., Program Differences... 2, and Generate Checksum... The "Program Differences..." option is highlighted. A toolbar below the menu contains icons for file operations like Open, Save, and Cut/Paste. On the left, a "Program Trees" panel shows a tree view of the "WallaceSrc.exe" file structure, including Headers, .textbss, .text, .rdata, .data, and .idata. In the center, there are two panes labeled "vt:/WallaceSrc.exe" and "vt:/WallaceSrc.exe". A modal dialog box titled "Determine Program Differences" is open over the main window. It contains a section "Do Differences On" with checkboxes for Bytes, References, Bookmarks, Labels, Program Context, Properties, Code Units, Comments, and Functions. All checkboxes are checked. Below this are buttons for "Select All" and "Deselect All". Under "Address Ranges To Diff", there is a checkbox for "Limit To Selection" which is unchecked. A text input field says "Entire Program". At the bottom of the dialog are "OK" and "Cancel" buttons.

Program Differences



WallaceSrc.exe x

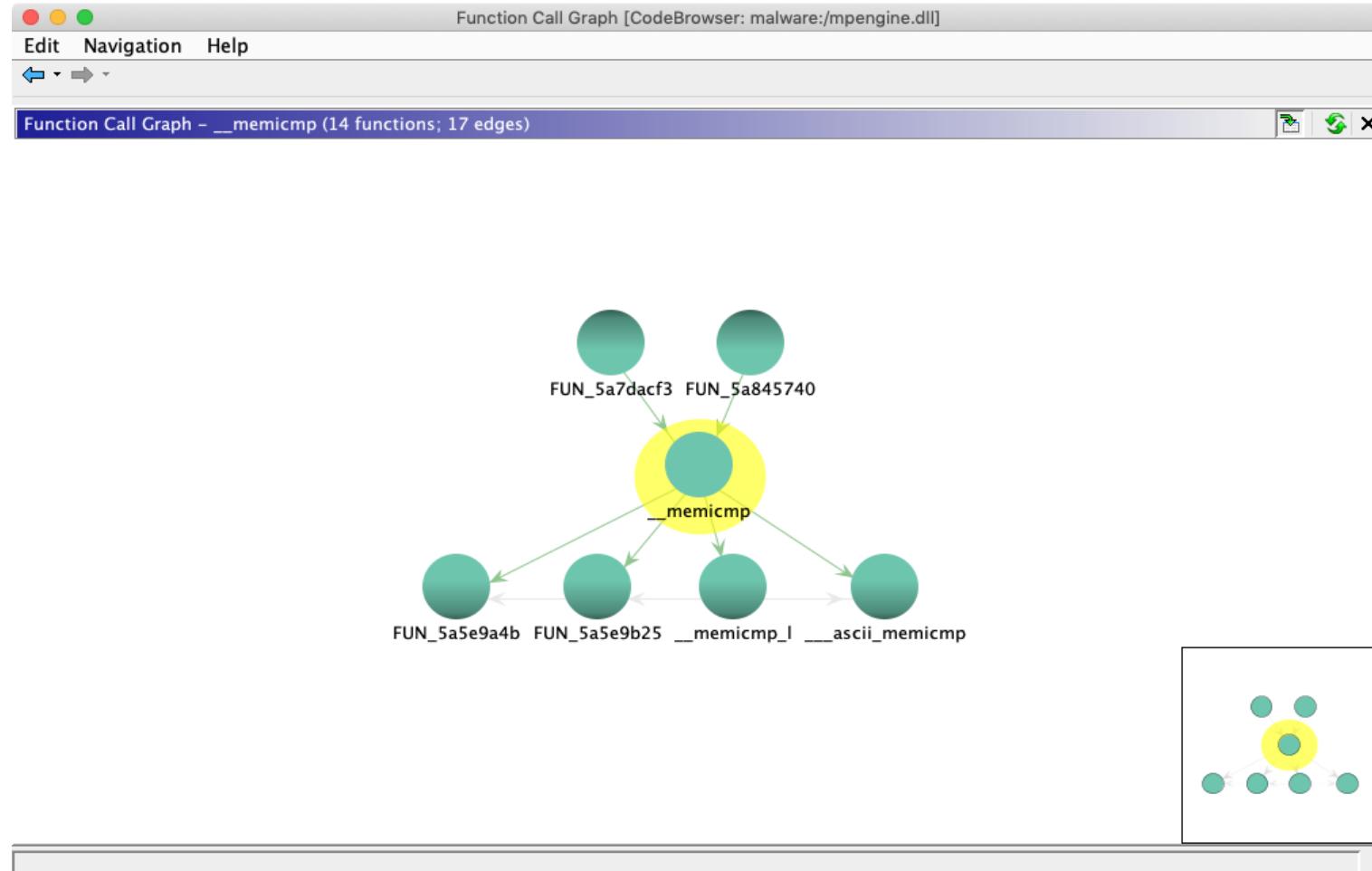
vt:/WallaceSrc.exe

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 PUSH LAB_00414b2e 41 00	
004118fa 64 a1 00 MOV EAX,FS:[0x0] 00 00 00	
00411900 50 PUSH EAX	
00411901 81 ec f4 SUB ESP,0xf4 00 00 00	
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 MOV FS:[0x0],EAX 00 00 00	
004118fd 6a 10 PUSH 0x10 004118ff e8 b4 f8 CALL operator_new ff ff	
00411904 83 c4 04 ADD ESP,0x4	
00411907 89 85 14 MOV dword ptr [local_f0 + EBPI],EAX ff ff ff	

Function Call Graph



Decompiler Slicing

- Forward Slice consists of all statements in program P that may be affected by the value of variable v in a statement S
- Backward Slice consists of all statements in program P that may affect the value of variable v in a statement S

A screenshot of a decompiler interface, likely Jadx or similar, showing assembly code. A context menu is open over a variable named `iVar5`. The menu items are:

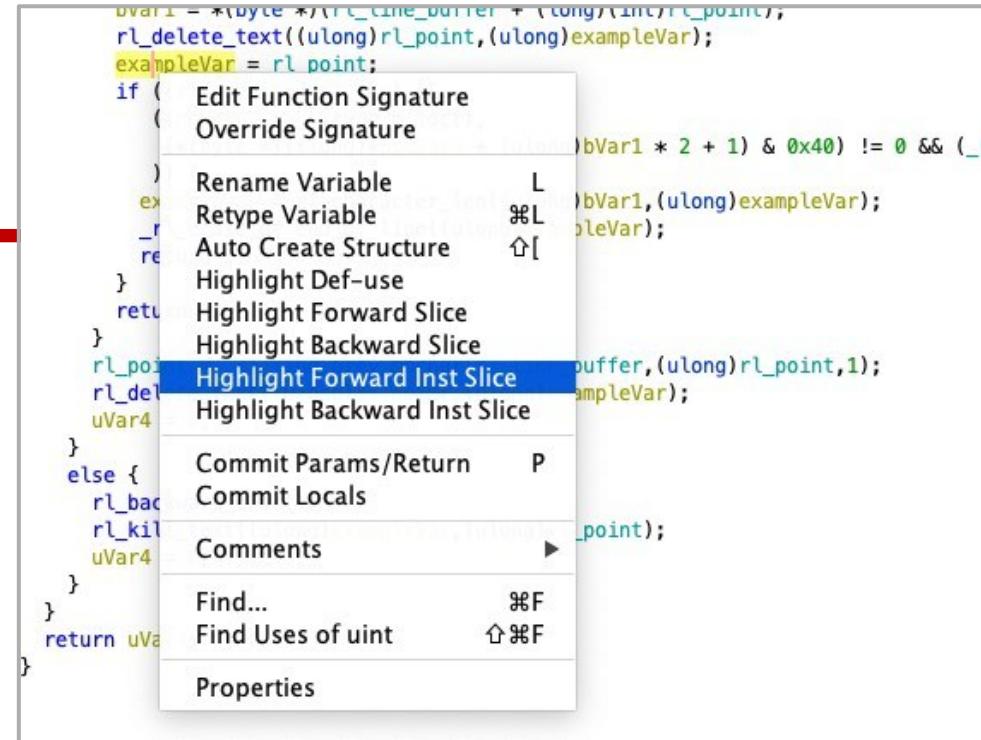
- exampleVar = r1_point,
- iVar5 = (int)uParm1;
- if (iVar5 < 0) {
- uVar4
- return
- }
- if (rl_
- rl_did
- uVar4
- }
- else {
- if ((
- sVa
- if
- r
- b
- r
- e
- i
-)
-)
-)
- }
- }
- }
- return 0;

The menu includes options for editing function signatures, renaming variables, retying variables, auto-creating structures, and highlighting def-use and slice regions. It also has commit and find functions.

Slicing

```
undefined8 _rl_rubout_char(ulong uParm1)
{
    byte bVar1;
    size_t sVar2;
    ushort **ppuVar3;
    undefined8 uVar4;
    int iVar5;
    uint exampleVar;

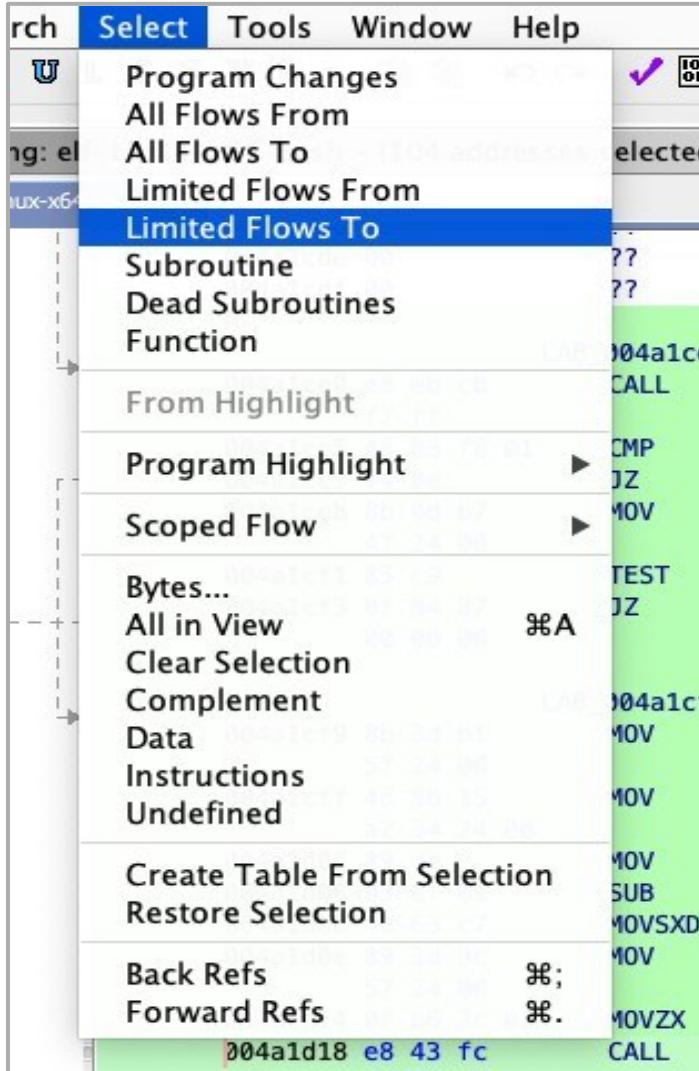
    exampleVar = rl_point;
    iVar5 = (int)uParm1;
    if (iVar5 < 0) {
        uVar4 = rl_delete(uParm1 & 0xffffffff00000000 | (ulong)(uint)-iVar5);
        return uVar4;
    }
    if (rl_point == 0) {
        rl_ding();
        uVar4 = 0xffffffff;
    }
    else {
        if ((iVar5 < 2) && (rl_explicit_arg == 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;
}
```



```
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
    )) {
    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;
}
```

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

```
MOVSDX    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.

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
NOT	SF = INT_SLESS \$Uac60, 0:1
	ZF = INT_EQUAL \$Uac60, 0:1
	\$U22f0:1 = BOOL_NEGATE ZF
	CBRANCH *[ram]0x401546:8, \$U22f0
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_SLESEQUAL, INT_LESS, INT_LESEQUAL
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_LESEQUAL
FP Conversion	INT2FLOAT, FLOAT2FLOAT, TRUNC, CEIL, FLOOR, ROUND
Branching	BRANCH, CBRANCH, BRANCHIND, CALL, CALLIND, RETURN
Extension / Truncation	INT_ZEXT, INT_SEXT, PIECE, SUBPIECE

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



Ca' Foscari
University
of Venice

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

- 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*



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



Ca' Foscari
University
of Venice

```
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>  
    undefined Stack[-0xc]:4 local_c  
    undefined 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
```



Ghidra is good for...

- Scripting reverse engineering
- Firmware / embedded systems analysis
- Analysis of software that Hex-Rays can't decompile
- Collaborative long-term professional RE
- Unique windows
 - Checksum Generator
 - Disassembled View
 - Data Type Preview
 - Function Tags
 - Symbol tree

Ghidra Pros



Ca' Foscari
University
of Venice

- 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

Ghidra Pros



- 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

Contributing to Ghidra



Ca' Foscari
University
of Venice

- Ghidra code is available on Github
 - Apache License 2.0
- NSA has been responsive to community questions and bug reports posted on Github
- Official site: ghidra-sre.org
- Open source: github.com/NationalSecurityAgency/ghidra
github.com/NationalSecurityAgency/ghidra-data

Conclusions



Ca' Foscari
University
of Venice

- 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

