HEX-RAYS DECOMPILER

The most advanced binary analysis tool for programmers in the software security, validation and verification fields

Waste of time is the most extravagant of all expenses.
Theophrastus

c. 372 - c.287 BC

Hex-Rays helps professionals take a closer look at programs

- Secure coding
- Malware analysis
- Software forensics
- Debugging software
- Optimizing software
- COTS software validation
- Lost source code recovery
- Software vulnerability research
- Patent infringement investigations
- OS documentation and verification

Hex-Rays is the first decompiler to handle real world apps!

Hex-Rays Decompiler

The Hex-Rays Decompiler converts executable programs into a human readable C-like pseudo code text. The pseudo code text is generated on the fly. Our technology is fast enough to analyze most functions within a few seconds.

Currently the decompiler supports compiler-generated code for the Intel x86/x64 processors and ARM. We have an SDK to allow our customers to implement their own analysis methods. Vulnerability research, software validation, coverage analysis are the areas that can benefit from such customizations.

The decompiler runs on MS Windows, Linux and Mac OS X. Both the GUI and text IDA versions are supported; however, in the text mode, only batch operation is available.

It comes with one year of free updates and online support.
Call us now for pricing information.

Facts about Hex-Rays Decompiler

- The decompiler supports compiler-generated Intel x86/x64 code
- It supports ARM code, including Thumb
- It can handle code generated by any mainstream C/C++ compiler
- It is very fast. Most functions are analyzed instantaneously
- Floating point instructions are supported
- It has interactive and batch modes
- Can debug with pseudocode
- It is an IDA plugin

In comparison to low level assembly language, high level language representation in Hex-Rays has several advantages:

- Concise: Requires less time to read it
- Structured: Program logic is more obvious
- Dynamic: Variable names and types can be changed on the fly
- Familiar: No need to be an expert ASM programmer
- Practical: Handles real world applications

Bundle discount when purchased with a new IDA license

How much is your time worth?

(877) 943-2776 **www.ccso.com** **ida9@ccso.com**

1 **Hexadecimal code: incomprehensible for humans**

```
00010C70  5E C2 04 00 53 56 57 FF  15 34 24 01 00 8B D8 33
00010C80  FF BE 8C 26 01 00 56 E8  B6 F7 FF FF 50 8D 04 1F
00010C90  50 56 FF 15 40 24 01 00  83 C4 0C 85 C0 74 0F 47
00010CA0  81 FF 00 30 00 00 7C DE  33 C0 5F 5E 5B C3 8B C7
```

2 **Disassembler output: makes sense but lengthy**

```
00010C74 check_for_system_process proc near      ; CODE XREF: start+A0↓p
00010C74                push    ebx
00010C75                push    esi
00010C76                push    edi
00010C77                call    ds:IoGetCurrentProcess
00010C7D                mov     ebx, eax
00010C7F                xor     edi, edi
00010C81                mov     esi, offset aSystem ; "System"
00010C86
00010C86 loc_10C86:                              ; CODE XREF: check_for_system_process+32↓j
00010C86                push    esi
00010C87                call    strlength
00010C8C                push    eax              ; size_t
00010C8D                lea     eax, [edi+ebx]
00010C90                push    eax              ; char *
00010C91                push    esi              ; char *
00010C92                call    ds:strncmp
00010C98                add     esp, 0Ch
00010C9B                test    eax, eax
00010C9D                jz      short loc_10CAE
00010C9F                inc     edi
00010CA0                cmp     edi, 12288
00010CA6                jl      short loc_10C86
00010CA8                xor     eax, eax
00010CAA
00010CAA @@exitsub:                              ; CODE XREF: check_for_system_process+3C↓j
00010CAA                pop     edi
00010CAB                pop     esi
00010CAC                pop     ebx
00010CAD                retn
```

3 **Decompiler output: concise and familiar to programmers**

```
signed int __cdecl check_for_system_process()
{
  PEPROCESS peprocess; // ebx@1
  signed int count; // edi@1
  size_t slen; // eax@2

  peprocess = IoGetCurrentProcess();
  count = 0;
  while ( 1 )
  {
    slen = strlength("System");
    if ( !strncmp("System", (const char *)peprocess + count, slen) )
      break;
    ++count;
    if ( count >= 12288 )
      return 0;
  }
  return count;
}
```