IDA PRO – the state-of-the-art binary code analysis tool

IDA Pro is the flagship product of Hex-Rays, the software provider in reverse engineering. Being an interactive and programmable disassembler and debugger, IDA Pro provides excellent quality performance on different platforms and is compatible with many processors. IDA Pro has become the de-facto standard for the analysis of hostile code, vulnerability research and commercial off-the-shelf validation.

IDA Pro comes with different types of licenses: Named, Computer, Floating and Educational license to meet different business' scales and demands of usage.

A disassembler | A debugger | interactive | Programmable

Key features

Multi-processor Disassembler

- Disassembler modules for a large number of processors. The free SDK even allows you to run your custom disassembler;
- Full and extensible interactivity;
- Programmable: IDA can be extended in line with user's own requirement with IDC or IDAPython;
- Open plugin architecture: external plugins enable extension of IDA's capability;
- FLIRT technology (Fast library identification and recognition technology);
- Code graphing;
- Lumina server holds metadata with a large number of well-known functions;

Multi-target Debugger

- The debugger adds the dynamic analysis of the information collected statically by the disassembler;
- Offers all the features expected from a debugger and more: "remote" function and tracking. Remote debugger: for Windows, Linux, Mac OS X, and other machines in any combination;

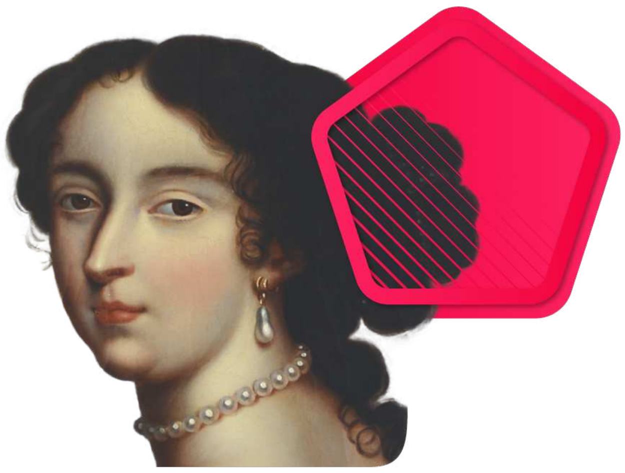
More features and upgrades are introduced along with new IDA version releases!

IDA 8.0 Highlights

As usual, IDA has improved on many fronts, but the most important feature this time around - and the one that justifies the major version bump - is the introduction of the brand-new IDA Teams.

IDA Teams is out!

We've been hard at work putting the finishing touches to our new product: an integrated set of tools to bring seamless collaboration among teams of [reverse-]engineers.



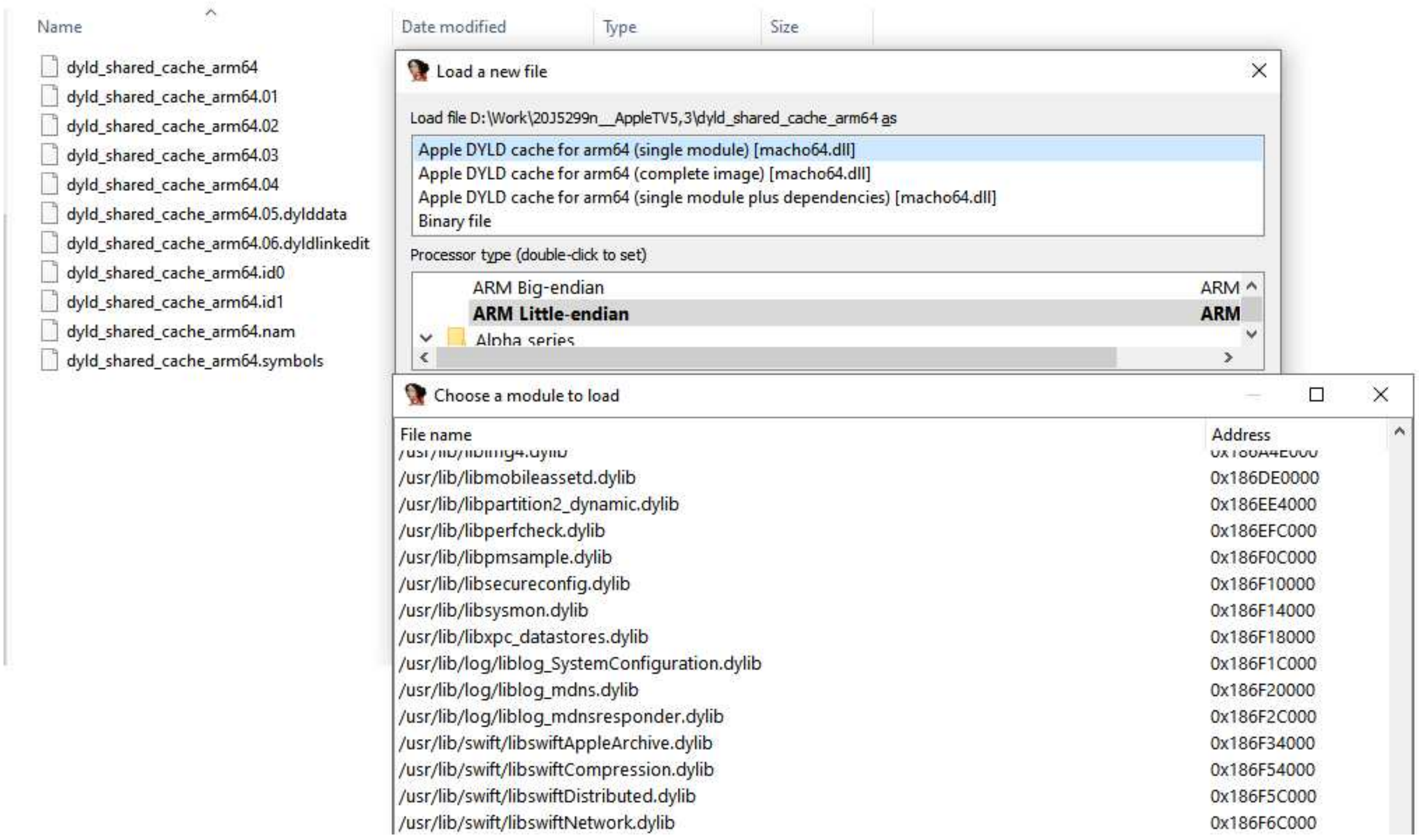
Inspired by the best revision control tools out there, IDA Teams brings their key concepts to IDA databases & workflows:

- revision control,
- changes attribution,
- diffing & merging

We're very proud - and excited! - by what IDA Teams can accomplish, and hope you will appreciate its potential. But we won't stop there: we already have great ideas for the next versions!

iOS 16 dyld shared cache support

Once again, Apple has modified the internal format of the system files on their mobile operating systems to improve performance and memory usage, so we have updated IDA to support the new tweaks.



```
Name                                    Date modified    Type    Size
dyld_shared_cache_arm64
dyld_shared_cache_arm64.01
dyld_shared_cache_arm64.02
dyld_shared_cache_arm64.03
dyld_shared_cache_arm64.04
dyld_shared_cache_arm64.05.dylddata
dyld_shared_cache_arm64.06.dyldlinkedit
dyld_shared_cache_arm64.id0
dyld_shared_cache_arm64.id1
dyld_shared_cache_arm64.nam
dyld_shared_cache_arm64.symbols
```

Load a new file

Load file D:\Work\20J5299n__AppleTV5,3\dyld_shared_cache_arm64 as

- Apple DYLD cache for arm64 (single module) [macho64.dll]
- Apple DYLD cache for arm64 (complete image) [macho64.dll]
- Apple DYLD cache for arm64 (single module plus dependencies) [macho64.dll]
- Binary file

Processor type (double-click to set)

ARM Big-endian	ARM
ARM Little-endian	**ARM**
Alpha series	

Choose a module to load

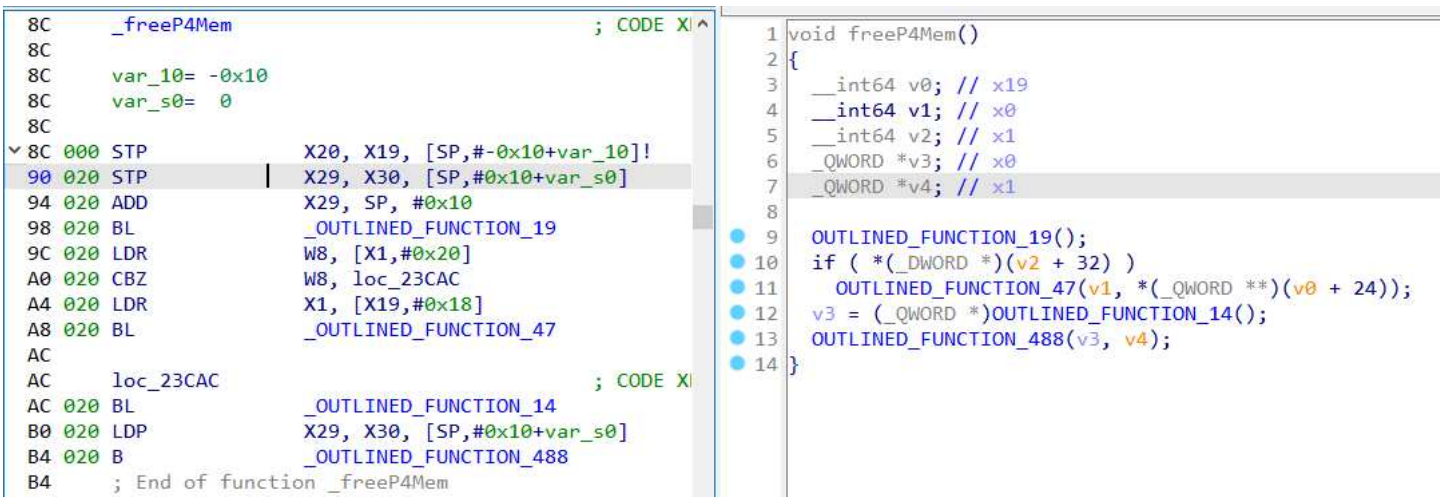
File name	Address
/usr/lib/libmobileassetd.dylib	0x186DE0000
/usr/lib/libpartition2_dynamic.dylib	0x186EE4000
/usr/lib/libperfcheck.dylib	0x186EFC000
/usr/lib/libpmsample.dylib	0x186F0C000
/usr/lib/libsecureconfig.dylib	0x186F10000
/usr/lib/libsysmon.dylib	0x186F14000
/usr/lib/libxpc_datastores.dylib	0x186F18000
/usr/lib/log/liblog_SystemConfiguration.dylib	0x186F1C000
/usr/lib/log/liblog_mdns.dylib	0x186F20000
/usr/lib/log/liblog_mdnsresponder.dylib	0x186F2C000
/usr/lib/swift/libswiftAppleArchive.dylib	0x186F34000
/usr/lib/swift/libswiftCompression.dylib	0x186F54000
/usr/lib/swift/libswiftDistributed.dylib	0x186F5C000
/usr/lib/swift/libswiftNetwork.dylib	0x186F6C000

Outlined functions

Another new trick used by Apple on recent iOS versions is outlined functions. This is a code size optimization where the compiler extracts common instruction sequences used in several functions into a new pseudo-function which is then "called" from multiple places.

This behavior tends to make decompiled pseudocode ugly, especially when the outlined functions use arbitrary registers or stack variables from the parent function. Now you can mark such functions with a special attribute and the decompiled code will look much nicer.

For example, here's a fragment of a file which has been compiled with such optimization:

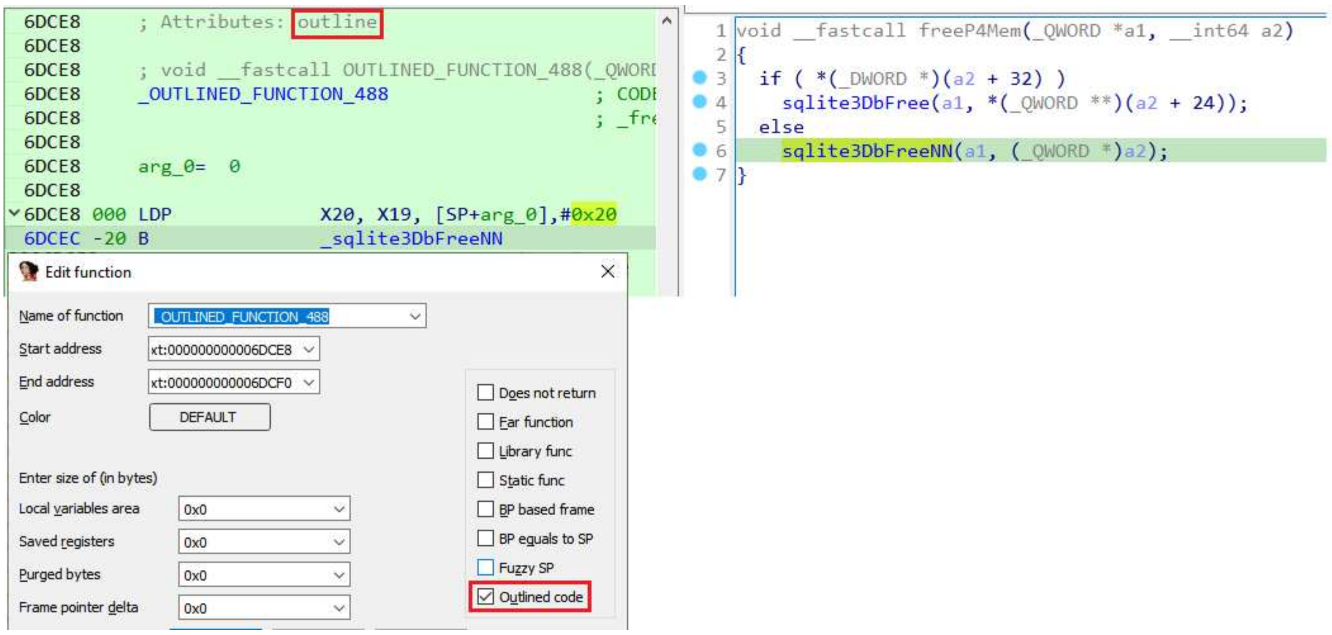


```
8C      _freeP4Mem                               ; CODE XI
8C
8C      var_10= -0x10
8C      var_s0=  0
8C
8C 000 STP              X20, X19, [SP,#-0x10+var_10]!
90 020 STP              X29, X30, [SP,#0x10+var_s0]
94 020 ADD              X29, SP, #0x10
98 020 BL               _OUTLINED_FUNCTION_19
9C 020 LDR              W8, [X1,#0x20]
A0 020 CBZ              W8, loc_23CAC
A4 020 LDR              X1, [X19,#0x18]
A8 020 BL               _OUTLINED_FUNCTION_47
AC
AC      loc_23CAC                                ; CODE XI
AC 020 BL               _OUTLINED_FUNCTION_14
B0 020 LDP              X29, X30, [SP,#0x10+var_s0]
B4 020 B                _OUTLINED_FUNCTION_488
B4      ; End of function _freeP4Mem
```

```
void freeP4Mem()
{
  __int64 v0; // x19
  __int64 v1; // x0
  __int64 v2; // x1
  _QWORD *v3; // x0
  _QWORD *v4; // x1

  OUTLINED_FUNCTION_19();
  if ( *(_DWORD *)(v2 + 32) )
    OUTLINED_FUNCTION_47(v1, *(_QWORD **)(v0 + 24));
  v3 = (_QWORD *)OUTLINED_FUNCTION_14();
  OUTLINED_FUNCTION_488(v3, v4);
}
```

In initial pseudocode, we can see explicit calls to the compiler-generated _OUTLINED_FUNCTION_NN fragments, and some variables marked in orange because the decompiler considers them to be spoiled by the calls (which is the case with calls to regular functions).

After marking those pseudo-functions with ‘outline’ attribute and refreshing the pseudocode, the decompiler inlines everything and there are no more orange variables:



```
6DCE8      ; Attributes: outline
6DCE8
6DCE8      ; void __fastcall OUTLINED_FUNCTION_488(_QWORD
6DCE8      _OUTLINED_FUNCTION_488                        ; CODE
6DCE8                                                    ; _fre
6DCE8
6DCE8      arg_0=  0
6DCE8
6DCE8 000 LDP             X20, X19, [SP+arg_0],#0x20
6DCEC -20 B               _sqlite3DbFreeNN
```

```
void __fastcall freeP4Mem(_QWORD *a1, __int64 a2)
{
  if ( *(_DWORD *)(a2 + 32) )
    sqlite3DbFree(a1, *(_QWORD **)(a2 + 24));
  else
    sqlite3DbFreeNN(a1, (_QWORD *)a2);
}
```

Edit function

Name of function	OUTLINED_FUNCTION_488
Start address	xt:000000000006DCE8
End address	xt:000000000006DCF0
Color	DEFAULT

Enter size of (in bytes)

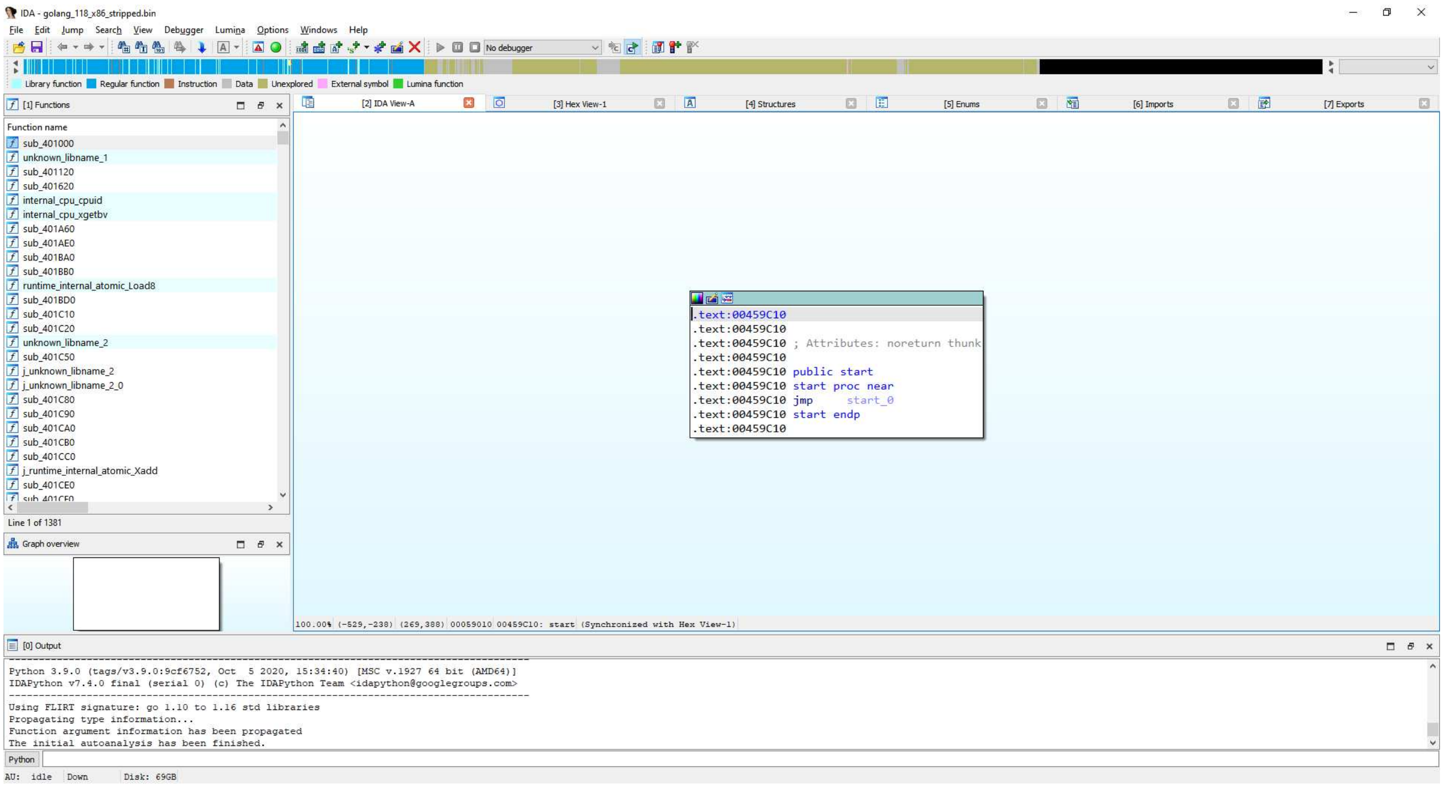
Local variables area	0x0
Saved registers	0x0
Purged bytes	0x0
Frame pointer delta	0x0

- [ ] Does not return
- [ ] Far function
- [ ] Library func
- [ ] Static func
- [ ] BP based frame
- [ ] BP equals to SP
- [ ] Fuzzy SP
- [✓] Outlined code

Golang 1.18

The new version of the language introduced changes to the layout of the metadata. New IDA version can parse it and nicely recover function names.

For example, here’s a stripped golang 1.18 binary in IDA 7.7:



IDA - golang_118_x86_stripped.bin

Function name

```
sub_401000
unknown_libname_1
sub_401120
sub_401620
internal_cpu_cpuid
internal_cpu_xgetbv
sub_401A60
sub_401AE0
sub_401BA0
sub_401BB0
runtime_internal_atomic_Load8
sub_401BD0
sub_401C10
sub_401C20
unknown_libname_2
sub_401C50
j_unknown_libname_2
j_unknown_libname_2_0
sub_401C80
sub_401C90
sub_401CA0
sub_401CB0
sub_401CC0
j_runtime_internal_atomic_Xadd
sub_401CE0
sub_401CF0
```

Line 1 of 1381

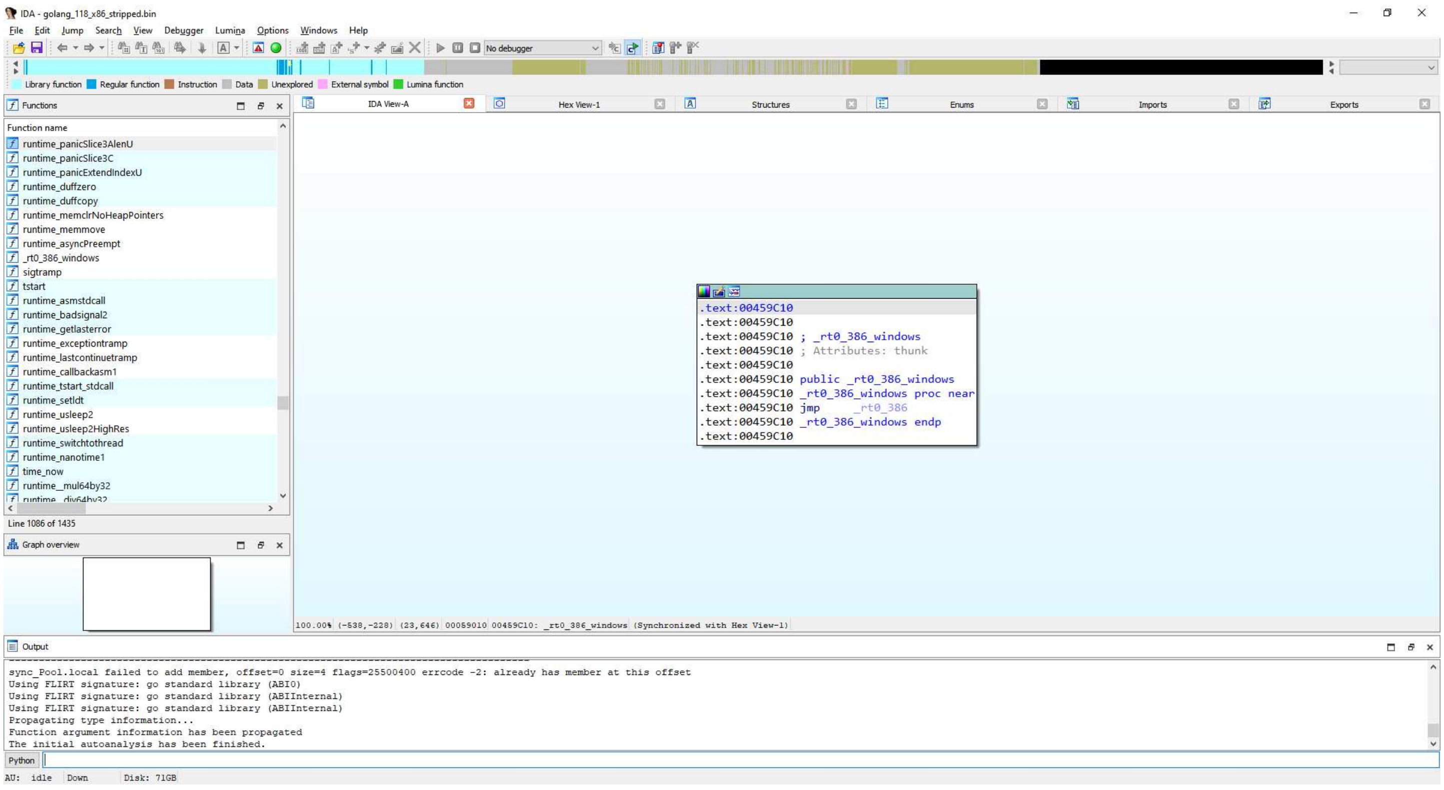
```
.text:00459C10
.text:00459C10
.text:00459C10 ; Attributes: noreturn thunk
.text:00459C10
.text:00459C10 public start
.text:00459C10 start proc near
.text:00459C10 jmp     start_0
.text:00459C10 start endp
.text:00459C10
```

```
100.00% (-529,-238) (269,388) 00059010 00459C10: start (Synchronized with Hex View-1)
```

```
---------------------------------------------------------------------------------
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)]
IDAPython v7.4.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------
Using FLIRT signature: go 1.10 to 1.16 std libraries
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```

```
AU:  idle   Down      Disk: 69GB
```

And the same binary in IDA 8.0:


IDA - golang_118_x86_stripped.bin
File Edit Jump Search View Debugger Lumina Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
Graph overview
Output
Python

```
Function name
runtime_panicSlice3AlenU
runtime_panicSlice3C
runtime_panicExtendIndexU
runtime_duffzero
runtime_duffcopy
runtime_memclrNoHeapPointers
runtime_memmove
runtime_asyncPreempt
_rt0_386_windows
sigtramp
tstart
runtime_asmstdcall
runtime_badsignal2
runtime_getlasterror
runtime_exceptiontramp
runtime_lastcontinuetramp
runtime_callbackasm1
runtime_tstart_stdcall
runtime_setldt
runtime_usleep2
runtime_usleep2HighRes
runtime_switchtothread
runtime_nanotime1
time_now
runtime__mul64by32
runtime__div64by32
Line 1086 of 1435
```

```
.text:00459C10
.text:00459C10
.text:00459C10 ; _rt0_386_windows
.text:00459C10 ; Attributes: thunk
.text:00459C10
.text:00459C10 public _rt0_386_windows
.text:00459C10 _rt0_386_windows proc near
.text:00459C10 jmp     _rt0_386
.text:00459C10 _rt0_386_windows endp
.text:00459C10
```

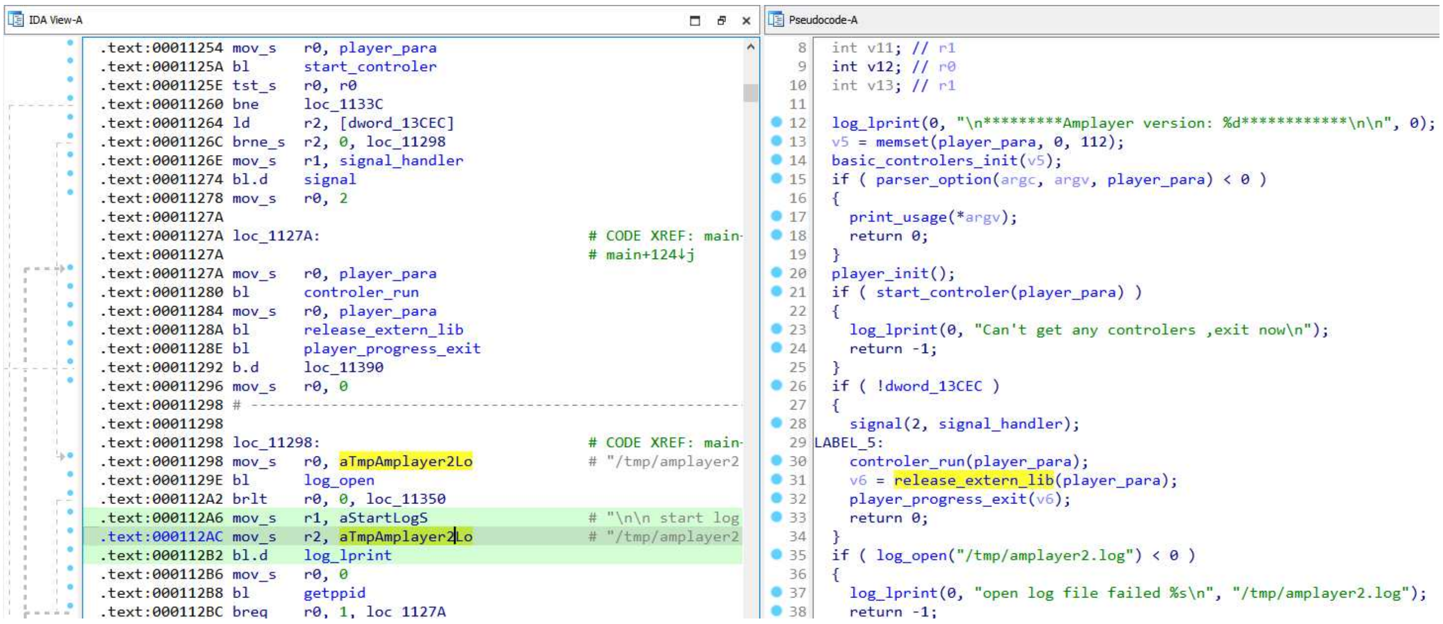
```
100.00% (-538,-228) (23,646) 00059010 00459C10: _rt0_386_windows (Synchronized with Hex View-1)
```

```
------------------------------------------------------------------------------------------
sync_Pool.local failed to add member, offset=0 size=4 flags=25500400 errcode -2: already has member at this offset
Using FLIRT signature: go standard library (ABI0)
Using FLIRT signature: go standard library (ABIInternal)
Using FLIRT signature: go standard library (ABIInternal)
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```

```
AU:  idle   Down      Disk: 71GB
```

ARC decompiler

ARC processors are used in many embedded devices such as SSD drives or Wi-Fi chipsets. It was also the core used in the infamous Intel ME until several years ago. The new decompiler supports all ARC instruction set variants currently supported by IDA: classic 32-bit ISA, 32/16-bit ARCompact and the new ARCv2. Delay slots, conditional instructions and zero-overhead loops are supported out of the box.



```
.text:00011254 mov_s   r0, player_para
.text:0001125A bl      start_controler
.text:0001125E tst_s   r0, r0
.text:00011260 bne     loc_1133C
.text:00011264 ld      r2, [dword_13CEC]
.text:0001126C brne_s  r2, 0, loc_11298
.text:0001126E mov_s   r1, signal_handler
.text:00011274 bl.d    signal
.text:00011278 mov_s   r0, 2
.text:0001127A
.text:0001127A loc_1127A:                          # CODE XREF: main-
.text:0001127A                                     # main+124↓j
.text:0001127A mov_s   r0, player_para
.text:00011280 bl      controler_run
.text:00011284 mov_s   r0, player_para
.text:0001128A bl      release_extern_lib
.text:0001128E bl      player_progress_exit
.text:00011292 b.d     loc_11390
.text:00011296 mov_s   r0, 0
.text:00011298 # ---------------------------------------------------------
.text:00011298
.text:00011298 loc_11298:                          # CODE XREF: main-
.text:00011298 mov_s   r0, aTmpAmplayer2Lo     # "/tmp/amplayer2
.text:0001129E bl      log_open
.text:000112A2 brlt    r0, 0, loc_11350
.text:000112A6 mov_s   r1, aStartLogS          # "\n\n start log
.text:000112AC mov_s   r2, aTmpAmplayer2Lo     # "/tmp/amplayer2
.text:000112B2 bl.d    log_lprint
.text:000112B6 mov_s   r0, 0
.text:000112B8 bl      getppid
.text:000112BC breq    r0, 1, loc_1127A
```

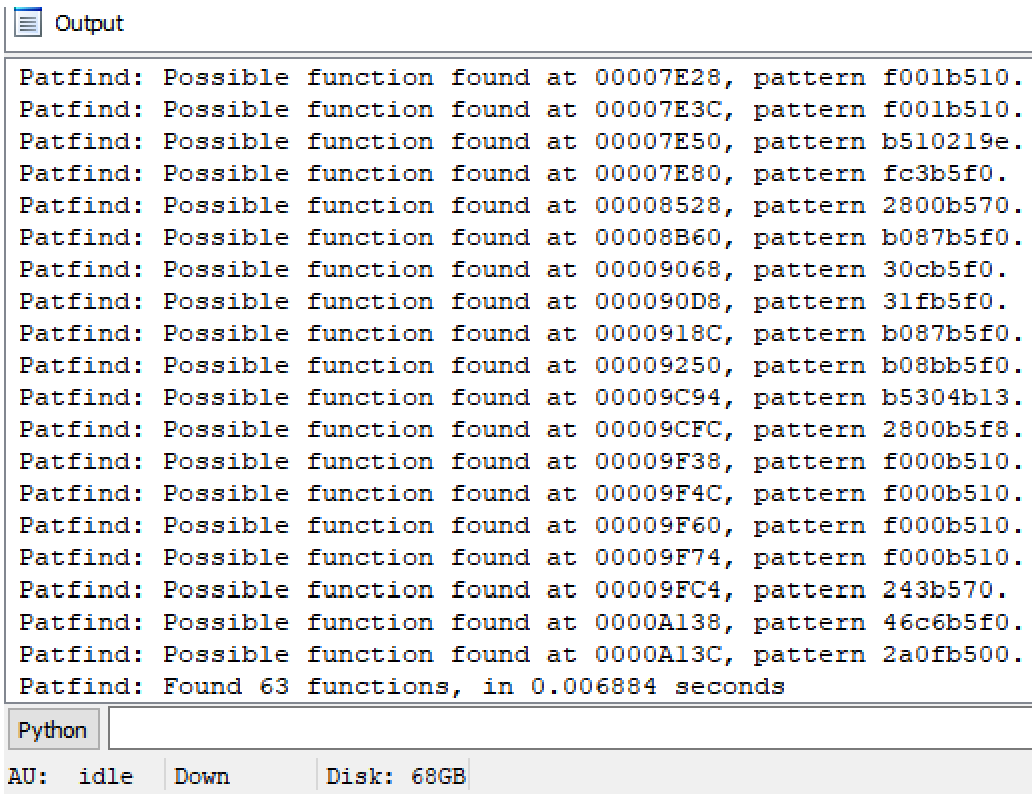
```
  int v11; // r1
  int v12; // r0
  int v13; // r1

  log_lprint(0, "\n*********Amplayer version: %d************\n\n", 0);
  v5 = memset(player_para, 0, 112);
  basic_controlers_init(v5);
  if ( parser_option(argc, argv, player_para) < 0 )
  {
    print_usage(*argv);
    return 0;
  }
  player_init();
  if ( start_controler(player_para) )
  {
    log_lprint(0, "Can't get any controlers ,exit now\n");
    return -1;
  }
  if ( !dword_13CEC )
  {
    signal(2, signal_handler);
LABEL_5:
    controler_run(player_para);
    v6 = release_extern_lib(player_para);
    player_progress_exit(v6);
    return 0;
  }
  if ( log_open("/tmp/amplayer2.log") < 0 )
  {
    log_lprint(0, "open log file failed %s\n", "/tmp/amplayer2.log");
    return -1;
```

No more Python 2

It's been over two years since Python 2.7 support has ended, and it's time to let go. IDA 8.0 will support only Py- thon 3.x, including the latest 3.10 (Windows installer includes Python 3.10.5).

Better firmware analysis thanks to the function finder plugin (patfind)

Firmware binaries usually do not have any symbols or other metadata which would help IDA to find code in the unmarked loaded data, so users had to do it manually. In the new release, we've added a plugin which makes use of the pattern format used in Ghidra (with minor extensions). This plugin is enabled by default for binary and binary-like formats and helps IDA discover more code automatically. It can also be invoked manually for normal structured files to find otherwise unreferenced code.



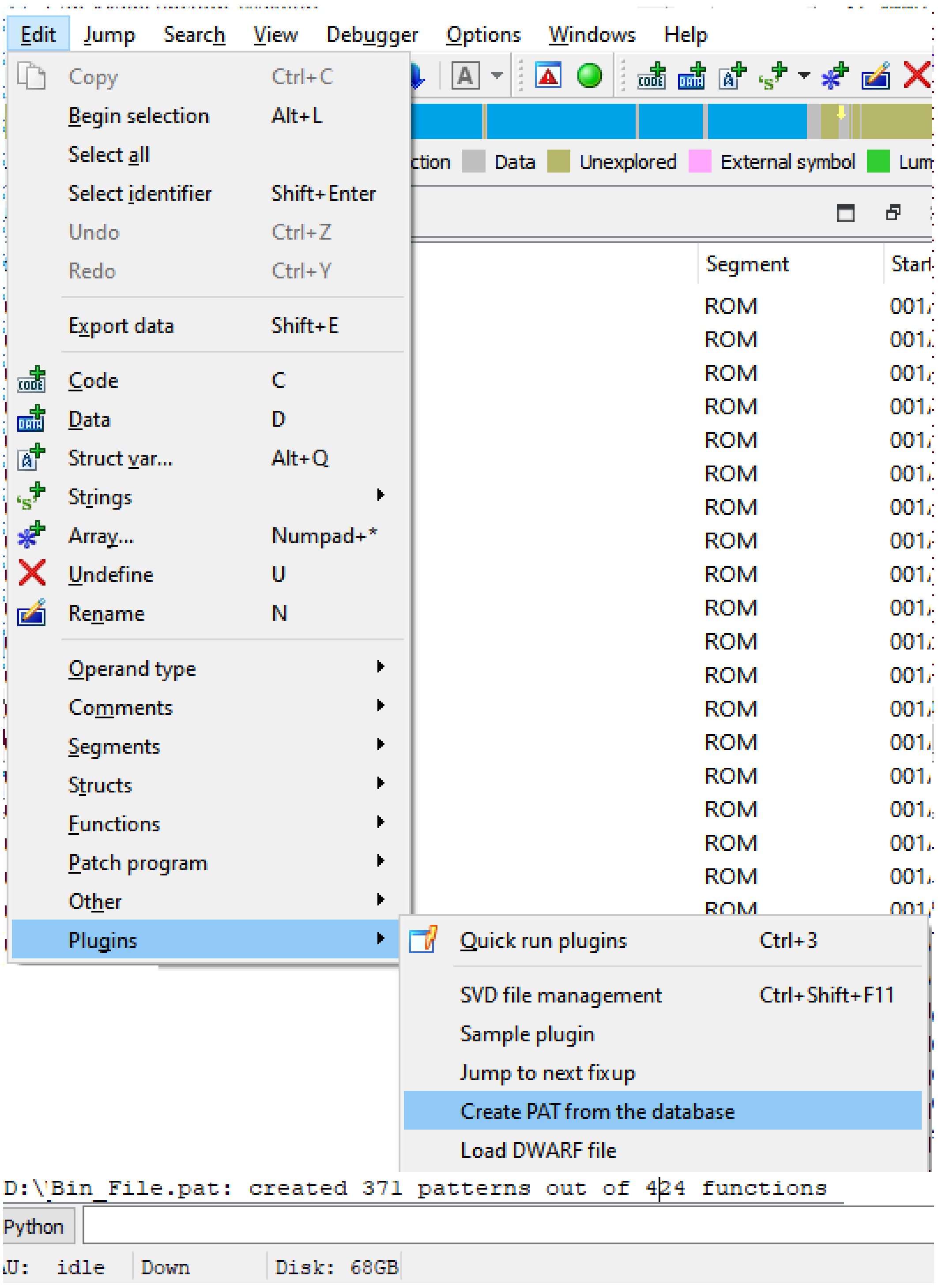
```
Patfind: Possible function found at 00007E28, pattern f001b510.
Patfind: Possible function found at 00007E3C, pattern f001b510.
Patfind: Possible function found at 00007E50, pattern b510219e.
Patfind: Possible function found at 00007E80, pattern fc3b5f0.
Patfind: Possible function found at 00008528, pattern 2800b570.
Patfind: Possible function found at 00008B60, pattern b087b5f0.
Patfind: Possible function found at 00009068, pattern 30cb5f0.
Patfind: Possible function found at 000090D8, pattern 31fb5f0.
Patfind: Possible function found at 0000918C, pattern b087b5f0.
Patfind: Possible function found at 00009250, pattern b08bb5f0.
Patfind: Possible function found at 00009C94, pattern b5304b13.
Patfind: Possible function found at 00009CFC, pattern 2800b5f8.
Patfind: Possible function found at 00009F38, pattern f000b510.
Patfind: Possible function found at 00009F4C, pattern f000b510.
Patfind: Possible function found at 00009F60, pattern f000b510.
Patfind: Possible function found at 00009F74, pattern f000b510.
Patfind: Possible function found at 00009FC4, pattern 243b570.
Patfind: Possible function found at 0000A138, pattern 46c6b5f0.
Patfind: Possible function found at 0000A13C, pattern 2a0fb500.
Patfind: Found 63 functions, in 0.006884 seconds
```

Comparison of the number of functions discovered after initial autoanalysis with default settings:

File type	IDA 7.7	IDA 8.0
Cortex M0 firmware (8KB)	0	59
ARM9 firmware (23KB)	0	97
ARM11 firmware (300KB)	84	1697
PowerPC firmware (2MB)	0	2223

FLAIR pattern generator (makepat)

The FLAIR toolkit, available to all IDA Pro customers, allows creating FLIRT signatures from static libraries. How- ever, such libraries are not always available - sometimes all you have is just the final linked binary. The makepat plugin supports creating .pat files for functions in an arbitrary IDA database. It uses the functionality added for Lumina to mark variable bytes in the patterns. The .pat can then be compiled to a signature file using sigmake and applied to other files



```
D:\Bin_File.pat: created 371 patterns out of 424 functions
```

```
U:  idle  Down   Disk: 68GB
```

Full changelist: https://www.hex-rays.com/products/ida/news/8_0/

Previous releases

IDA Version 7.7 – Service Pack 1 - Release date: 18th January 2022

Highlights: The Service Pack 1 of IDA 7.7 is primarily a bug fixes release that provides fixes for a few errors that might affect many users.

Full changelist: https://www.hex-rays.com/products/ida/news/7_7sp1/

IDA Version 7.7 – Release date: 24th December 2020

Highlights: IDA 7.7 introduced DA Teams, iOS 16 dyld shared cache support, Outlined functions Golang 1.18, New decompiler: ARC, Better firmware analysis thanks to the function finder plugin (patfind), FLAIR pattern generator (makepat) And much more!

Full changelist: https://www.hex-rays.com/products/ida/news/7_7/

IDA Version 7.6SP1 - Release date: 28th April 2021

Highlights: The Service Pack 1 of IDA 7.6 is primarily a bug fixes release that provides fixes for a few errors that might affect many users.

Full changelist: https://www.hex-rays.com/products/ida/news/7_6sp1/

IDA Version 7.6 - Release date: 22nd March 2021

Highlights: IDA 7.6 introduced Apple Silicon's full support, Golang analysis, thorough improvements for Hex-Rays Decompiler, two new processor: RISC-V and RL78, further supports for compressed macOS and iOS ker- nelcache and for Python 3.9

Full changelist: https://www.hex-rays.com/products/ida/news/7_6/

IDA Version 7.5 - Release date: May 2020

Highlights: IDA 7.5 introduced the tree-like folder view that helped with information organization and hence incredibly increase efficiency when it comes to large binaries analysis. MIPS decompiler was added to the lineup, with Lumina now available for both MIPS and PPC binaries. IDA 7.5 comprised many iOS/macOS improvements such as the just-added type libraries with the most major APIs and additional frameworks from macOS and iP- hone SDKs.

Full changelist: https://www.hex-rays.com/products/ida/news/7_5/

IDA Version 7.5 – Service Pack 1 - Release date: 19th June 2020

Highlights: This Service Pack was released to improve user experience especially for newly released features such as the tree-like folder view function and the MIPS Decompiler.

Full changelist: https://www.hex-rays.com/products/ida/news/7_5sp1/