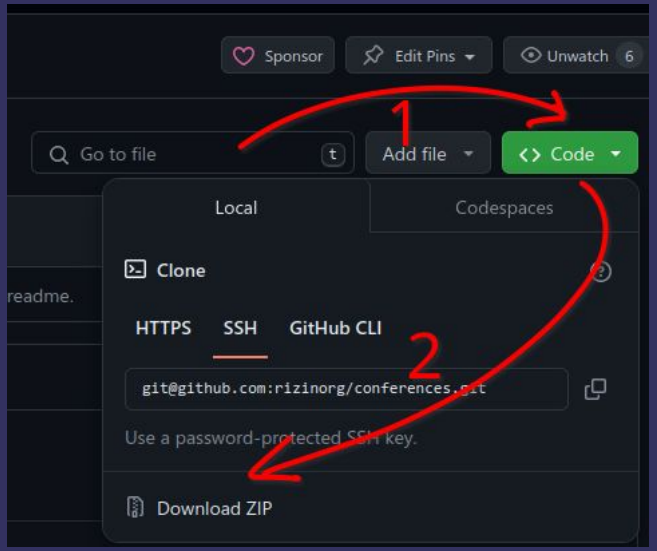
Rizin Workshop - Essential Material

1. Go to: https://github.com/rizinorg/conferences
2. Download the zip (see image)
3. Unzip the downloaded file.
4. Open the terminal
   - Windows users should use: powershell
5. Install rizin using the enclosed script.
   - Windows users: `download-rizin.ps1`
   - Linux & Mac users: `install-rizin.sh`
6. Check if rizin works by executing

```
rizin -qc "clippy hello"
```



Auditing proprietary apps using Rizin

Free and Open Source Reverse Engineering Framework

fossasia

RizinOrg

Who Are We

Anton Kochkov
Co-Founder

Giovanni Grazioli
Co-Founder

RizinOrg

What is Reverse Engineering

Reverse engineering is the process of analyzing a compiled program or software system to understand its inner workings, design, and functionality. This is especially useful to understand legacy systems, recovering lost source code, analyzing malware, security audits and more...

RizinOrg

rizin

Rizin is a Unix-friendly approach on reverse engineering.

It can be used in a shell-like environment tailored specifically to analyzing binaries directly from the command line without unnecessary weight. Rizin comes with various tools that enrich capabilities of a reverse engineer.

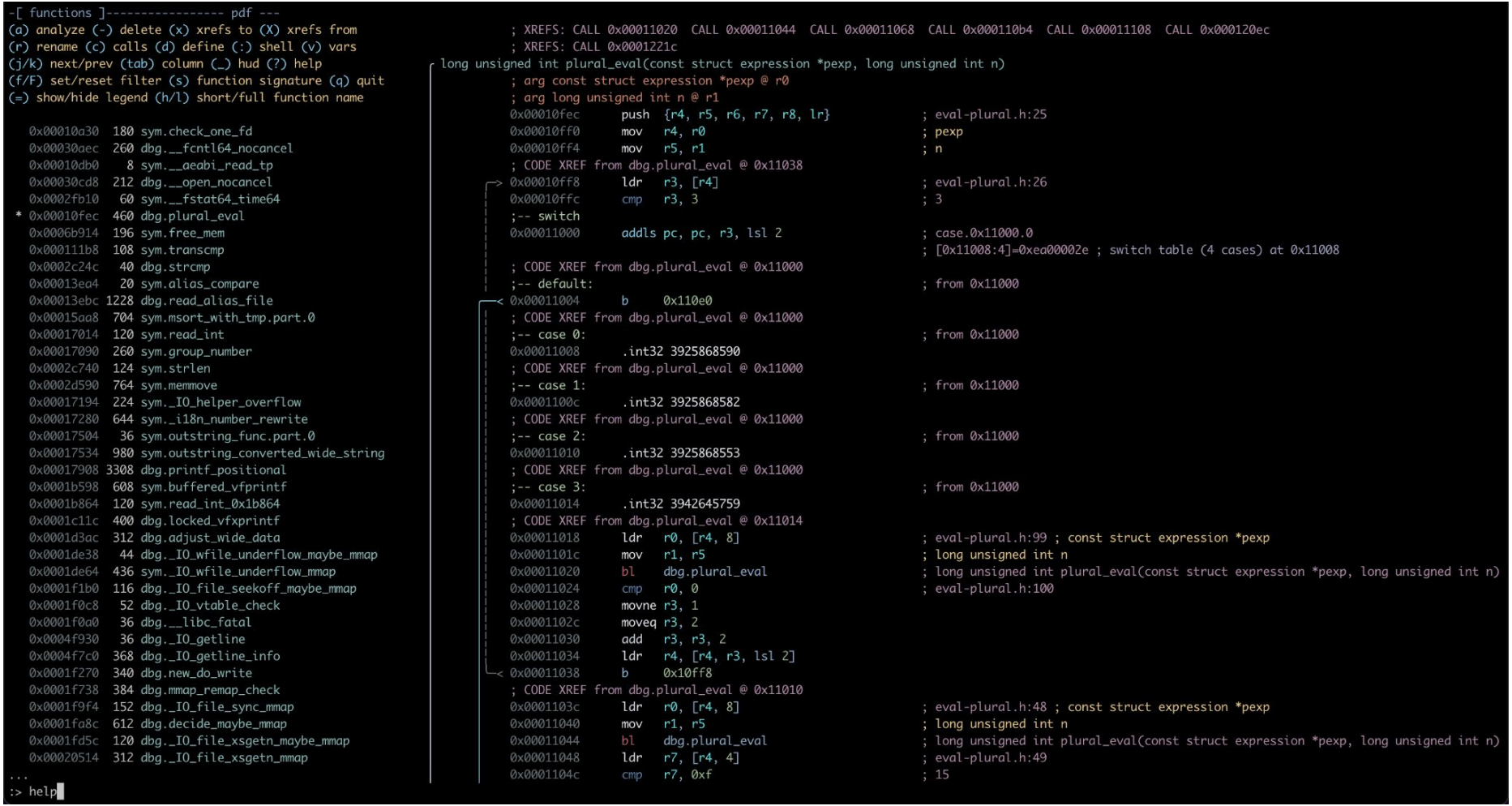
What is Rizin?

RizinOrg

Rizin tools

1. Rizin is a framework, which is packaged as a C library (librz), thus can be used separately from Rizin tools, a good example is Cutter that uses it like that.
2. Main “entrypoint” and shell is the **rizin** tool itself
3. Other, task-specific tools are available:
   a. **rz-bin** - provides all kind of information about binary formats
   b. **rz-asm** - a command-line assembler and disassemblers
   c. **rz-diff** - a tool to compare two binaries as raw data or analyzed executables
   d. **rz-hash** - allows to calculate different hashes or even encrypt data
   e. **rz-gg** - a small "eggs" code generator useful for exploitation purposes
   f. **rz-find** - binary analog of find tool, allowing to search patterns and bit masks
   g. **rz-sign** - tool to create, convert and parse FLIRT signatures
   h. **rz-ax** - a calculator and number format converter
   i. **rz-run** - a tool that allows to specify running environment and arguments for debugged file

RizinOrg


-[functions]---------- pdf ---
(a) analyze (-) delete (x) xrefs to (X) xrefs from
(r) rename (c) calls (d) define (:) shell (v) vars
(j/k) next/prev (tab) column (_) hud (?) help
(f/F) set/reset filter (s) function signature (q) quit
(=) show/hide legend (h/l) short/full function name
0x00010a30 180 sym.check_one_fd
0x00030aec 260 dbg.__fcntl64_nocancel
0x00010db0 8 sym.__aeabi_read_tp
0x00030cd8 212 dbg.__open_nocancel
0x0002fb10 60 sym.__fstat64_time64
* 0x00010fec 460 dbg.plural_eval
0x0006b914 196 sym.free_mem
0x000111b8 108 sym.transcmp
0x0002c24c 40 dbg.strcmp
0x00013ea4 20 sym.alias_compare
0x00013ebc 1228 dbg.read_alias_file
0x00015aa8 704 sym.msort_with_tmp.part.0
0x00017014 120 sym.read_int
0x00017090 260 sym.group_number
0x0002c740 124 sym.strlen
0x0002d590 764 sym.memmove
0x00017194 224 sym._IO_helper_overflow
0x00017280 644 sym._i18n_number_rewrite
0x00017504 36 sym.outstring_func.part.0
0x00017534 980 sym.outstring_converted_wide_string
0x00017908 3308 dbg.printf_positional
0x0001b598 608 sym.buffered_vfprintf
0x0001b864 120 sym.read_int_0x1b864
0x0001c11c 400 dbg.locked_vfxprintf
0x0001d3ac 312 dbg.adjust_wide_data
0x0001de38 44 dbg._IO_wfile_underflow_maybe_mmap
0x0001de64 436 sym._IO_wfile_underflow_mmap
0x0001f1b0 116 dbg._IO_file_seekoff_maybe_mmap
0x0001f0c8 52 dbg._IO_vtable_check
0x0001f0a0 36 dbg.__libc_fatal
0x0004f930 36 dbg._IO_getline
0x0004f7c0 368 dbg._IO_getline_info
0x0001f270 340 dbg.new_do_write
0x0001f738 384 dbg.mmap_remap_check
0x0001f9f4 152 dbg._IO_file_sync_mmap
0x0001fa8c 612 dbg.decide_maybe_mmap
0x0001fd5c 120 dbg._IO_file_xsgetn_maybe_mmap
0x00020514 312 dbg._IO_file_xsgetn_mmap
...
:> help
; XREFS: CALL 0x00011020 CALL 0x00011044 CALL 0x00011068 CALL 0x000110b4 CALL 0x00011108 CALL 0x000120ec
; XREFS: CALL 0x0001221c
long unsigned int plural_eval(const struct expression *pexp, long unsigned int n)
; arg const struct expression *pexp @ r0
; arg long unsigned int n @ r1
0x00010fec push {r4, r5, r6, r7, r8, lr} ; eval-plural.h:25
0x00010ff0 mov r4, r0 ; pexp
0x00010ff4 mov r5, r1 ; n
; CODE XREF from dbg.plural_eval @ 0x11038
0x00010ff8 ldr r3, [r4] ; eval-plural.h:26
0x00010ffc cmp r3, 3 ; 3
;-- switch
0x00011000 addls pc, pc, r3, lsl 2 ; case.0x11000.0
; [0x11008:4]=0xea00002e ; switch table (4 cases) at 0x11008
; CODE XREF from dbg.plural_eval @ 0x11000
;-- default:
0x00011004 b 0x110e0 ; from 0x11000
; CODE XREF from dbg.plural_eval @ 0x11000
;-- case 0:
0x00011008 .int32 3925868590 ; from 0x11000
; CODE XREF from dbg.plural_eval @ 0x11000
;-- case 1:
0x0001100c .int32 3925868582 ; from 0x11000
; CODE XREF from dbg.plural_eval @ 0x11000
;-- case 2:
0x00011010 .int32 3925868553 ; from 0x11000
; CODE XREF from dbg.plural_eval @ 0x11000
;-- case 3:
0x00011014 .int32 3942645759 ; from 0x11000
; CODE XREF from dbg.plural_eval @ 0x11014
0x00011018 ldr r0, [r4, 8] ; eval-plural.h:99 ; const struct expression *pexp
0x0001101c mov r1, r5 ; long unsigned int n
0x00011020 bl dbg.plural_eval ; long unsigned int plural_eval(const struct expression *pexp, long unsigned int n)
0x00011024 cmp r0, 0 ; eval-plural.h:100
0x00011028 movne r3, 1
0x0001102c moveq r3, 2
0x00011030 add r3, r3, 2
0x00011034 ldr r4, [r4, r3, lsl 2]
0x00011038 b 0x10ff8
; CODE XREF from dbg.plural_eval @ 0x11010
0x0001103c ldr r0, [r4, 8] ; eval-plural.h:48 ; const struct expression *pexp
0x00011040 mov r1, r5 ; long unsigned int n
0x00011044 bl dbg.plural_eval ; long unsigned int plural_eval(const struct expression *pexp, long unsigned int n)
0x00011048 ldr r7, [r4, 4] ; eval-plural.h:49
0x0001104c cmp r7, 0xf ; 15

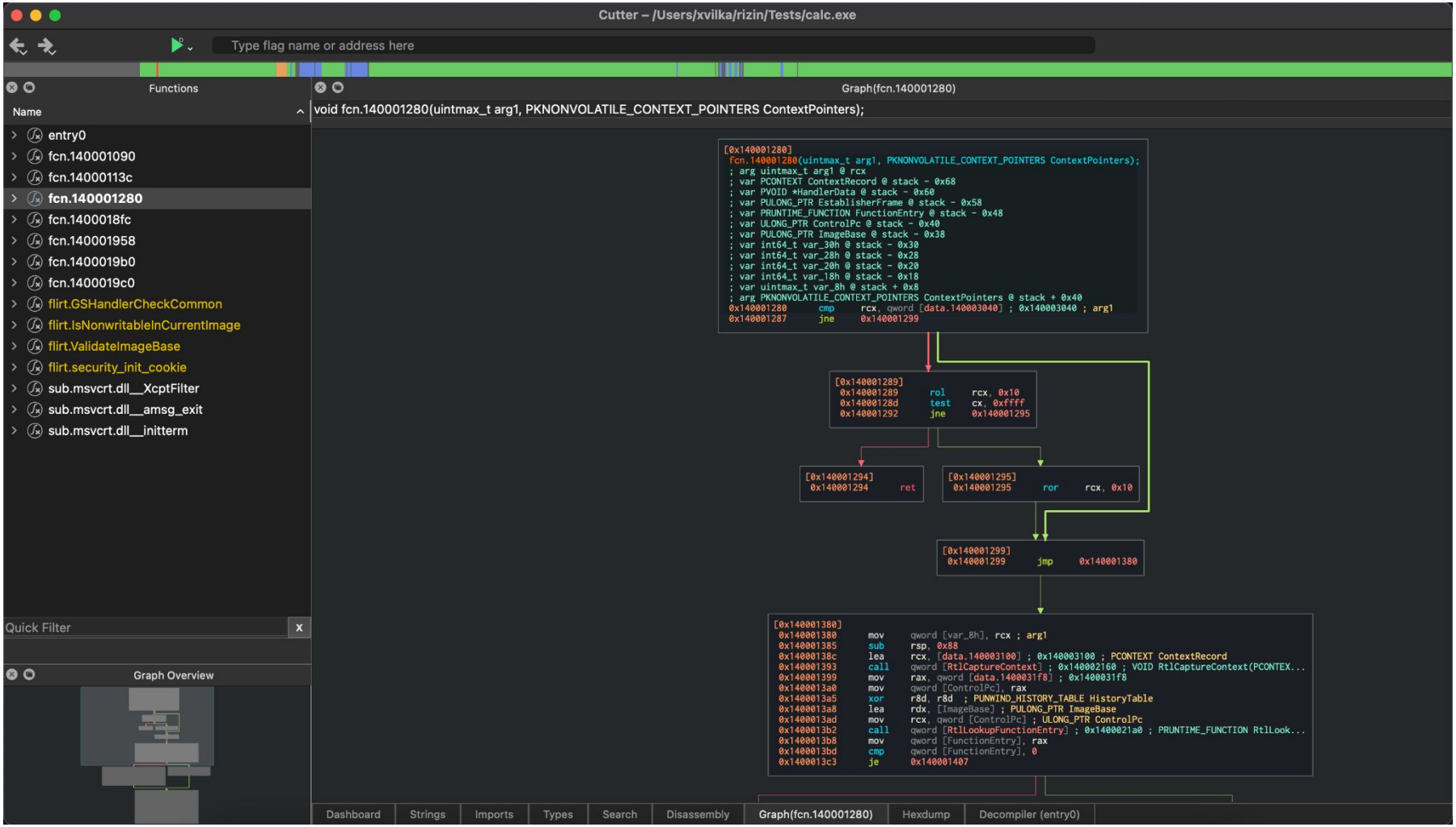
What is Cutter?

Cutter

Cutter is an advanced reverse-engineering platform powered by Rizin.

Provides an easy to use GUI with widgets and features to make your Reverse Engineering experience as comfortable as possible.

RizinOrg


Cutter – /Users/xvilka/rizin/Tests/calc.exe
Type flag name or address here
Functions
Name
entry0
fcn.140001090
fcn.14000113c
fcn.140001280
fcn.1400018fc
fcn.140001958
fcn.1400019b0
fcn.1400019c0
flirt.GSHandlerCheckCommon
flirt.IsNonwritableInCurrentImage
flirt.ValidateImageBase
flirt.security_init_cookie
sub.msvcrt.dll__XcptFilter
sub.msvcrt.dll__amsg_exit
sub.msvcrt.dll__initterm
Quick Filter
Graph Overview
Graph(fcn.140001280)
void fcn.140001280(uintmax_t arg1, PKNONVOLATILE_CONTEXT_POINTERS ContextPointers);
[0x140001280]
fcn.140001280(uintmax_t arg1, PKNONVOLATILE_CONTEXT_POINTERS ContextPointers);
; arg uintmax_t arg1 @ rcx
; var PCONTEXT ContextRecord @ stack - 0x68
; var PVOID *HandlerData @ stack - 0x60
; var PULONG_PTR EstablisherFrame @ stack - 0x58
; var PRUNTIME_FUNCTION FunctionEntry @ stack - 0x48
; var ULONG_PTR ControlPc @ stack - 0x40
; var PULONG_PTR ImageBase @ stack - 0x38
; var int64_t var_30h @ stack - 0x30
; var int64_t var_28h @ stack - 0x28
; var int64_t var_20h @ stack - 0x20
; var int64_t var_18h @ stack - 0x18
; var uintmax_t var_8h @ stack + 0x8
; arg PKNONVOLATILE_CONTEXT_POINTERS ContextPointers @ stack + 0x40
0x140001280 cmp rcx, qword [data.140003040] ; 0x140003040 ; arg1
0x140001287 jne 0x140001299
[0x140001289]
0x140001289 rol rcx, 0x10
0x14000128d test cx, 0xffff
0x140001292 jne 0x140001295
[0x140001294]
0x140001294 ret
[0x140001295]
0x140001295 ror rcx, 0x10
[0x140001299]
0x140001299 jmp 0x140001380
[0x140001380]
0x140001380 mov qword [var_8h], rcx ; arg1
0x140001385 sub rsp, 0x88
0x14000138c lea rcx, [data.140003100] ; 0x140003100 ; PCONTEXT ContextRecord
0x140001393 call qword [RtlCaptureContext] ; 0x140002160 ; VOID RtlCaptureContext(PCONTEX...
0x140001399 mov rax, qword [data.1400031f8] ; 0x1400031f8
0x1400013a0 mov qword [ControlPc], rax
0x1400013a5 xor r8d, r8d ; PUNWIND_HISTORY_TABLE HistoryTable
0x1400013a8 lea rdx, [ImageBase] ; PULONG_PTR ImageBase
0x1400013ad mov rcx, qword [ControlPc] ; ULONG_PTR ControlPc
0x1400013b2 call qword [RtlLookupFunctionEntry] ; 0x1400021a0 ; PRUNTIME_FUNCTION RtlLook...
0x1400013b8 mov qword [FunctionEntry], rax
0x1400013bd cmp qword [FunctionEntry], 0
0x1400013c3 je 0x140001407
Dashboard
Strings
Imports
Types
Search
Disassembly
Graph(fcn.140001280)
Hexdump
Decompiler (entry0)

Architectures

Real ISA: X86 (16,32,64), ARM (32,64), PowerPC, SystemZ (S390), RISC-V, MIPS, SPARC, M68k, M680x, HP PA-RISC, M-Core, PIC, Hexagon (QDSP), V810, V850/RH850, Tricore, RX, RL78, 8051, AVR, ARC, CR16, TI TMS320x, SuperH, H8300, VAX, ...

Bytecode: Java, Dalvik (Android), Python, Lua, EFI (EBC), WebAssembly (WASM), Malbolge, ...

Full list: rz-asm -L

Formats

ELF, PE, {Fat}MachO, COFF, DEX, ART, OMF, MZ, LE, NE, TE, DMP{64}/MDMP, Dyldcache (iOS/macOS), Kernelcache (iOS/macOS), QNX, zing, psxexe, NES, ninds, nin3ds, ningb, nro, nso, ...

Plus also automatic parsing of the bytecode file formats.

Full list: rz-bin -L

RizinOrg

First step: opening a file

Rizin is an terminal application, thus you will need to open a terminal to invoke rizin.

First step: opening a file

Rizin is an terminal application, thus you will need to open a terminal to invoke rizin.

To open a file in rizin you just need to type rizin, followed by the filename.

Example:

```
$ rizin /bin/ls
```

RizinOrg 

First step: opening a file

Rizin is an terminal application, thus you will need to open a terminal to invoke rizin.

To open a file in rizin you just need to type rizin, followed by the filename.

Example:

```
$ rizin /bin/ls
```

Once rizin opens, we can see that the console prompt has changed.

Try to type help and press enter, to see what happens.

```
[0x0000000]> help
```

RizinOrg

First inspection

The first operations we will learn about:

- Check binary details
- View strings
- View symbols
- View debug information
- Perform analysis
- Close rizin & visual modes

First inspection: Binary details

Cheat Sheet

To check the binary details we will use the command `i`.

Add a ? (question mark) after the command to see all the commands.

`[0x0000000]> i?`

```
[0x00000000]> i?
Usage: i[?]   # Get info about opened binary file
| i[jqt]                  # Show info of current file
| ia[jq]                  # Show a summary of all info
| iA[jqt]                 # List archs
| ic[?]                   # List classes, fields and methods
| iC[j]                   # Show signature info
| id[jqp]                 # Debug commands
[. . .]
```

RizinOrg

First inspection: Strings

Cheat Sheet
binary info - i
Help - <cmd>?

To view the strings we will use the command **iz**.

Add a **~..** (tilde dot dot) after the command to enter in less mode (**q** to exit).

[0x0000000]> iz~..

```
[0x00004f80]> iz
nth paddr      vaddr      len size section type  string
-------------------------------------------------------------------------------------------------------------------------
0   0x00016007 0x00016007 5   6    .rodata ascii =fff?
1   0x00016620 0x00016620 46  47   .rodata ascii Copyright %s %d Free Software Foundation, Inc.
2   0x00016651 0x00016651 10  11   .rodata ascii KMGTPEZYRQ
3   0x000166a8 0x000166a8 10  11   .rodata ascii ?pcdb-lswd
4   0x000166b8 0x000166b8 10  11   .rodata ascii sort_files
5   0x00016730 0x00016730 6   7    .rodata ascii posix-
6   0x00016840 0x00016840 65  66   .rodata ascii # Configuration file for dircolors, a utility to help you set the
7   0x00016882 0x00016882 72  73   .rodata ascii # LS_COLORS environment variable used by GNU ls with the --color option.
8   0x000168cb 0x000168cb 56  57   .rodata ascii # Copyright (C) 1996-2024 Free Software Foundation, Inc.
9   0x00016904 0x00016904 70  71   .rodata ascii # Copying and distribution of this file, with or without modification,
[. . .]
```

RizinOrg

First inspection: Strings

Cheat Sheet
binary info - i
Help - <cmd>?

There is one more special mode which is particularly useful for searching strings.

Add a `~...` (tilde dot dot dot) after the command to enter in HUD mode (`remove string and enter` to exit).

```
[0x0000000]> iz~...
```

```
0> while|
 - 16   0x000e8130 0x000e8130 5     6     .rodata ascii   while
   63   0x000e8d00 0x000e8d00 5     6     .rodata ascii   while
   1723 0x000f200a 0x000f200a 46    47    .rodata ascii   got bad TLS record (len:%d) while expecting %s
   2156 0x000f3ed1 0x000f3ed1 32    33    .rodata ascii   too many leases while loading %s
   2233 0x000f44c1 0x000f44c1 16    17    .rodata ascii   error while %s%s
   2607 0x000f6019 0x000f6019 6     7     .rodata ascii   while
   2719 0x000f6532 0x000f6532 5     6     .rodata ascii   while
   4153 0x000fdace 0x000fdace 38    39    .rodata ascii   File %s is a %s while file %s is a %s\n
```

RizinOrg

First inspection: Symbols

To view the symbols we will use the command is.

```
[0x0000000]> is
```

```
[0x00004f80]> is
nth paddr      vaddr      bind   type   size lib name
-------------------------------------------------------------------------------
114 0x0001f0a0 0x000200a0 GLOBAL OBJ    8        obstack_alloc_failed_handler
1   ---------- ---------- GLOBAL FUNC   0        imp.__ctype_toupper_loc
2   ---------- ---------- GLOBAL FUNC   0        imp.getenv
3   ---------- ---------- GLOBAL FUNC   0        imp.cap_to_text
4   ---------- ---------- GLOBAL OBJ    0        imp.__progname
5   ---------- ---------- GLOBAL FUNC   0        imp.sigprocmask
6   ---------- ---------- GLOBAL FUNC   0        imp.__snprintf_chk
7   ---------- ---------- GLOBAL FUNC   0        imp.raise
[. . .]
```

```
Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
```

RizinOrg

First inspection: Debug information

Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is

Sometimes binaries contains debug information (like DWARF structures etc..).

To view the debug information we will use the command **id**.

[0x0000000]> id

```
[0x00004f80]> id
.debug_abbrevs content:
Abbrev table for offset: 0x00000000
1 DW_TAG_formal_parameter       DW_CHILDREN_no (0x0)
        DW_AT_type      DW_FORM_ref4

2 DW_TAG_formal_parameter       DW_CHILDREN_no (0x7)
        DW_AT_type      DW_FORM_ref4
        DW_AT_artificial        DW_FORM_flag_present

3 DW_TAG_subprogram     DW_CHILDREN_yes (0x10)
        DW_AT_external  DW_FORM_flag_present
[. . .]
```

RizinOrg

First inspection: Analysis

Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is
View debug info - id

Sometimes some info are not available without analysis (this depends by the binary, architecture and language used).

To perform the binary analysis we will use the command aaa.

```
[0x0000000]> aaa
```

```
[0x00009690]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls
[x] Analyze len bytes of instructions for references
[x] Check for classes
[x] Analyze local variables and arguments
[x] Type matching analysis for all functions
[x] Applied 0 FLIRT signatures via sigdb
[x] Propagate noreturn information
[x] Integrate dwarf function information.
[x] Resolve pointers to data sections
[. . .]
```

RizinOrg

First inspection: Close rizin & visual modes

To close a rizin session, just type q (or quit or exit)

Rizin has many visual modes (for example less mode), and these can be closed by pressing q.

```
Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is
View debug info - id
Run analysis - aaa
```

RizinOrg

Workshop Task: First inspection

Open `task-first-inspection.bin` with rizin

1. What is the binary type?
2. What is the architecture (name & bits)?
3. What compiler has been used?
4. Is the binary stripped?
5. What is the name of the binary?
6. What language and standard is the source code of this compiled binary written in?

Cheat Sheet

```
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is
View debug info - id
Run analysis - aaa
Close rizin/views - q
```

RizinOrg

Workshop Task: First inspection (Solutions)

1. What is the binary type?

RizinOrg

Workshop Task: First inspection (Solutions)

1. Binary type? ELF 64
2. What is the architecture (name & bits)?

RizinOrg

Workshop Task: First inspection (Solutions)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. What compiler has been used?

RizinOrg

Workshop Task: First inspection (Solutions)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. Compiler? GCC: (GNU) 13.2.0
4. Is the binary stripped?

RizinOrg

Workshop Task: First inspection (Solutions)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. Compiler? GCC: (GNU) 13.2.0
4. Is stripped? No
5. What is the name of the binary?

RizinOrg

Workshop Task: First inspection (Solutions)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. Compiler? GCC: (GNU) 13.2.0
4. Is stripped? No
5. Binary name? ilovefruit
6. What language and standard is the source code of this compiled binary written in?

RizinOrg

Workshop Task: First inspection (Solutions)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. Compiler? GCC: (GNU) 13.2.0
4. Is stripped? No
5. Binary name? ilovefruit
6. Language & standard of src? C++11

RizinOrg

Workshop Task: First inspection (How)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. Compiler? GCC: (GNU) 13.2.0
4. Is stripped? No
5. Binary name? ilovefruit
6. Language & standard of src? C++11

`Command i`

RizinOrg

Workshop Task: First inspection (How)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. Compiler? GCC: (GNU) 13.2.0
4. Is stripped? No — Command `id`
5. Binary name? ilovefruit
6. Language & standard of src? C++11

RizinOrg

Workshop Task: First inspection (How)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. Compiler? GCC: (GNU) 13.2.0
4. Is stripped? No
5. Binary name? ilovefruit `Command iz`
6. Language & standard of src? C++11

RizinOrg

Workshop Task: First inspection (How)

1. Binary type? ELF 64
2. Architecture (name & bits)? RISC-V 64 bits
3. Compiler? GCC: (GNU) 13.2.0
4. Is stripped? No
5. Binary name? ilovefruit
6. Language & standard of src? C++11

`Command i & is`

RizinOrg

Dependencies

The next operations we will learn about:

- View syscalls called
- View shared & linked libraries
- Recover golang data (libraries, strings, functions etc..)
- FLIRT signatures

RizinOrg

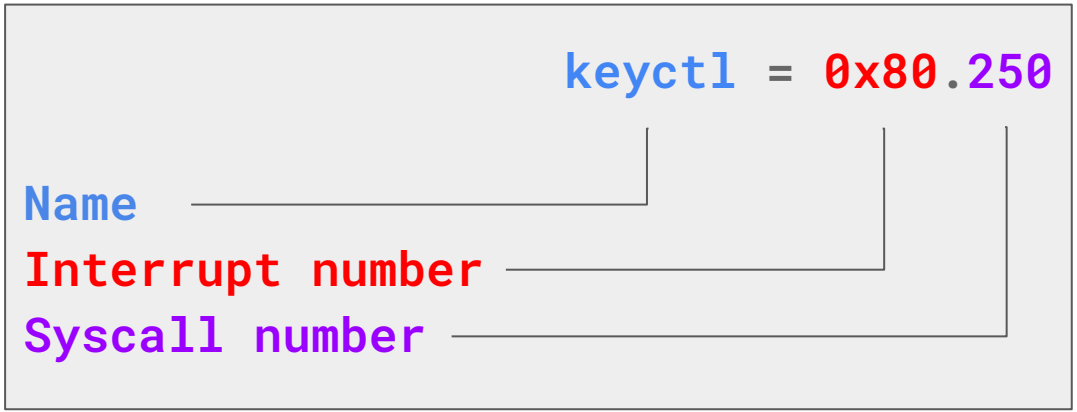
Dependencies: Syscalls

It is possible on rizin to view the used syscalls of the binary (this is only possible after running the analysis).

To view the syscalls we will use the command asl.

```
[0x0000000]> asl
```

```
[0x000109d0]> asl
keyctl = 0x80.250
setitimer = 0x80.38
fchownat = 0x80.260
fchmod = 0x80.91
io_getevents = 0x80.208
msgctl = 0x80.71
process_mrelease = 0x80.448
getxattr = 0x80.191
link = 0x80.86
set_mempolicy = 0x80.238
[. . .]
```




```
Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is
View debug info - id
Run analysis - aaa
Close rizin/views - q
```

RizinOrg

Dependencies: Shared & linked libraries

It is possible on rizin to view the shared & linked library dependencies.

To view the binary the libraries we will use the command **il**. You can also search for a specific string by using **~text** (tilde)

[0x0000000]> il

```
[0x000109d0]> il
library
------------------------
libkcupslib.so.6.0.2
libKF6IconWidgets.so.6
libKF6KIOCore.so.6
libKF6CoreAddons.so.6
libKF6WidgetsAddons.so.6
libQt6Widgets.so.6
libKF6I18n.so.6
[. . .]
```

Not all binaries supports libraries, and most of the stripped libraries don't provide info about the statically linked libraries, but there are exceptions, like with golang.

```
Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is
View debug info - id
Run analysis - aaa
Close rizin/views - q
View Syscalls - asl
```

RizinOrg

Dependencies: Recover golang data

Golang binaries are special, the provide a lot of info also when stripped.

To recover the golang data we can just analyze the binary using the command **aaa** or manually via command **aalg**.

[0x0000000]> aalg

```
[0x0046bf80]> aalg
[x] Found go 1.20+ pclntab data.
[x] Recovered 11092 symbols and saved them at sym.go.*
[x] Recovered 169 go packages
[x] Analyze all flags starting with sym.go. (aF @@f:sym.go.*)
[x] Recovering go strings from bin maps
[x] Analyze all instructions to recover all strings used in sym.go.*
[x] Recovered 8347 strings from the sym.go.* functions.
[. . .]
```

```
Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is
View debug info - id
Run analysis - aaa
Close rizin/views - q
View Syscalls - asl
View libraries - il
Grep mode - ~text
```

RizinOrg

Dependencies: FLIRT signatures

Rizin implements FLIRT signature format; it can be used to detect software in stripped binaries. To create a signature, command **Fc** and to apply a signature use **Fs**.

[0x0000000]> F?

```
[0x00009690]> F?
Usage: F<cdsfal>   # FLIRT signature management
| Fc <filename> # Create a FLIRT file (.pat or .sig)
| Fd <filename> # Open a FLIRT file (.pat or .sig) and dumps its contents
| Fs <filename> # Open a FLIRT file (.pat or .sig) and tries to apply the
signatures
| Ff            # Outputs the flirt function signature info
| Fa [<filter>] # Apply signatures from sigdb
| Fl[t]         # Lists all available signatures in sigdb
[. . .]
```

```
Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is
View debug info - id
Run analysis - aaa
Close rizin/views - q
View Syscalls - asl
View libraries - il
Grep mode - ~text
Golang Analysis - aalg
```

RizinOrg

Workshop Task: Dependencies

1. Open task-dependencies-0.bin
2. Check the libraries and then run the go analysis, and then check the libraries again.
3. How many libraries (go packages) did it recover?
4. Open task-dependencies-1.bin and run the analysis.
5. Find if the execve syscall is called.
6. Create a FLIRT sig file from task-dependencies-1.bin
7. Close the session and open task-dependencies-2.bin
8. Analyze the binary and apply the signature just created.
9. How many signatures did it found?

```
Cheat Sheet
binary info - i
Help - <cmd>?
View strings - iz
Less mode - ~..
HUD mode - ~...
View symbols - is
View debug info - id
Run analysis - aaa
Close rizin/views - q
View Syscalls - asl
View libraries - il
Grep mode - ~text
Golang Analysis - aalg
FLIRT sig file - F
```

RizinOrg

Workshop Task: Dependencies (Solutions)

1. Open task-dependencies-0.bin
2. Check the libraries and then run the go analysis, and then check the libraries again.
3. How many libraries (go packages) did it recover?

RizinOrg 

Workshop Task: Dependencies (Solutions)

1. Open task-dependencies-0.bin
2. Check the libraries and then run the go analysis, and then check the libraries again.
3. How many libraries (go packages) did it recover? 139
4. Open task-dependencies-1.bin and run the analysis.
5. Is execve syscall called?

RizinOrg

Workshop Task: Dependencies (Solutions)

1. Open task-dependencies-0.bin
2. Check the libraries and then run the go analysis, and then check the libraries again.
3. How many libraries (go packages) did it recover? 139
4. Open task-dependencies-1.bin and run the analysis.
5. Is execve syscall called? Yes
6. Create a FLIRT sig file from task-dependencies-1.bin
7. Close the session and open task-dependencies-2.bin
8. Analyze the binary and apply the signature just created.
9. How many signatures did it found?

RizinOrg

Workshop Task: Dependencies (Solutions)

1. Open task-dependencies-0.bin
2. Check the libraries and then run the go analysis, and then check the libraries again.
3. How many libraries (go packages) did it recover? 139
4. Open task-dependencies-1.bin and run the analysis.
5. Is execve syscall called? Yes
6. Create a FLIRT sig file from task-dependencies-1.bin
7. Close the session and open task-dependencies-2.bin
8. Analyze the binary and apply the signature just created.
9. How many signatures did it found? 536

RizinOrg

Workshop Task: Dependencies (How)

1. Open task-dependencies-0.bin
2. Check the libraries and then run the go analysis, and then check the libraries again.
3. How many libraries (go packages) did it recover? 139

`aalg`

4. Open task-dependencies-1.bin and run the analysis.
5. Is execve syscall called? Yes
6. Create a FLIRT sig file from task-dependencies-1.bin
7. Close the session and open task-dependencies-2.bin
8. Analyze the binary and apply the signature just created.
9. How many signatures did it found? 536

RizinOrg

Workshop Task: Dependencies (How)

1. Open task-dependencies-0.bin
2. Check the libraries and then run the go analysis, and then check the libraries again.
3. How many libraries (go packages) did it recover? 139
4. Open task-dependencies-1.bin and run the analysis.
5. Is execve syscall called? Yes

asl~execve

6. Create a FLIRT sig file from task-dependencies-1.bin
7. Close the session and open task-dependencies-2.bin
8. Analyze the binary and apply the signature just created.
9. How many signatures did it found? 536

RizinOrg

Workshop Task: Dependencies (How)

1. Open task-dependencies-0.bin
2. Check the libraries and then run the go analysis, and then check the libraries again.
3. How many libraries (go packages) did it recover? 139
4. Open task-dependencies-1.bin and run the analysis.
5. Is execve syscall called? Yes
6. Create a FLIRT sig file from task-dependencies-1.bin
7. Close the session and open task-dependencies-2.bin
8. Analyze the binary and apply the signature just created.
9. How many signatures did it found? 536

```
Fc file.sig
Fs file.sig
```

RizinOrg

Application audit

The last operations we will learn about:

- Bin information
- Diffing binaries
- Checking for license violation
- Creating FLIRT signatures

RizinOrg

Application audit: Bin information

There are two ways to do that - inside Rizin shell and using the rz-bin tool. You can use the **-I (dash i) option** to get the same binary information as the **i command in rizin**:

rz-bin -I <file>

```
$ rz-bin -I file.bin
[Info]
…
va       true
sanitiz  false
static   false
linenum  true
lsyms    true
canary   true
PIE      true
RELROCS  true
NX       true
```

RizinOrg

Application audit: Diffing binaries

The framework has a specific tool rz-diff to run the analysis of two files simultaneously and then perform the diff. It is very useful for comparing binaries of two different versions. To compare functions use "`-t functions`" (dash T) followed by the files

`$ rz-diff -t functions <file1> <file2>`

```
$ rz-diff -h
bytes      | compare raw bytes in the files (only for small files)
lines      | compare text files
functions  | compare functions found in the files
classes    | compare classes found in the files
command    | compare command output returned when executed in both files
entries    | compare entries found in the files
fields     | compare fields found in the files
graphs     | compare 2 functions and outputs in graphviz/dot format
imports    | compare imports found in the files
libraries  | compare libraries found in the files
sections   | compare sections found in the files
strings    | compare strings found in the files
symbols    | compare symbols found in the files
```

RizinOrg

Application audit: Checking for license violation

You can use rz-find to search across multiple binaries for content within them. The simplest first step is to search for strings. The Rizin framework provides a binary grep alternative - rz-find.

```
rz-find -X -s <string> <file/dir>
```

It can also search for function names or symbols:

```
rz-find -X -S <symbol> <file/dir>
```

RizinOrg

Application audit: Creating FLIRT signatures

The framework has a specific tool rz-sign which allows to directly creates, convert or parse FLIRT signatures. It is very useful for generating signatures from multiple files. To generate signature functions use **-o** (dash O) followed by the output file and input file. To dump the signature use **-d** (dash D) followed by the signature file.

$ rz-sign -o output.sig <file>

```
$ rz-sign -h
Usage: rz-sign [options] [file]
 -h                          Show this help
 -a [-a]                     Add extra 'a' to analysis command (available only with -o option)
 -e [k=v]                    Set an evaluable config variable (available only with -o option)
 -c [output.pat] [input.sig] Parse a FLIRT signature and convert it to its other format
 -o [output.sig] [input.bin] Perform an analysis on the binary and generate the FLIRT signature
 -d [flirt.sig]              Parse a FLIRT signature and dump its content
 -q                          Quiet mode
 -v                          Show version information
Examples:
  rz-sign -d signature.sig
  rz-sign -c new_signature.pat old_signature.sig
  rz-sign -o libc.sig libc.so.6
```

RizinOrg

Workshop Task: Application Audit

1. Is PIE support enable in info.bin?
2. Are there canaries in info.bin?
3. Try to search for GPL binaries, how many did you find?
4. Try to match the functions of compare_0 to compare_1.
5. Try to generate a FLIRT signature from compare_0 using rz-sign and dump its content.

Cheat Sheet

```
rz-bin -I file
rz-diff -t functions f0 f1
rz-find -X -s string path
rz-sign -o file.sig file
rz-sign -d file.sig
```

RizinOrg

Workshop Task: Application Audit (Solutions)

1. Is PIE support enable in info.bin?

RizinOrg

Workshop Task: Application Audit (Solutions)

1. Is PIE support enable in info.bin? No
2. Are there canaries in info.bin?

RizinOrg

Workshop Task: Application Audit (Solutions)

1. Is PIE support enable in info.bin? No
2. Are there canaries in info.bin? Yes
3. Try to search for GPL binaries, how many did you find?

RizinOrg 

Workshop Task: Application Audit (Solutions)

1. Is PIE support enable in info.bin? No
2. Are there canaries in info.bin? Yes
3. How many GPL binaries? 3 (binary_1, binary_2, binary_3)

RizinOrg

Workshop Task: Application Audit (How)

1. Is PIE support enable in info.bin? No
2. Are there canaries in info.bin? Yes

```
rz-bin -I info.bin
```

3. How many GPL binaries? 3
4. Try to match the functions of compare_0 to compare_1.
5. Try to generate a FLIRT signature from compare_0 using rz-sign and dump its content.

RizinOrg

Workshop Task: Application Audit (How)

1. Is PIE support enable in info.bin? No
2. Are there canaries in info.bin? Yes
3. How many GPL binaries? 3 `rz-find -X -s GPL .`
4. Try to match the functions of compare_0 to compare_1.
5. Try to generate a FLIRT signature from compare_0 using rz-sign and dump its content.

RizinOrg

Workshop Task: Application Audit (How)

1. Is PIE support enable in info.bin? No
2. Are there canaries in info.bin? Yes
3. How many GPL binaries? 3
4. Try to match the functions of compare_0 to compare_1.

```
rz-diff -t functions compare_0 compare_1
```

5. Try to generate a FLIRT signature from compare_0 using rz-sign and and dump its content.

RizinOrg

Workshop Task: Application Audit

1. Is PIE support enable in info.bin? No
2. Are there canaries in info.bin? Yes
3. How many GPL binaries? 3
4. Try to match the functions of compare_0 to compare_1.
5. Try to generate a FLIRT signature from compare_0 using rz-sign and and dump its content.

```
rz-sign -o file.sig compare_0
rz-sign -o file.sig
```

RizinOrg

Learn more about Rizin & Cutter at:

1. Book: https://book.rizin.re
2. Cutter documentation: https://cutter.re/docs
3. Rizin Blog: https://rizin.re/posts
4. Rizin community server (Mattermost): https://im.rizin.re/
5. Rizin GitHub organization: https://github.com/rizinorg

RizinOrg

Thank you for attending

@akochkov

@akochkov@mastodon.world

@der0ad

@deroad@social.treehouse.systems

@rizin@fosstodon.org

rizinorg

cutter_re

fossasia

RizinOrg