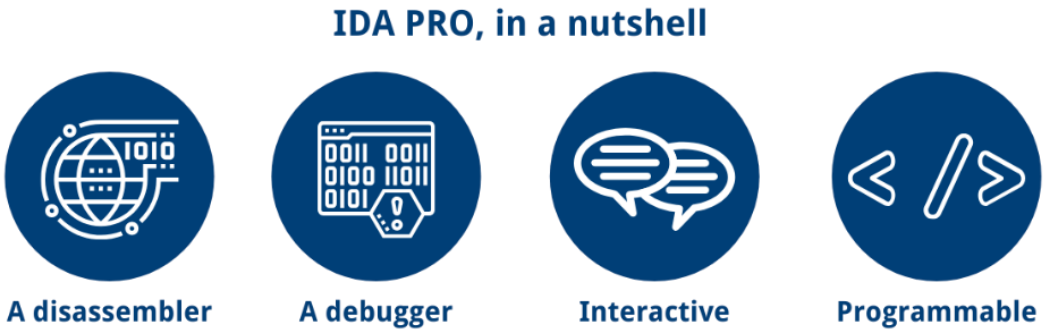
Hex-Rays

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.



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 PRO Version 7.6

Highlights of new features and improvements:

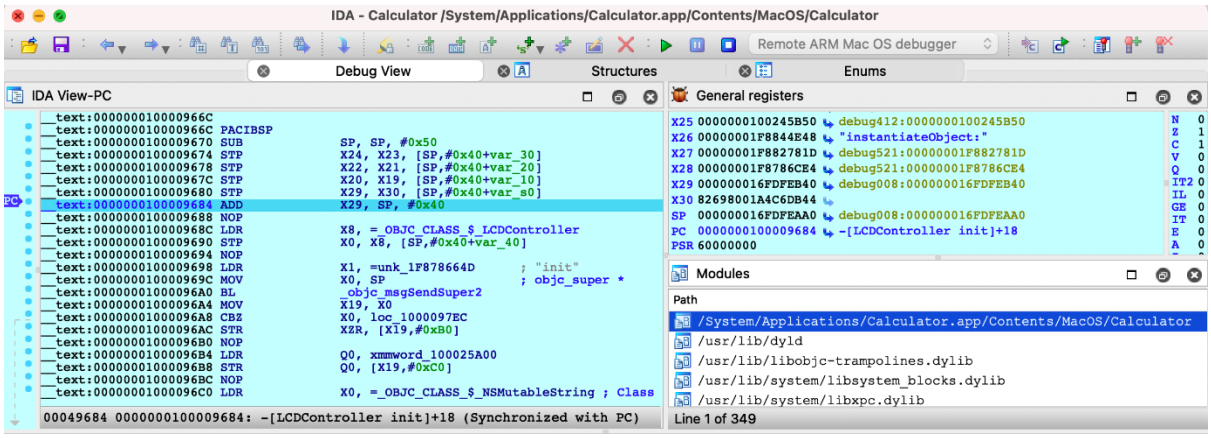
- **Apple Silicon support:**

IDA for macOS is now available as a native ARM64 binary which can make full use of the M1 chip's incredible performance.

It is hard to overstate just how much IDA benefits from the new speed boost. Auto-analysis completes much quicker, the UI is noticeably snappier, and almost every other feature in IDA seems smoother when running on M1.

Our beta testers reported that IDA 7.6 is "incredibly stable" and "way faster" on Apple Silicon.

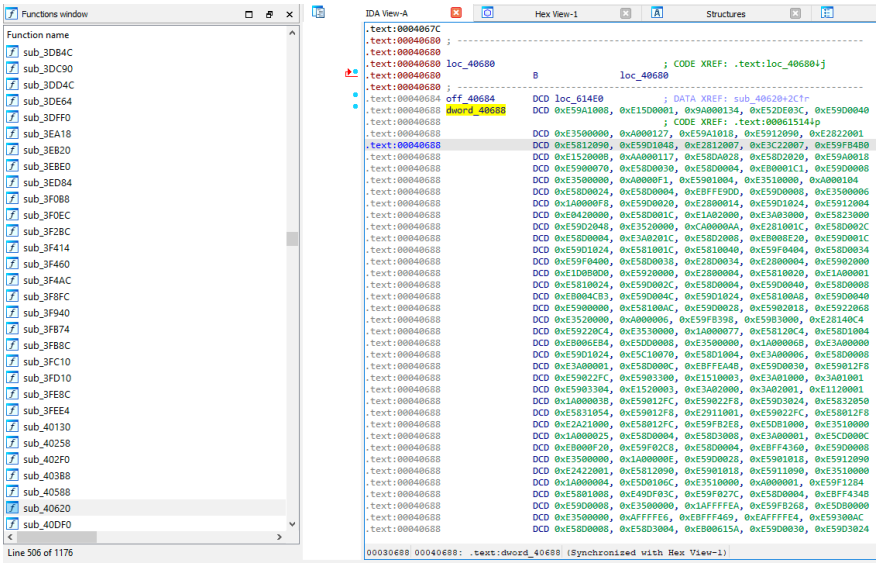
Debugging native arm64 processes is also supported on M1, including arm64e:



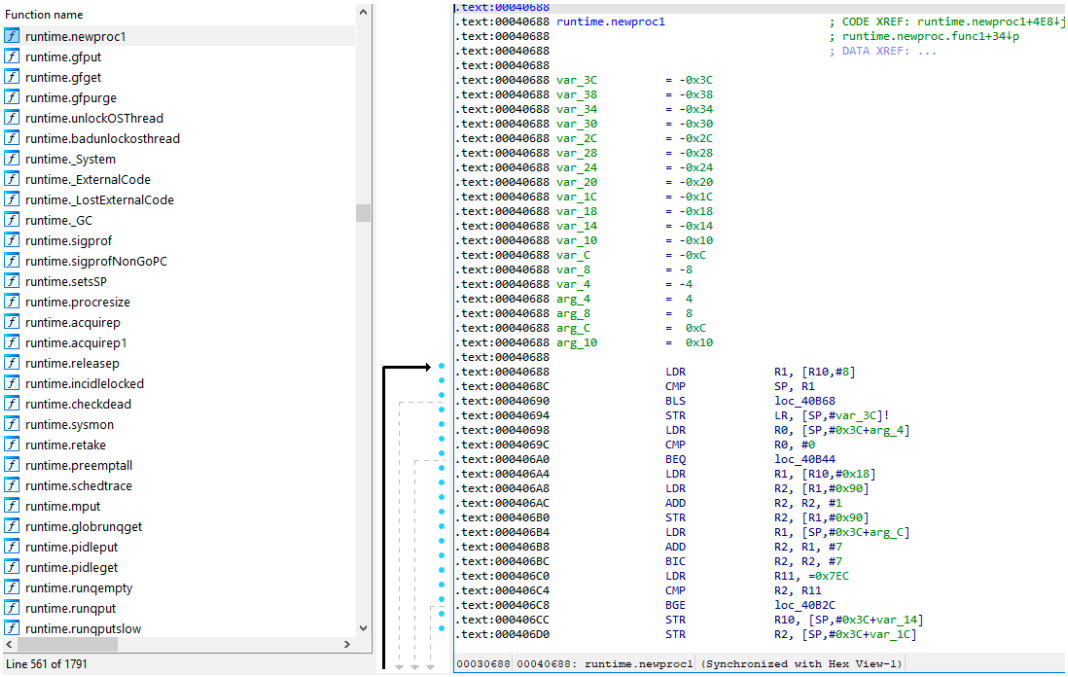
- **Golang analysis:**

The Go language (aka *golang*) from Google is getting popular thanks to its ease of use, performance, and self-contained binaries not requiring dependencies. Due to some of the language designers' decisions the *golang* binaries are quite different from those produced by other compilers and some changes were required in IDA to properly support its peculiarities.

Here's an example of how a stripped golang binary for ARM looks like in IDA 7.5:



and in 7.6:



Among additions:

- parsing of golang-specific metadata to recover function names and boundaries
- support for stack-based parameters and return values even on platforms that usually use registers (ARM, x64)
- detection of golang-specific string literals

- **Decompiler improvements:**
  - automatic renaming of variables
    Although interactivity is IDA's selling point, it still tries to do as much as possible to automate mundane tasks. With this release the decompiler will try to automatically assign names to variables and structure fields based on assignments and function calls.

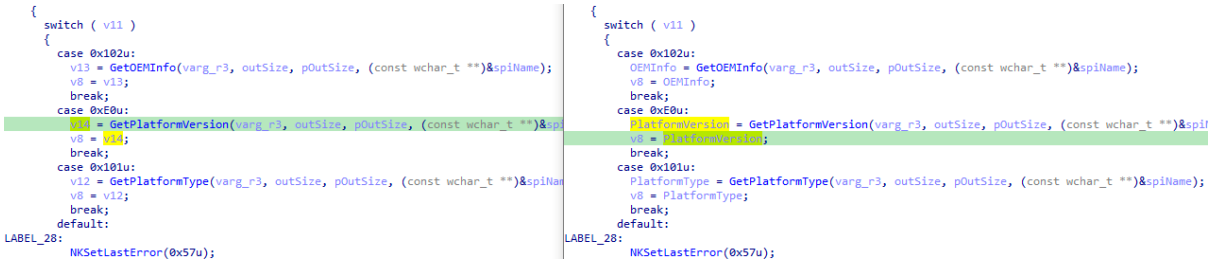
See two snippets from decompilation of the same binary.

IDA 7.5:

```
latencyState = OALWakeupLatency_GetCurrentState();
v7 = g_pTimerRegs->TCRR;
if ( (int)(dword_61874 - v7) >= dword_61864 )
{
  if ( (int)(dword_61874 - v7) >= g_wakeupLatencyConstraintTickCount )
    v12 = g_wakeupLatencyConstraintTickCount;
  else
    v12 = dword_61874 - v7;
  v2 = OALWakeupLatency_GetDelayInTicks(latencyState);
  if ( v12 < v2 )
  {
    latencyState = OALWakeupLatency_FindStateByMaxDelayInTicks(v12);
    v2 = OALWakeupLatency_GetDelayInTicks(latencyState);
  }
  if ( v12 >= v2 )
  {
    OEMWriteDebugLED(0x1Eu, 1u);
    if ( OALWakeupLatency_IsChipOff(latencyState) )
      OALContextSave();
    OALWakeupLatency_PushState(latencyState);
    dword_61874 -= v2;
    OALTimerSetCompare(dword_61874);
    g_pPrcmPrm->pOMAP_WKUP_PRM->PM_WKST_WKUP = 256;
    g_pPrcmPrm->pOMAP_WKUP_PRM->PM_WKEN_WKUP |= 0x100u;
    fnOALCPUIdle(g_pCPUInfo);
    g_pPrcmPrm->pOMAP_WKUP_PRM->PM_WKEN_WKUP &= 0xFFFFFEFF;
    OALWakeupLatency_PopState();
    v1 = g_pPrcmPrm->pOMAP_WKUP_PRM->PM_WKST_WKUP;
    data = g_pPrcmPrm->pOMAP_CORE_PRM->PM_PREPWSTST_CORE;
    prevMpuState = g_pPrcmPrm->pOMAP_MPU_PRM->PM_PREPWSTST_MPU;
    prevPerState = g_pPrcmPrm->pOMAP_PER_PRM->PM_PREPWSTST_PER;
    OALContextRestore(prevMpuState, data, prevPerState);
    v11 = g_pTimerRegs->TCRR;
    PrcmProcessPostMpuWakeup();
```

```
latencyState = OALWakeupLatency_GetCurrentState();
TCRR = g_pTimerRegs->TCRR;
if ( (int)(dword_61874 - TCRR) >= dword_61864 )
{
  if ( (int)(dword_61874 - TCRR) >= g_wakeupLatencyConstraintTickCount )
    v12 = g_wakeupLatencyConstraintTickCount;
  else
    v12 = dword_61874 - TCRR;
  DelayInTicks = OALWakeupLatency_GetDelayInTicks(latencyState);
  if ( v12 < DelayInTicks )
  {
    latencyState = OALWakeupLatency_FindStateByMaxDelayInTicks(v12);
    DelayInTicks = OALWakeupLatency_GetDelayInTicks(latencyState);
  }
  if ( v12 >= DelayInTicks )
  {
    OEMWriteDebugLED(0x1Eu, 1u);
    if ( OALWakeupLatency_IsChipOff(latencyState) )
      OALContextSave();
    OALWakeupLatency_PushState(latencyState);
    dword_61874 -= DelayInTicks;
    OALTimerSetCompare(dword_61874);
    g_pPrcmPrm->pOMAP_WKUP_PRM->PM_WKST_WKUP = 256;
    g_pPrcmPrm->pOMAP_WKUP_PRM->PM_WKEN_WKUP |= 0x100u;
    fnOALCPUIdle(g_pCPUInfo);
    g_pPrcmPrm->pOMAP_WKUP_PRM->PM_WKEN_WKUP &= ~0x100u;
    OALWakeupLatency_PopState();
    PM_WKST_WKUP = g_pPrcmPrm->pOMAP_WKUP_PRM->PM_WKST_WKUP;
    data = g_pPrcmPrm->pOMAP_CORE_PRM->PM_PREPWSTST_CORE;
    prevMpuState = g_pPrcmPrm->pOMAP_MPU_PRM->PM_PREPWSTST_MPU;
    prevPerState = g_pPrcmPrm->pOMAP_PER_PRM->PM_PREPWSTST_PER;
    OALContextRestore(prevMpuState, data, prevPerState);
    v11 = g_pTimerRegs->TCRR;
    PrcmProcessPostMpuWakeup();
```

IDA 7.6:

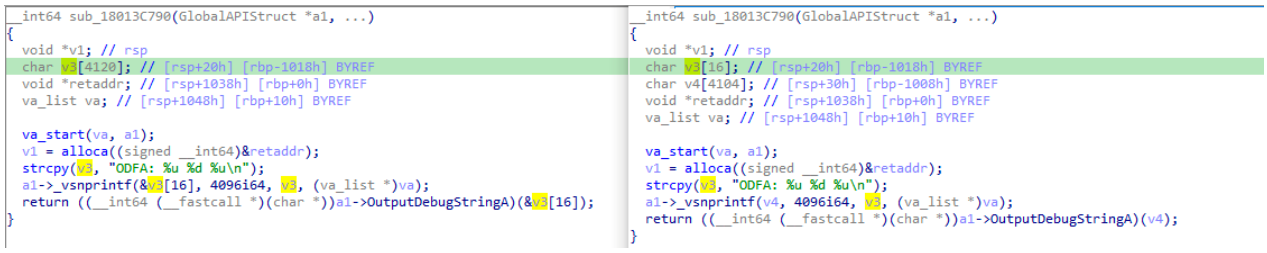


```
{
  switch ( v11 )
  {
    case 0x102u:
      v13 = GetOEMInfo(varg_r3, outSize, pOutSize, (const wchar_t **)&spiName);
      v8 = v13;
      break;
    case 0xE0u:
      v14 = GetPlatformVersion(varg_r3, outSize, pOutSize, (const wchar_t **)&sp
      v8 = v14;
      break;
    case 0x101u:
      v12 = GetPlatformType(varg_r3, outSize, pOutSize, (const wchar_t **)&spiNa
      v8 = v12;
      break;
    default:
LABEL_28:
      NKSetLastError(0x57u);
```

```
{
  switch ( v11 )
  {
    case 0x102u:
      OEMInfo = GetOEMInfo(varg_r3, outSize, pOutSize, (const wchar_t **)&spiName);
      v8 = OEMInfo;
      break;
    case 0xE0u:
      PlatformVersion = GetPlatformVersion(varg_r3, outSize, pOutSize, (const wchar_t **)&spi
      v8 = PlatformVersion;
      break;
    case 0x101u:
      PlatformType = GetPlatformType(varg_r3, outSize, pOutSize, (const wchar_t **)&spiName);
      v8 = PlatformType;
      break;
    default:
LABEL_28:
      NKSetLastError(0x57u);
```

- improved recognition of stack arrays

  Arrays on stack can be difficult to detect automatically since usually only their first elements are referenced explicitly. We have added heuristics which recover arrays in many typical situations, reducing the need for manual intervention.



```
__int64 sub_18013C790(GlobalAPIStruct *a1, ...)
{
  void *v1; // rsp
  char v3[4120]; // [rsp+20h] [rbp-1018h] BYREF
  void *retaddr; // [rsp+1038h] [rbp+0h] BYREF
  va_list va; // [rsp+1048h] [rbp+10h] BYREF

  va_start(va, a1);
  v1 = alloca((signed __int64)&retaddr);
  strcpy(v3, "ODFA: %u %d %u\n");
  a1->_vsnprintf(&v3[16], 4096i64, v3, (va_list *)va);
  return ((__int64 (__fastcall *)(char *))a1->OutputDebugStringA)(&v3[16]);
}
```

```
__int64 sub_18013C790(GlobalAPIStruct *a1, ...)
{
  void *v1; // rsp
  char v3[16]; // [rsp+20h] [rbp-1018h] BYREF
  char v4[4104]; // [rsp+30h] [rbp-1008h] BYREF
  void *retaddr; // [rsp+1038h] [rbp+0h] BYREF
  va_list va; // [rsp+1048h] [rbp+10h] BYREF

  va_start(va, a1);
  v1 = alloca((signed __int64)&retaddr);
  strcpy(v3, "ODFA: %u %d %u\n");
  a1->_vsnprintf(v4, 4096i64, v3, (va_list *)va);
  return ((__int64 (__fastcall *)(char *))a1->OutputDebugStringA)(v4);
}
```

- empty lines for better readability

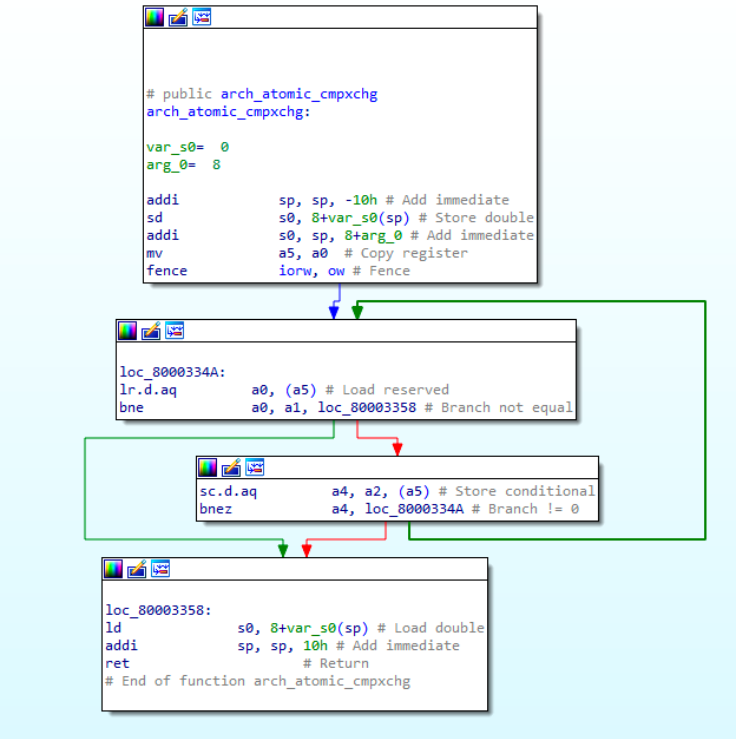
  If you add GENERATE_EMPTY_LINES = YES to hexrays.cfg, the decompiler will add extra empty lines between compound statements and before labels, which improves readability of long functions.

```
    if ( type == GUMBO_TOKEN_START_TAG )
    {
      v119 = a2->v.start_tag.tag;
      if ( v119 == GUMBO_TAG_MATH )
      {
        reconstruct_active_formatting_elements(a1);
        adjust_mathml_attributes(a1, a2);
        adjust_foreign_attributes(a1, a2);
        v120 = 2;
        goto LABEL_289;
      }

      if ( v119 == GUMBO_TAG_SVG )
      {
        reconstruct_active_formatting_elements(a1);
        adjust_svg_attributes(a1, a2);
        adjust_foreign_attributes(a1, a2);
        v120 = 1;

LABEL_289:
        insert_foreign_element(a1, a2, v120);
        if ( !a2->v.doc_type.force_quirks )
          return 1;

LABEL_290:
        pop_current_node(a1);
        a1->_parser_state->_self_closing_flag_acknowledged = 1;
        return 1;
      }
    }

    memset(&v131[16], 0, 80);
    v131[128] = 1;
    memset(&v131[129], 0, 22);
```

- **New processor modules: RISC-V and RL78**

  Our processor selection continues to expand steadily.

- RISC-V is an open ISA which is starting to become available in various hardware such as the latest iteration of the Espressif Systems wireless platform, ESP32-C3.



- RL78 from Renesas is a 16-bit descendant of the 8-bit NEC 78k0(s) family previously supported by IDA and is used in various automotive and consumer applications.

```
ROM:00D8     ; =============== S U B R O U T I N E =======================================
ROM:00D8
ROM:00D8
ROM:00D8     sub_D8:                                 ; CODE XREF: ROM:0EBD↓p
ROM:00D8
ROM:00D8     var_2           = -2
ROM:00D8
ROM:00D8 000                 PUSH    HL
ROM:00D9 002                 CLR1    PSW.IE          ; Program status word/Interrupt enable
ROM:00DC 002                 CALL    sub_EC3
ROM:00DF 002                 CALL    sub_265
ROM:00E2 002                 CLRW    AX
ROM:00E3 002                 PUSH    AX
ROM:00E4 004                 MOVW    AX, #9
ROM:00E7 004                 PUSH    AX
ROM:00E8 006                 MOV     [SP+6+var_2], #2
ROM:00EB 006                 MOVW    DE, #4002h
ROM:00EE 006                 MOVW    BC, #100h
ROM:00F1 006                 MOVW    AX, #0FE00h
ROM:00F4 006                 CALL    sub_163A
ROM:00F7 006                 ADDW    SP, #4
ROM:00F9 002                 CALL    sub_434
ROM:00FC 002                 MOV     A, #3
ROM:00FE 002                 CALL    sub_50B
ROM:0101 002                 SET1    PSW.IE          ; Program status word/Interrupt enable
ROM:0104 002                 CALL    sub_57C
ROM:0107 002                 OR      A, word_FFAF0+1
ROM:010A 002                 XCH     A, X
ROM:010B 002                 OR      A, word_FFAF0
ROM:010E 002                 XCH     A, X
ROM:010F 002                 MOVW    word_FFAF0, AX
ROM:0112 002                 OR      A, X
ROM:0114 002                 BNZ     loc_119
ROM:0116 002                 ONEB    A
ROM:0117 002                 MOV     [SP+2+var_2], A
ROM:0119
ROM:0119     loc_119:                                ; CODE XREF: sub_D8+3C↑j
ROM:0119 002                 MOV     A, [SP+2+var_2]
ROM:011B 002                 MOV     byte_FFAE0, A
ROM:011E
ROM:011E     loc_11E:                                ; CODE XREF: sub_D8+18A↓j
ROM:011E 002                 MOVW    BC, word_FFAEA
ROM:0121 002                 MOVW    AX, word_FFAEC
ROM:0124 002                 CMPW    AX, BC
```

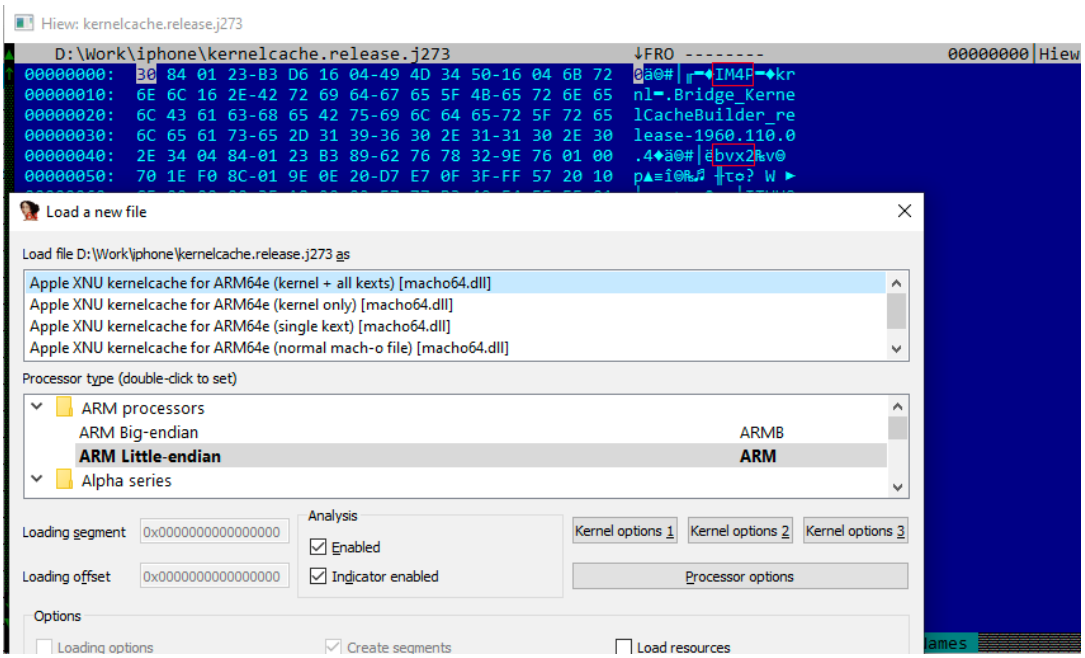
- **UI improvements**
  - processor list in the Load File dialog is now organized using folder view which can be filtered using Ctrl-F
  - you can now use cut & paste in folder views instead of dragging things with the mouse
  - a new, separate bookmarks view with the global list of bookmarks which can be grouped into folders
  - the lines that have a bookmark will now have an additional background color for visual feedback

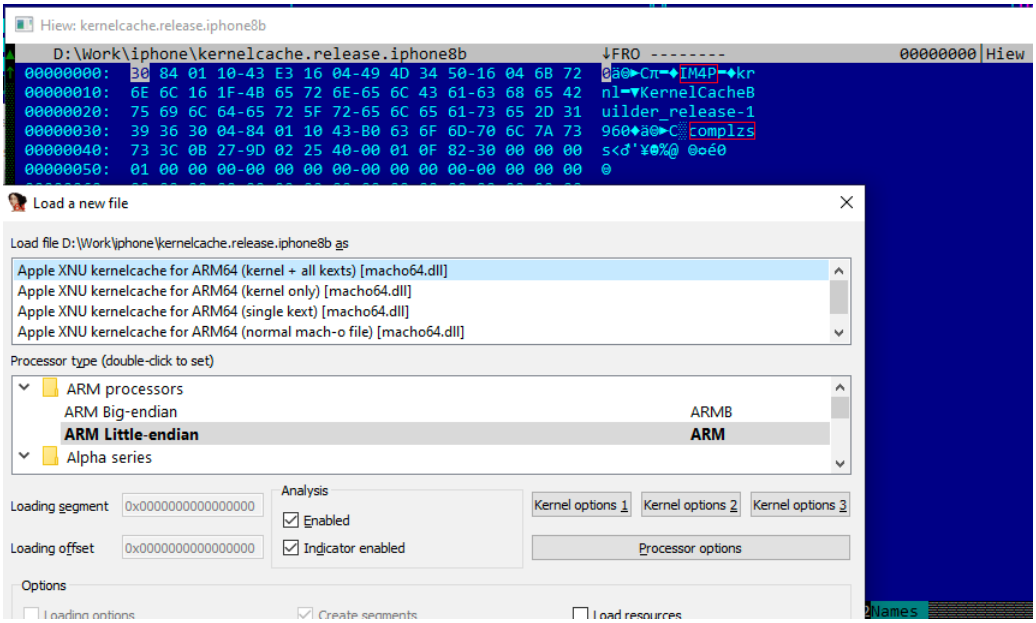
- **UI: Strings list is now cached in the database**

  The Strings window is one of the most commonly used views in IDA for quick reconnaissance. However, depending on the settings it can take a long to to scan the whole database which had to be repeated each time on reopening the window or reloading the database. Now we cache the list so opening it the second time is almost instant

- **Compressed macOS and iOS kernelcache support**

  In the recent iOS and macOS versions, the kernelcache files are compressed. Although there are tools available which can decompress them, it's one more thing to remember. Now IDA handles the standard compressed formats transparently so you can simply load them as standard Mach-O files. Since IDA can also handle ZIP files, you can open them directly from the IPSW updates!

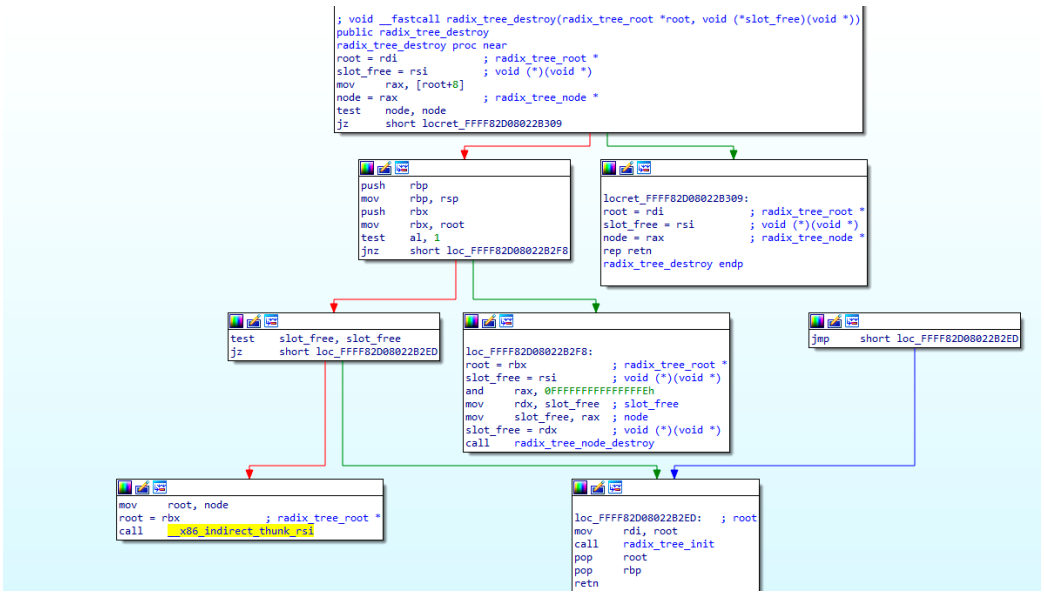




- **Retpoline handling**

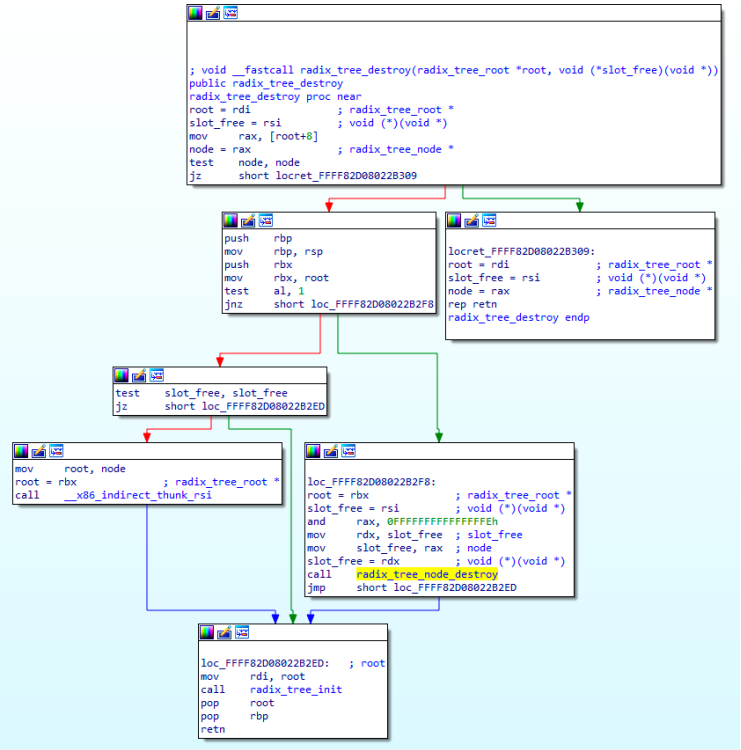
  Retpoline (return trampoline) is a compile-time mitigation against the Spectre speculative execution vulnerability disclosed in 2017. Binaries compiled with this option use special thunk functions for indirect jumps which tend to break standard control flow analysis. IDA now detects and handles these thunks transparently, resulting in nice and clean function graphs and pseudocode.

  In IDA 7.5:




```
void __fastcall radix_tree_destroy(radix_tree_root *root, void (*slot_free)(void *))
{
  radix_tree_node *v2; // rax

  v2 = root->rnode;
  if ( v2 )
  {
    if ( ((unsigned __int8)v2 & 1) != 0 )
      radix_tree_node_destroy(root, (radix_tree_node *)((unsigned __int64)v2 & 0xFFFFFFFFFFFFFFFELL), slot_free);
    if ( slot_free )
      __x86_indirect_thunk_rsi(v2);
    radix_tree_init(root);
  }
}
```

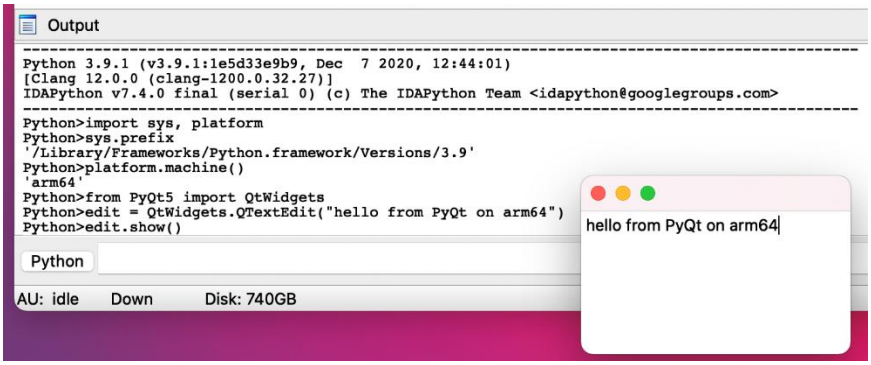
and in IDA 7.6:




```
void __fastcall radix_tree_destroy(radix_tree_root *root, void (*slot_free)(void *))
{
  radix_tree_node *rnode; // rax

  rnode = root->rnode;
  if ( rnode )
  {
    if ( ((unsigned __int8)rnode & 1) != 0 )
    {
      radix_tree_node_destroy(root, (radix_tree_node *)((unsigned __int64)rnode & 0xFFFFFFFFFFFFFFFELL), slot_free);
    }
    else if ( slot_free )
    {
      slot_free(rnode);
    }
    radix_tree_init(root);
  }
}
```

- **Python 3.9 support**

  Python 3.9 was released after IDA 7.5 and changed the layout of some internal structures leading to crashes in scripts or plugins using PyQt. IDA 7.6 adds official support for 3.9 (as well as still supporting previous 3.x versions and 2.7). Python 3.9.1 is also officially available for macOS on ARM64 and can be used by IDA there.



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

Hex-Rays

IDA PRO 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 iPhone SDKs.

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

IDA PRO 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/

IDA PRO Version 7.5 – Service Pack 2

Release date: 28th July 2020

Highlights: This release fixes some immediate issues with the new macOS11/iOS14 binaries and focuses principally on enhancing the static analysis for new file formats. MH_FILESET kernelcache format is fully supported, Objective-C metadata is improved and type libraries for MacOSX11.0.sdk and iPhoneOS14.0.sdk was added.

Full changelist: https://www.hex-rays.com/products/ida/news/7_5sp2/

IDA PRO Version 7.5 – Service Pack 3

Release date: 28th October 2020

Highlights: This service pack introduces a handful of new and interesting features specific to the soon-to-be-released macOS 11 (Big Sur) and provides fixes for numerous minor issues. macOS11 kernel debugging with VMware Fusion 12 and symbolication of MH_FILESET kernelcaches were both improved.

Full changelist: https://www.hex-rays.com/products/ida/news/7_5sp3/