

# RELYZE

QUICK START GUIDE

28 November 2022

## CONTENTS

Relyze .....	1
GUI Overview .....	3
Analyzing a File .....	4
Analysis Overview .....	6
Structure View .....	7
Code Flat View .....	8
Code Flow View .....	9
Code Pseudo View .....	10
References .....	11
Searching .....	12
Edit an Instruction .....	13
Edit a Jump Table .....	14
Call Graph View .....	15
Split View .....	16
Binary Diffing .....	17
Graph References .....	19
Custom Data Types .....	20
Analysis Options .....	21
Saving Analysis Archives .....	22
Batch Analysis .....	23
Plugin Editor .....	24
Keyboard Shortcuts .....	26
Command Line Usage .....	27
Support .....	28

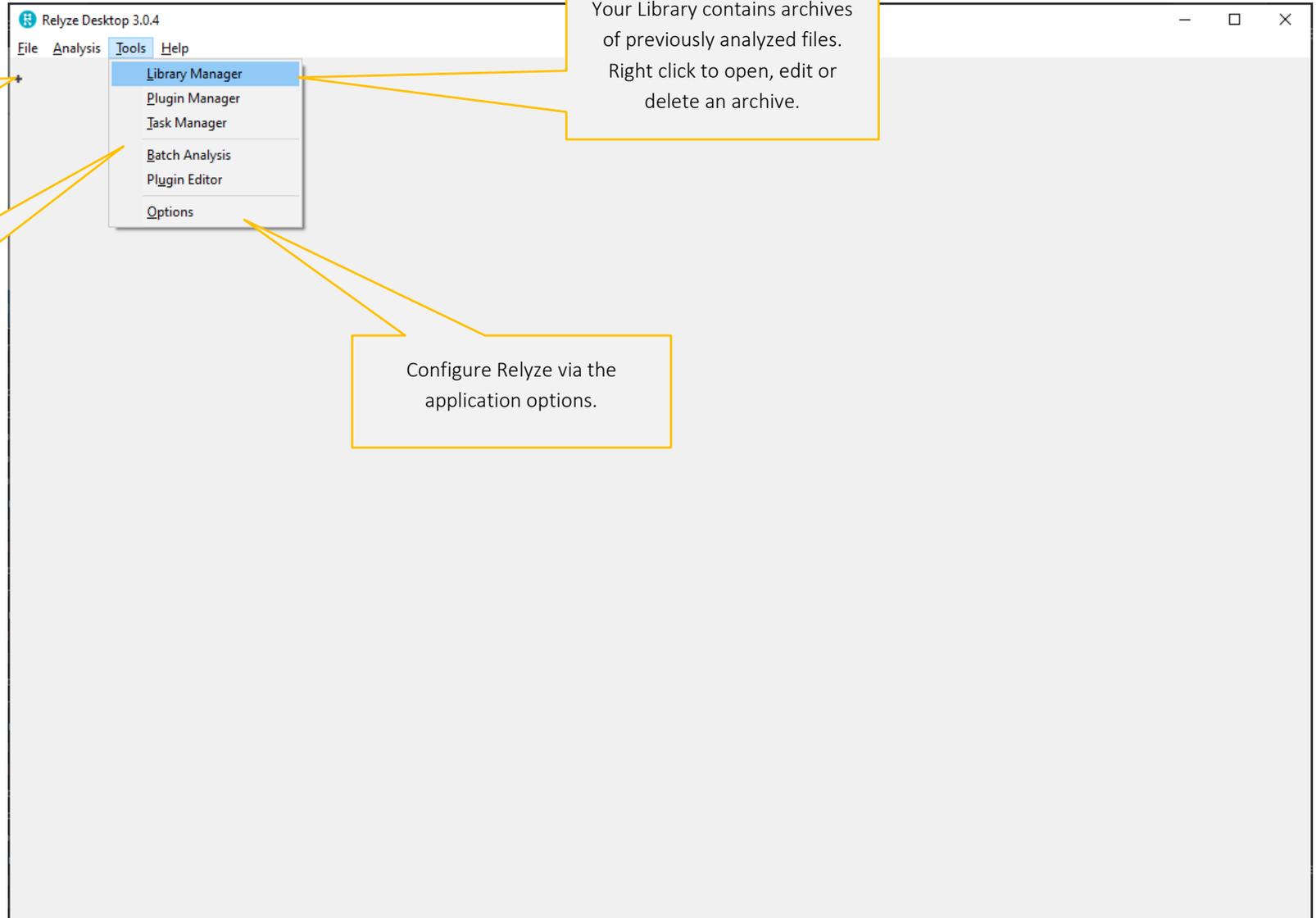
## GUI OVERVIEW

Analyze a file either by clicking the + button, dragging a file onto the application or select Open from the File menu.

Your Library contains archives of previously analyzed files. Right click to open, edit or delete an archive.

You can access the applications task manager, batch analysis and the plugin editor here. Press F1 at any time to display the Plugin Editor.

Configure Relyze via the application options.



Either drop a file onto the application or click the + button in the main window to bring up the Open dialog

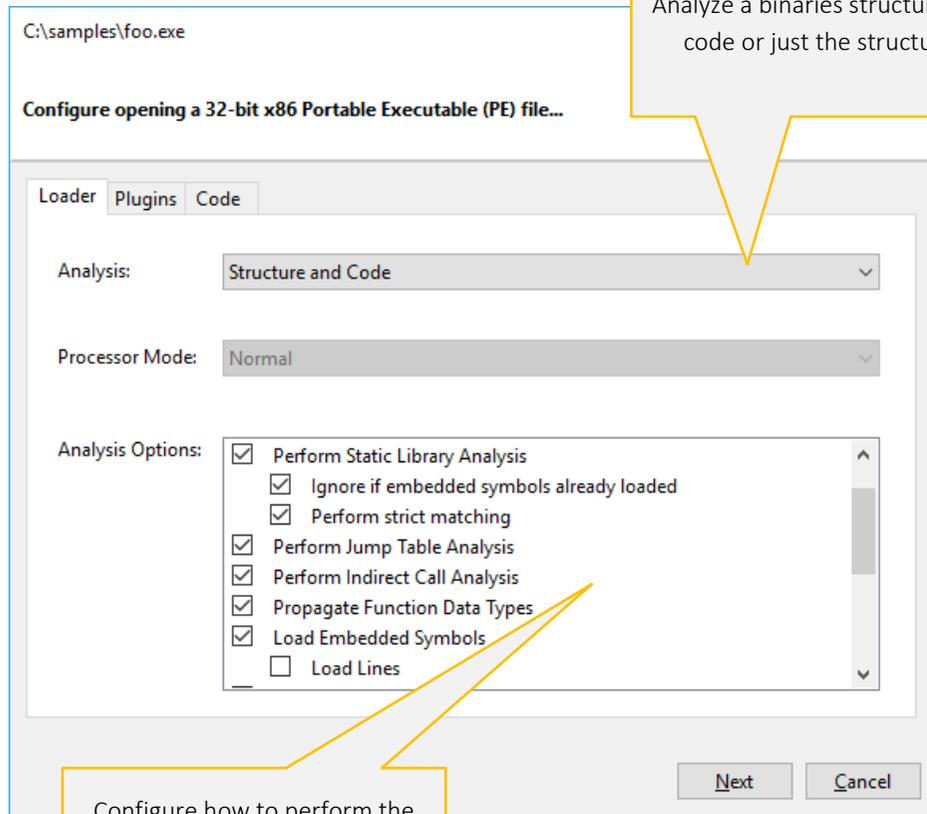
The screenshot shows an 'Open' dialog box with the following elements:

- Title: Open
- Instruction: Select either a local or remote file to open...
- Location: A text field containing 'C:\samples\foo1' with a dropdown arrow and a browse button ('...').
- Decode: A dropdown menu currently set to 'None'.
- Buttons: 'Next' and 'Cancel' buttons at the bottom right.

Callouts from the surrounding text boxes point to the 'Location' field, the 'Decode' dropdown, and the '...' browse button.

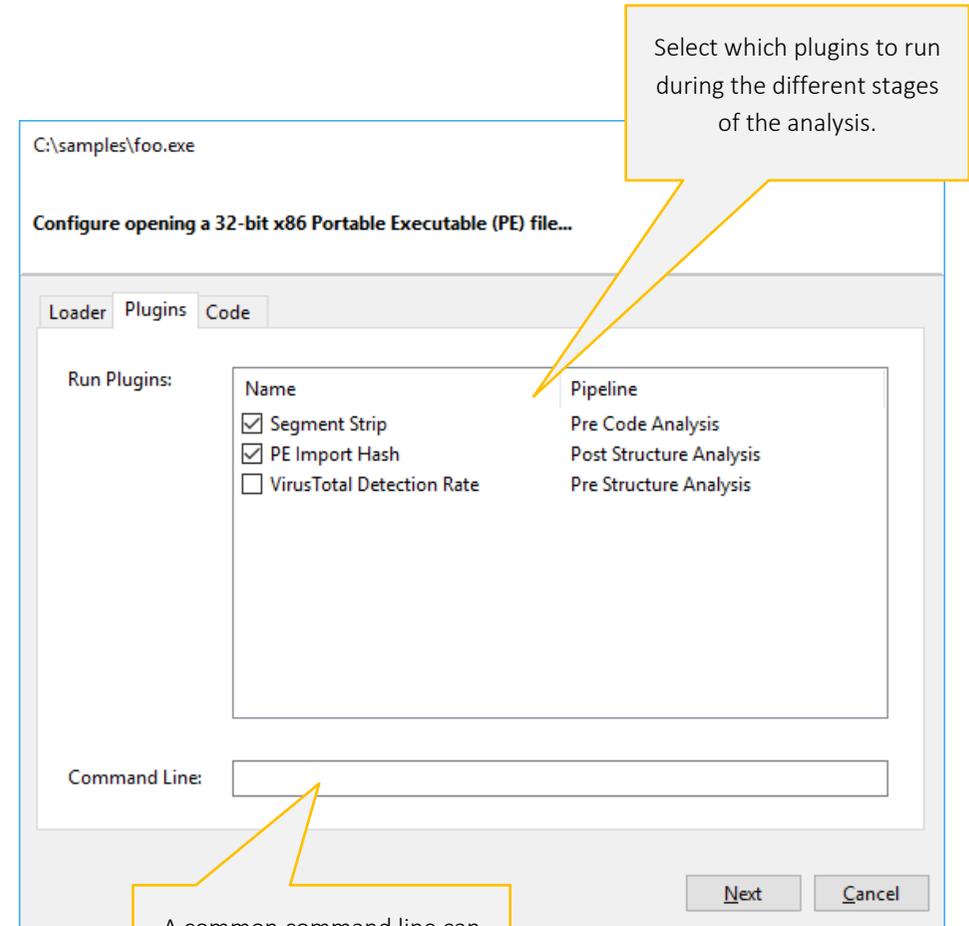
Browse for a local file or enter a remote file location such as a UNC path, HTTP or FTP location.

If needed, a decoder plugin may be used to decode an input source into an expected format



Analyze a binaries structure and code or just the structure.

Configure how to perform the analysis.



Select which plugins to run during the different stages of the analysis.

A common command line can be passed to plugins if they require extra configuration.

# ANALYSIS OVERVIEW

Switch between the analysis overview, structure, code and diff views.

Access the Data Type Manager and Plugin Editor here.

Access the analysis options.

Save an analysis via the File menu. Access more options via the Analysis menu.

Double click items with an arrow to jump to a relevant location in either the structure or code view.

Right click the entropy graph to jump into the code or structure.

Group	Title	Information
File	Name	psping.exe
File	Type	32-bit x86 Portable Executable (PE)
File	Size	249.66 KB (255,648 bytes)
File	Creation Time	Wednesday 29 June 2016, 04:58
File	Accessed Time	Tuesday 25 February 2020, 11:32
File	Modified Time	Tuesday 25 February 2020, 11:32
Hash	File MD5	829BF469365FE504C673D887BE7D3436
Hash	File SHA1	C4CCD065F6D551A01DE86599E36BF90390E101CD
Hash	File SHA256	C8453110682D999223A84146462B0B4FC6979F40A01B660A7B925783B71B2D6FF
Format	PE Internal Name	PsPing
Format	PE Product Name	Sysinternals PsPing
Format	PE Company Name	Sysinternals - www.sysinternals.com
Format	PE Legal Copyright	Copyright (C) 2012-2016 Mark Russinovich
Format	PE Product Version	2.10
Format	PE File Description	PsPing - ping, latency, bandwidth
Format	PE File Version	2.10
Format	PE Original Filename	psping.exe
Security	Signature	PKCS#7 signature from a certificate for 'Microsoft Corporation' issued by 'Microsoft Code Signing PC'
Security	Mitigations	ASLR, DEP, SafeSEH, Stack Cookies
Analysis	Oldest Time Stamp	Tuesday 28 June 2016, 18:57
Analysis	Newest Time Stamp	Tuesday 28 June 2016, 18:57

ENTROPY SIGNATURE STOPWATCH

Highlighting '.text section', Offset 0x00000400 (1024), Length 141431 bytes. Current Offset 0x0000CD94 (52628) 0.671822.

# STRUCTURE VIEW

Explore the files structure.

The structures navigation bar; blue depicts areas where the structure metadata exists, grey is non structure metadata.

Name	Value	Comment
MinorImageVersion	0x0000	
MajorSubsystemVersion	0x0006	
MinorSubsystemVersion	0x0000	
Win32VersionValue	0x00000000	
SizeOfImage	0x00041000	260.00 KB (266,240 bytes)
SizeOfHeaders	0x00000400	1.00 KB (1,024 bytes)
Checksum	0x0004106E	The checksum is correct
Subsystem	0x0003	IMAGE_SUBSYSTEM_WINDOWS_CUI
DllCharacteristics	0x8140	3 flags set IMAGE_DLLCHARACTERISTICS_DYNAMIC_ IMAGE_DLLCHARACTERISTICS_NX_COMPA IMAGE_DLLCHARACTERISTICS_TERMINAL
SizeOfStackReserve	0x00100000	1.00 MB (1,048,576 bytes)
SizeOfStackCommit	0x00001000	4.00 KB (4,096 bytes)
SizeOfHeapReserve	0x00100000	1.00 MB (1,048,576 bytes)
SizeOfHeapCommit	0x00001000	4.00 KB (4,096 bytes)

Hex View

```
0x00000150 00 10 04 00 00 04 00 00 6E 10 04 00 03 00 40 81 .....n...@.  
0x00000160 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00 .....  
0x00000170 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 .....  
0x00000180 28 F7 02 00 C8 00 00 00 00 D0 03 00 88 05 00 00 .....8...@...  
0x00000190 00 00 00 00 00 00 00 00 00 A8 03 00 A0 3E 00 00 .....  
0x000001A0 00 E0 03 00 04 20 00 00 E0 42 02 00 38 00 00 00 .....B..B...  
0x000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0x000001C0 00 00 00 00 00 00 00 00 38 E5 02 00 40 00 00 00 .....8...@...  
0x000001D0 00 00 00 00 00 00 00 00 40 02 00 44 02 00 00 00 .....@..D...  
0x000001E0 64 F5 02 00 60 00 00 00 00 00 00 00 00 00 00 00 d...`...text..  
0x000001F0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00 .....text..  
0x00000200 77 28 02 00 10 00 00 00 2A 02 00 00 04 00 00 00 w(...*...  
0x00000210 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 .....`...  
0x00000220 2E 72 64 61 74 61 00 00 60 C3 00 00 00 40 02 00 ..rdata...@..  
0x00000230 00 C4 00 00 00 02 E0 02 00 00 00 00 00 00 00 00 00 .....@..@.data...  
0x00000240 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00
```

Selection: 0x00000158 (344) - 0x0000015C (348) - 0x4 (4) bytes

Right click to decode or disassemble selected bytes. You can also perform searches and many more operations.

# CODE FLAT VIEW

Navigate back to a previous location (Or press 'ESC' key)

The screenshot shows a debugger window with the following components:

- Navigation Bar:** A horizontal bar at the top with colored segments representing different memory regions.
- Code View:** A window showing assembly code with addresses, hex values, and mnemonics. The current location is highlighted with an orange band.
- Disassembly View:** A window showing a linear disassembly of the binary.
- Bookmark List:** A list of bookmarks with columns for RVA, Type, Length, and Value.
- Filtering:** A search box at the bottom of the bookmark list to filter items.

An orange band displays your current location in the navigation bar. Red is code, Purple is static library code. Blue is data, Green is string data, Grey is unmapped memory.

Display a linear disassembly of the binary with the Flat view.

Add a new bookmark with the 'B' key. Use bookmarks to identify and quickly navigate between interesting locations.

The analysis will add automatic comments. Press the ';' key to either add or edit a comment.

Filter items by text to find useful information.

# CODE FLOW VIEW

Flow displays the logical view of a function including local variables while Flat displays a linear disassembly with no function analysis. Call displays the analysis call graph.

Hover over immediate hex values, references or graph edges to preview more information.

Right click on a reference to interact with it. Hold the Tab key to flash the analysis and see the raw reference.

```

int __cdecl __cinit( int p1 )
{
    unsigned int local_0x8;

    push ebp
    mov ebp, esp
    cmp dword ptr [data_0x259E0], 0x0
    jz code_0xFACA
}

code_0xFAB1:
    push va_ptr data_0x259E0
    __IsNonwritableInCurrentImage
    pop ecx
    test eax, eax
    jz code_0xFACA
    
```

Everything	Segments	Strings	Imports	Functions	Signatures
RVA ▲		Type		Length	Value
0x0000FAA5		Function		132	__cinit
0x0000FB75		Function		54	__initterm
0x0000FBAB		Function		34	__initterm_e
0x0000FDCF		Function		115	__initstdio
0x000107C6		Function		88	__initMTAoncurrentthread
0x0001089B		Function		80	__uninitMTAoncurrentthread
0x00010C44		Function		47	__onexitinit
0x000113A4		Function		17	__init_ah_hooks
init					

Pseudo displays the decompiled pseudocode of the current function.

Right click on a variable to interact with it. You can rename, retype a variable and display its cross references

Hold the Tab key to flash the displaying of casts in the pseudocode.

```

0x0000FAA5:00001 int __cdecl __cinit( int p1 )
0x0000FAA5:00002 {
0x0000FAA5:00003     int v1; // eax
0x0000FAA5:00004     int v2; // eax
0x0000FAA5:00005     int v3; // eax
0x0000FAA5:00006
0x0000FAA5:00007     v1 = __IsNonwritableInCurrentImage( &data_0x259E0 );
0x0000FAA5:00008     if( v1 != 0 ) {
0x0000FAA5:00009         __fpmath( p1 );
0x0000FAA5:00010     }
0x0000FAA5:00011     __initp_misc_cfltcvt_tab();
0x0000FAA5:00012     v2 = __initterm_e( &data_0x2429C, &data_0x242B4 );
0x0000FAA5:00013     if( v2 == 0 ) {
0x0000FAA5:00014         _atexit( &func_0x14D2C );
0x0000FAA5:00015         __initterm( &data_0x24244, &data_0x24298 );
0x0000FAA5:00016         if( data_0x3C3C8 != 0 ) {
0x0000FAA5:00017             v3 = __IsNonwritableInCurrentImage( &data_0x3C3C8 );
0x0000FAA5:00018             if( v3 != 0 ) {
0x0000FAA5:00019                 data_0x3C3C8( 0, 2, 0 );
0x0000FAA5:00020             }
    
```

Everything	Segments	Strings	Imports	Functions	Signatures	Value
0x0000FAA5		Function	132			__cinit
0x0000FB75		Function	54			__initterm
0x0000FBAB		Function	34			__initterm_e
0x0000FDCF		Function	115			__initstdio
0x000107C6		Function	88			__initMTAoncurrentthread
0x0001089B		Function	80			__uninitMTAoncurrentthread
0x00010C44		Function	47			__onexitinit
0x000113A1		Function	17			__init_sh_hooks

Select a label and press 'X' to bring up the references dialog.

Double click on a reference to go to that location. You can navigate backwards via the 'ESC' key.

Found 47 references to \_\_getptd

RVA ▼	Name	Content
0x0001C2A8	___InternalCxxFrameHandler+0x6	call __getptd; unsigned long __cdecl __getptd
0x0001C296	___FrameUnwindToState+0xCD	call __getptd; unsigned long __cdecl __getptd
0x0001C1F0	___FrameUnwindToState+0x27	call __getptd; unsigned long __cdecl __getptd
0x0001B542	___CreateFrameInfo+0x1A	call __getptd; unsigned long __cdecl __getptd
0x0001B534	___CreateFrameInfo+0xC	call __getptd; unsigned long __cdecl __getptd

frame X 5 items

Filter the references to your target label.

The image shows a 'Search...' dialog box with the following fields and options:

- Find:** A text input field containing the regular expression `(pop|push) e[sd]*`.
- Mode:** A dropdown menu set to 'Regular Expression Mode'.
- Options:** A group of checkboxes including:
  - Case Sensitive
  - Search Code
  - Search Function Locals
  - Search Data
- Buttons:** 'Find Next', 'Find All', and 'Cancel'.

Three callout boxes provide additional information:

- Top-left: 'Use the right click menu in the disassembly or press the 'S' key to bring up the search dialog.'
- Top-right: 'Enter the expression you wish to search for.'
- Right: 'You can perform a regular text search, a regex search or a binary search.'

## EDIT AN INSTRUCTION

To edit an existing instruction, select the line containing the instruction in either the flat or flow view. You can then either use the right click menu and select Block -> Edit Instruction, or you can use the default keyboard shortcut 'e'. The following dialog is then displayed allowing you to edit the selected instruction.

The screenshot shows the 'Edit Instruction...' dialog box with the following fields and options:

- Assembly:** A text input field containing the assembly code `xor eax, 0x41424344`.
- Encoding:** A text input field containing the hexadecimal encoding `35 44 43 42 41`.
- Options:** A section containing a checked checkbox labeled 'Insert padding if required'.
- Buttons:** 'Ok' and 'Cancel' buttons at the bottom right.

Callouts provide additional information:

- A callout pointing to the Assembly field: "Edit the assembly of an existing instruction and click Ok to update the analysis."
- A callout pointing to the Encoding field: "The new instruction encoding is updated as you edit the assembly."
- A callout pointing to the 'Insert padding if required' checkbox: "If the edited instruction overwrites an existing instruction boundary, padding bytes can automatically be inserted."
- A callout pointing to the bottom of the dialog: "Any errors in assembling will be displayed here."

If a jump table was analysed incorrectly you can manually edit it. Select the jump instruction and right click to edit it (Or press the 'J' key).

Select the type of jump table you are editing.

**Edit Jump Table...**

Table Type:

Table RVA:

Table Count:

Entry Width:

Entry Base RVA:

Configure the location of the table and the number of entries held there.

You can optionally specify a custom base address to add to a table entry.

# CALL GRAPH VIEW

If you enable the split view, double clicking a node will go to that location in the other view.

Visualise the binaries call graph using either circular, force directed or hierarchical layouts.

Search for nodes in the graph containing some interesting text.

Zoom in and out using Ctrl + mouse wheel. Right click to export the graph to either SVG, DOT or PNG formats.

Everything	Segments	Strings	Imports	Functions	Signatures
RVA ▲		Type		Length	Value
0x0000FAA5		Function		132	__cinit
0x0000FB75		Function		54	__initterm
0x0000FBAB		Function		34	__initterm_e
0x0000FDCF		Function		115	__initstdio
0x000107C6		Function		88	__initMTAoncurrentthread
0x0001089B		Function		80	__uninitMTAoncurrentthread
0x00010C44		Function		47	__onexitinit
0x000113AA		Function		17	__init_ah_hooks

# SPLIT VIEW

Activate the split view.

The navigation bar will display two halved markers for each location in the split view.

```
code_0xFAB1:  
  push va_ptr data_0x259E0  
  call __IsNonwritableInCurrentImage  
  pop ecx  
  test eax, eax  
  jz code_0xFACA
```

```
code_0xFAC0:  
  push dword ptr [p1]  
  call dword ptr [data_0x259E0]
```

```
code_0xFAC4:  
  call __initp_misc_cfltcvt_  
  push va_ptr data_0x242B4
```

Force the two views to be linked, so navigating to a location in one view will go to that location in the other view.

Everything	Segments	Strings	Imports	Functions	Signatures
RVA ▲	Type	Length	Value		
0x0000FAA5	Function	132	__cinit	int __cdecl __cinit( int p1 )	
0x0000FB75	Function	54	__initterm	void __cdecl __initterm( int p1, ir	
0x0000FBAB	Function	34	__initterm_e	int __cdecl __initterm_e( int p1, ·	
0x0000FDCF	Function	115	__initstdio	int __cdecl __initstdio( void )	
0x000107C6	Function	88	__initMTAoncurrentthread	int __cdecl __initMTAoncurrentthrea	
0x0001089B	Function	80	__uninitMTAoncurrentthread	void __cdecl __uninitMTAoncurrenttl	
0x00010C44	Function	47	__onexitinit	int __cdecl __onexitinit( void )	
0x000113A4	Function	17	__init_ah_hooks	void __cdecl __init_ah_hooks( void	

# BINARY DIFFING

1. Open the two files you want to compare in new tabs.

2. Click to select the second file and begin the differential analysis. The task manager will display the progress.

4. Unmodified blocks are white. Blocks with at least 1 modification are a light orange. Removed blocks are red and added blocks are green.

5. Modified lines are orange. Removed lines are red and added lines are green. If you toggle linking the split view, scrolling and selecting a line will be synced in both views.

3. Browse the results and see the items that are equal, modified, removed or added. Right click for more options.

The screenshot shows a binary diffing tool interface with two side-by-side code windows. The left window is labeled 'A: psping.exe' and the right window is labeled 'B: psping.exe'. Both windows show assembly code with a central diffing area between them. The diffing area shows a comparison of code blocks. The left window shows a block 'code\_0x88DF' with assembly instructions: 'push dword ptr [data\_0x27EB8]', 'call func\_0x8360 ; void \_\_cdecl( int p1', 'pop ecx', and 'ret'. The right window shows a block 'code\_0xFE55' with assembly instructions: 'push dword ptr [data\_0x3C3C0]', 'call \_free ; void \_\_cdecl(', 'and dword ptr [data\_0x3C3C0]', 'pop ecx', and 'ret'. The diffing area shows that the 'and' instruction in the right window is highlighted in green, indicating it is an added line. The 'call' instruction in the right window is highlighted in orange, indicating it is a modified line. The 'push' and 'pop' instructions in the right window are highlighted in light orange, indicating they have at least one modification. The 'ret' instruction in the right window is highlighted in white, indicating it is unmodified. The bottom of the interface shows a comparison table with columns for 'Difference', 'Item Type', 'Diff Type', 'Item A', and 'Item B'. The table lists several functions that are modified, removed, or added. The 'Difference' column shows percentages ranging from 6.67% to 8.33%. The 'Item Type' column lists 'Function'. The 'Diff Type' column lists 'Modified', 'Removed', and 'Added'. The 'Item A' and 'Item B' columns list the names of the functions being compared. The table is filtered to show 349 items.

Difference	Item Type	Diff Type	Item A	Item B
8.33%	Function	Modified	_wctomb_s	_wctomb_s
7.69%	Function	Modified	__cftoe	__cftoe
7.69%	Function	Modified	_swprintf_s	_swprintf_s
7.42%	Function	Modified	__close_nolock	__close_nolock
7.22%	Function	Modified	__lock_file2	__lock_file2
7.14%	Function	Modified	_cfltcvt	__cfltcvt
6.67%	Function	Modified		
6.67%	Function	Modified		

Relyze Desktop 3.0.4

File Analysis Tools Help

foo.dll x psping.exe x psping.exe x +

Select to perform a pseudocode diff of the current function.

A: psping.exe

```

0x000088CC:00001 void __cdecl ___endstdio( void )
0x000088CC:00002 {
0x000088CC:00003     __flushall();
0x000088CC:00004     if( data_0x26BAC != 0 ) {
0x000088CC:00005         func_0xD2B6();
0x000088CC:00006     }
0x000088CC:00007     func_0x8360( data_0x27EB8 );
0x000088CC:00008 }

```

B: psping.exe

```

0x0000FE42:00001 void __cdecl ___endstdio( void )
0x0000FE42:00002 {
0x0000FE42:00003     __flushall();
0x0000FE42:00004     if( data_0x39C58 != 0 ) {
0x0000FE42:00005         func_0x15680();
0x0000FE42:00006     }
0x0000FE42:00007     _free( data_0x3C3C0 );
0x0000FE42:00008     data_0x3C3C0 = 0;
0x0000FE42:00009 }

```

Difference	Item Type	Diff Type	Item A	Item B
8.33%	Function	Modified	_wctomb_s	_wctomb_s
7.69%	Function	Modified	__cftoe	__cftoe
7.69%	Function	Modified	_swprintf_s	_swprintf_s
7.42%	Function	Modified	__close_nolock	__close_nolock
7.22%	Function	Modified	__lock_file2	__lock_file2
7.14%	Function	Modified	__cfltcvt	__cfltcvt
6.67%	Function	Modified	___endstdio	___endstdio
6.67%	Function	Modified	fast error exit	fast error exit

Enter Filter Text... X 349 items

1. Right click on either a label or reference and select "Graph Reference". You can graph references either to or from a selected location. A new reference graph is displayed in the split view.

2. Right click on a graph node to select a start and end path. Double click a node to display it.

3. Browse the paths or sort them by depth.

The screenshot shows the Relyze Desktop interface with the following components:

- Control Flow Graph (CFG):** A graph with nodes `__cinit`, `__initterm_e`, `_atexit`, and `func_0x10C73`. Edges represent control flow between these functions.
- Assembly View:** Shows the assembly code for `code_0xFAB1:` and `int __cdecl __cinit( int p1 )`. The assembly includes instructions like `push ebp`, `mov ebp, esp`, `cmp dword ptr [data_0x259E0], 0x0`, and `jz code_0xFACA`.
- Path List:** A table showing selected paths:
 

Path	Depth
Path 1	4
Path 2	5
- Function List:** A table listing functions with their RVA, Type, Length, and Value:
 

RVA	Type	Length	Value
0x0000FAA5	Function	132	<code>__cinit</code>
0x0000FB75	Function	54	<code>__initterm</code>
0x0000FBAB	Function	34	<code>__initterm_e</code>
0x0000FDCF	Function	115	<code>__initstdio</code>
0x000107C6	Function	88	<code>__initMTAoncurrentthread</code>
0x0001089B	Function	80	<code>__uninitMTAoncurrentthread</code>
0x00010C44	Function	47	<code>__onexitinit</code>
0x000113A4	Function	17	<code>__init_ah_hooks</code>

Open the Data Type Manager via either the analysis menu or the default keyboard shortcut 'F3'.

The screenshot shows the 'Data Type Manager (87 Types)' window. On the left, there are settings for 'Input Source' (Snippet), 'Language' (C), and 'Compatibility' (MSVC), along with an 'Import...' button. The main area displays a C code snippet:

```
1
2 #ifndef RELYZE_IMPORT_H
3 #define RELYZE_IMPORT_H
4
5 // Note: Include data_type_manager.h in order to access e
6 #include <data_type_manager.h>
7
8 struct _THREAD_DATA {
9     BOOLEAN block;
10    void * thread_ptr;
11 };
12
13 #endif
```

At the bottom right of the window is a 'Close' button.

Import a custom data type by defining it in C/C++. You can then set a data block to this new type.

The 'Set Data Type...' dialog box is shown with the following fields:

- Type: Structure
- Name: THREAD\_DATA
- Start Stack RVA: 0x0040000C
- End Stack RVA: 0x00400014
- Length: 0x8 (8 bytes)

Buttons for 'Ok' and 'Cancel' are at the bottom right.

Select either a label or reference to a data block and then set its type via the right click menu or the default keyboard shortcut 'T'.

Open the analysis options dialog via the analysis menu or the keyboard shortcut "F2".

The screenshot shows a dialog box titled "Options" with a "Code" tab selected. It contains the following settings:

- Address Mode:** A dropdown menu set to "Relative Virtual Address".
- Custom Image Base:** A text input field containing "0x10000000".
- Symbol Names:** A dropdown menu set to "Unmangle Names".
- Comments:** A checkbox labeled "Automatic Comments" which is checked.

At the bottom right of the dialog are "OK" and "Cancel" buttons.

By default all addresses are displayed as Relative Virtual Addresses. You can select to use Virtual Addresses instead and optionally specify a custom base address to instantly rebase the analysis.

SAVING ANALYSIS ARCHIVES

Save...

Location: My Library

Name: foo.exe

Encrypt:  Password...

Tags:

- PE
- x86
- MyCustomTag
- +

Description:

Save Cancel

Save the analysis archive either to your library or to an external file.

Default tags will automatically be listed here. You can add in custom tags as needed. Right click to rename or re-colour a tag.

Access your library via the application main Tools menu. Double click on an item to open the analysis.

Use the application main menu or the keyboard shortcut "Ctrl-S" to save an analysis to your library.

Relyze Library Manager

Name	Saved ▼	Tags
mount	Thursday 25 May 2017, 11:56	ARM ELF
test_thumb2_it.elf	Thursday 18 May 2017, 15:39	ARM ELF
devmgr.dll.testcase	Monday 15 May 2017, 09:42	x86 PE
Vcl.Consts.o	Monday 27 March 2017, 10:41	x64 ELF
arm_switch	Tuesday 28 February 2017, 14:01	ARM ELF
test_fracture.exe	Monday 27 February 2017, 13:49	x86 PE
stat	Monday 27 February 2017, 11:27	x64 ELF
test_fracture.exe	Wednesday 15 February 2017, 14:15	x86 PE
nethost.exe	Tuesday 24 January 2017, 16:55	x86 Malware PE
foo.exe	Friday 25 November 2016, 15:12	x86 MyCustomTag PE

Enter Filter Text...

Open the batch analysis dialog via the main application tab.

Configure the directory and the file extensions to process.

Batch analysis saves the results to your library; configure how to handle duplicates when an existing analysis already exists.

Add custom tags, in addition to the default tags that will be added, to all files processed during batch analysis.

Configure analysis options and select which plugins to automatically run during batch analysis.

**Batch Analysis - Configure Input Options...**

Directory: C:\Samples\

Extensions:  .exe, .dll

Parallel Instances: 12

Options:

- Recurse into sub directories
- Skip files with unknown file formats
- Verbose logging
- Save to library

**Batch Analysis - Configure Analysis Options...**

Duplicates: Skip - Skip new analysis in favor of existing analysis in library

Tags:

- Batch Analysis Run #1
- Malware

Batch analysis is a convenient way to process many files at once. Every file successfully processed can be saved to your library. Custom plugins can be written to filter, modify and analyse files being processed during batch analysis. You can also run Relyze from the command line in a headless mode if you wish to perform any plugin based analysis without the GUI.

Press F1 at any time to display the Plugin Editor.

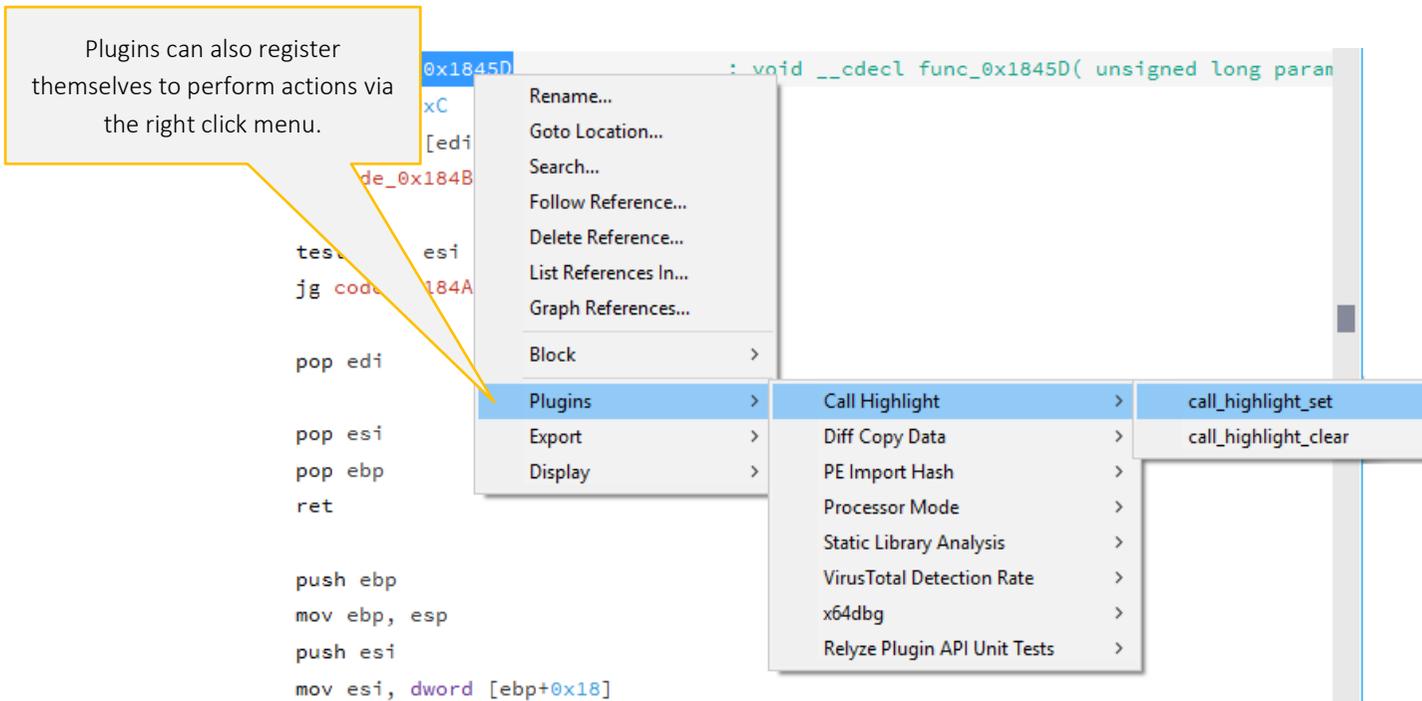
```
Relyze Plugin Editor
File Edit Plugin Help
New x +
1 require 'relyze/core'
2
3 class Plugin < Relyze::Plugin::Analysis
4
5   def initialize
6     super( {
7       :guid      => '{D7C262A6-DCD1-4D48-89A4-5F012ED22992}',
8       :name      => 'Your Plugin Name',
9       :description => 'Your Plugin Description',
10      :authors   => [ 'Your Name' ],
11      :license   => 'Your License',
12      :references => [ 'www.your.website' ]
13    } )
14   end
15
16   def run
17     print_message( "Hello World" )
18   end
19 end
```

Hello World

The main body of your plugin goes here. Plugins may also attach to the analysis pipeline to be invoked at different stages of analysis.

Every plugin has some associated metadata, including a unique GUID, a plugin name and description and information about the authors.

Press F5 to run the currently displayed plugin. The output console will display plugin output.



To learn more about how to create and run plugins, visit [http://support.relyze.com/knowledge\\_base/categories/plugins-3](http://support.relyze.com/knowledge_base/categories/plugins-3) and <https://www.relyze.com/docs/SDK/index.html>

## KEYBOARD SHORTCUTS

The default keyboard shortcuts are shown below. Additional keyboard shortcuts may be configured via the application options dialog.

Display Plugin Editor	F1	Block To Data	D
Display Analysis Options	F2	Block To Code	C
Display Analysis Task Manager	F3	Data Type Edit	T
Display Analysis Data Type Manager	F4	References In List	X
Plugin Editor - Run Plugin	F5	References Graph	Z
Analysis Save	Ctrl-S	Bookmark Edit	B
Navigate Back	ESCAPE	Comment Edit	;
Full Screen	F11	Jump Table Edit	J
Flash Instruction Analysis	Hold TAB	Copy Selected	Ctrl-C
Goto	G	Instruction Edit	E
Search	S	Cycle Between Flat and Flow Views	SPACE
Block Rename	N		

## COMMAND LINE USAGE

You can run Relyze from the command line to analyze a file without bringing up the GUI by using the `/analyze` switch, for example:

```
RelyzeCLI.exe /analyze "c:\samples\foo.dll"
```

Additional command line options for `/analyze` are shown below:

`/library <c:\path\to\library>` - Specify a directory to use for the library. All saved analysis archives will be placed in this directory.

`/tags "name1:hexcolor1,name2:hexcolor2"` - Specify custom tags to apply to the saved analysis archive.

`/nosave` - Don't save the analysed file to the library.

`/noflat` - If the input file is not a recognised executable file format, don't process it as a flat binary file.

`/skip` - If a duplicate analysis archive exists in the library, don't analyse the new file.

`/replace` - If a duplicate analysis archive exists in the library, replace the existing archive with the newly analysed archive.

`/add` - Add the new analysis archive to the library, ignoring any duplicate archive that may exist.

`/nosymbols` - Don't try to retrieve or use symbols during analysis.

`/decoder <GUID>` - Run a given decoder plugin over the input file before analysis.

`/plugin <File|GUID>` - Run a given analysis plugin during analysis. To specify multiple plugins use a separate `/plugin` argument for each one.

`/plugin_commandline "/opt1 /opt2=1234 /opt3=5678"` - Specify any custom options to pass to all the plugins.

You can run a plugin directly from the command line via the `/run` switch, for example:

```
RelyzeCLI.exe /run /plugin "c:\samples\scripts\testing.rb"
```

Additional command line options for `/run` are shown below:

`/plugin <File|GUID>` - Run a given analysis plugin. To specify multiple plugins use a separate `/plugin` argument for each one.

`/plugin_commandline "/opt1 /opt2=1234 /opt3=5678"` - Specify any custom options to pass to all the plugins.

`/log <c:\log.txt>` - Log all plugin messages to a specific log file.

`/library <c:\path\to\library>` - Specify a directory to use for the library. All saved analysis archives will be placed in this directory.

## SUPPORT

All online support options are available at <https://www.relyze.com/support>

The online plugin SDK documentation is available at <https://www.relyze.com/docs/SDK/index.html>

A repository of official and example plugins is available at <https://github.com/relyze-ltd/plugins>

For technical support please email [support@relyze.com](mailto:support@relyze.com)