[翻译]Ghidra简介

转载

Hu4 于 2019-03-15 13:41:11 发布 3914 收藏 7
原文链接：Ghidra: A quick overview for the curious

转自看雪大佬：crownless

Ghidra是由美国国家安全局（NSA）研究部门开发的软件逆向工程（SRE）套件，用于支持网络安全任务。它最近被公开出来，我对此感到好奇，想看看它是什么样的。

我还没有查过是否有其他人已经发过了类似的概述文章，不过，我还是决定写这篇文章，为了自己和那些不想自行运行Ghidra而只是想简单了解一下的人。

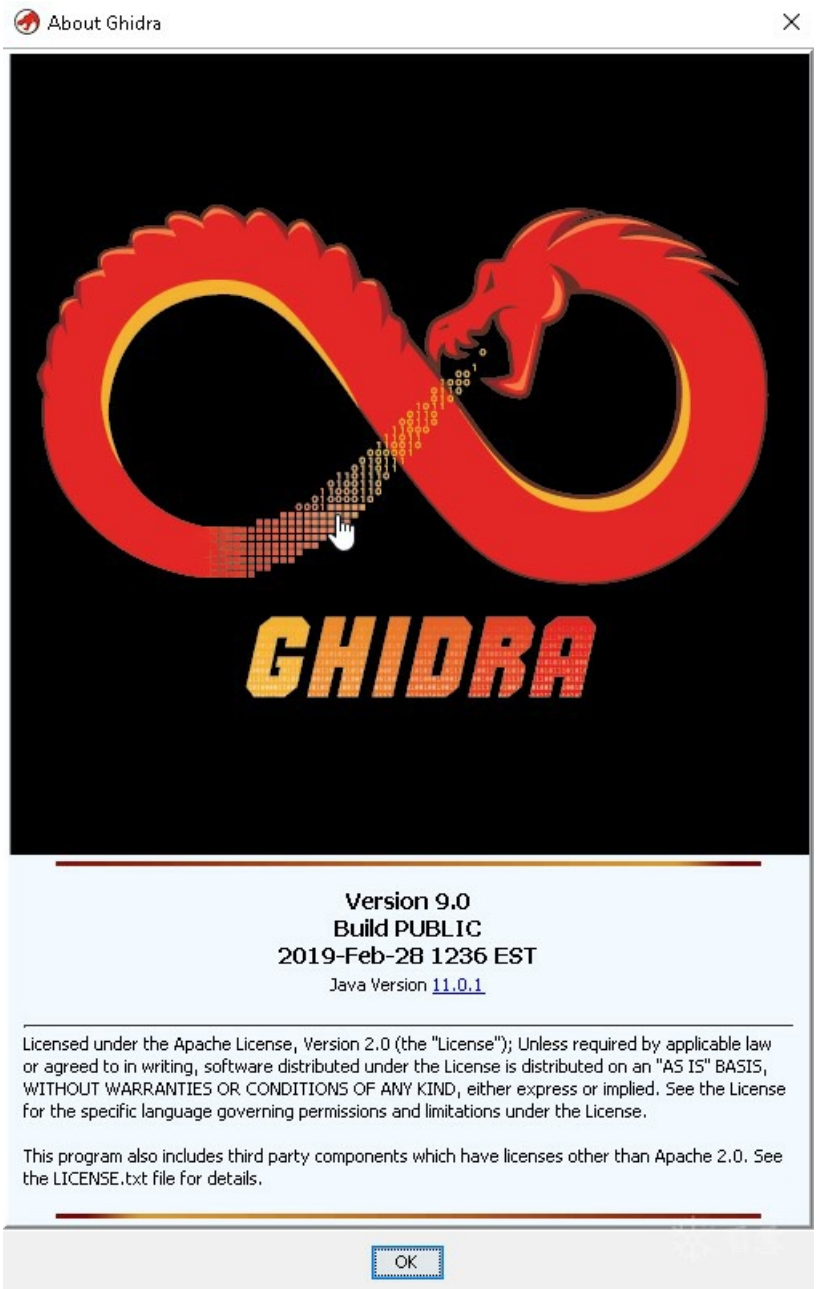
我知道将Ghidra与IDA Pro进行比较是不公平的，但我还是忍不住要比较，因为我是IDA Pro的长期用户，而且它是我在逆向工程工具方面的唯一参照物。

这篇文章很长，将包含很多截图。我刚刚开始上手Ghidra，因此，我可能是错的，或者可能会提供不准确或不完整的信息，所以请原谅我。

目录

总体概述

什么是Ghidra



> Ghidra是一个软件逆向工程（SRE）框架，包括一套功能齐全的高端软件分析工具，使用户能够在各种平台上分析编译后的代码，包括Windows、Mac OS和Linux。功能包括反汇编，汇编，反编译，绘图和脚本，以及数百个其他功能。Ghidra支持各种处理器指令集和可执行格式，可以在用户交互模式和自动模式下运行。用户还可以使用公开的API开发自己的Ghidra插件和脚本。

文件结构概述

我在解压缩的Ghidra安装档案中运行了tree命令。这是输出：

```
├──Configurations
│   └──Public_Release
│       ├──data
│       └──lib
├──Extensions
├──Features
```

```
|   ├──Base
|   |   ├──data
|   |   |   ├──formats
|   |   |   ├──parserprofiles
|   |   |   ├──stringngrams
|   |   |   ├──symbols
|   |   |   |   ├──win32
|   |   |   |   └──win64
|   |   |   └──typeinfo
|   |   |       ├──generic
|   |   |       ├──mac_10.9
|   |   |       └──win32
|   |   |           └──msvcrt
|   |   ├──ghidra_scripts
|   |   └──lib
|   ├──BytePatterns
|   |   ├──data
|   |   |   └──test
|   |   ├──ghidra_scripts
|   |   └──lib
|   ├──ByteViewer
|   |   ├──data
|   |   └──lib
|   ├──DebugUtils
|   |   └──lib
|   ├──Decompiler
|   |   ├──ghidra_scripts
|   |   ├──lib
|   |   └──os
|   |       ├──linux64
|   |       ├──osx64
|   |       └──win64
|   ├──DecompilerDependent
|   |   ├──data
|   |   └──lib
|   ├──FileFormats
|   |   ├──data
|   |   |   ├──android
|   |   |   ├──crypto
|   |   |   └──languages
|   |   |       └──Dalvik
|   |   ├──ghidra_scripts
|   |   └──lib
|   ├──FunctionGraph
|   |   ├──data
|   |   └──lib
|   ├──FunctionGraphDecompilerExtension
|   |   └──lib
```

```
|   ├──FunctionID
|   |   ├──data
|   |   ├──ghidra_scripts
|   |   └──lib
|   ├──GhidraServer
|   |   ├──data
|   |   |   └──yajsw-stable-12.12
|   |   |       ├──doc
|   |   |       ├──lib
|   |   |       |   ├──core
|   |   |       |   |   ├──commons
|   |   |       |   |   ├──jna
|   |   |       |   |   ├──netty
|   |   |       |   |   └──yajsw
|   |   |       |   └──extended
|   |   |       |       ├──abeille
|   |   |       |       ├──commons
|   |   |       |       ├──cron
|   |   |       |       ├──glazedlists
|   |   |       |       ├──groovy
|   |   |       |       ├──jgoodies
|   |   |       |       ├──keystore
|   |   |       |       ├──regex
|   |   |       |       ├──velocity
|   |   |       |       ├──vfs-dbx
|   |   |       |       ├──vfs-webdav
|   |   |       |       └──yajsw
|   |   |       └──templates
|   |   ├──lib
|   |   └──os
|   |       ├──linux64
|   |       ├──win32
|   |       └──win64
|   ├──GnuDemangler
|   |   ├──ghidra_scripts
|   |   └──lib
|   ├──GraphFunctionCalls
|   |   └──lib
|   ├──MicrosoftCodeAnalyzer
|   |   └──lib
|   ├──MicrosoftDemangler
|   |   └──lib
|   ├──MicrosoftDmang
|   |   └──lib
|   ├──PDB
|   |   ├──lib
|   |   ├──os
|   |   |   └──win64
```

```
|   |   └──src
|   |       └──pdb
|   |           ├──cpp
|   |           └──headers
|   ├──ProgramDiff
|   |   └──lib
|   ├──Python
|   |   ├──data
|   |   |   └──jython-2.7.1
|   |   ├──ghidra_scripts
|   |   └──lib
|   ├──Recognizers
|   |   └──lib
|   ├──SourceCodeLookup
|   |   └──lib
|   └──VersionTracking
|       ├──data
|       ├──ghidra_scripts
|       └──lib
├──Framework
|   ├──DB
|   |   └──lib
|   ├──Demangler
|   |   └──lib
|   ├──Docking
|   |   ├──data
|   |   └──lib
|   ├──FileSystem
|   |   └──lib
|   ├──Generic
|   |   ├──data
|   |   └──lib
|   ├──Graph
|   |   └──lib
|   ├──Help
|   |   └──lib
|   ├──Project
|   |   ├──data
|   |   └──lib
|   ├──SoftwareModeling
|   |   ├──data
|   |   |   └──languages
|   |   └──lib
|   └──Utility
|       └──lib
├──Processors
|   ├──6502
|   |   └──data
```

```
|   |       └──languages
|   ├──68000
|   |   ├──data
|   |   |   ├──languages
|   |   |   └──manuals
|   |   └──lib
|   ├──6805
|   |   └──data
|   |       └──languages
|   ├──8051
|   |   ├──data
|   |   |   ├──languages
|   |   |   |   └──old
|   |   |   └──manuals
|   |   └──ghidra_scripts
|   ├──8085
|   |   └──data
|   |       └──languages
|   ├──AARCH64
|   |   ├──data
|   |   |   ├──languages
|   |   |   └──patterns
|   |   └──lib
|   ├──ARM
|   |   ├──data
|   |   |   ├──languages
|   |   |   |   └──old
|   |   |   ├──manuals
|   |   |   └──patterns
|   |   └──lib
|   ├──Atmel
|   |   ├──data
|   |   |   ├──languages
|   |   |   └──manuals
|   |   └──lib
|   ├──CR16
|   |   └──data
|   |       ├──languages
|   |       └──manuals
|   ├──DATA
|   |   ├──data
|   |   |   └──languages
|   |   ├──ghidra_scripts
|   |   └──lib
|   ├──JVM
|   |   ├──data
|   |   |   ├──languages
|   |   |   └──manuals
```

```
202 |   |   └──lib
203 |   ├──MIPS
204 |   |   ├──data
205 |   |   |   ├──languages
206 |   |   |   ├──manuals
207 |   |   |   └──patterns
208 |   |   └──lib
209 |   ├──PA-RISC
210 |   |   └──data
211 |   |       ├──languages
212 |   |       ├──manuals
213 |   |       └──patterns
214 |   ├──PIC
215 |   |   ├──data
216 |   |   |   ├──languages
217 |   |   |   └──manuals
218 |   |   ├──ghidra_scripts
219 |   |   └──lib
220 |   ├──PowerPC
221 |   |   ├──data
222 |   |   |   ├──languages
223 |   |   |   |   └──old
224 |   |   |   ├──manuals
225 |   |   |   └──patterns
226 |   |   └──lib
227 |   ├──Sparc
228 |   |   ├──data
229 |   |   |   ├──languages
230 |   |   |   ├──manuals
231 |   |   |   └──patterns
232 |   |   └──lib
233 |   ├──TI_MSP430
234 |   |   └──data
235 |   |       ├──languages
236 |   |       └──manuals
237 |   ├──Toy
238 |   |   ├──data
239 |   |   |   └──languages
240 |   |   |       └──old
241 |   |   |           └──v01stuff
242 |   |   └──lib
243 |   ├──x86
244 |   |   ├──data
245 |   |   |   ├──languages
246 |   |   |   |   └──old
247 |   |   |   ├──manuals
248 |   |   |   └──patterns
249 |   |   └──lib
250 |   
251 |   
```

```
252 |   └──Z80
253 |       └──data
    |           ├──languages
    |           └──manuals
    └──Test
        └──IntegrationTest
            └──lib
```

可以看出这个项目非常有条理。深入挖掘，我注意到Ghidra已经包含了许多组件的源代码：

- 如果你搜索`* -src.zip`，会出现很多源代码文件。
- PDB插件源代码
- 源代码形式的200多个Java脚本
- 等等

我提到了源代码，因为在撰写本文时，Ghidra的GitHub仓库仍然不包含源代码，它的内容如下：

> 此仓库是完整开源版本的预备发布区。请放心，我们正在努力使这里的软件可用。在此期间，您可以在您的软件逆向工程（SRE）工作中使用Ghidra，开发自己的脚本和插件，并仔细阅读首次公开发行版本中发布的超过一百万行的Java和Sleigh代码。该版本可以从我们的项目主页下载。请考虑查看我们的贡献者指南，了解如何在项目可用时参与此开源项目。

处理器模块

在撰写本文时，Ghidra支持以下处理器模块：

- 6502
- 68000
- 6805
- 8051
- 8085
- AARCH64
- ARM
- Atmel
- CR16
- DATA
- JVM
- MIPS
- PA-RISC
- PIC
- PowerPC
- Sparc
- TI_MSP430
- Toy
- x86
- Z80

它们位于C:\ghidra_9.0\Ghidra\Processors。
处理器模块似乎是数据驱动的。它们的一些插件/扩展方面是用Java编写和实现的。例如，您可以在此处找到x86模块的一些源代码组件：C:\ghidra_9.0\Ghidra\Processors\x86\lib\x86-src.zip。
处理器模块的可编程部分包含“重定位解码器”、“文件格式解码器”、“分析插件”等。

```
├──app
│ ├──plugin
│ │ └──core
│ │ └──analysis
│ └──util
│ └──bin
│ └──format
│ ├──coff
│ │ └──relocation
│ └──elf
│ ├──extend
│ └──relocation
└──feature
└──fid
└──hash
```

有趣的是，处理器模块参考了外部工具（即IDA Pro）中相应的处理器模块：

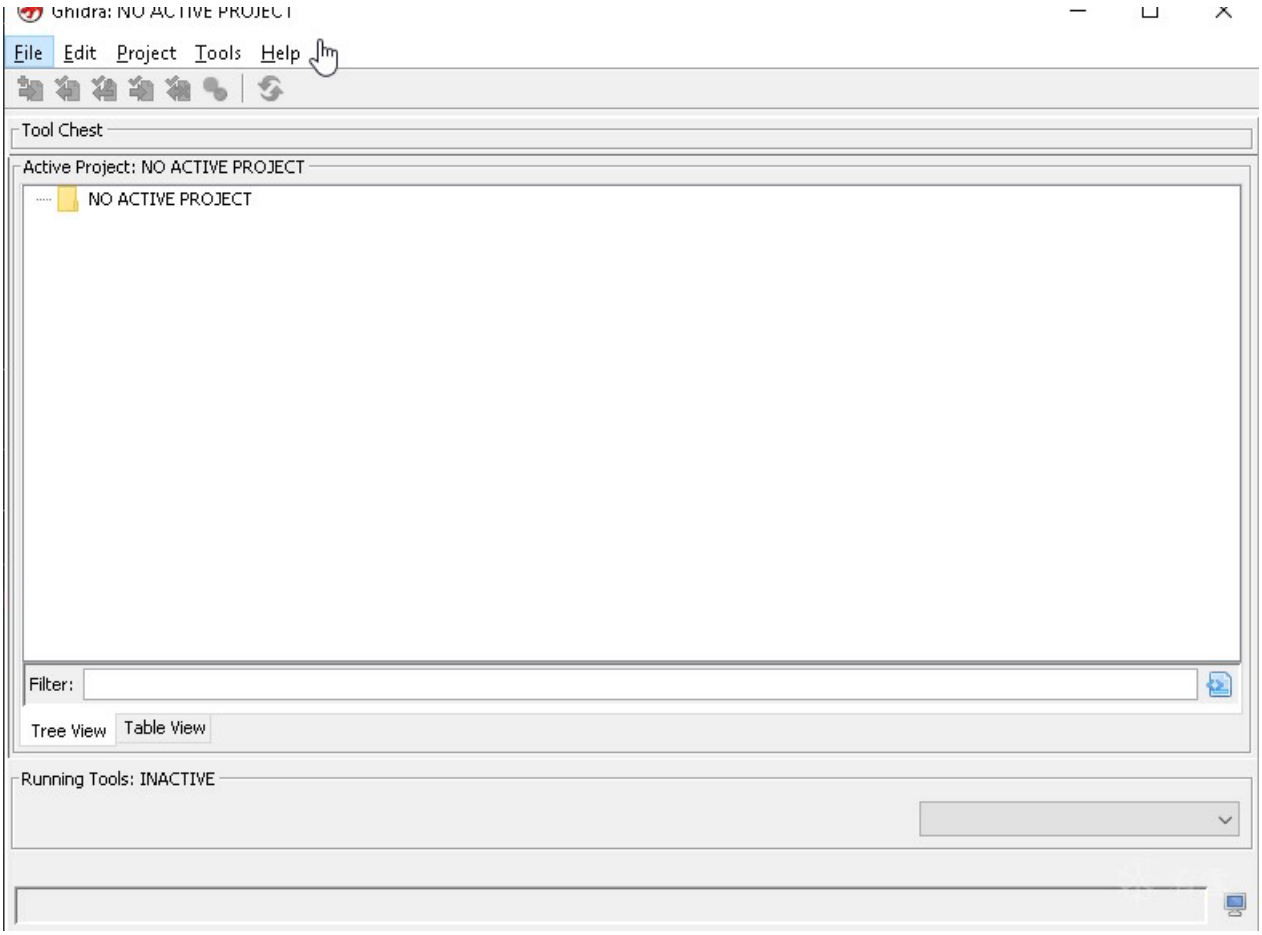
```
<language_definitions>

  <language processor="6502"
            endian="little"
            size="16"
            variant="default"
            version="1.0"
            slafile="6502.sla"
            processorspec="6502.pspec"
            id="6502:LE:16:default">
    <description>6502 Microcontroller Family</description>
    <compiler name="default" spec="6502.cspec" id="default"/>
    <external_name tool="IDA-PRO" name="m6502"/>
  <external_name tool="IDA-PRO" name="m65c02"/>
  </language>
```

Ghidra的功能

Ghidra功能齐全。它包括强大的代码浏览器，图形查看器，反编译器，数百个脚本，各种搜索工具，撤消/重做支持，协同工作服务器，程序差异比较工具等。由于Ghidra是巨大的，我无法涵盖每一个功能。那么，我将关注那些经验丰富的逆向工程师所认为的基础功能，也就是最重要和最有用的功能。

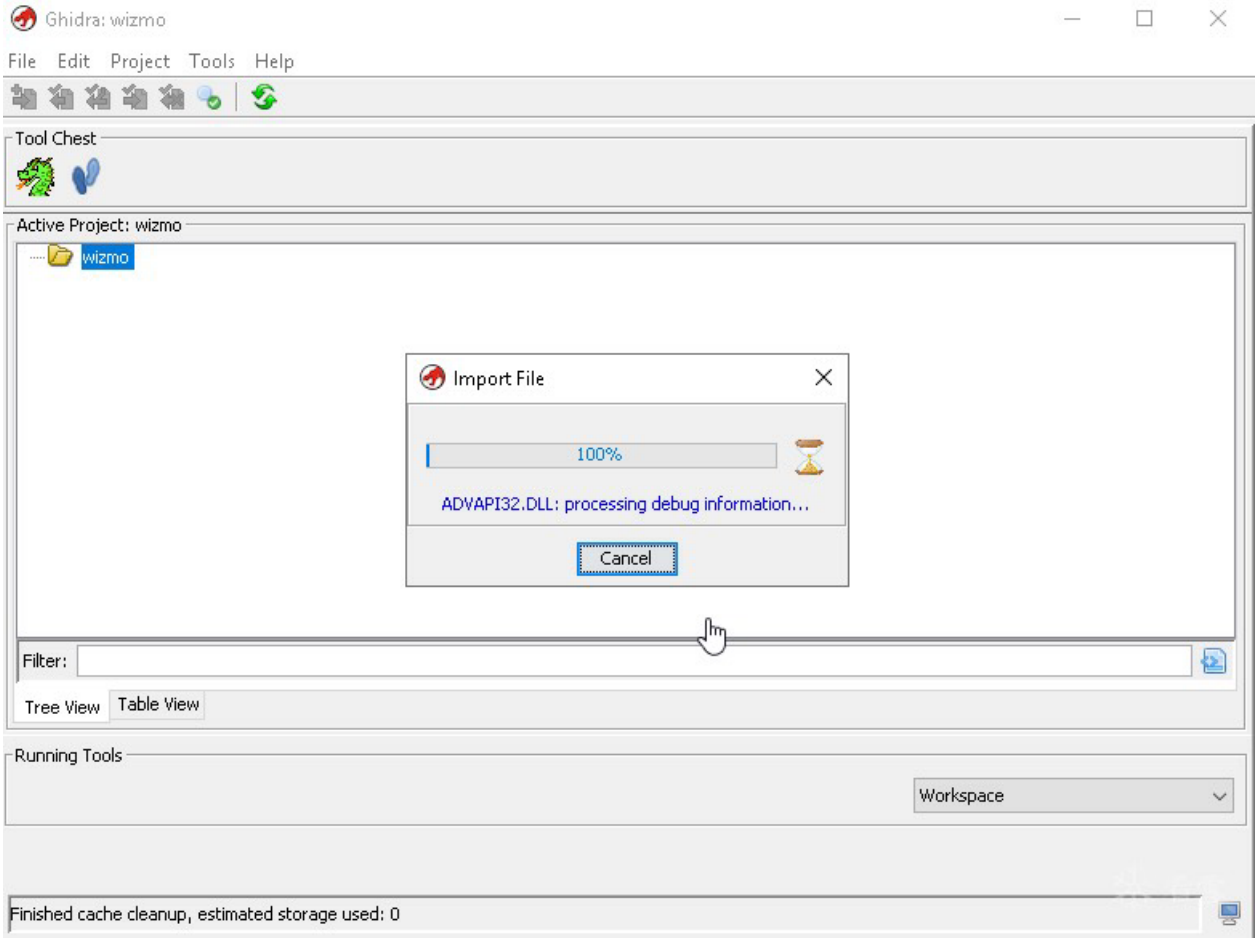
项目管理

在Ghidra中，任何东西都是一个项目。与IDA不同，您不会使用输入文件开始逆向工程会话，而是从创建项目开始。在第一次运行时没有项目，您将看到此对话框：



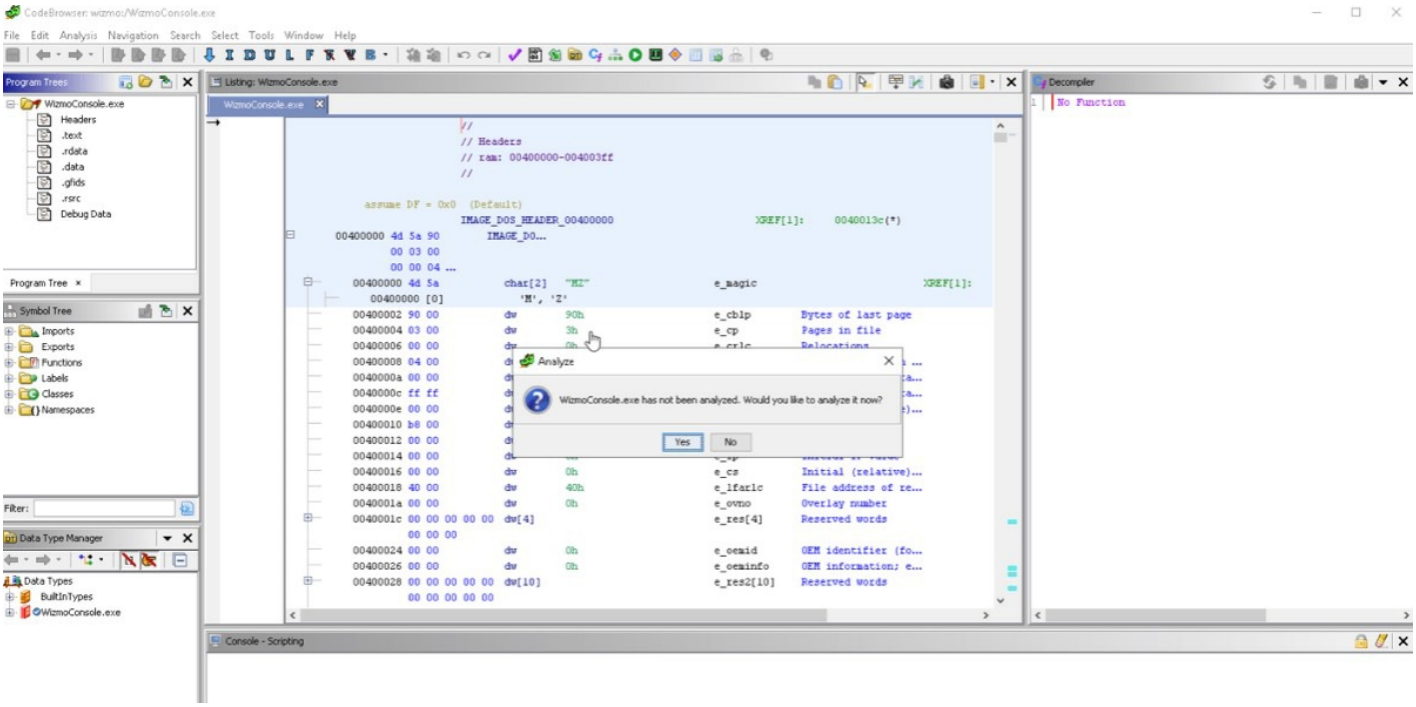
在本文中，我将对我的开源的Wizmo工具（可在此处找到）进行逆向工程。如果你想跟着我一起使用Ghidra，请下载二进制文件。

首先创建一个名为“Wizmo”的项目，然后导入“WizmoConsole.exe”程序：

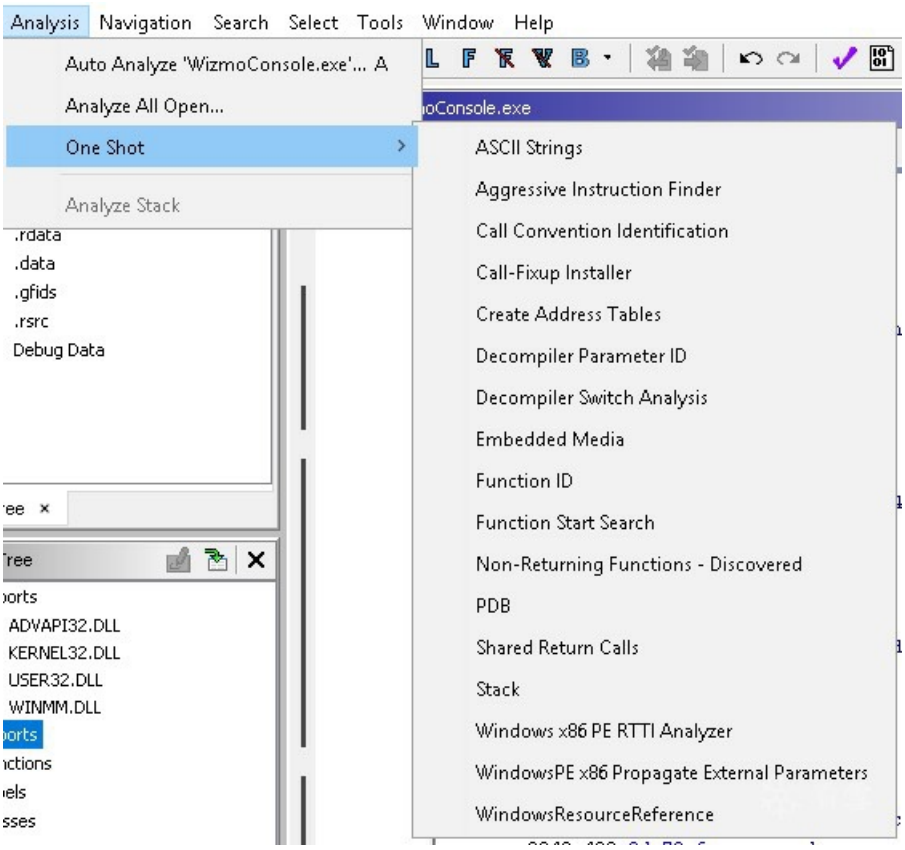


导入文件后，将显示导入结果摘要对话框：

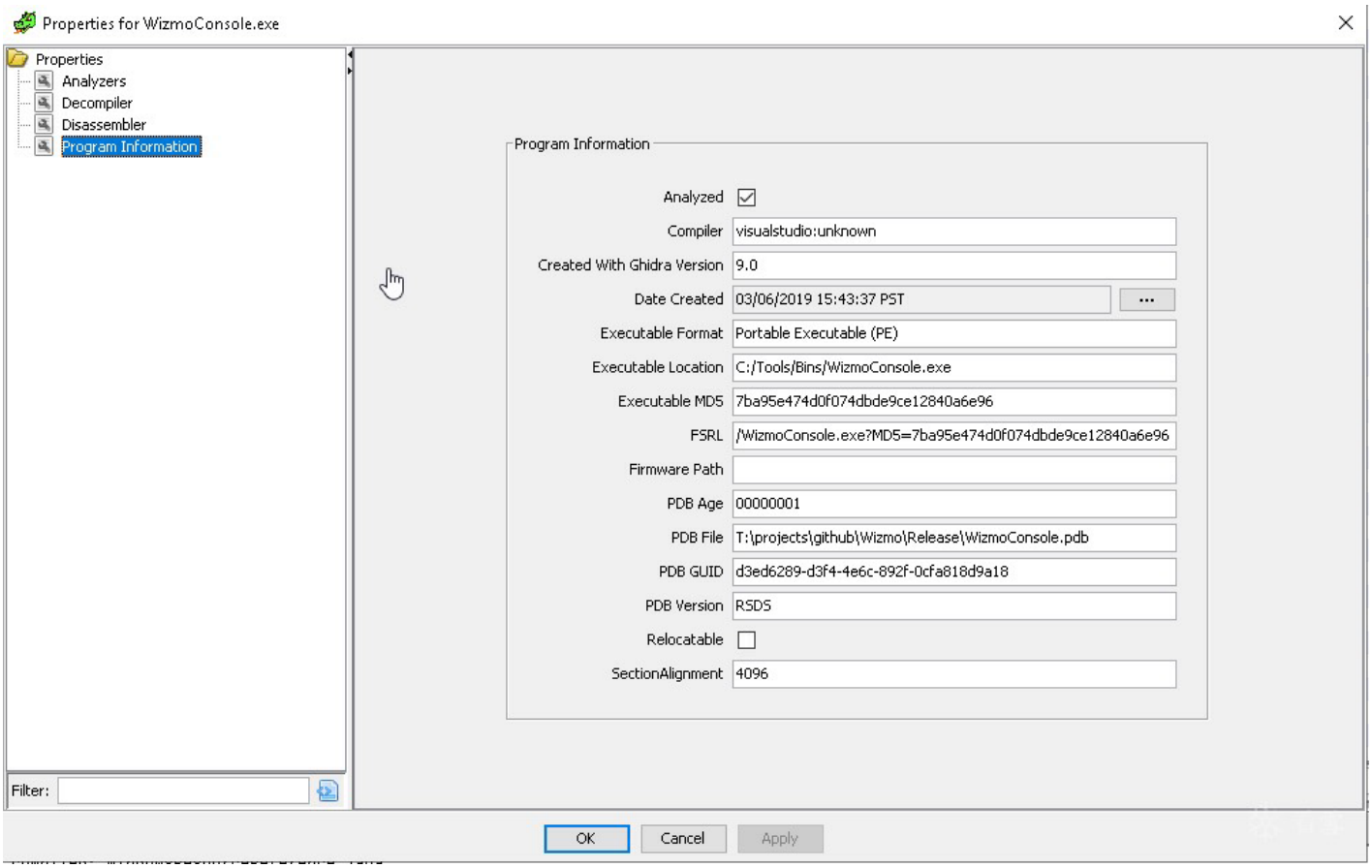
按“确定”后，您将看到代码浏览器窗口，并将被询问是否要开始分析该文件：



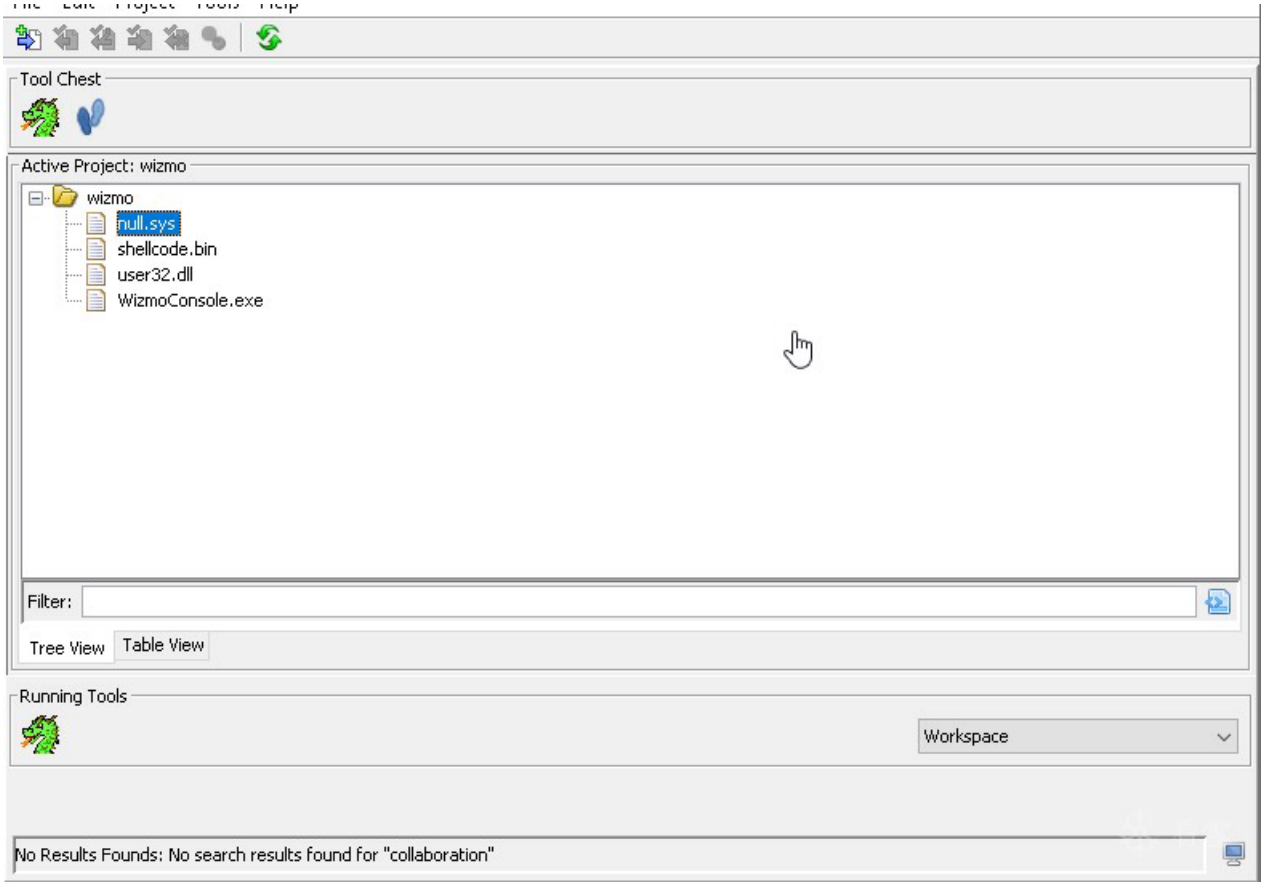
您可以随后从“分析”菜单中分析或重新分析该文件：



您还可以检查导入文件的属性：



您可以根据需要导入任意数量的文件。通常，导入项目的文件之间应该具有逻辑关系。例如EXE主程序及其DLL。

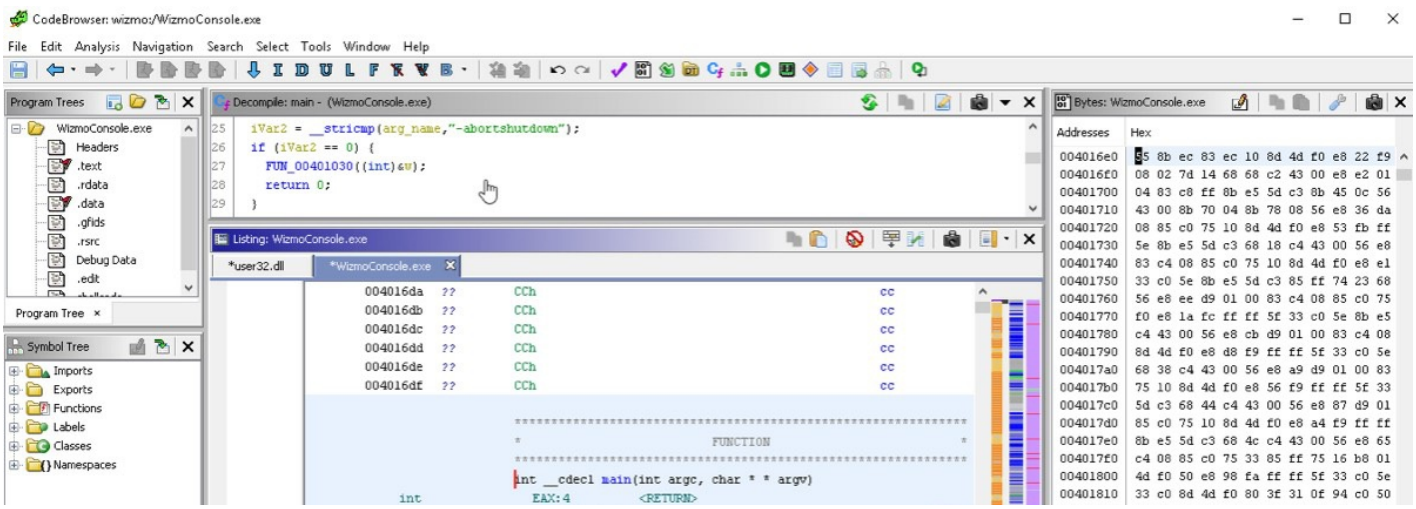


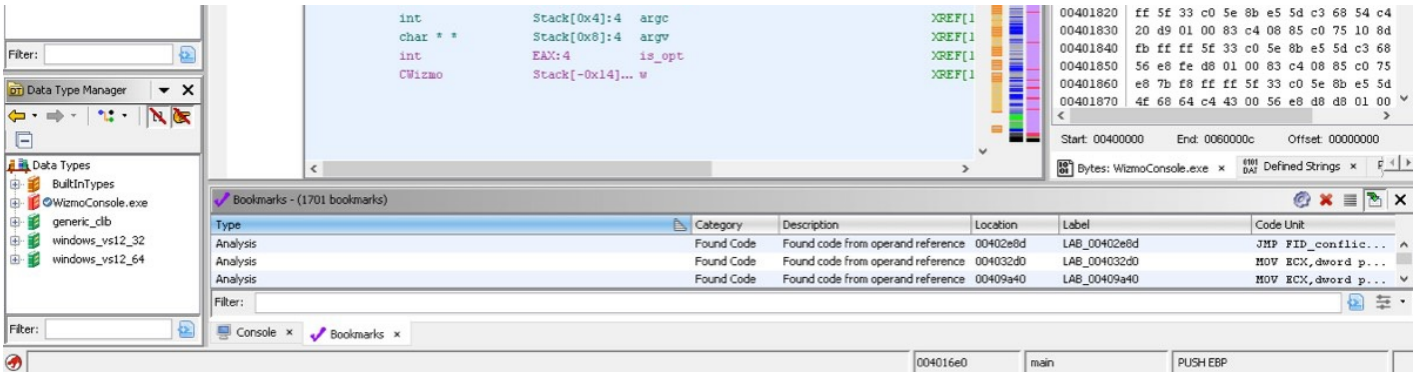
在上面的示例中，我导入了不相关的文件。 稍后，我们还将了解到可以通过编辑外部函数路径来创建从一个导入文件到另一个导入文件的链接。 例如，`WizmoConsole.exe`从`user32.dll`导入，因此我们可以链接WizmoConsole中导入的函数直接跳转到`user32.dll`。 这个功能真正构成了一个项目。 IDA Pro尚不支持项目概念。

代码浏览器

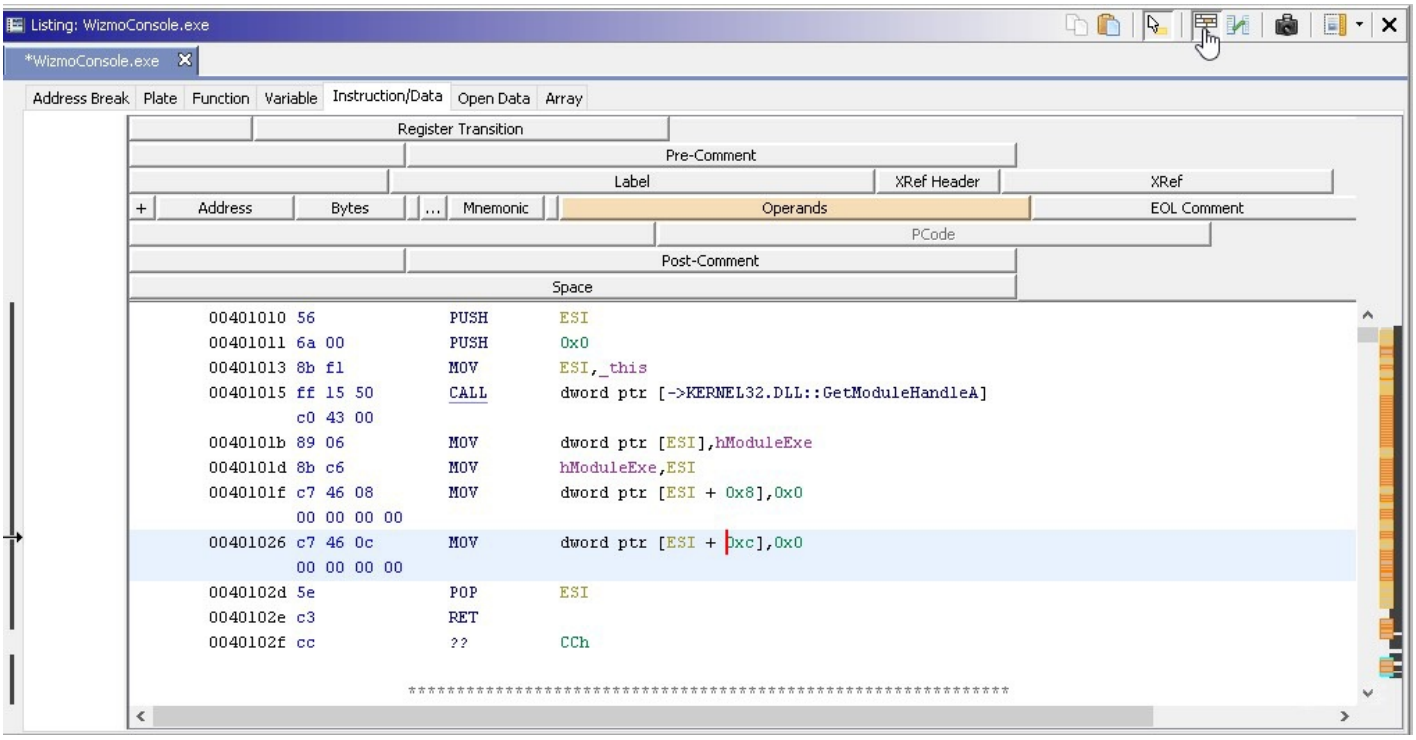
代码浏览器可以与IDA的主界面进行比较。 代码浏览器中有着Ghidra的所有可视元素：

- 主菜单
- 反汇编视图
- 符号树
- 程序树
- 字符串视图
- 数据类型管理器
- 反编译器视图
- 等等

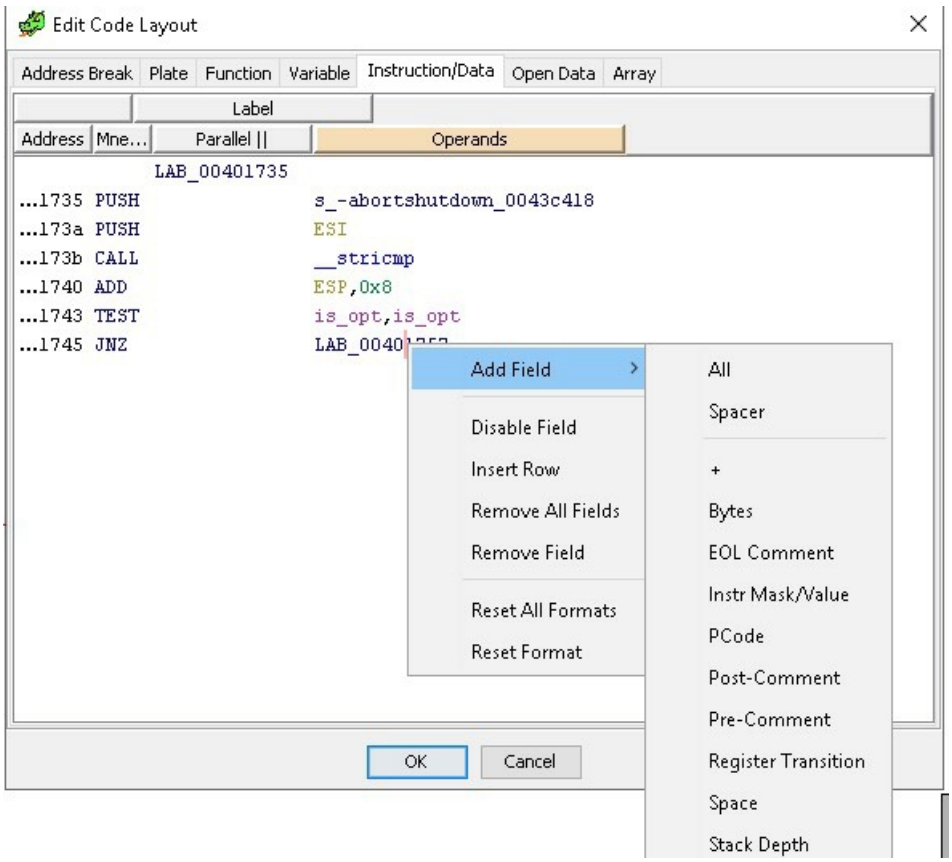


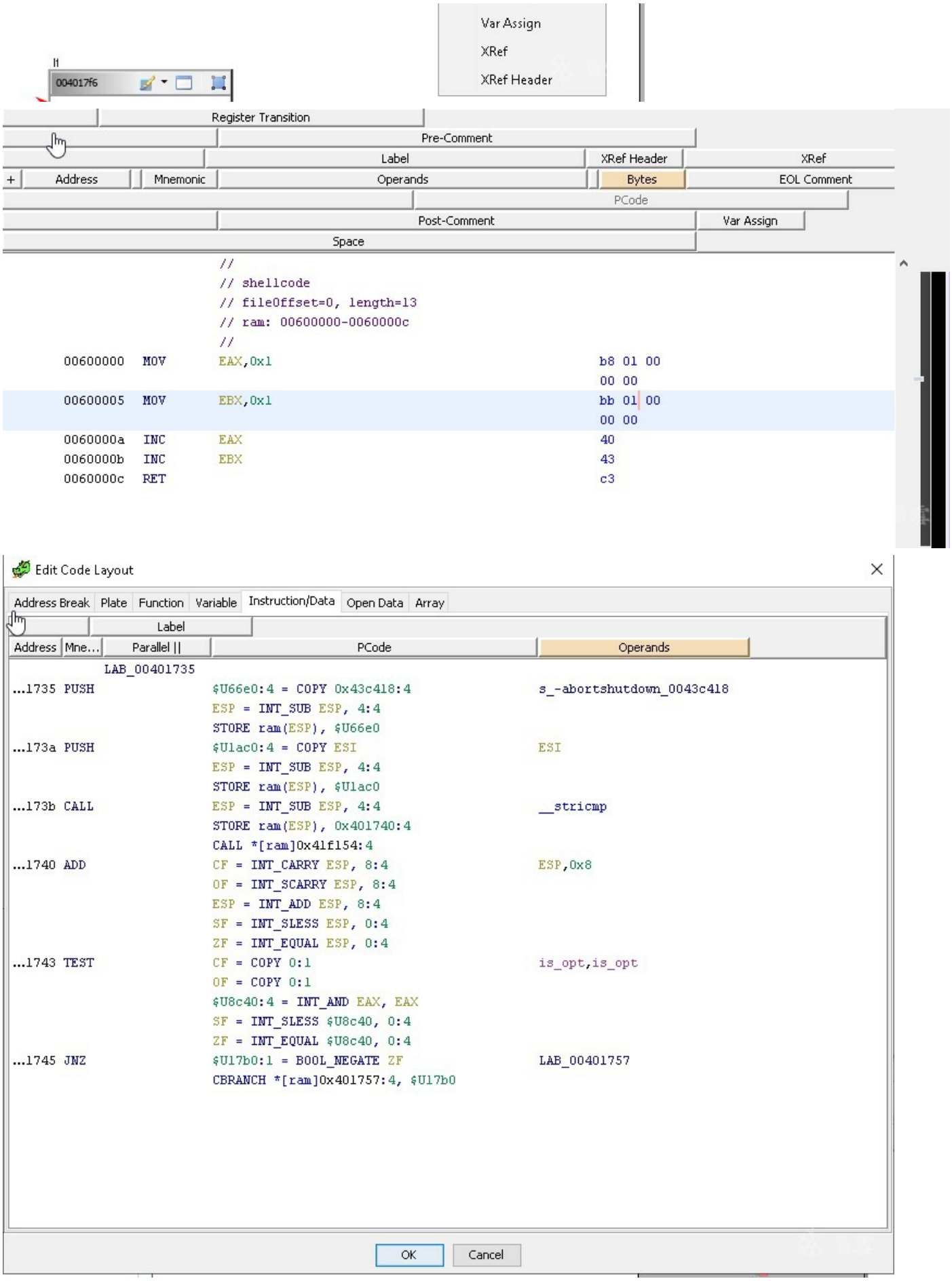


程序反汇编列表是高度可定制的。 只需按下“编辑列表字段”按钮（如光标所示）即可查看所有自定义选项：



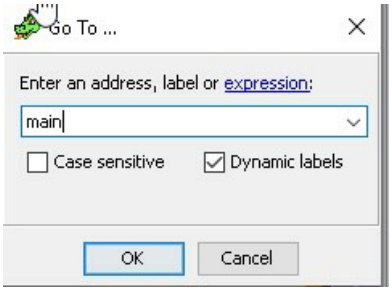
单击并拖动字段以重新排列反汇编列表（disasm视图）窗口中的可视元素。 这种高级可视化定制在IDA Pro中也不具备。



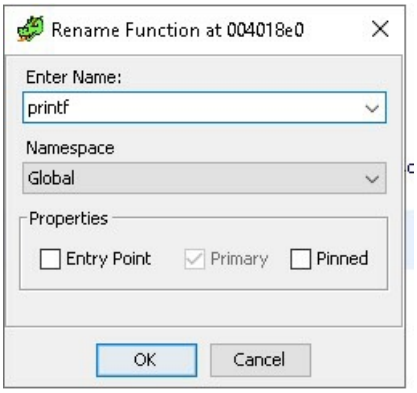


代码浏览器还可以显示其他辅助信息，例如程序概述和熵：

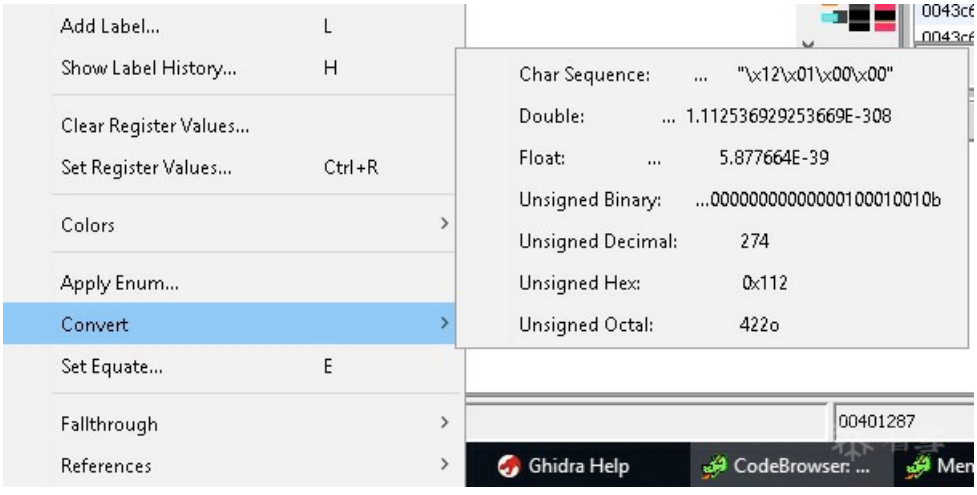
在代码浏览器反汇编列表中，您可以按“G”跳转到地址或标签：



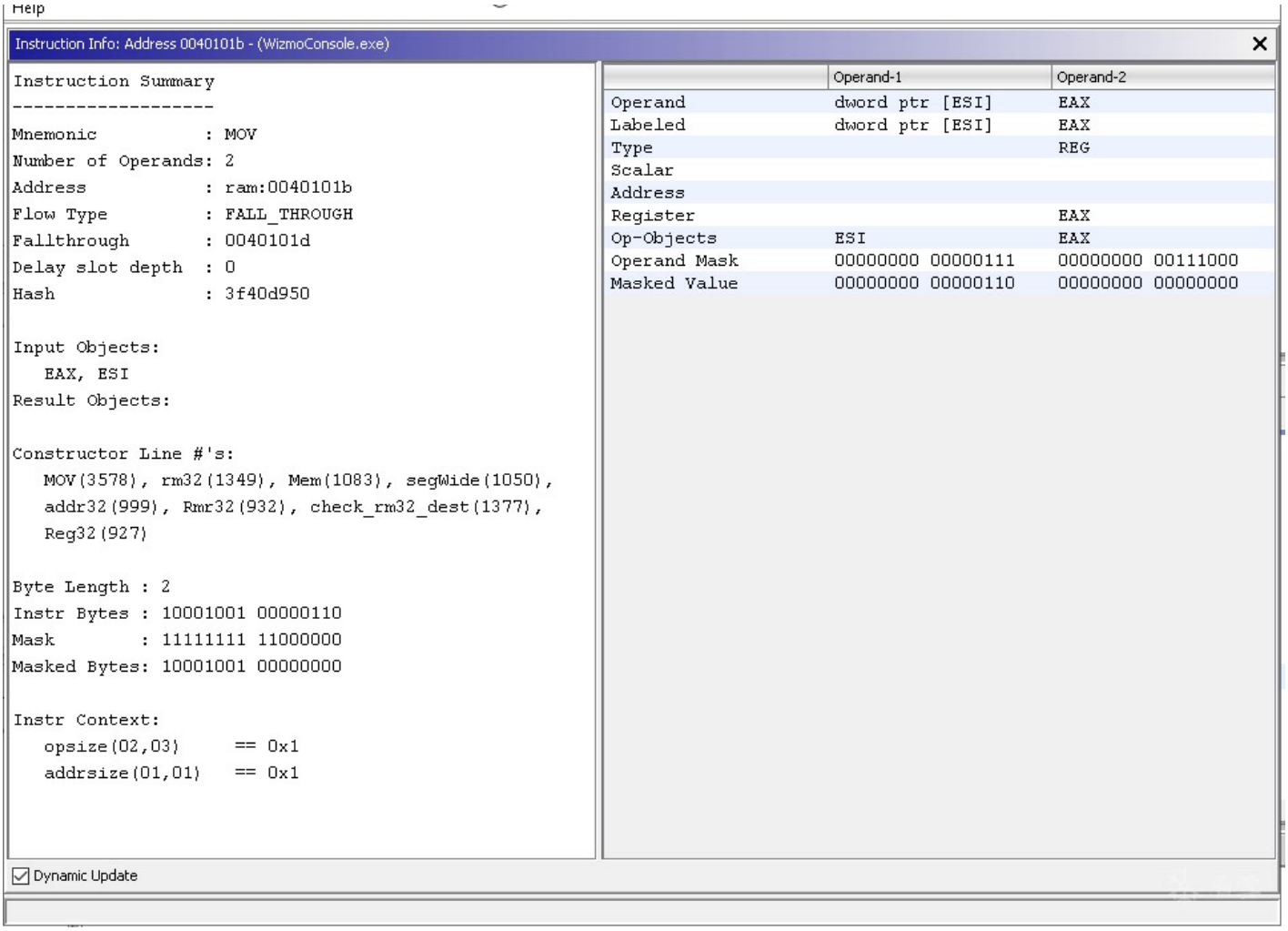
或者只是重命名一个函数或标签：



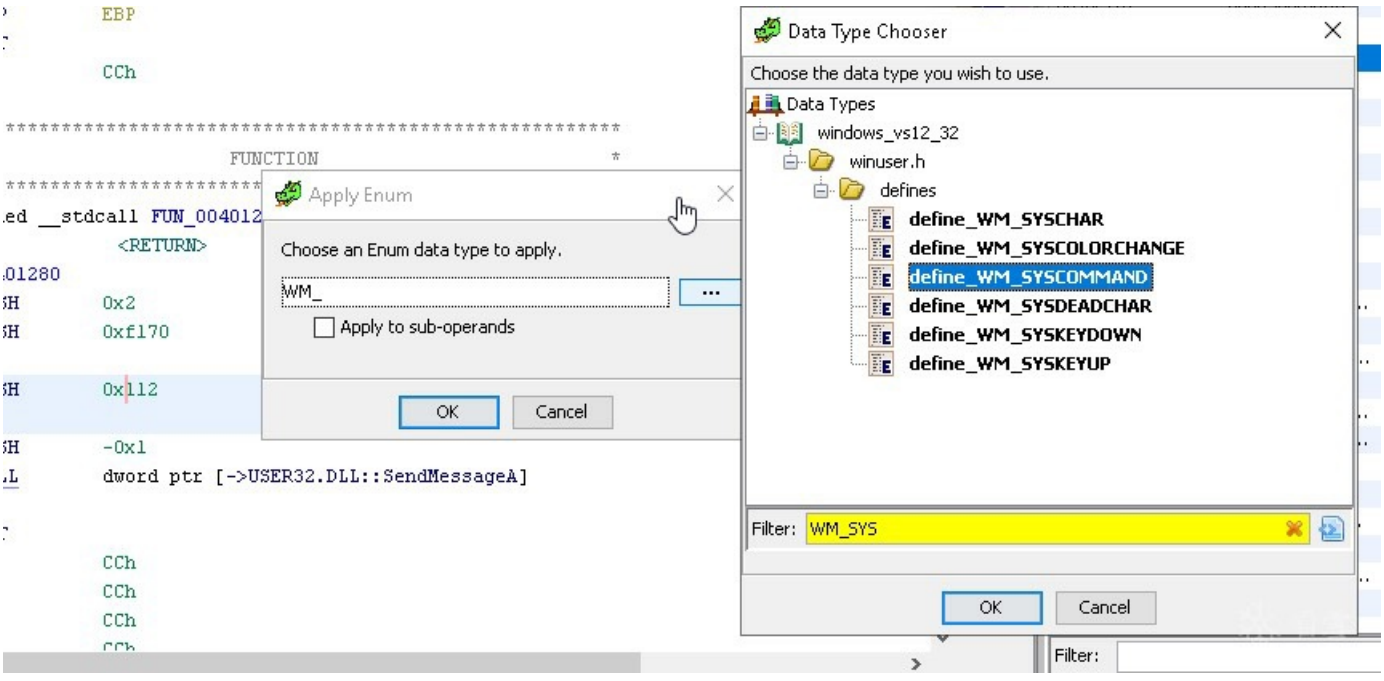
您还可以右键单击列表中的数字，将其转换为另一个数字表示形式：



要在代码浏览器中查看有关指令的信息，只需右键单击并选择“指令信息”：



回到反汇编列表自定义这一主题，您还可以将某些操作数转换为枚举常量：

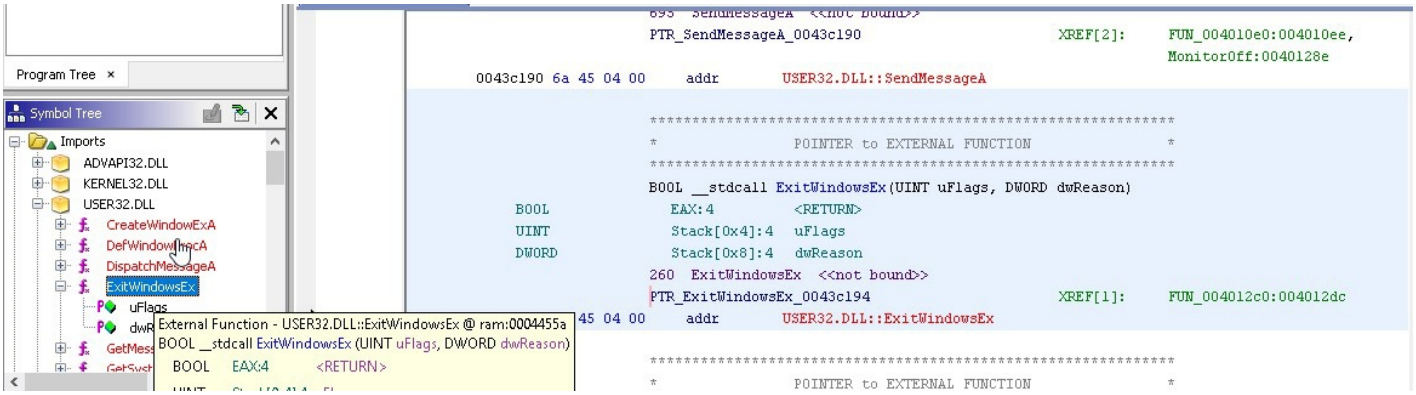


Ghidra提供了一个很好的数据类型选择器，它可以帮助您输入完整的类型名称或直观地选择它。

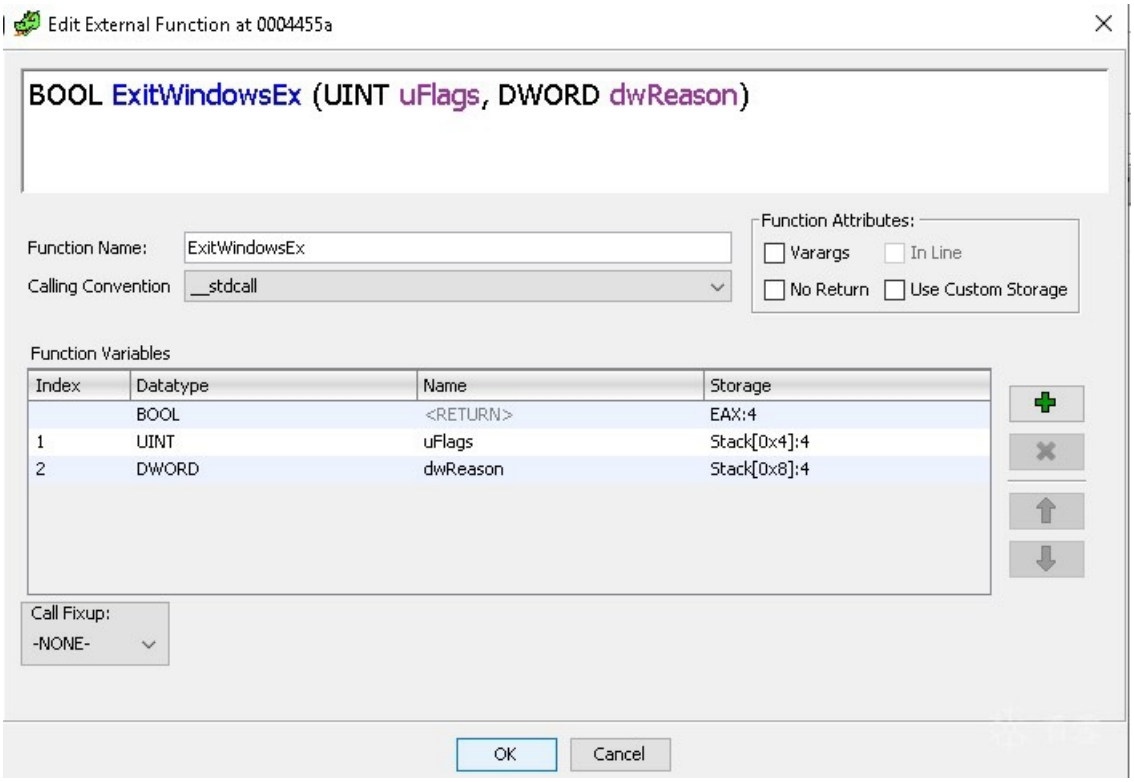
符号树

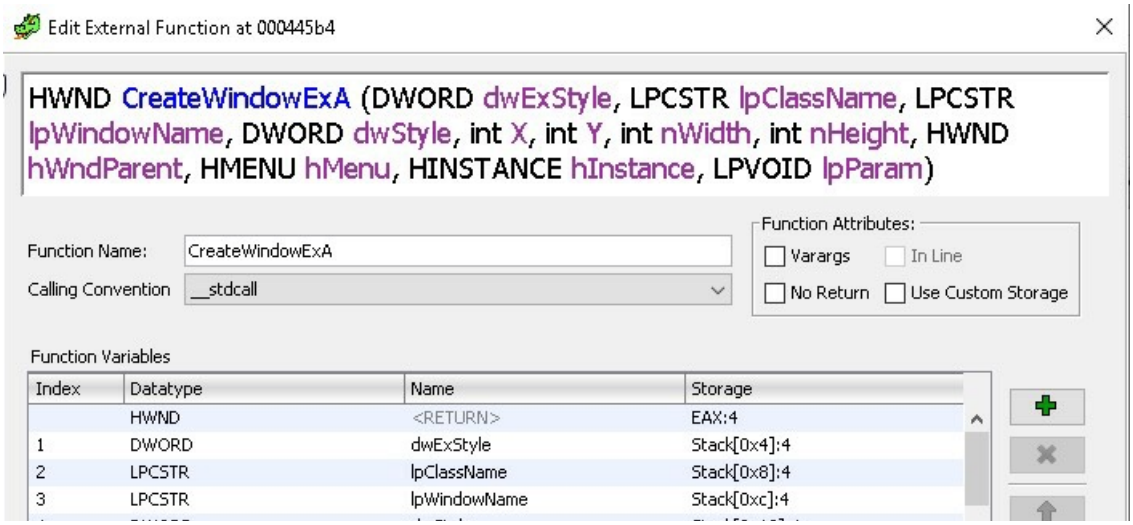
使用符号树窗口可以查看程序中的所有符号，例如导出符号，导入符号，类，函数，标签等。

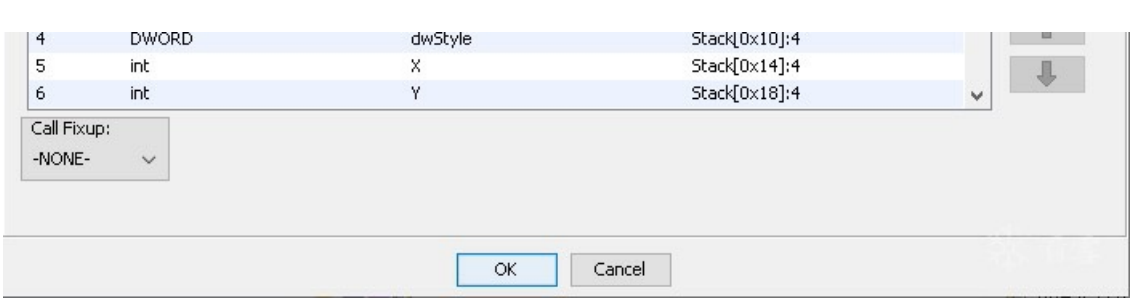
这里我正在浏览USER32.dll的导入符号：



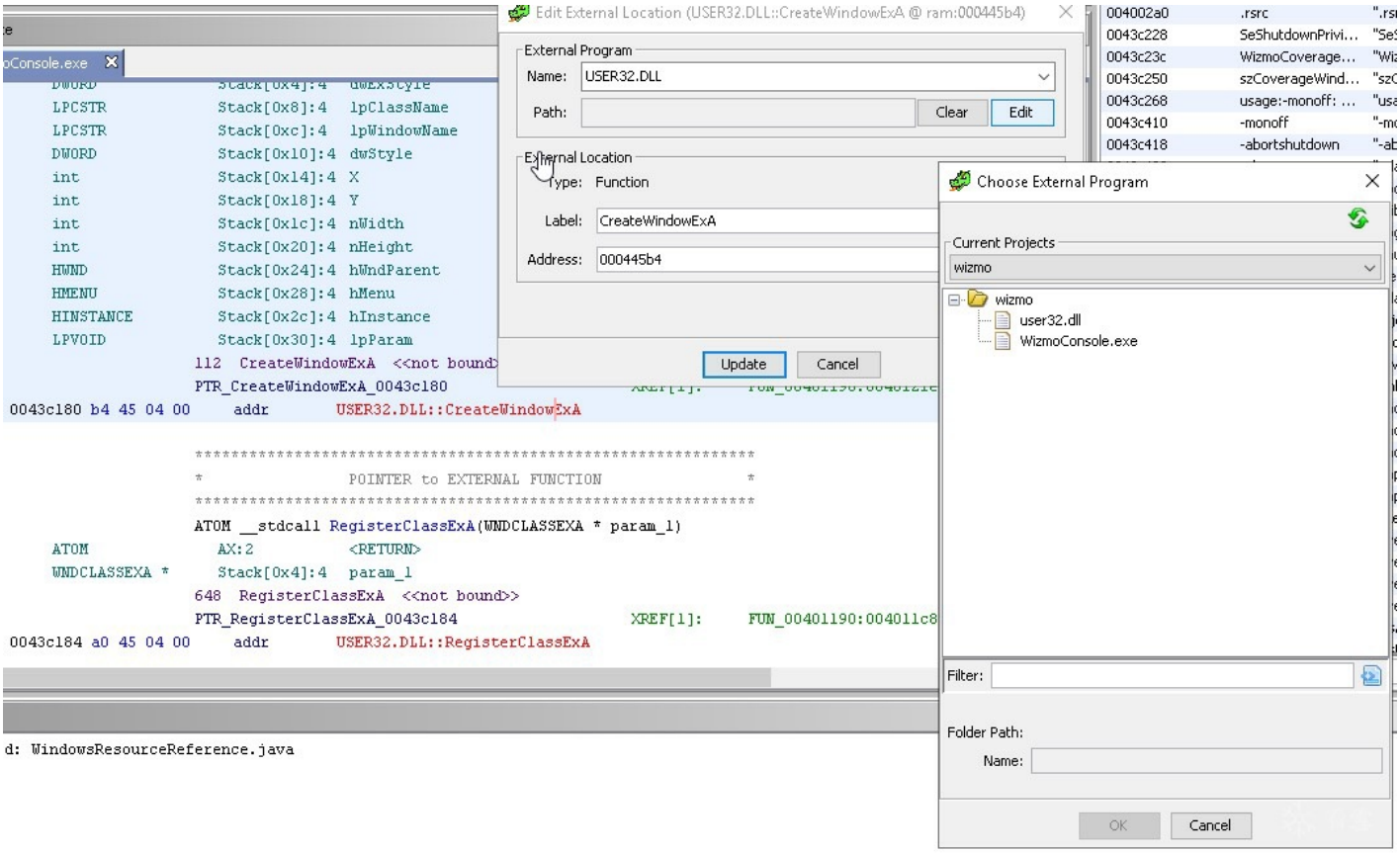
在浏览导入的条目时，可以双击以在代码浏览器中跳转到该条目。 此外，如果您对导入条目的原型不满意，您可以随时编辑它：







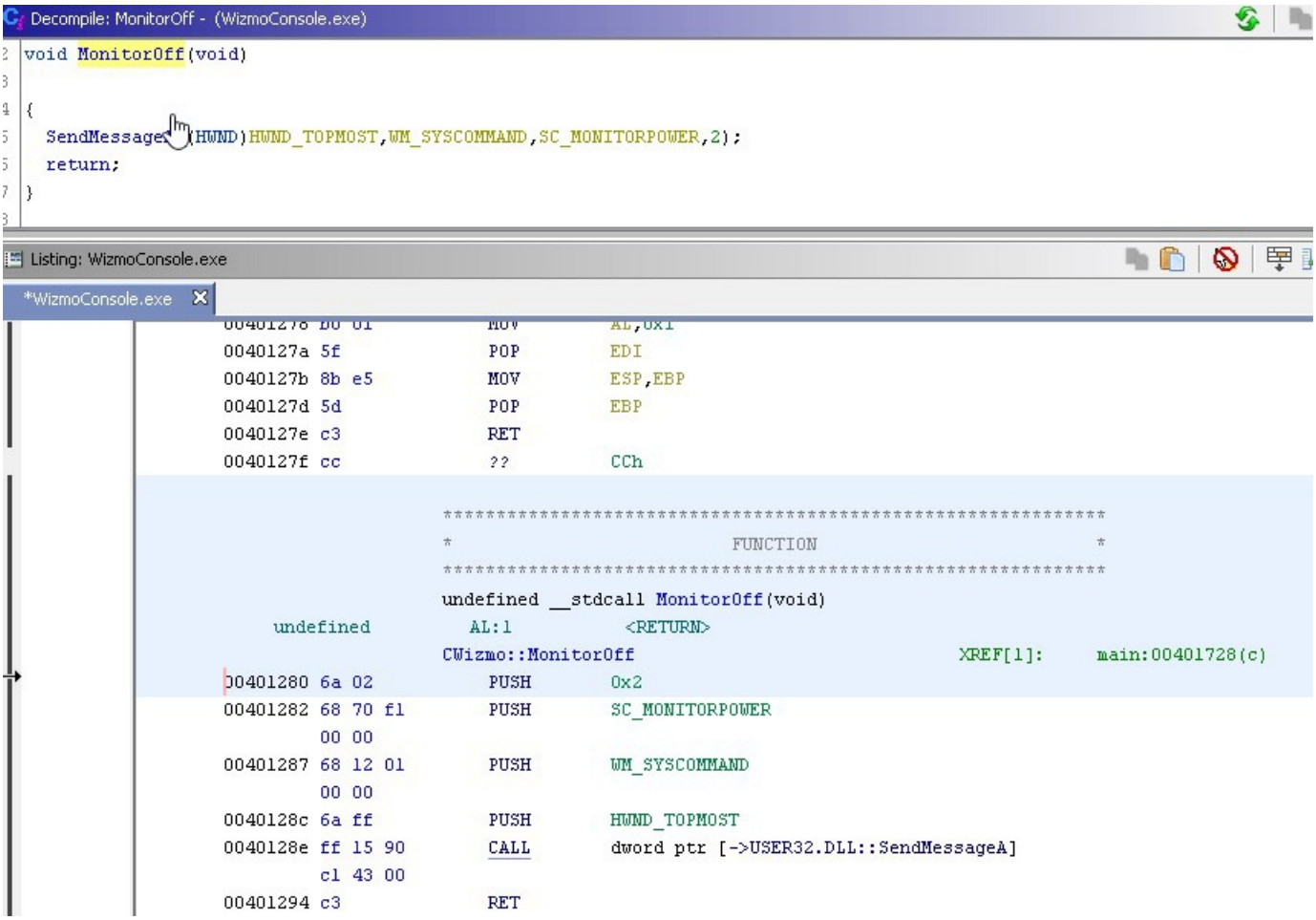
之前，我提到您可以将外部函数链接到另一个导入的文件。 由于我们知道所有这些函数都来自user32.dll，我们可以将这些函数链接到项目中的导入文件：



选择：“路径”->编辑->并选择相关的导入文件（user32.dll）。

反编译器

反编译器是Ghidra中一个很酷的且最受欢迎的功能：



您可以从“窗口”菜单切换到反编译器视图。 反编译视图与反汇编列表同步。 因此，在反编译器视图中浏览时，您将在列表窗口中看到相应的反汇编行。

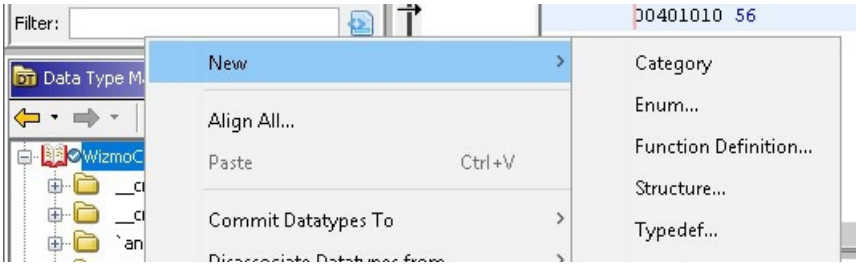
与IDA的Hex-Rays反编译器插件一样，Ghidra的反编译器具有交互性和可定制性：

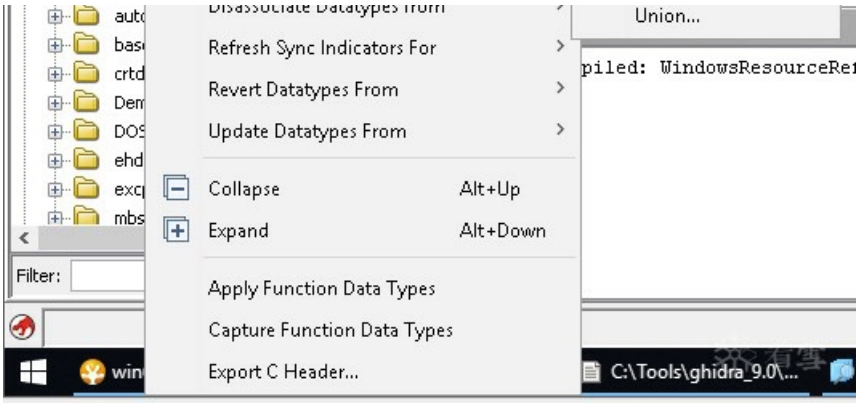
- 重命名函数
- 添加注释
- 改变函数原型
- 更改变量名称和类型
- 等等

例如，这是CWizmo::CWizmo构造函数的完整（手动清理过的）反编译结果：

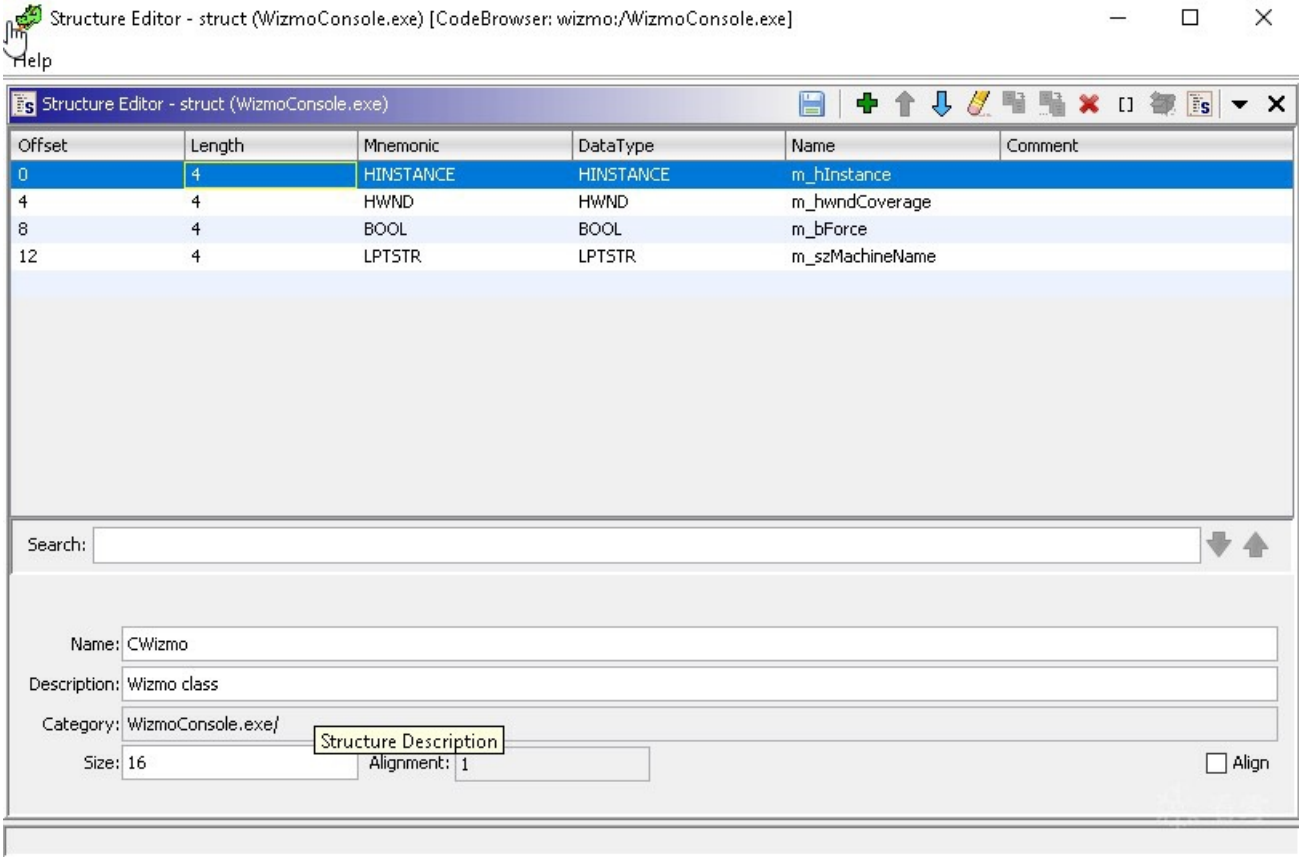
```
CWizmo * __fastcall CWizmo(CWizmo *_this)

{
  HMODULE hModuleExe;

  hModuleExe = GetModuleHandleA((LPCSTR)0x0);
  *(HMODULE *)&_this->m_hInstance = hModuleExe;
  _this->m_bForce = 0;
  _this->m_szMachineName = (LPTSTR)0x0;
  return _this;
}
```

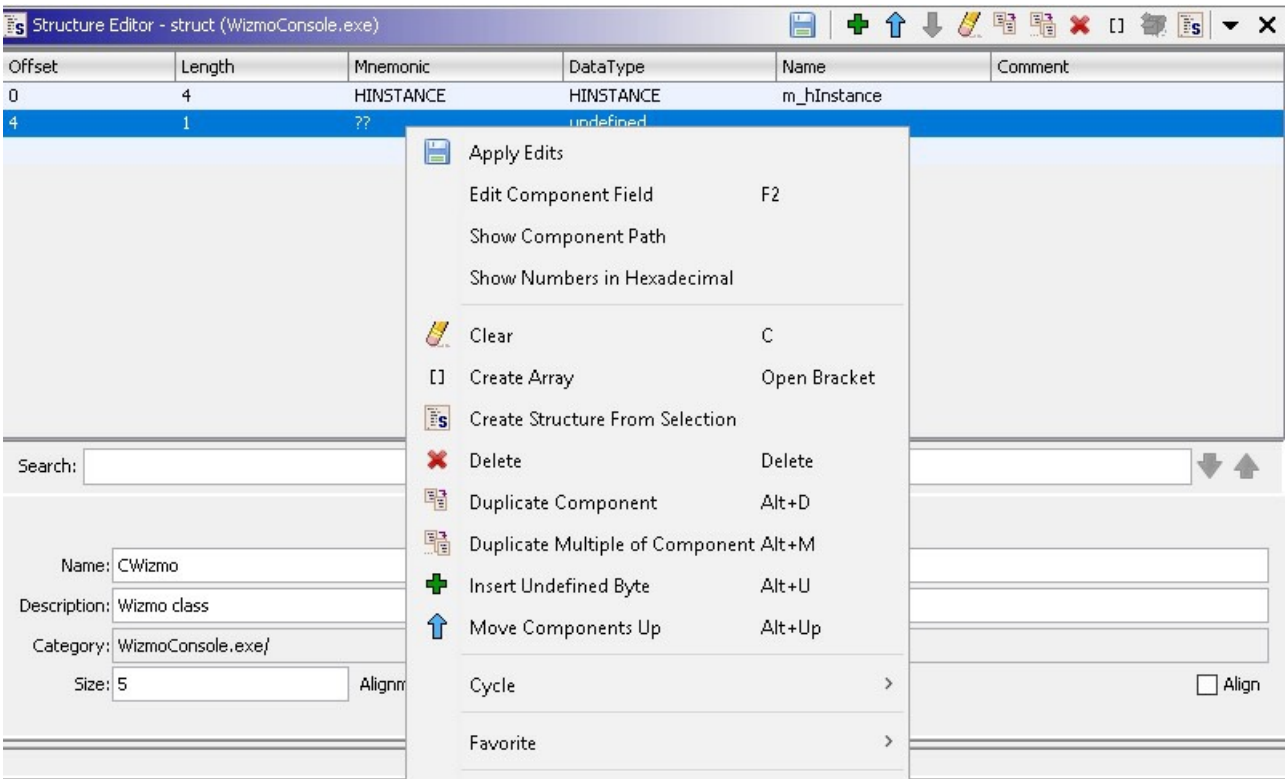
我必须首先使用“数据类型”窗口并选择“新建 -> 结构”来创建新的自定义结构：

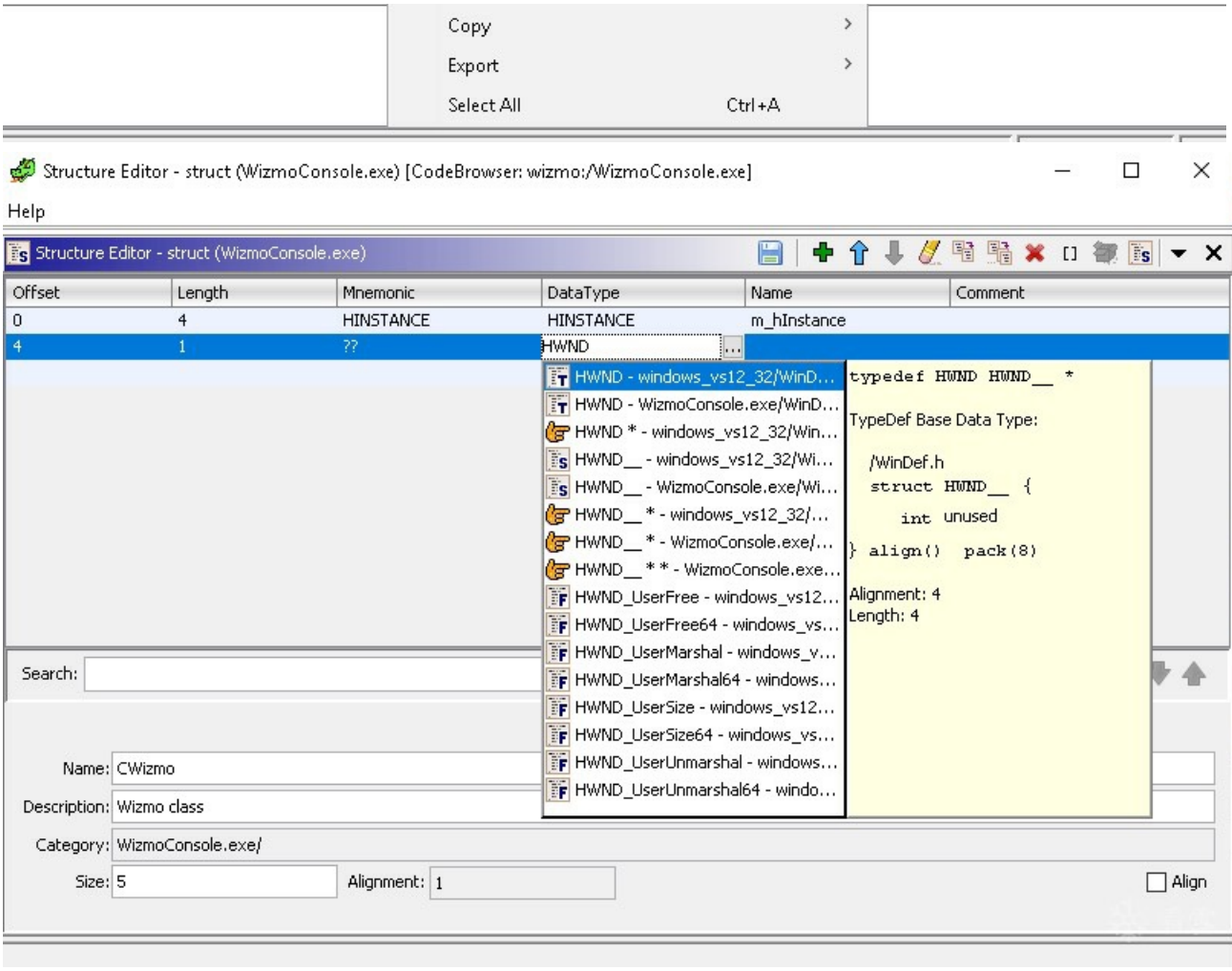


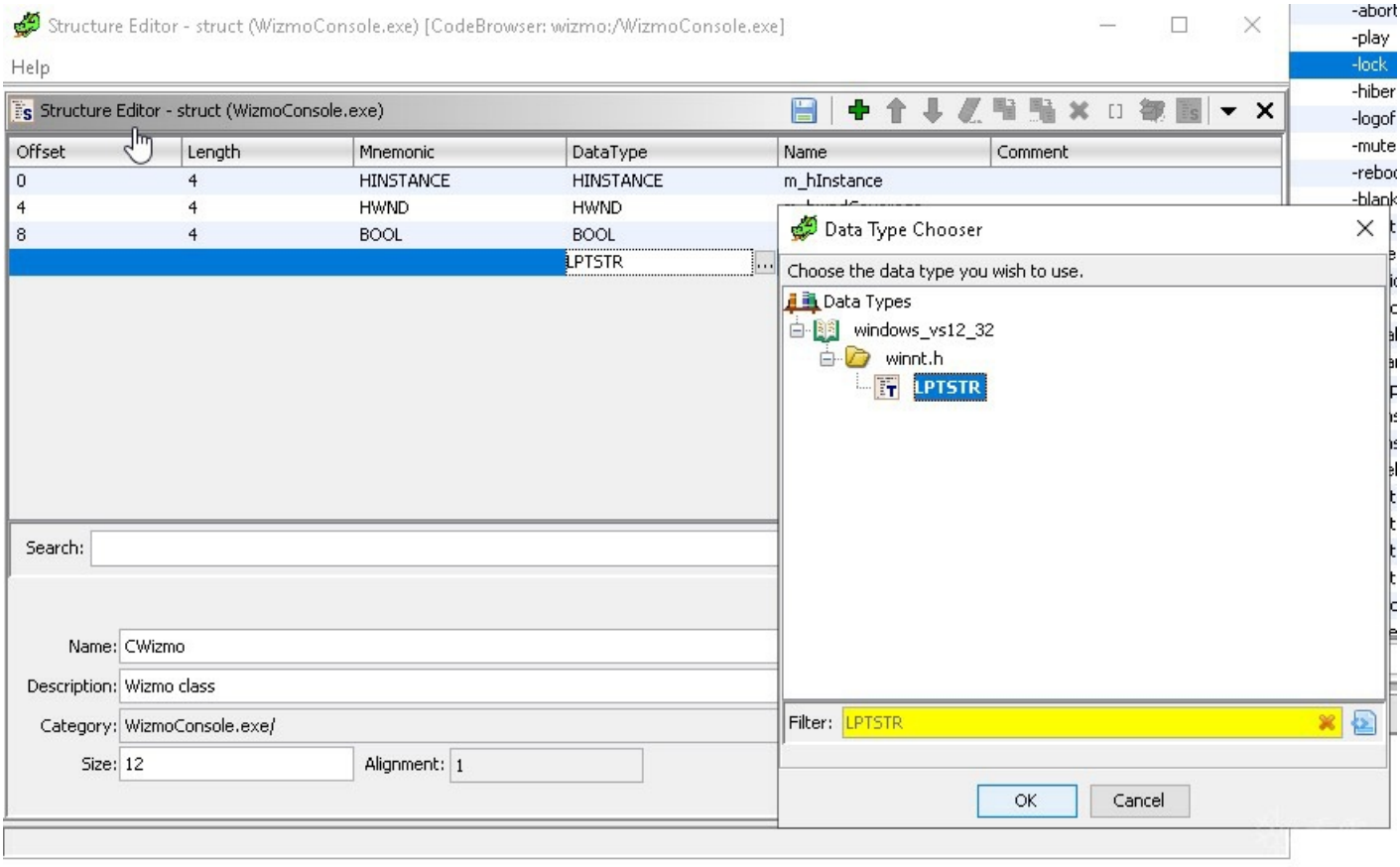


然后我填充了新的结构字段：

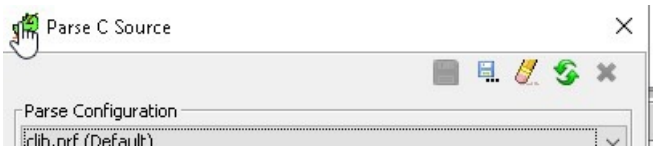


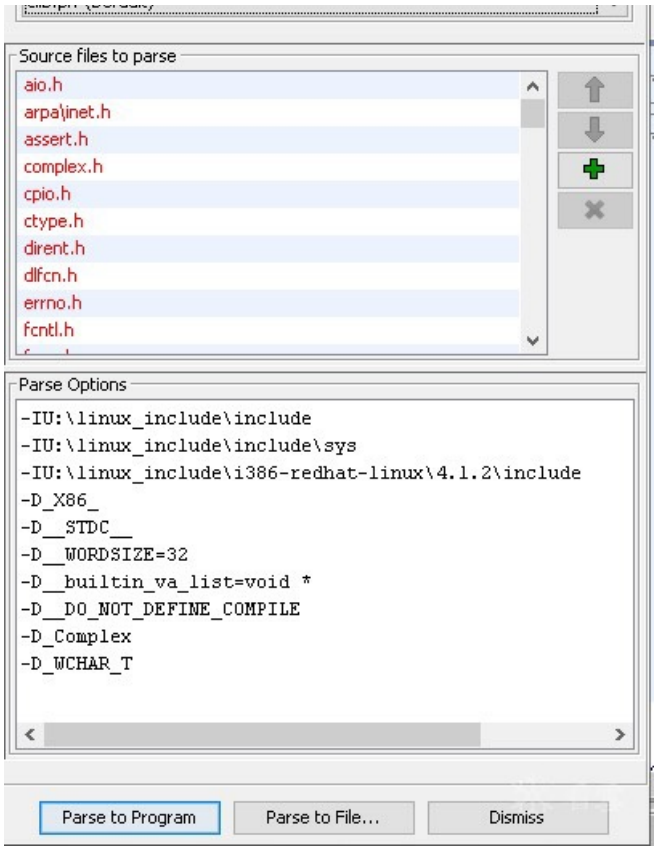




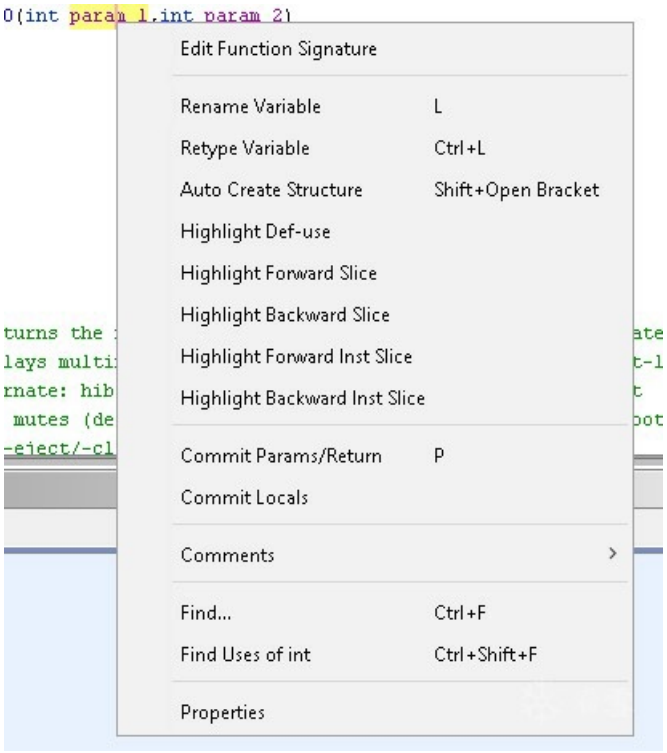


如果您不想手动创建自定义结构，也可以解析C头文件：

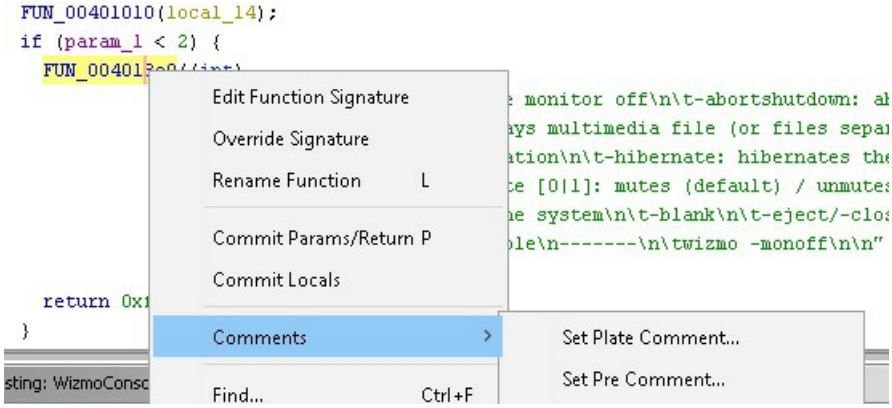




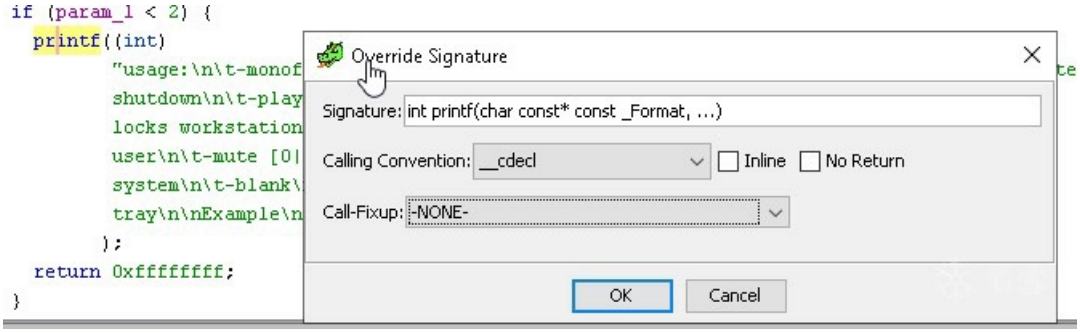
反编译器有一个上下文弹出菜单：



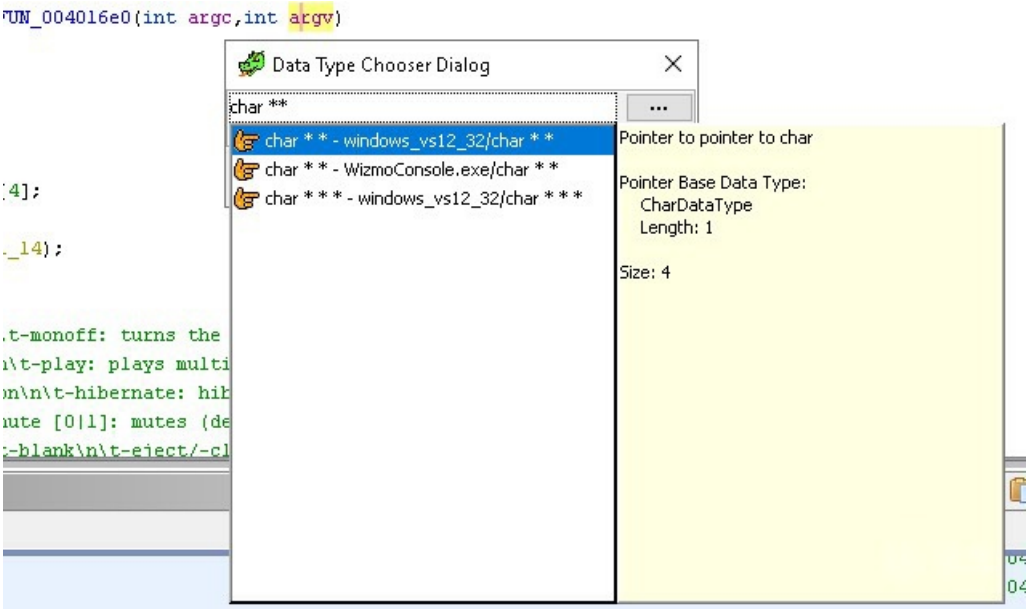
它允许您在反编译列表中设置注释：



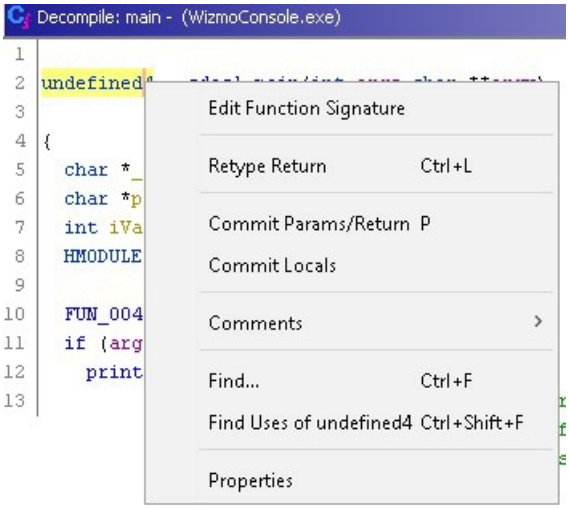
更改反编译的函数原型：



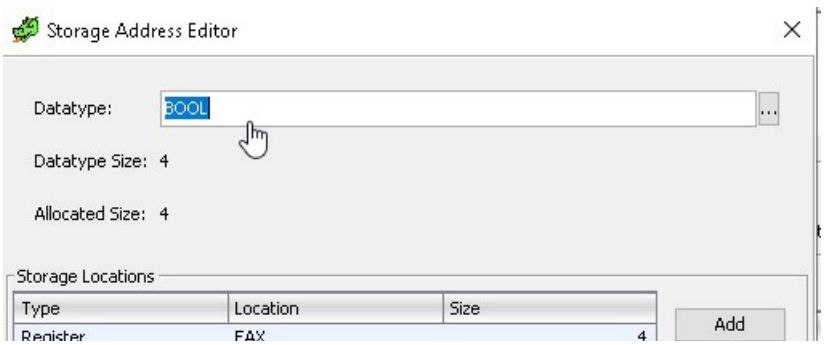
更改函数参数的原型：

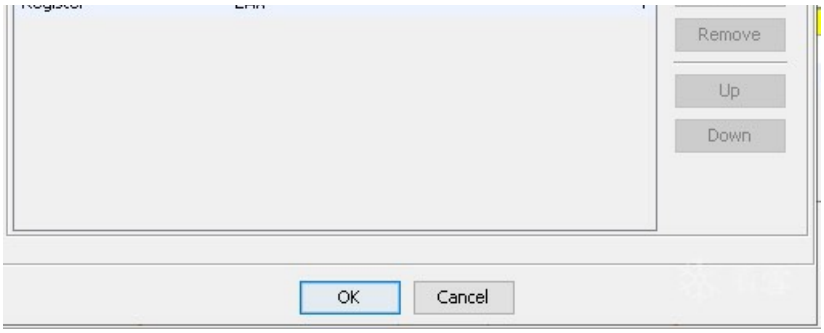


修改函数的返回类型、签名或执行搜索：

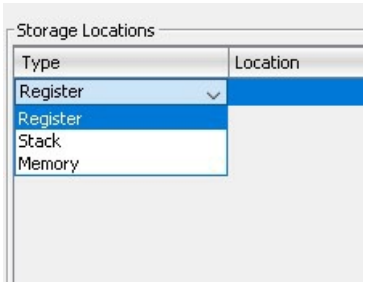


值得注意的是，函数编辑器（使用“F”热键切换）与IDA的函数原型功能一样强大。 您可以编辑参数并为它们指定自定义存储（堆栈，寄存器等）：



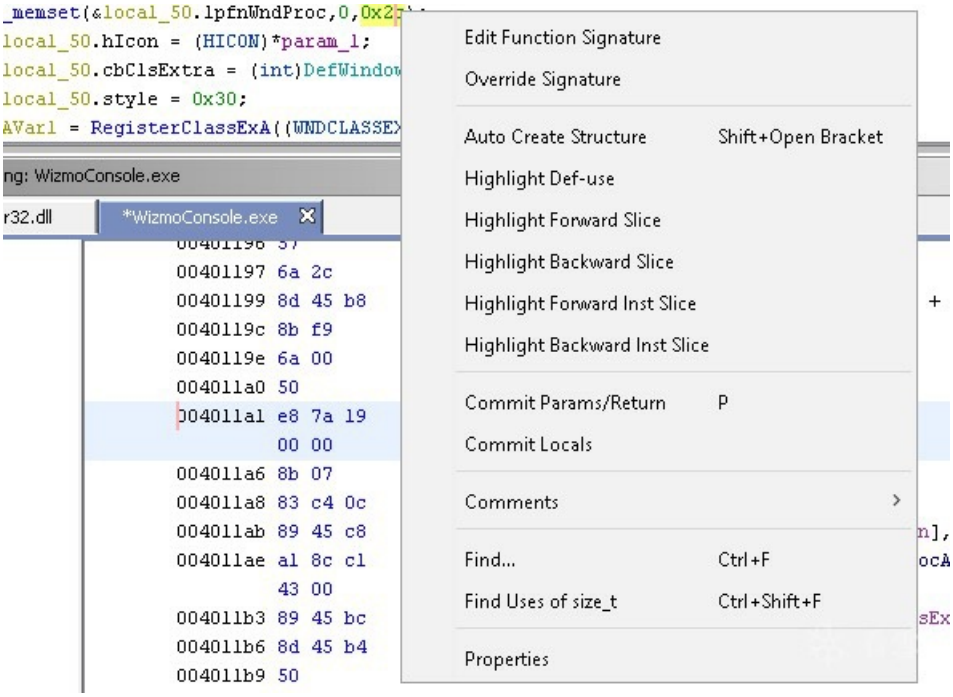


x86输入文件的一些受支持的存储类型：

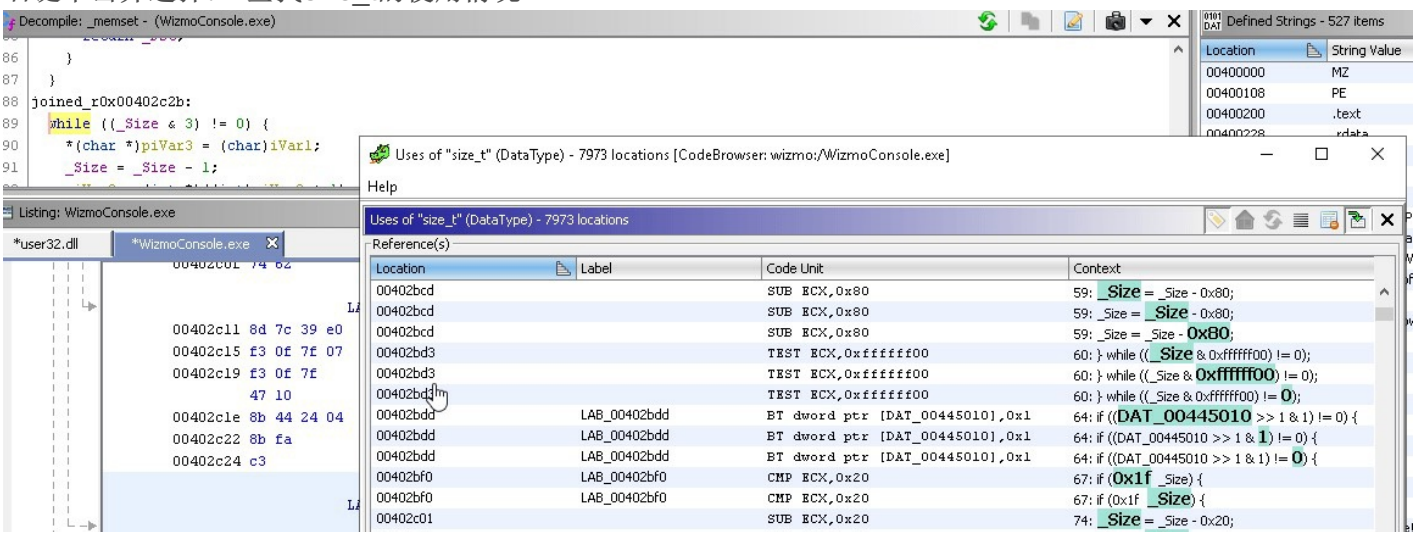


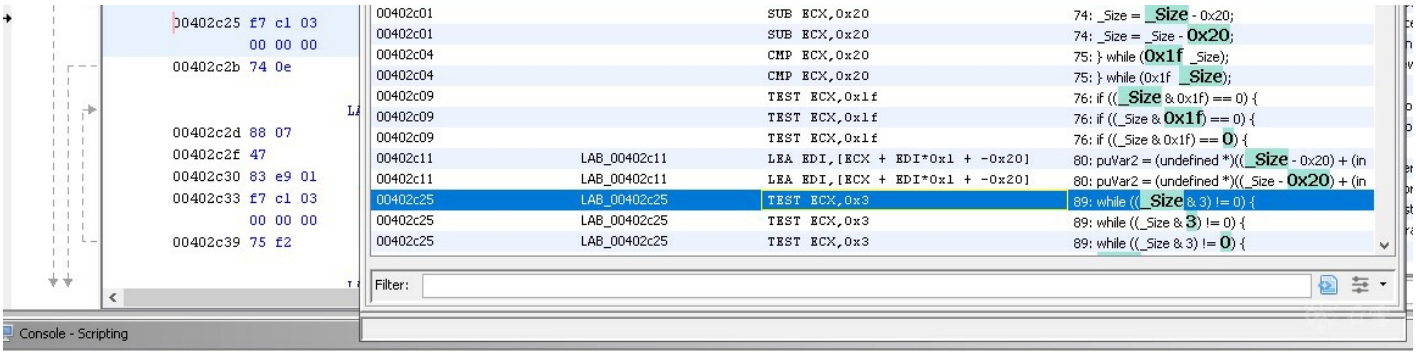
除了是一个交互式反编译器，您还拥有强大的搜索功能。 例如，我们可以从反编译列表中搜索给定数据类型的使用情况。

在这里，我们右键单击memset的最后一个参数（0x2c，size_t）来查找所有反编译函数中“size_t”类型的所有使用情况（对于漏洞研究非常方便）：



右键单击并选择：“查找size_t的使用情况”

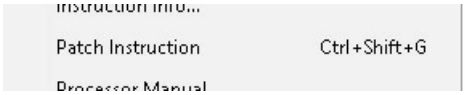




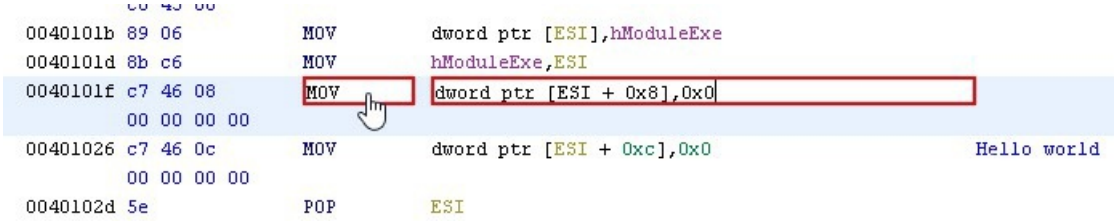
结果显示了所有使用“size_t”类型的变量。

代码修补和十六进制查看器

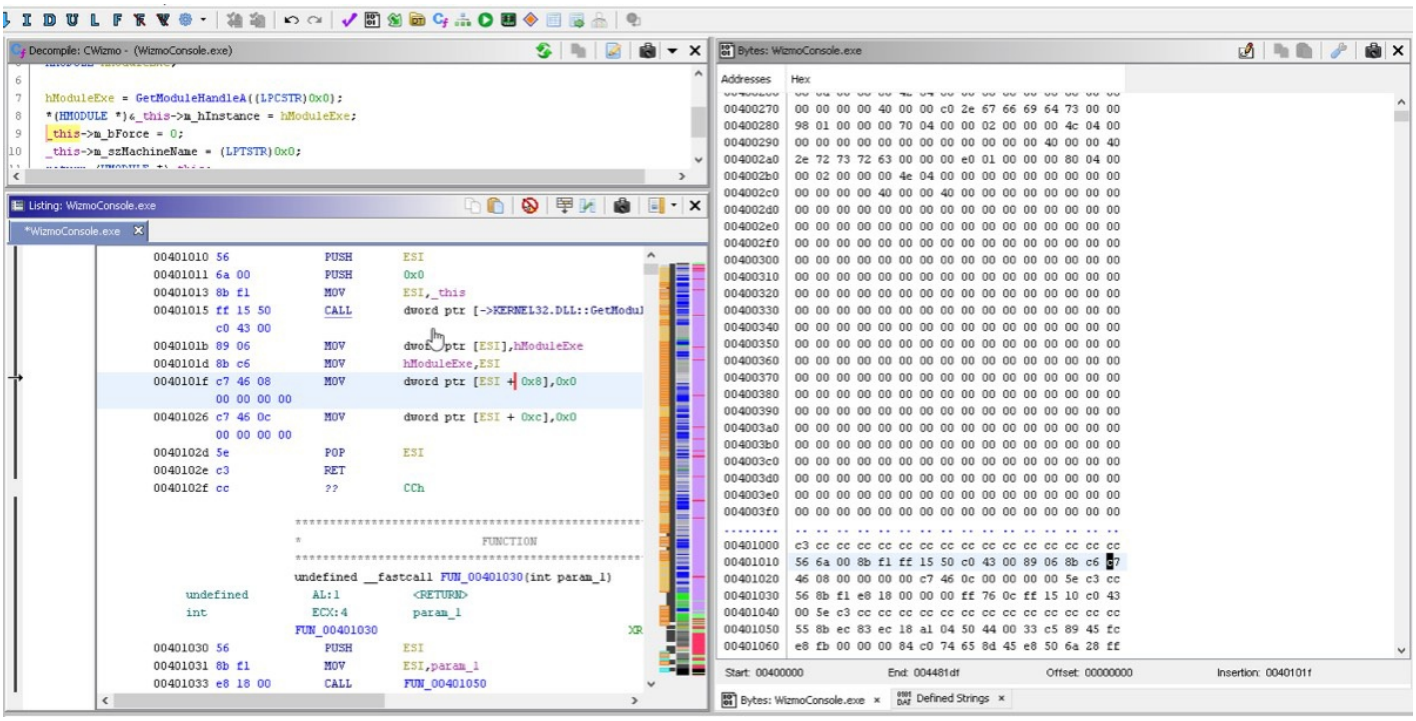
与IDA一样，Ghidra提供了许多功能来修补代码，然后保存修补结果。 要修补指令，只需右键单击并选择：



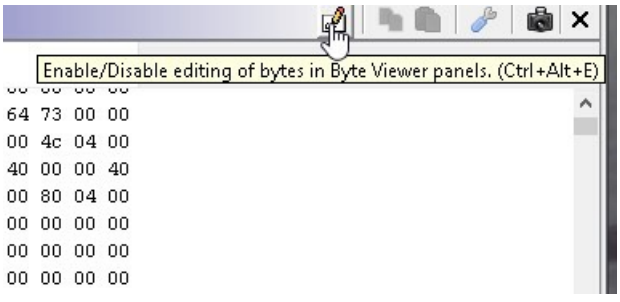
然后，您将看见指令编辑器/汇编器：



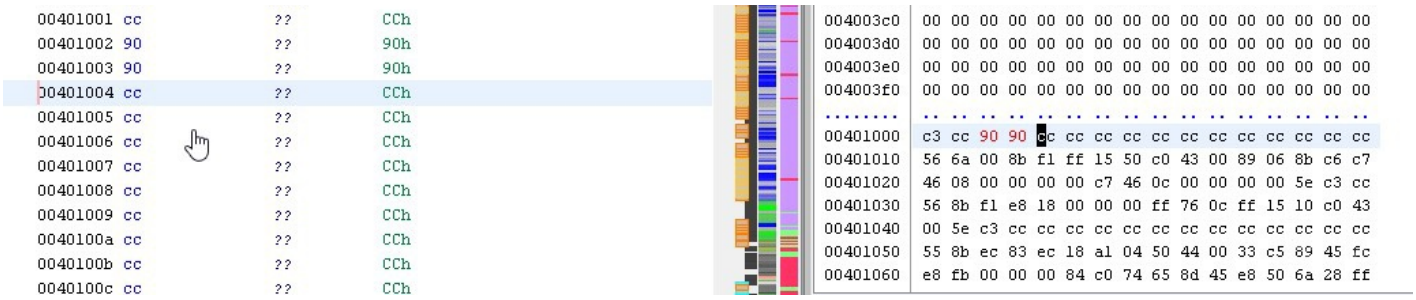
如果您希望像精英黑客一样从十六进制查看器中修补代码，只需从“窗口/字节”菜单切换十六进制视图：



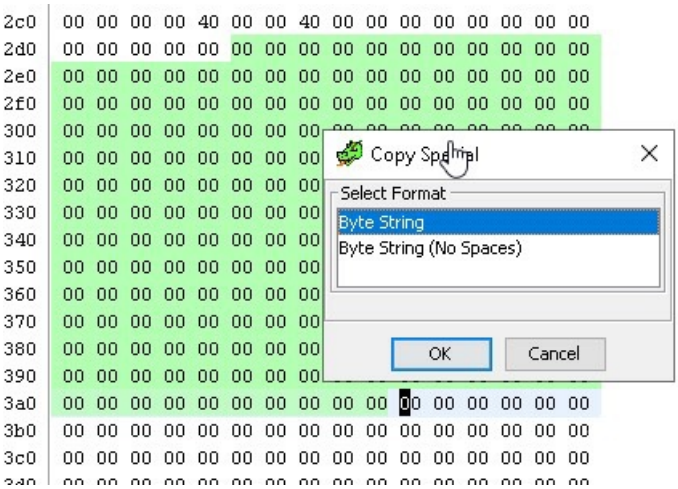
然后使字节视图可编辑：



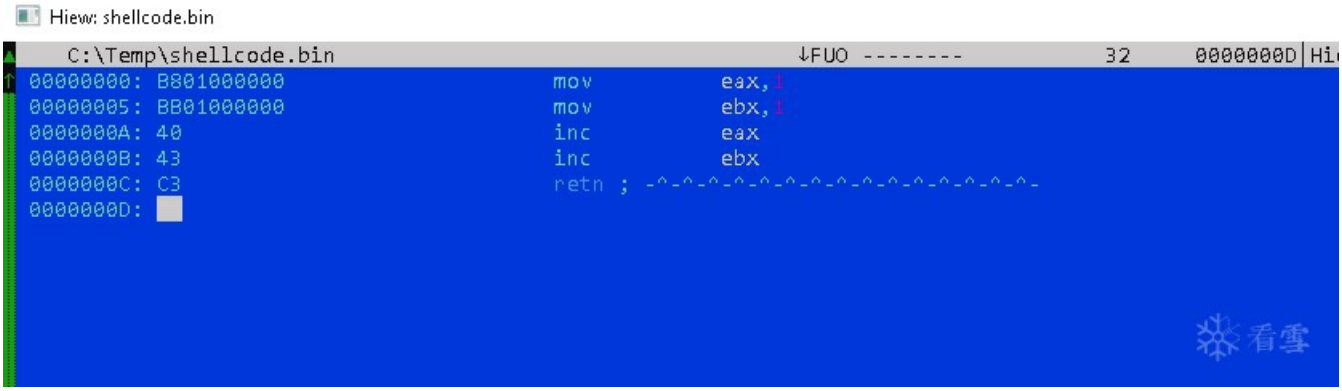
您现在可以编辑该程序：

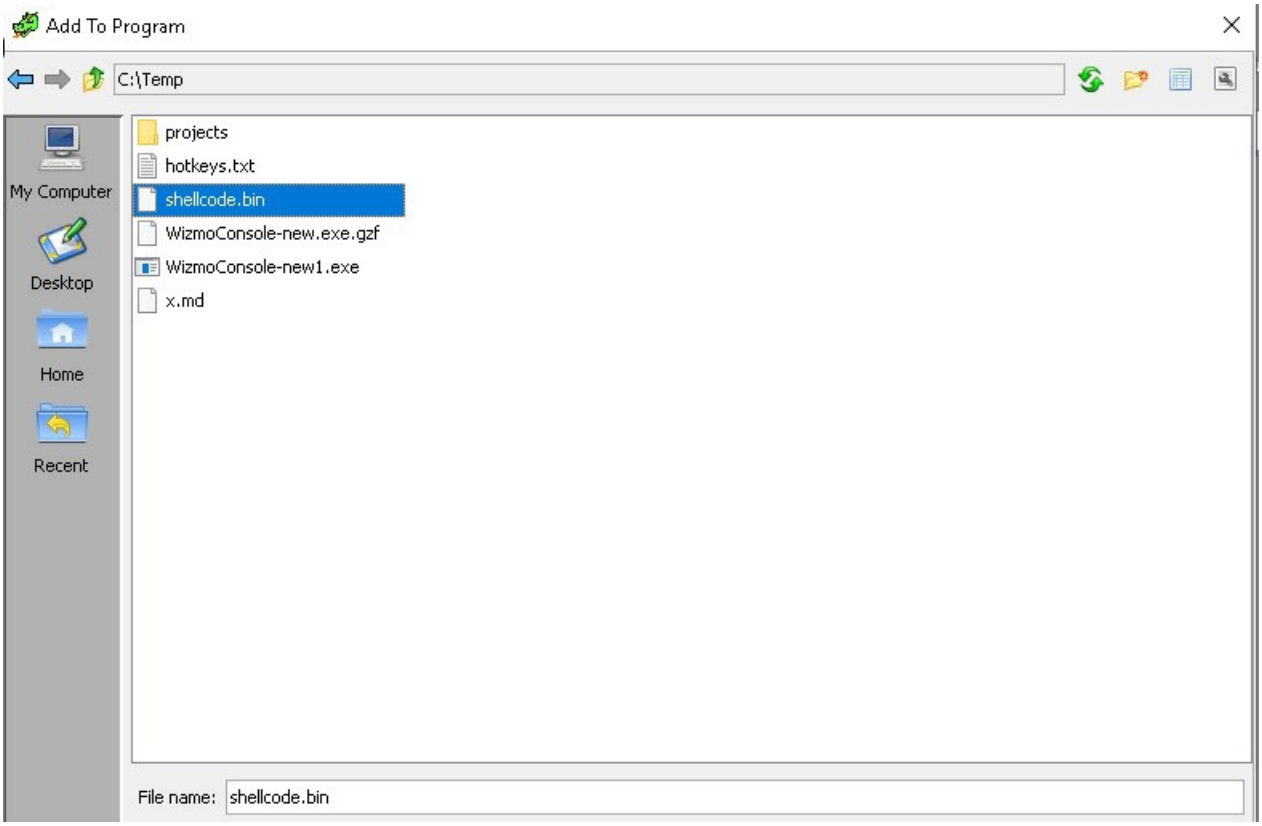


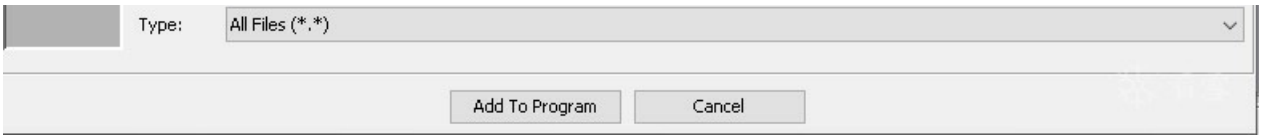
十六进制查看器有一个上下文菜单，允许您复制字节，例如：



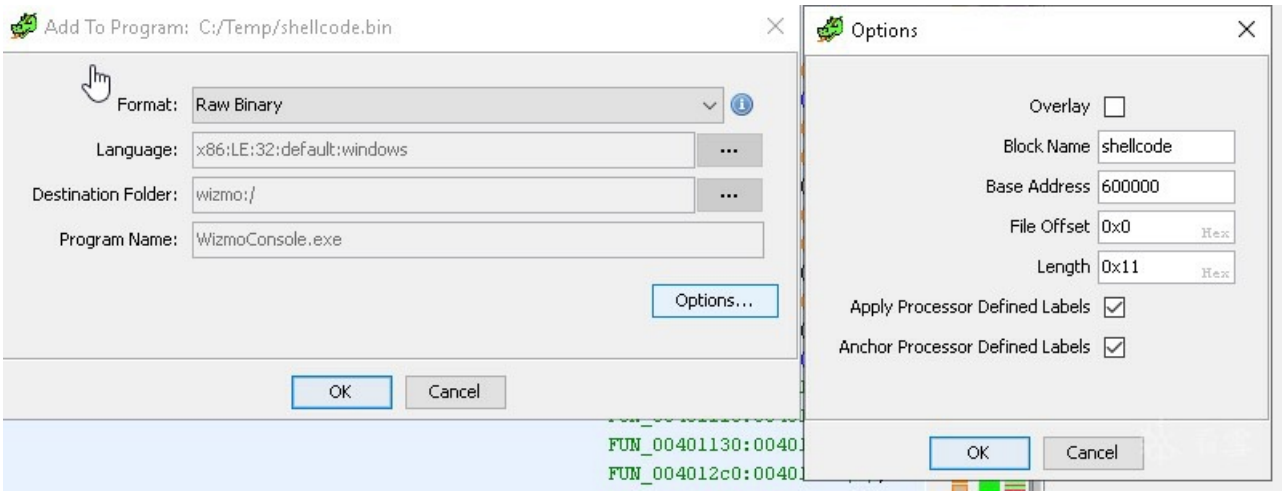
与在IDA Pro中一样，您可以通过从“文件”菜单中选择“添加到程序”来“加载其他二进制文件”：



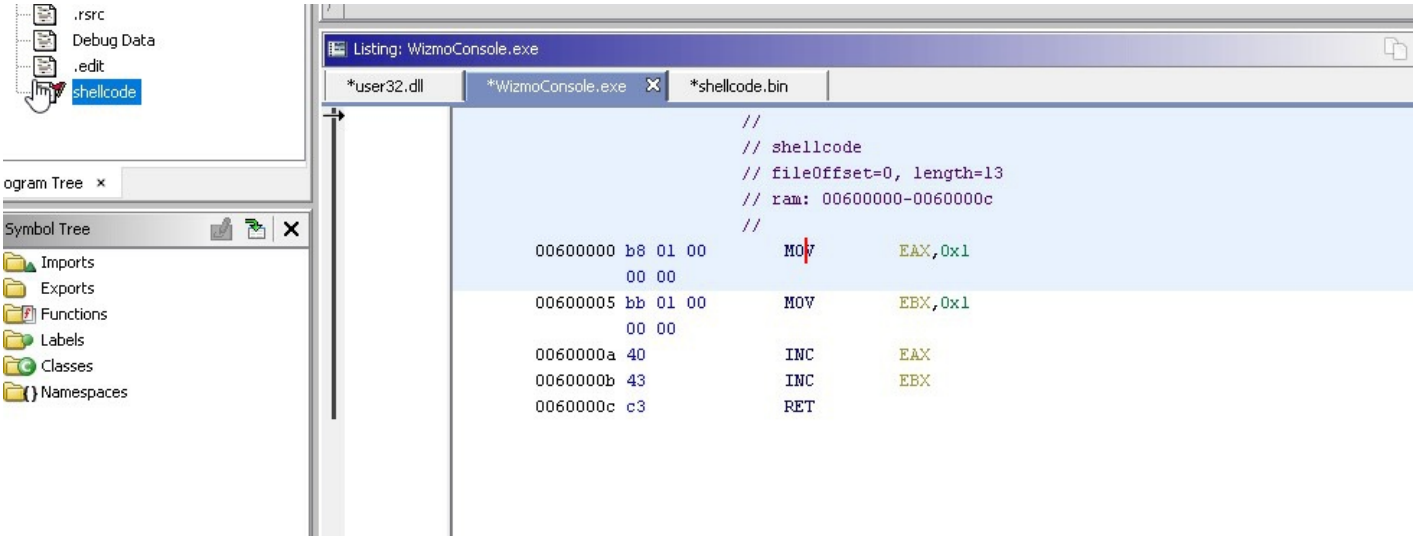




选择要添加的文件后，可以指定其他加载选项（块名称、基地址等）：

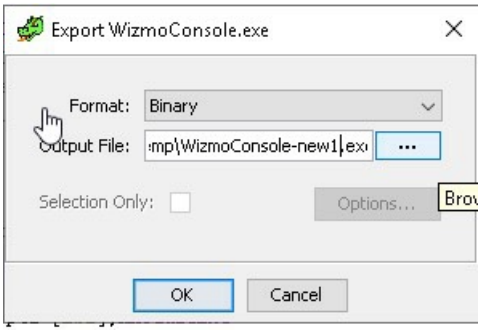


这非常有用，例如，如果要加载shellcode并在程序中对其进行分析：

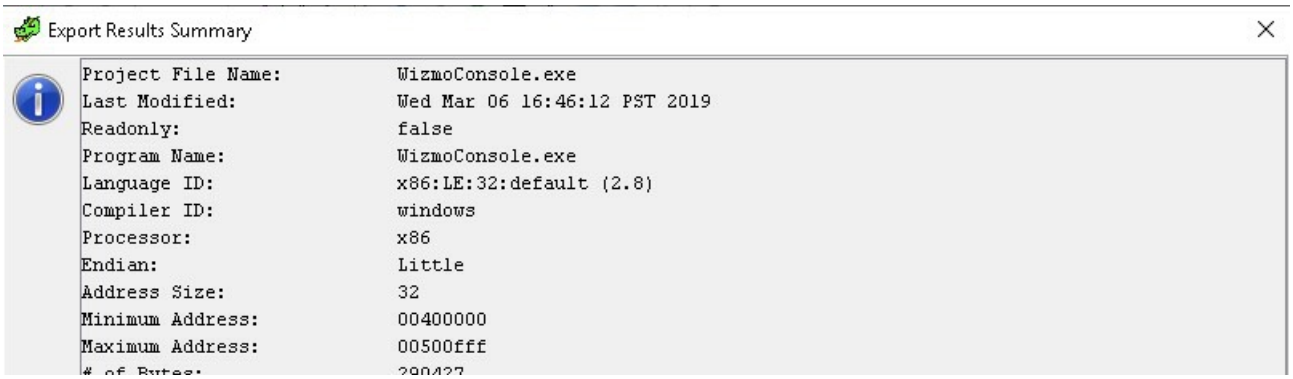


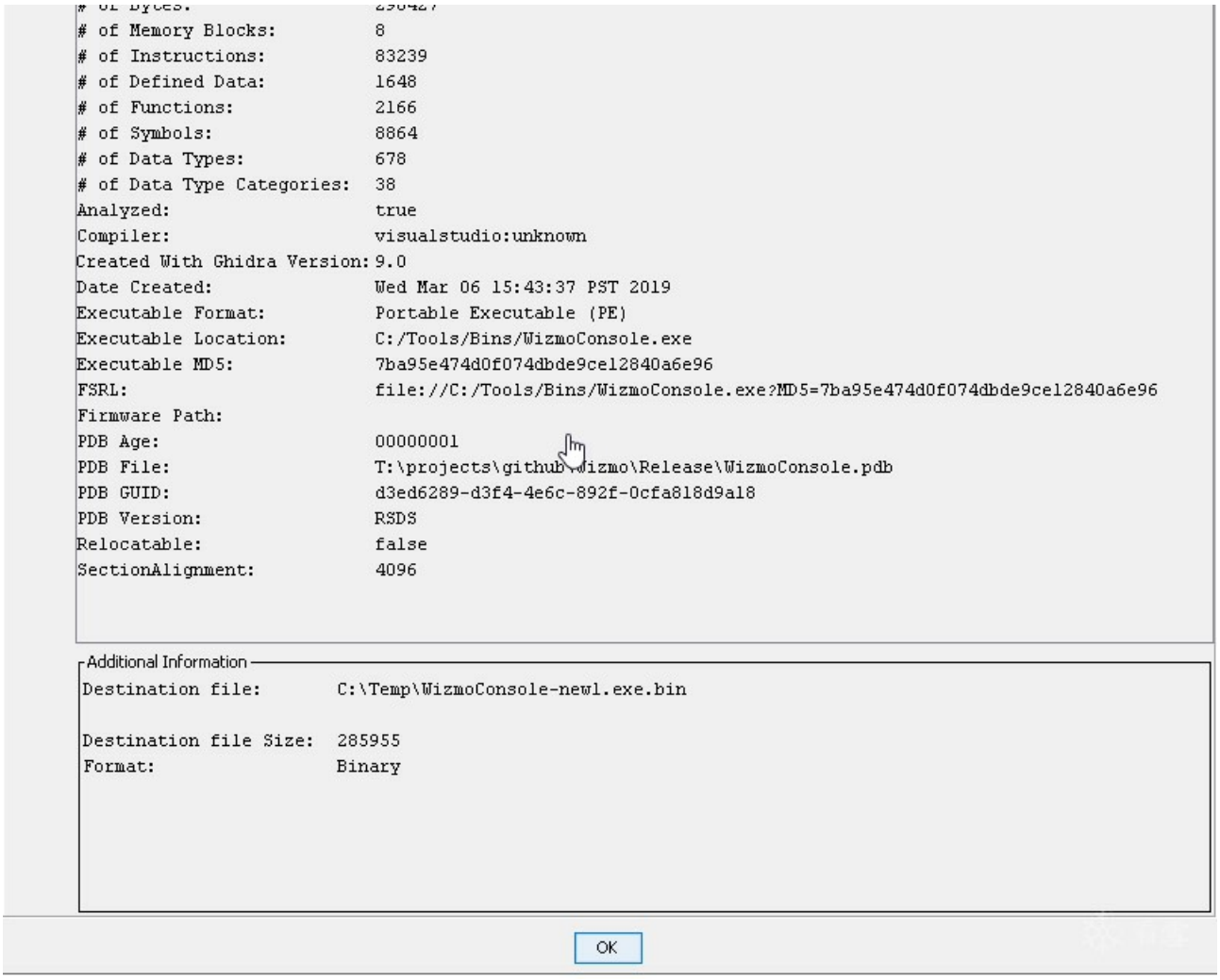
然后，新代码在代码浏览器中以其自己的块名称很好地显示。

如果没有导出/应用到外部，则不会完成修补。 Ghidra和IDA一样，允许你导出你的改动：

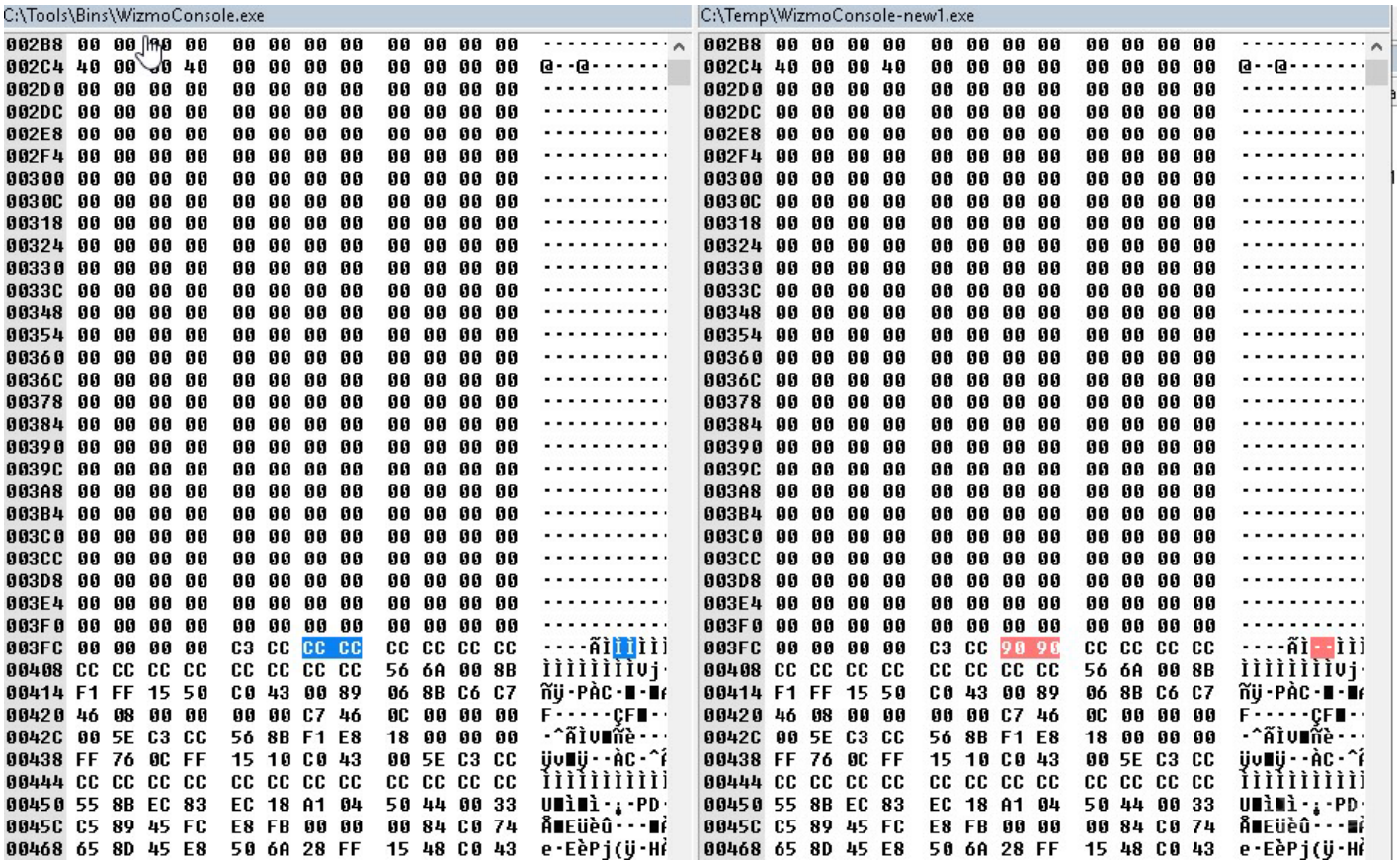


导出为二进制格式。 成功导出后，您将看到摘要：





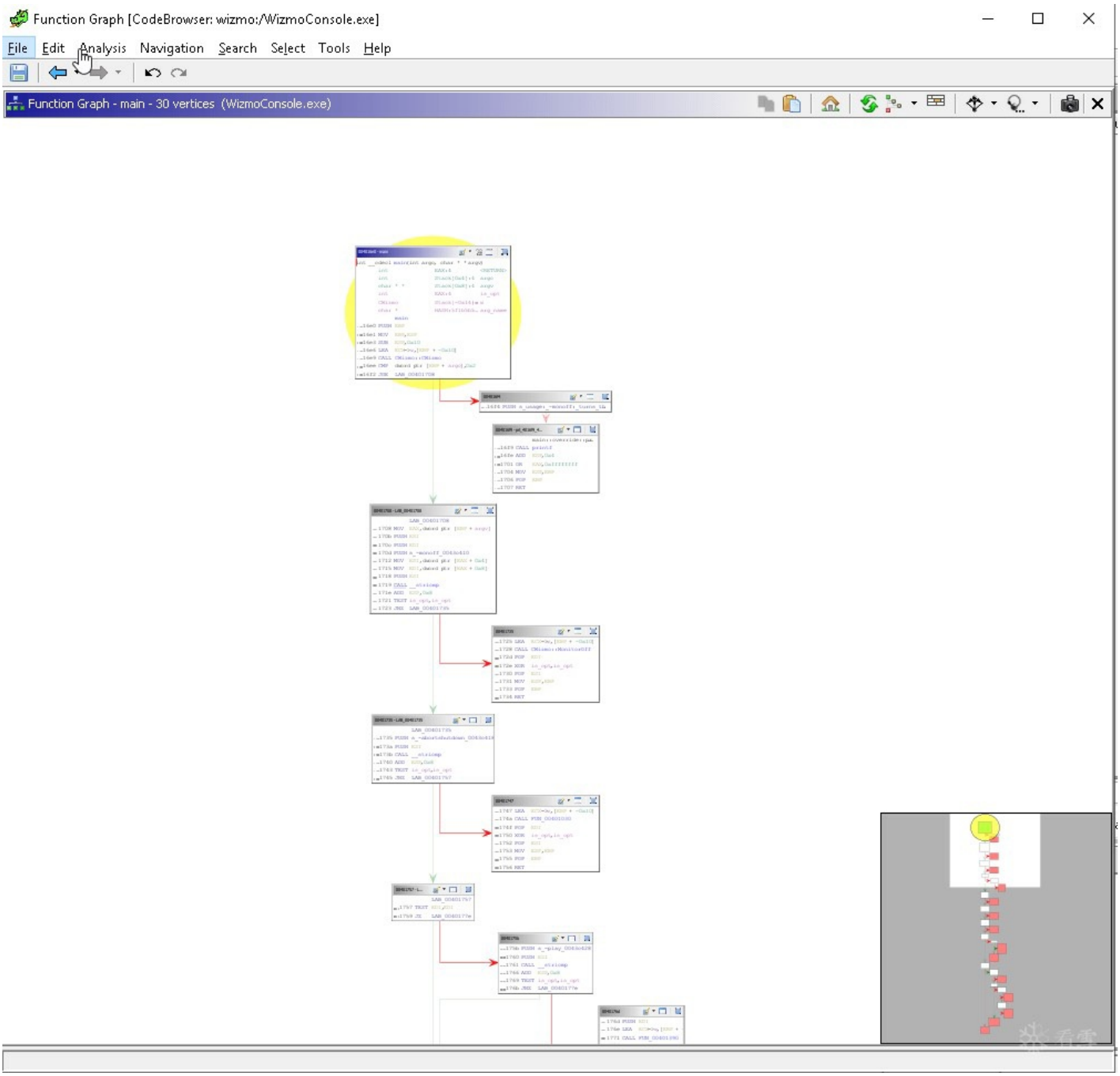
如果您比较原始文件和修补文件，您应该看到差异，说明修补是被正确应用的：



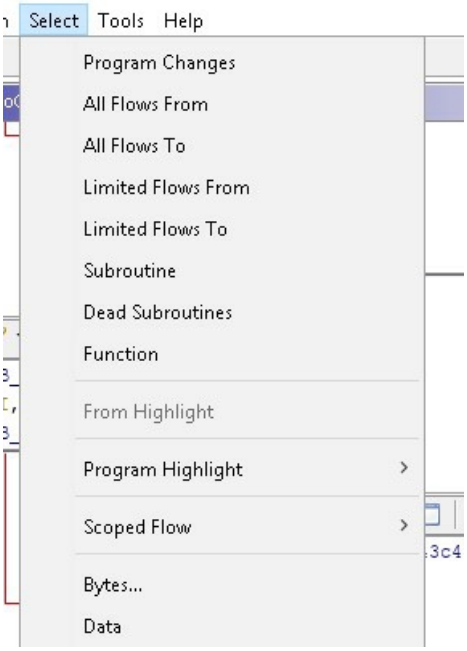
等等

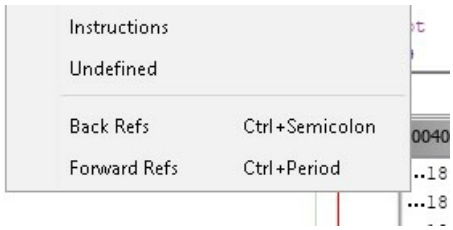
图表视图

与IDA一样，Ghidra也支持图表视图。 结合“选择”菜单中的功能，图表视图成为一个强大的工具：

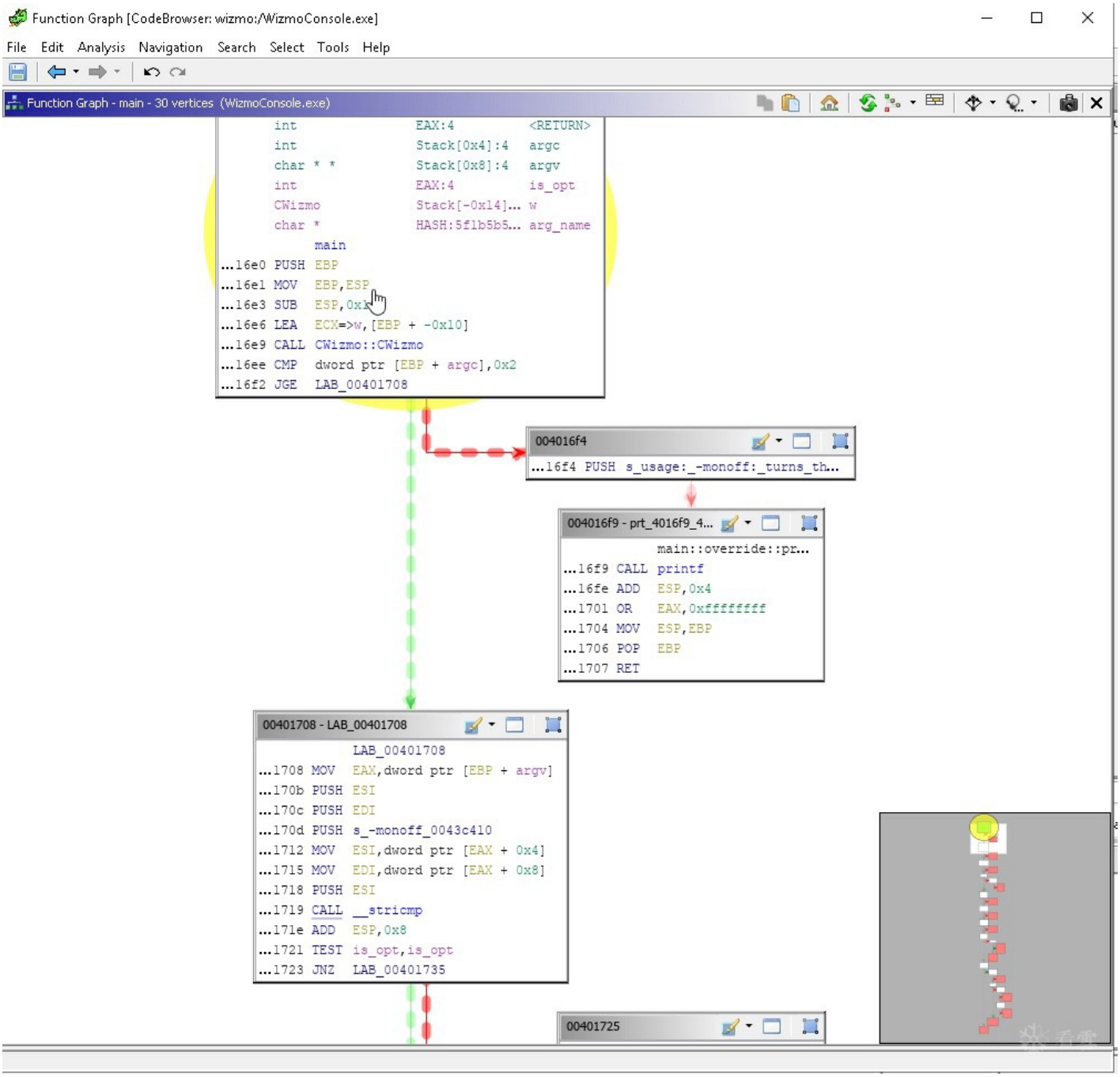


“选择”菜单：

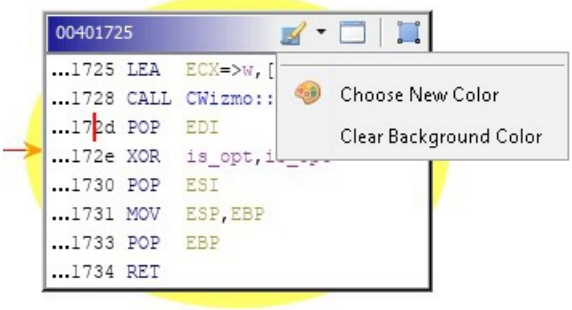




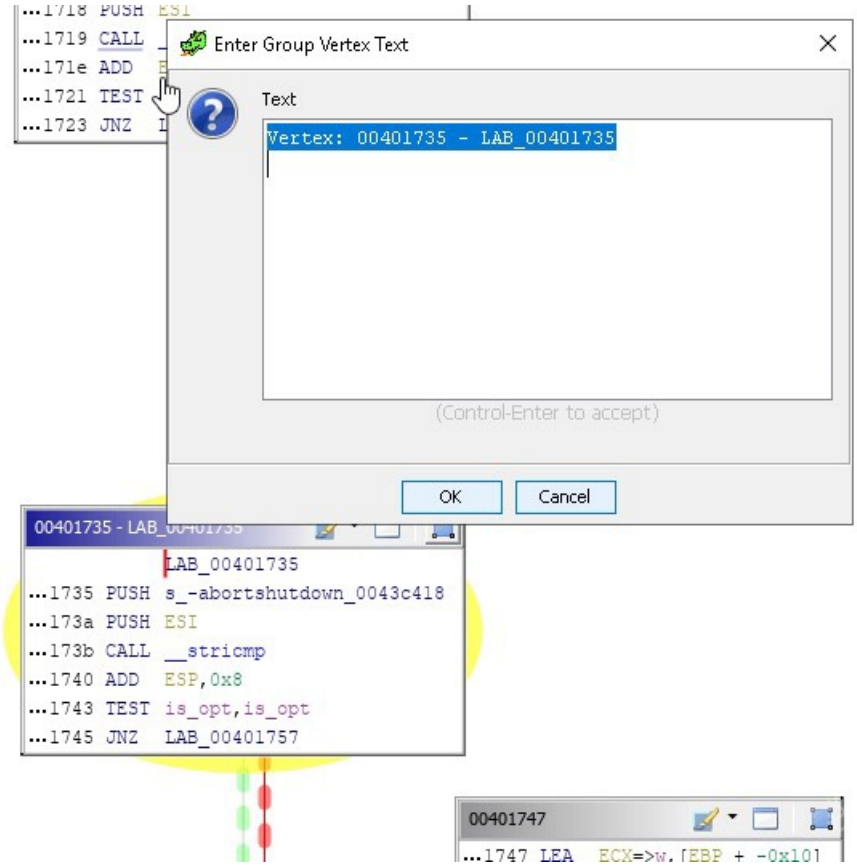
你可以放大：



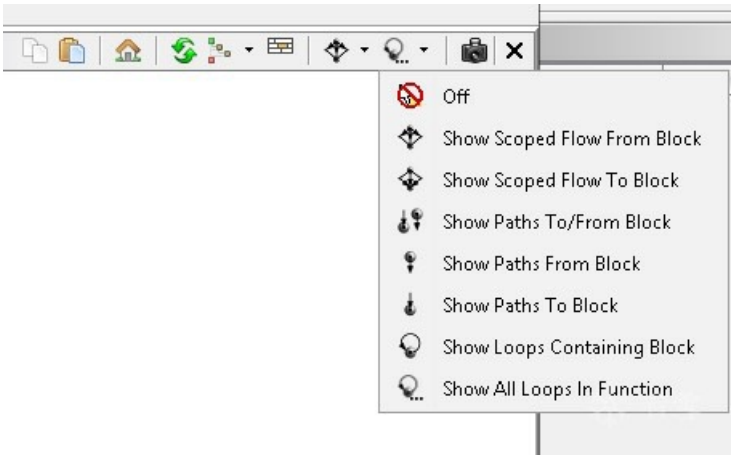
您还可以更改基本块的颜色：



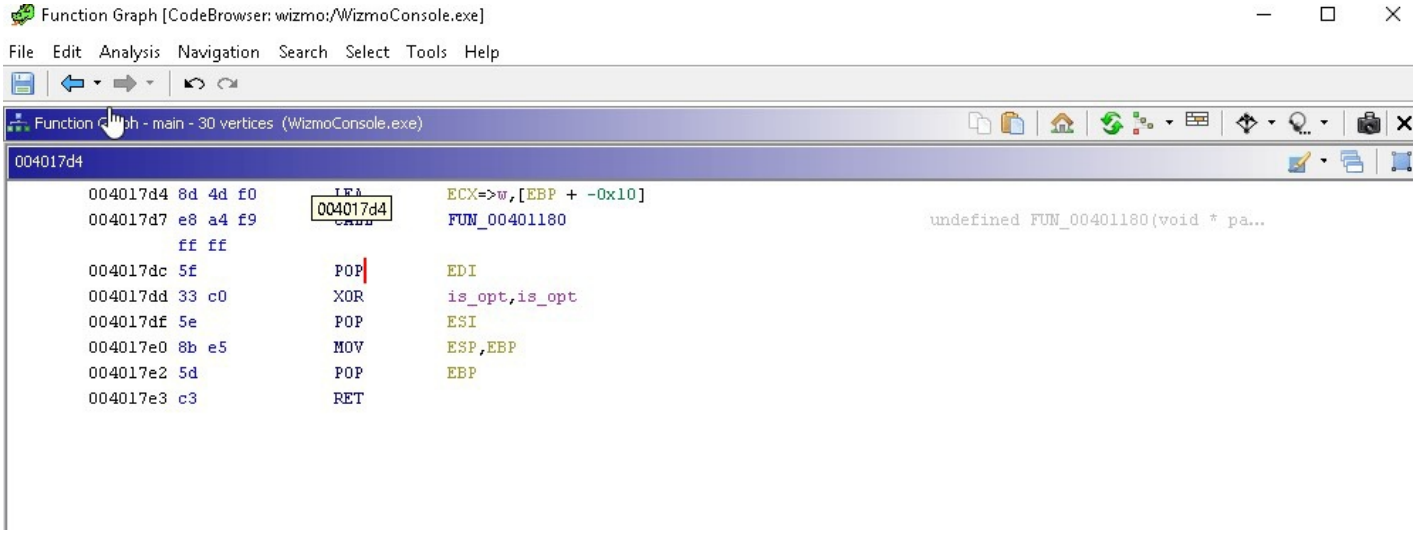
或者将基本块的内容折叠为具有您选择的标签的单个块：



您还可以尝试各种视觉辅助工具：

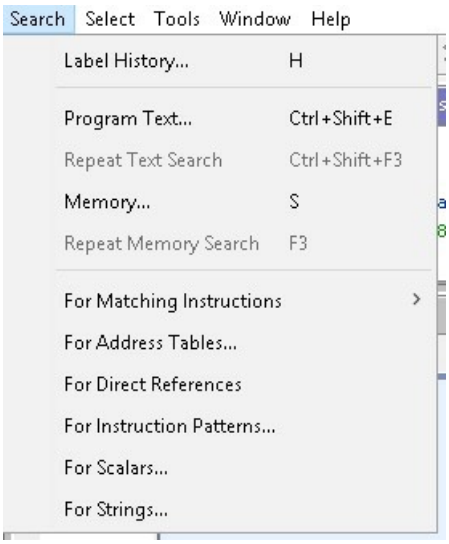


最后但并非最不重要的是，您可以在给定的基本块上选择“全屏”以更好地检查它：

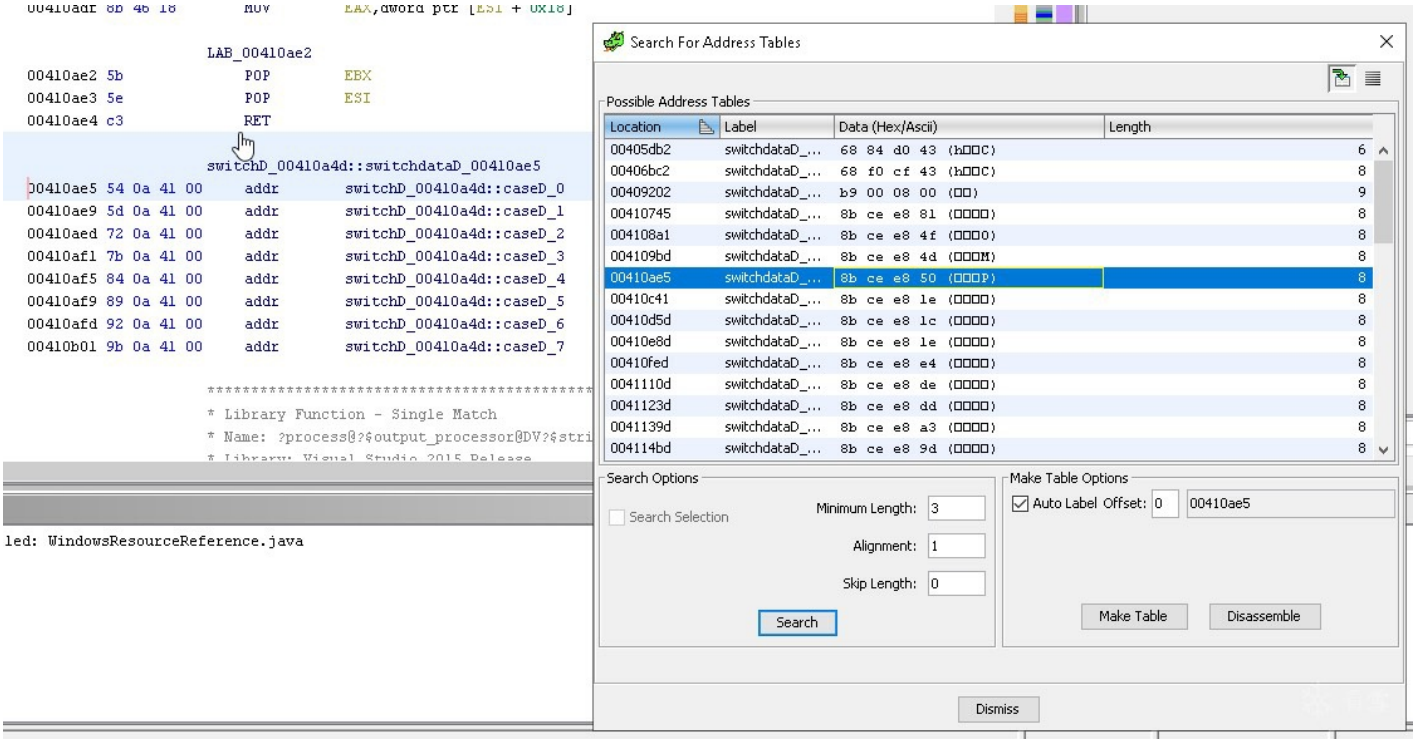


搜索功能

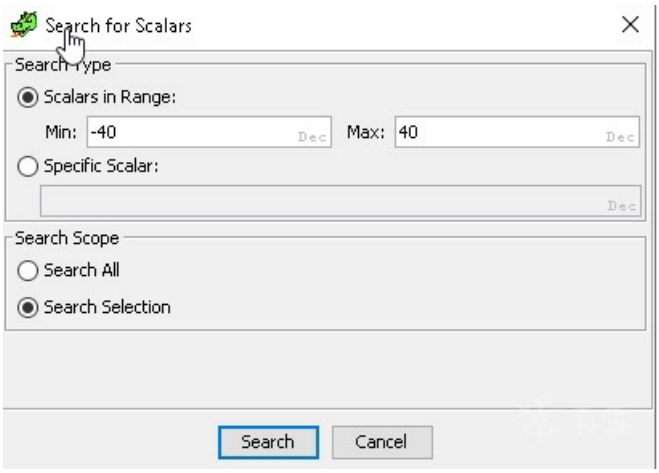
Ghidra在“搜索”菜单下提供了各种搜索功能：



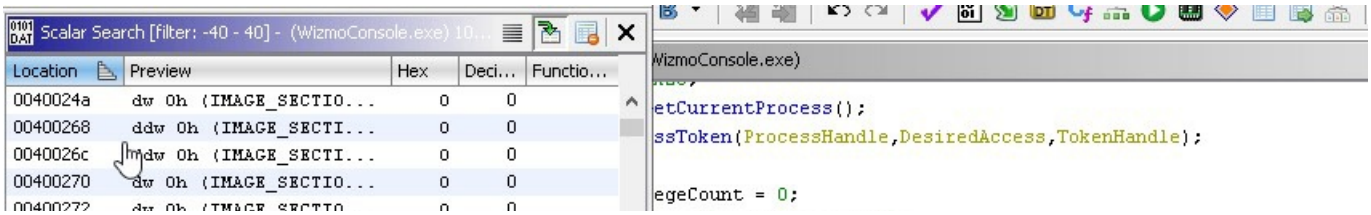
您可以搜索地址表，例如：

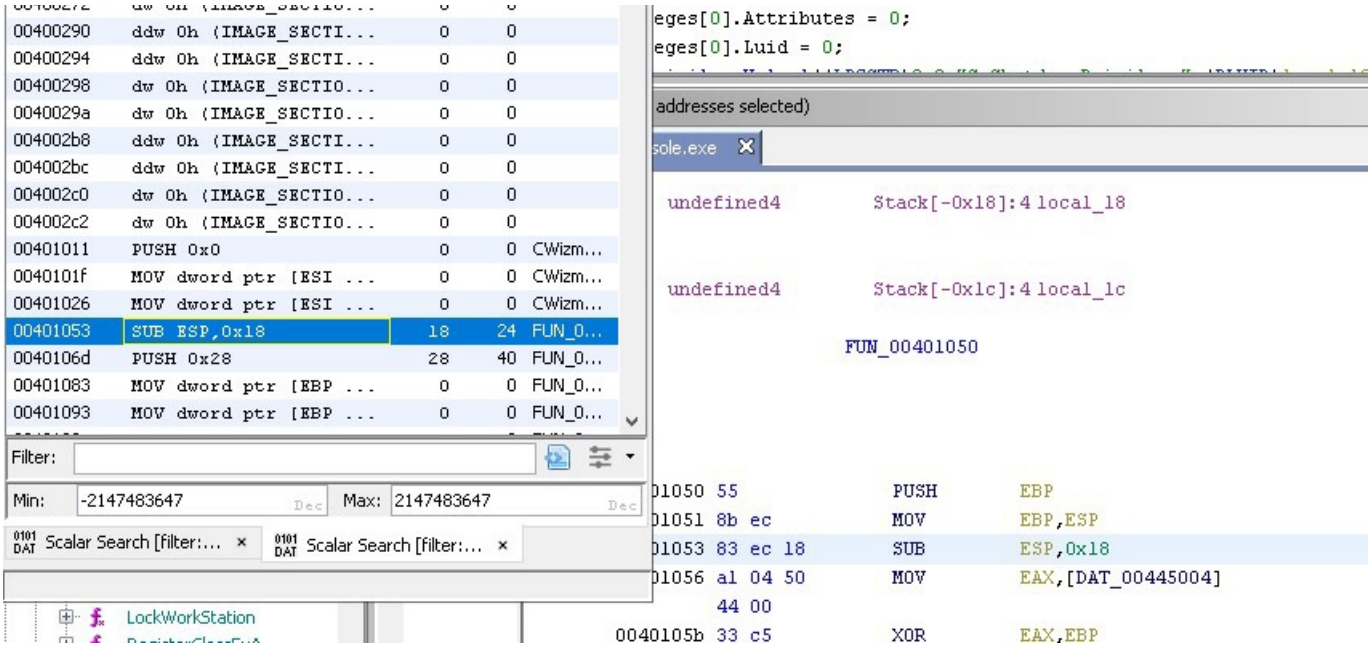


您可以同样搜索标量（IDA中的“立即数搜索”）：

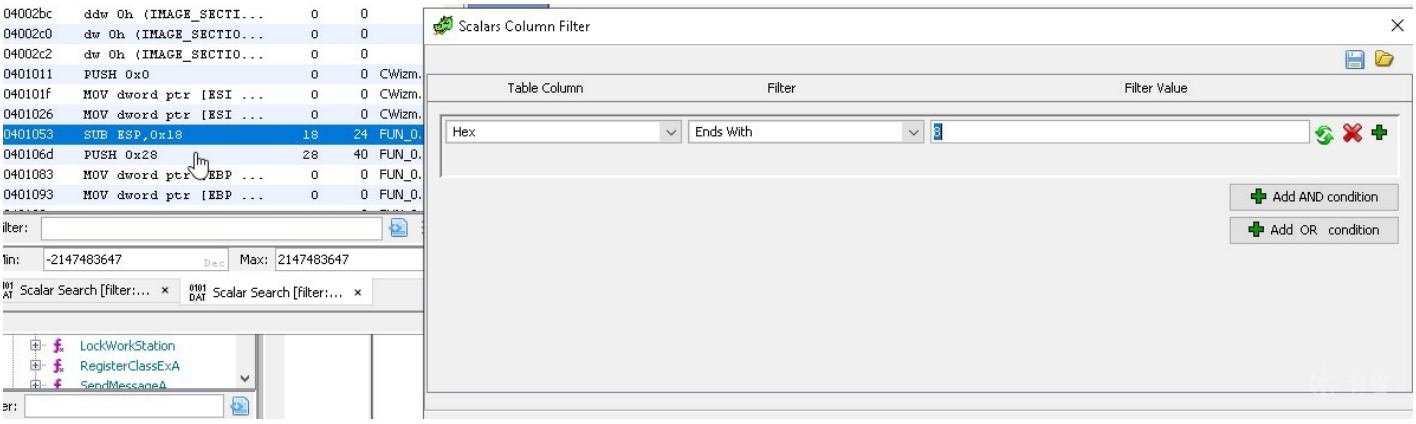


找到结果后：

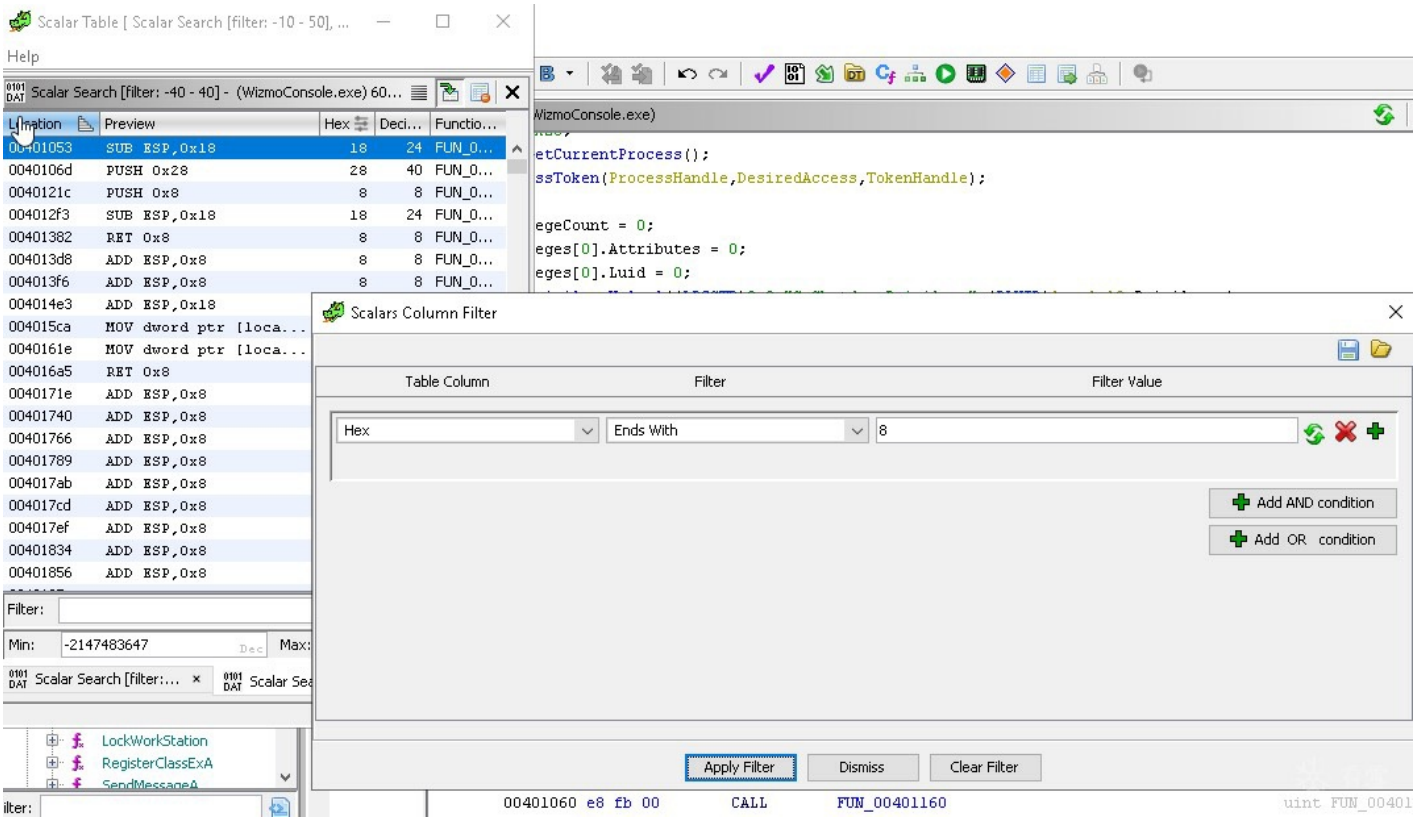




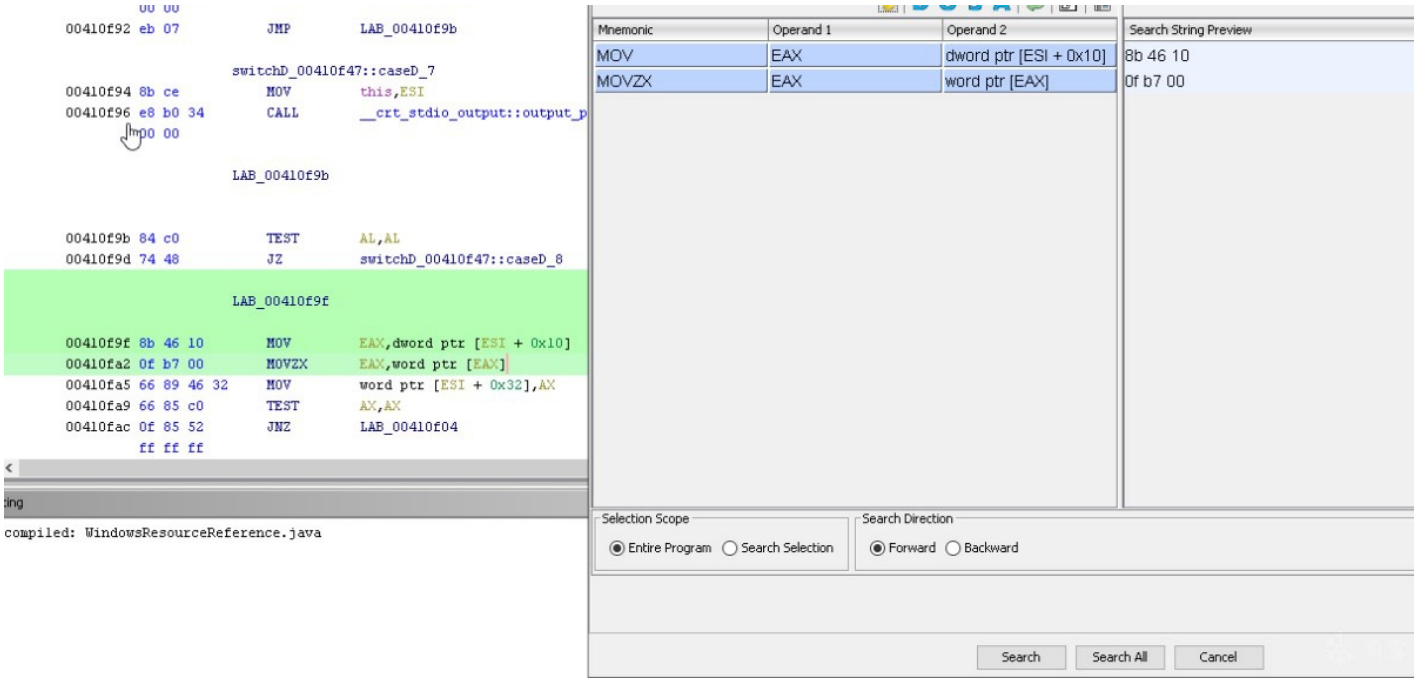
您可以应用其他过滤器：



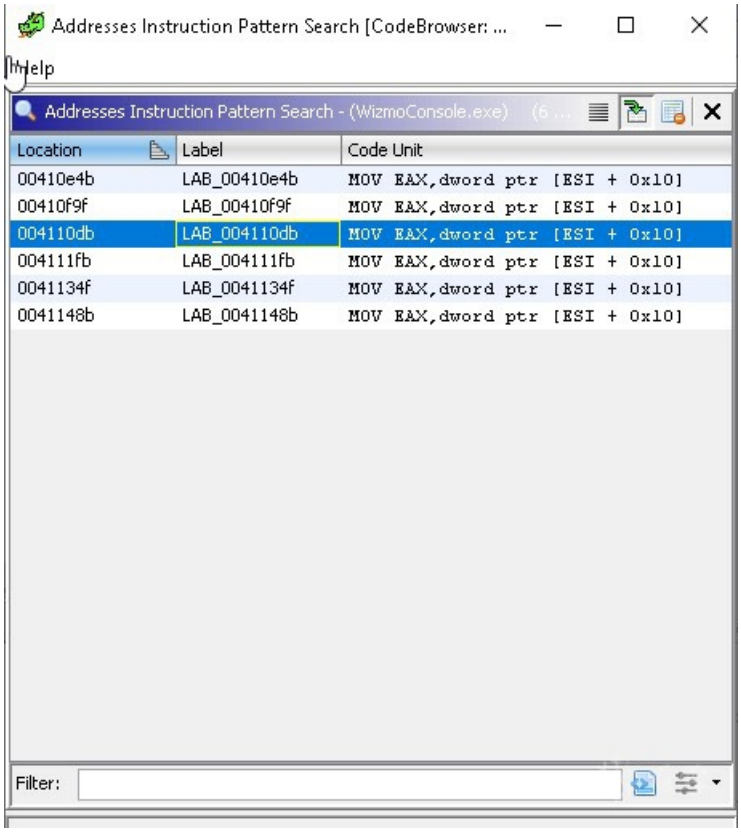
应用过滤器时，搜索结果会进一步细化：



如果要查找某些指令序列，可以从代码浏览器中选择一个或多个指令：

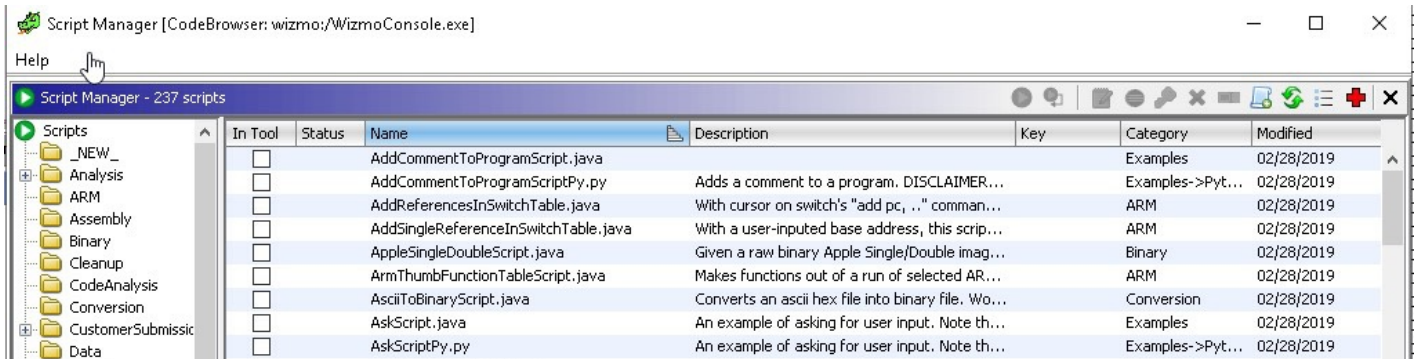


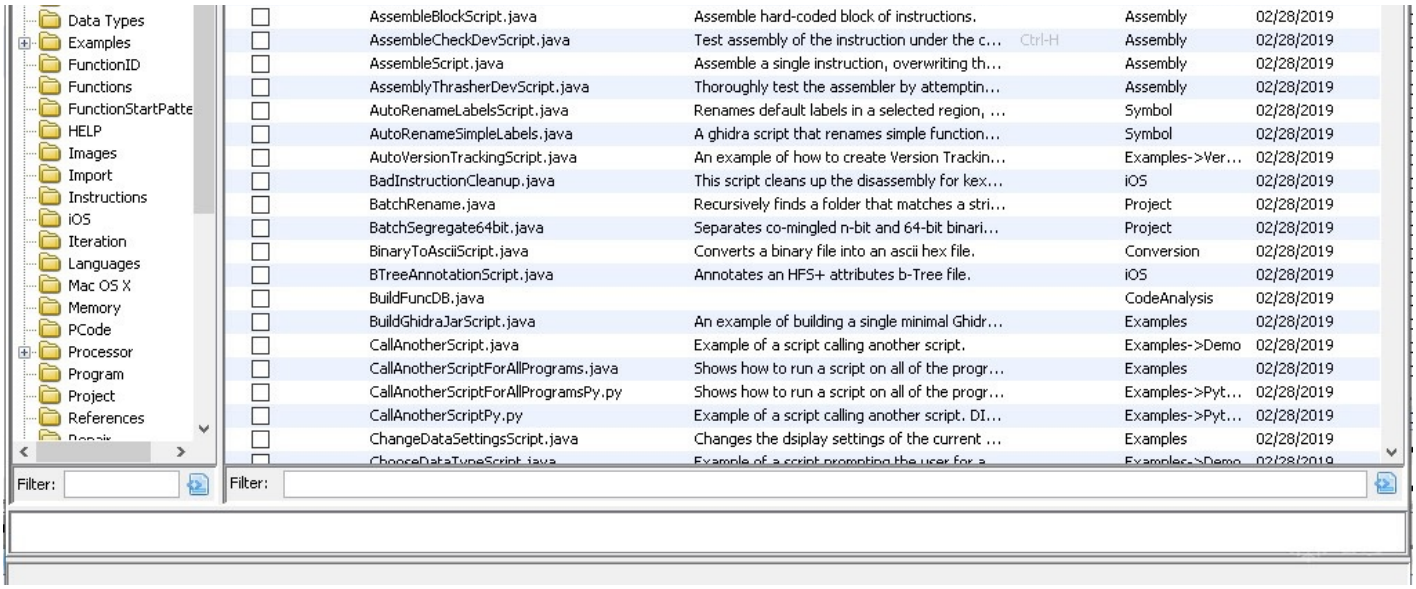
然后从搜索菜单中选择“For Instruction Pattern”以执行搜索：



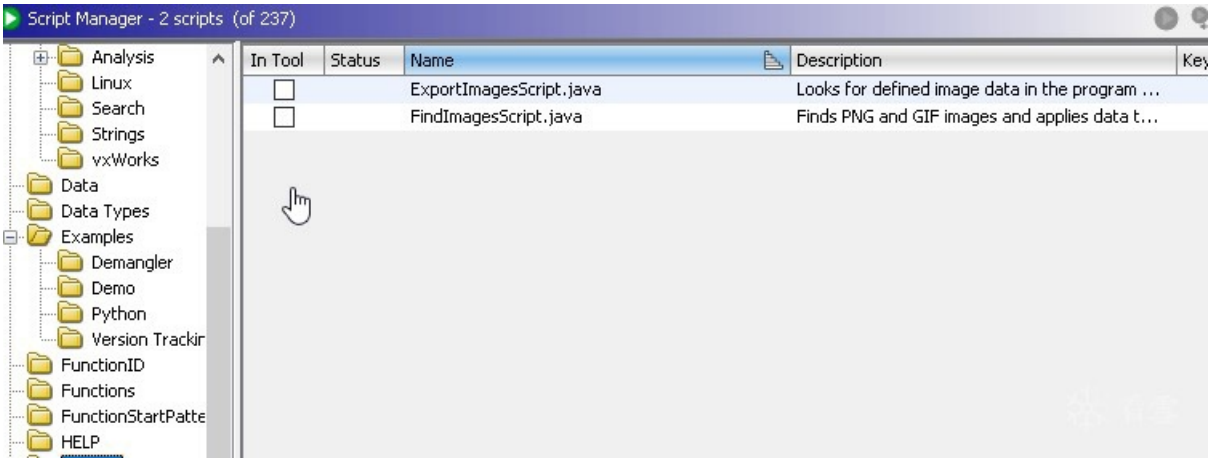
脚本功能

如果一款软件逆向工程（SRE）工具没有强大的脚本工具，那么它不能说是完整的（从“窗口/脚本管理器”菜单中选择脚本）。 开箱即用的Ghidra附带了200多个用Java编写的脚本：





例如，FindImagesScript.java脚本在输入文件中查找PNG和GIF图像：



这些脚本使用Ghidra的API：

FindImagesScript.java

```
import ghidra.program.model.listing.Data;
import ghidra.program.model.mem.Memory;
import ghidra.program.model.mem.MemoryBlock;

import java.util.ArrayList;
import java.util.List;

public class FindImagesScript extends GhidraScript {

    @Override
    public void run() throws Exception {

        int numValidImagesFound = 0;
        //Look for potential GIF images in binary using image header byte patterns
        println("Looking for GIF and PNG images in " + currentProgram.getName());
        List<Address> foundGIFS = scanForGIF87aImages();
        foundGIFS.addAll(scanForGIF89aImages());

        //Loop over all potential found GIFs
        for (int i = 0; i < foundGIFS.size(); i++) {
            boolean foundGIFImage = false;
            //See if already applied GIF
            Data data = getDataAt(foundGIFS.get(i));
            //If not already applied, try to apply GIF data type
            if (data == null) {
                println("Trying to apply GIF datatype at " + foundGIFS.get(i).toString());
                try {
                    Data newGIF = createData(foundGIFS.get(i), new GifDataType());
                    if (newGIF != null) {
                        println("Applied GIF at " + newGIF.getAddressString(false, true));
                        foundGIFImage = true;
```

```
        }
    }
```

```java
List<Address> scanForImages(byte[] imageBytes) {
    Memory memory = currentProgram.getMemory();
    MemoryBlock[] blocks = memory.getBlocks();

    byte maskBytes[] = null;

    List<Address> foundImages = new ArrayList<Address>();

    for (int i = 0; i < blocks.length; i++) {
        if (blocks[i].isInitialized()) {
            Address start = blocks[i].getStart();
            Address found = null;
            while (true) {
                if (monitor.isCancelled()) {
                    break;
                }
                found =
                    memory.findBytes(start, blocks[i].getEnd(), imageBytes, maskBytes, true,
                        monitor);
                if (found != null) {
                    foundImages.add(found);
                    start = found.add(1);
                }
                else
                    break;
            }
        }
    }
    return foundImages;
```

如果您不喜欢Java，可以使用Python（使用Jython托管）来编写脚本：

Python Interpreter

The Ghidra *Python Interpreter* provides a full general-purpose Python interactive shell and allows you to interact with your current Ghidra session by exposing Ghidra's powerful Java API through the magic of Jython.

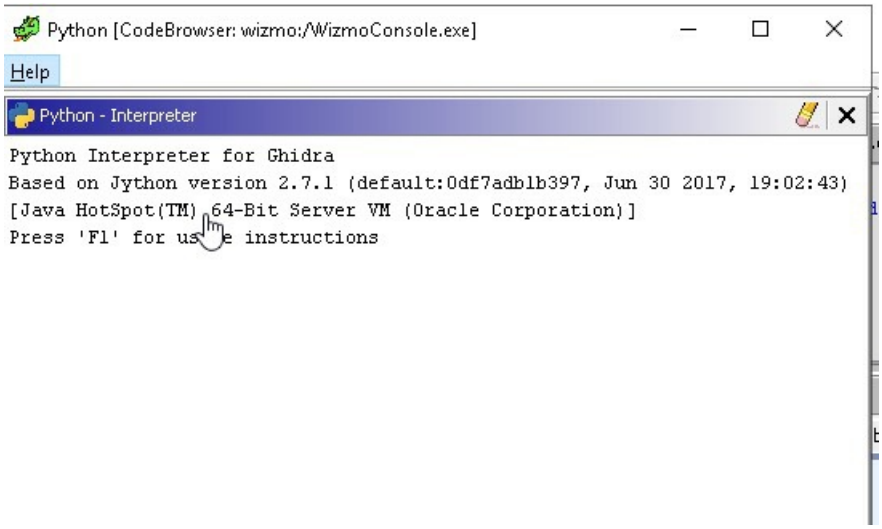
Environment

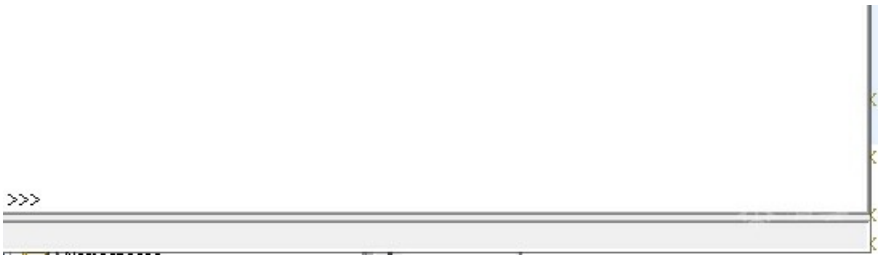
The Ghidra *Python Interpreter* is configured to run in a similar context as a Ghidra script. Therefore, you immediately have access to variables such as `currentProgram`, `currentSelection`, `currentAddress`, etc without needing to import them. These variables exist as Java objects behind the scenes, but Jython allows you to interact with them through a Python interface, which is similar to Java in some ways.

As in Java, classes outside of your current package/module need to be explicitly imported. For example, consider the following code snippet:

```
# Get a data type from the user
tool = state.getTool()
dtm = currentProgram.getDataTypeManager()
from ghidra.app.util.datatype import DataTypeSelectionDialog
from ghidra.util.data.DataTypeParser import AllowedDataTypes
selectionDialog = DataTypeSelectionDialog(tool, dtm, -1, AllowedDataTypes.FIXED_LENGTH)
tool.showDialog(selectionDialog)
dataType = selectionDialog.getUserChosenDataType()
if dataType != None: print "Chosen data type: " + str(dataType)
```

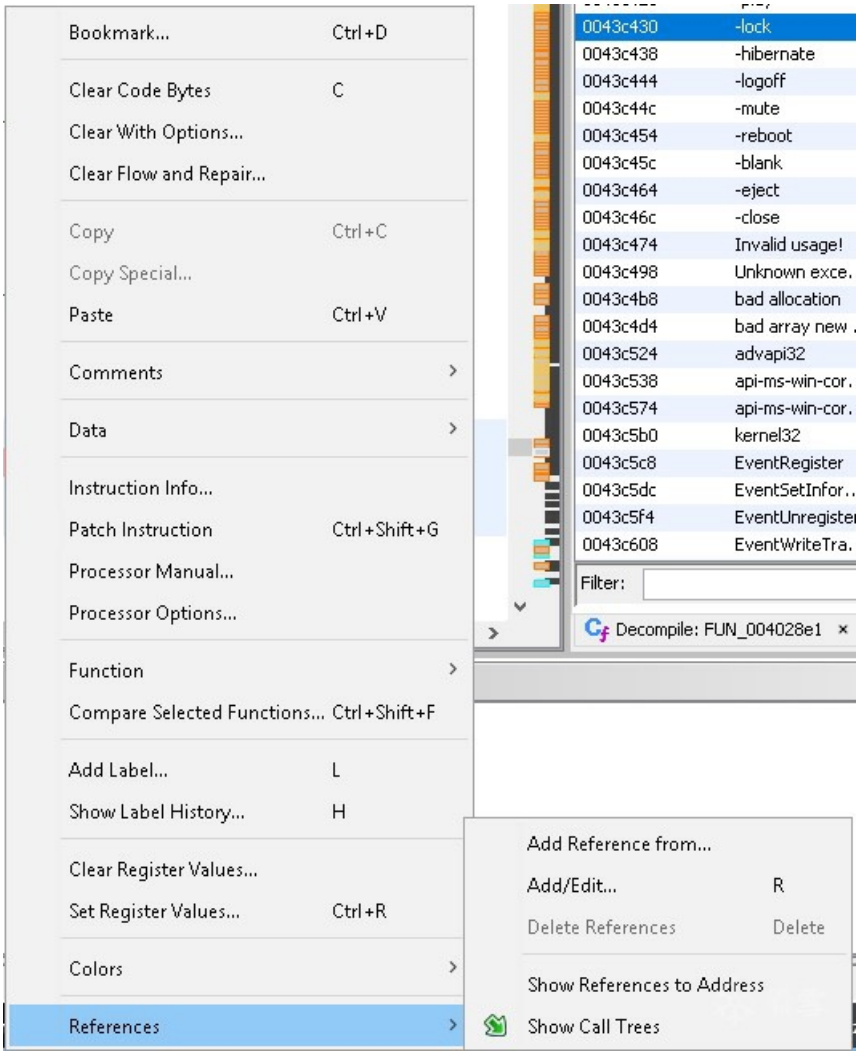
`currentProgram` and `state` are defined within the Ghidra scripting class hierarchy, so nothing has to be explicitly imported before they can be used. However, because the `DataTypeSelectionDialog` class and `AllowedDataType` enum reside in different packages, they must be explicitly imported. Failure to do so will result in a Python `NameError`.



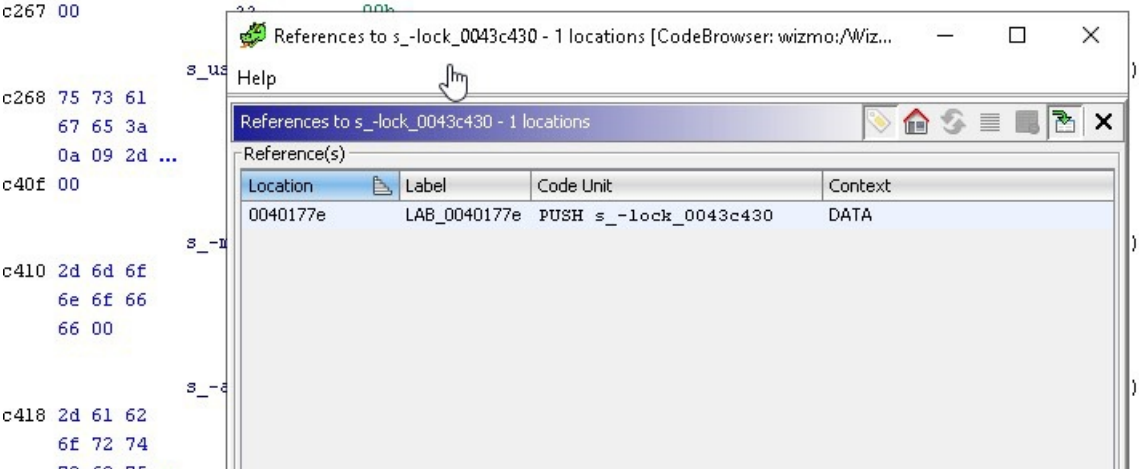


杂项功能

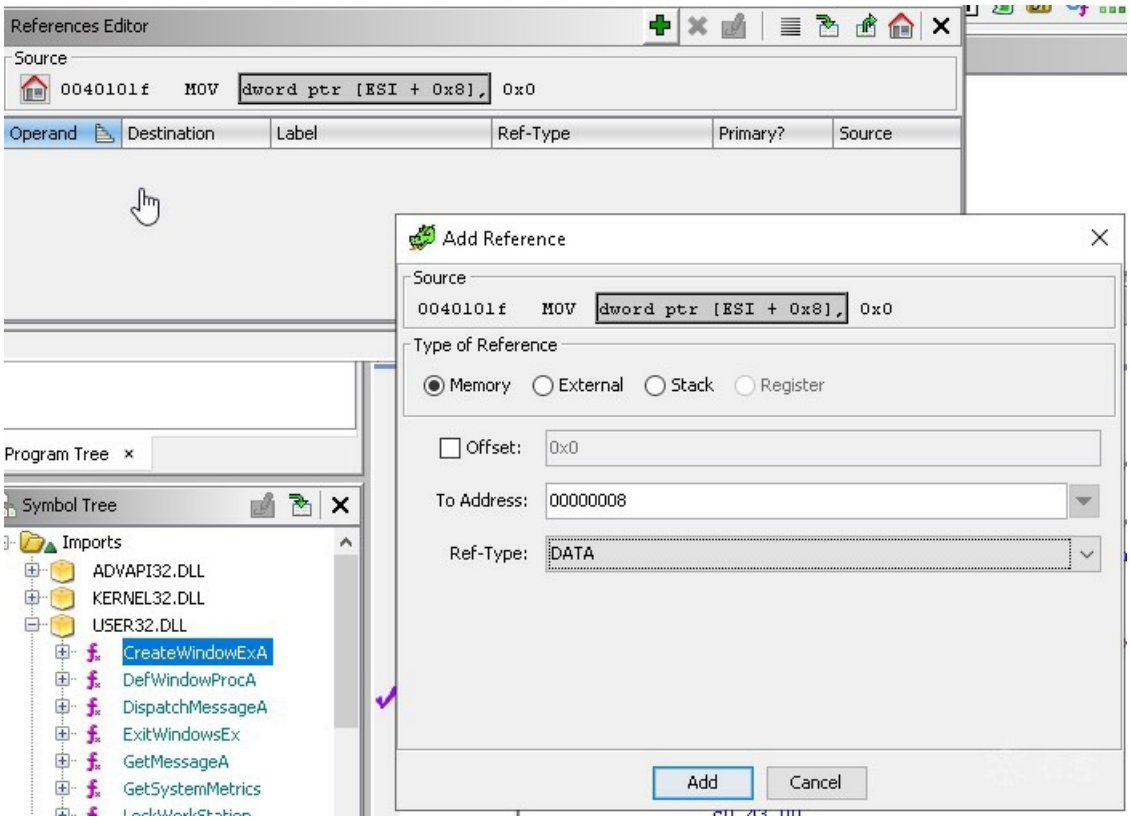
Ghidra还有许多值得一提的杂项功能。
让我们从交叉引用功能开始。
您可以让Ghidra计算几乎任何项目（字符串，指令，寄存器等）的交叉引用。
例如，我们正在寻找字符串窗口中给定字符串的交叉引用：



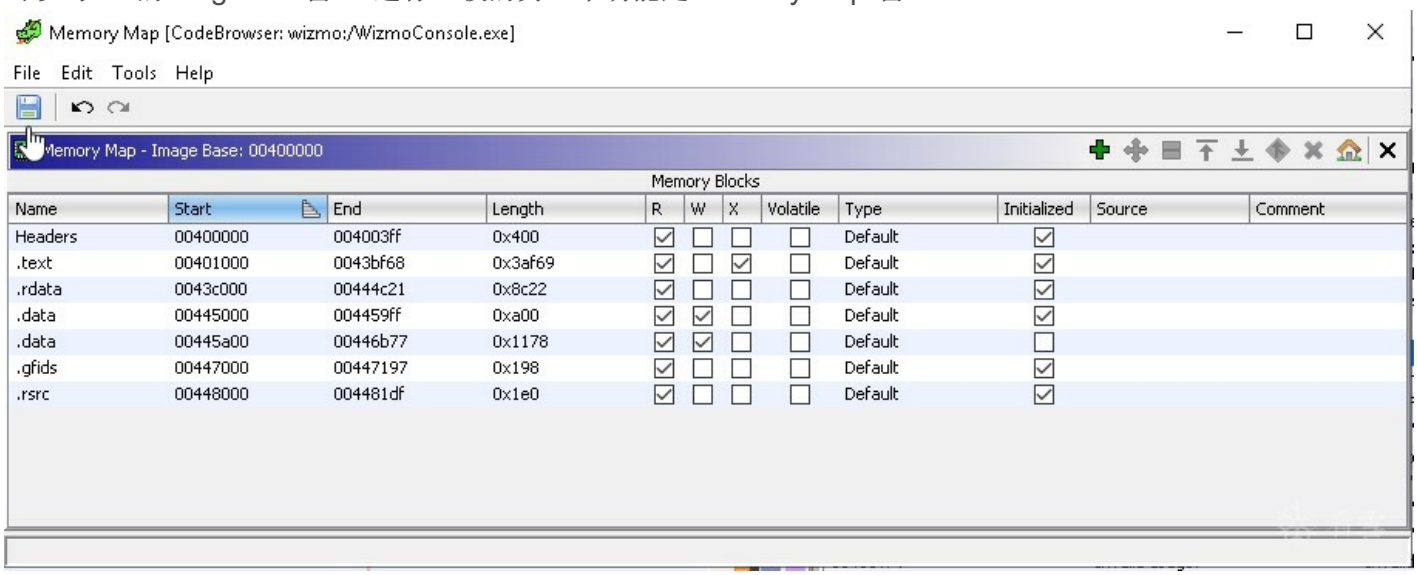
使用字符串交叉引用，您可以发现恶意字符串或找到引用/实现某些功能的代码（基于您找到的字符串文本）。



与IDA一样，您可以手动创建交叉引用：

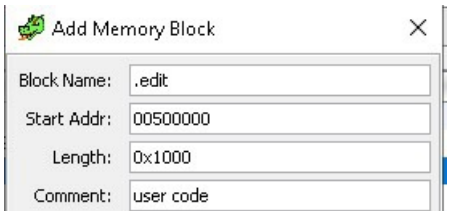


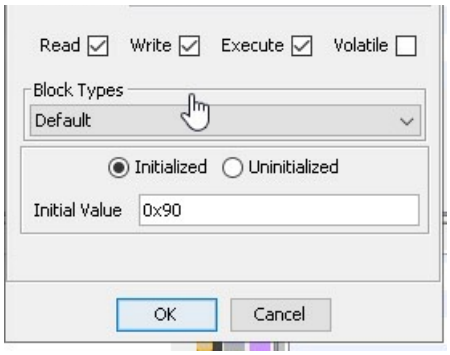
可以与IDA的“Segments窗口”进行比较的另一个功能是“Memory map”窗口：



在内存映射中，您可以看到程序分区（如果输入文件包含分区，如PE或ELF文件）。

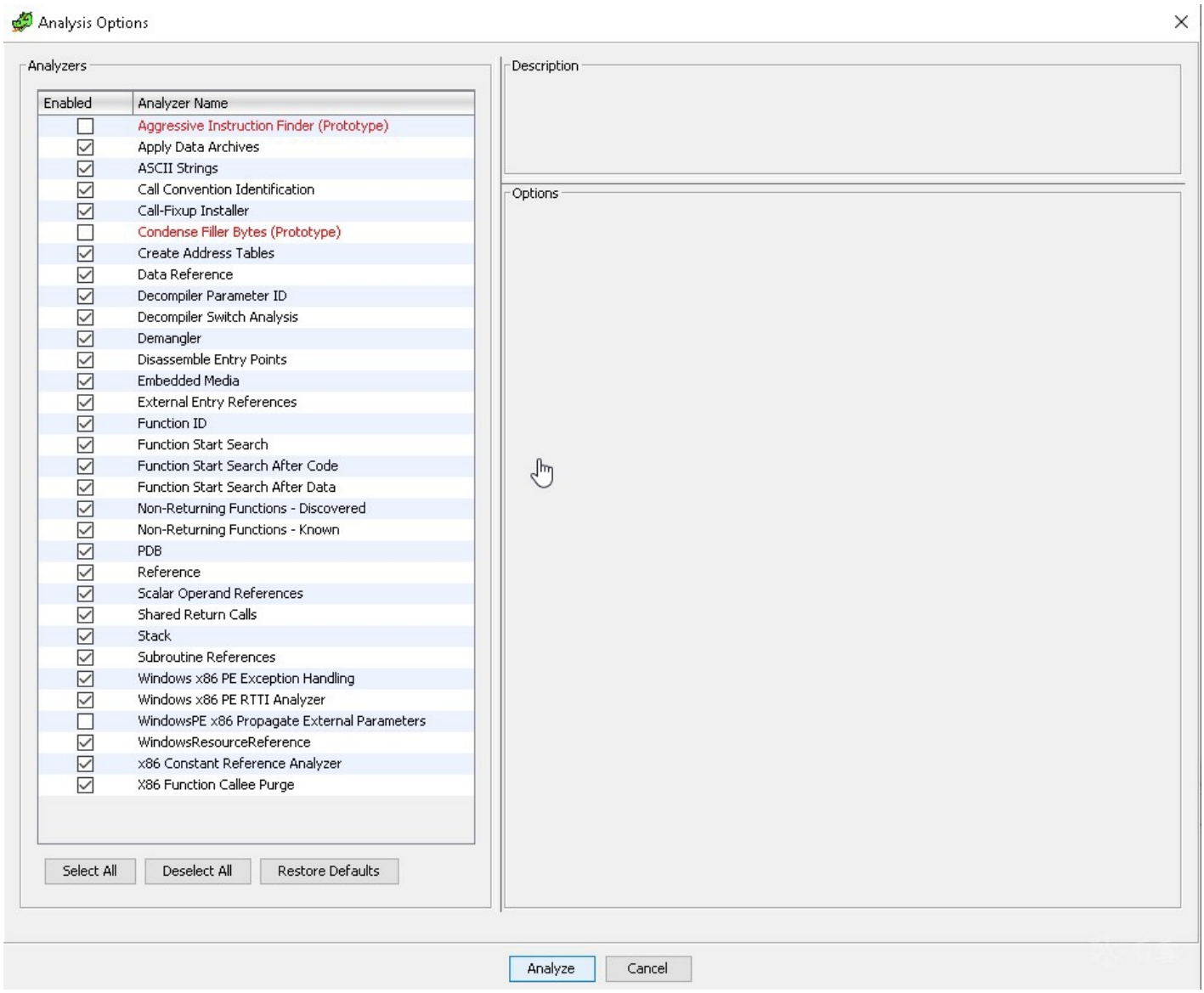
此外，您可以手动创建新分区：

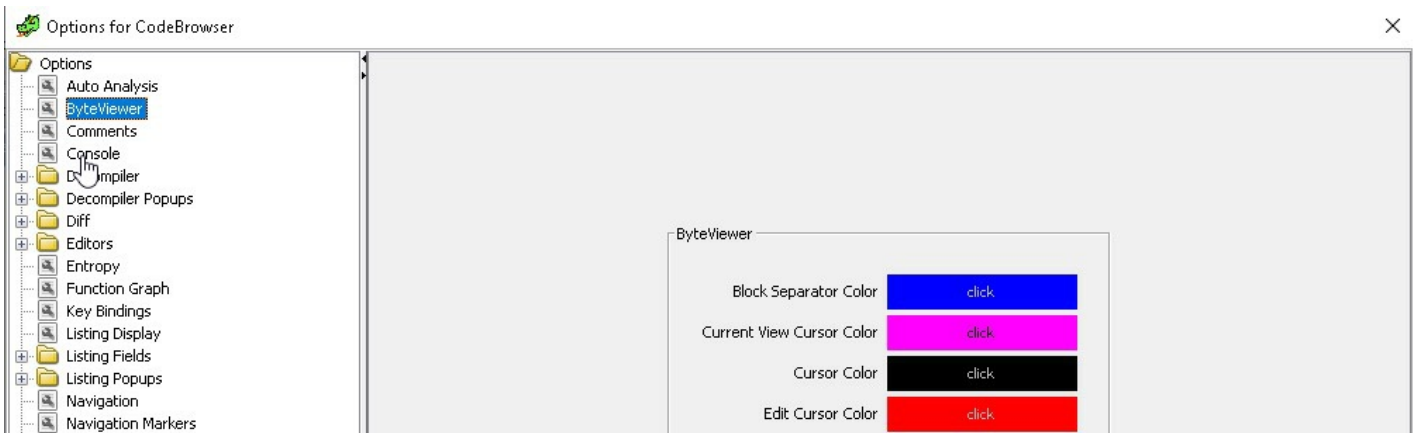


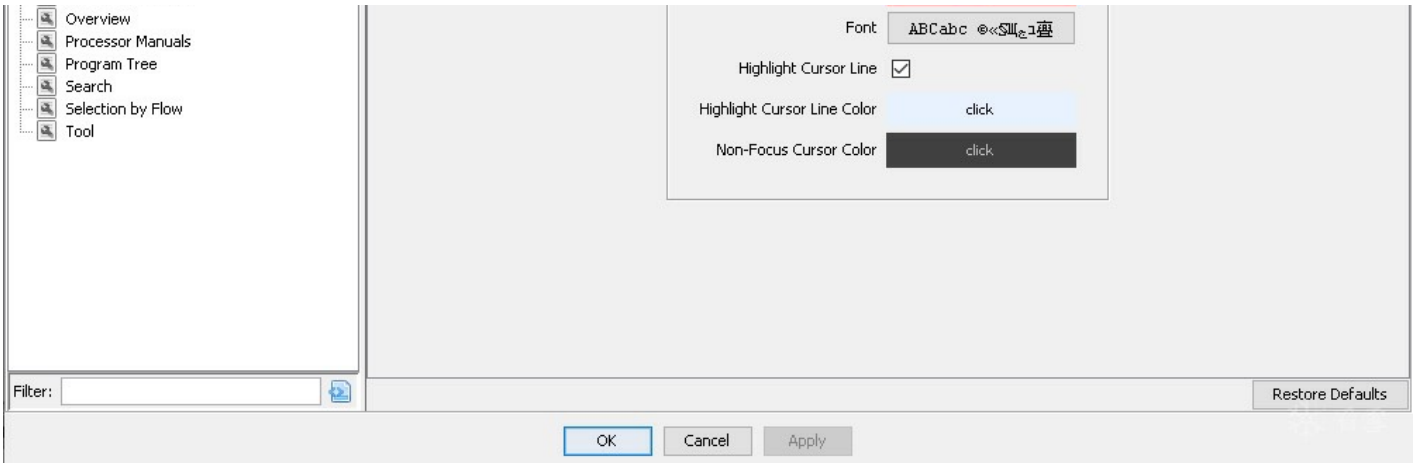


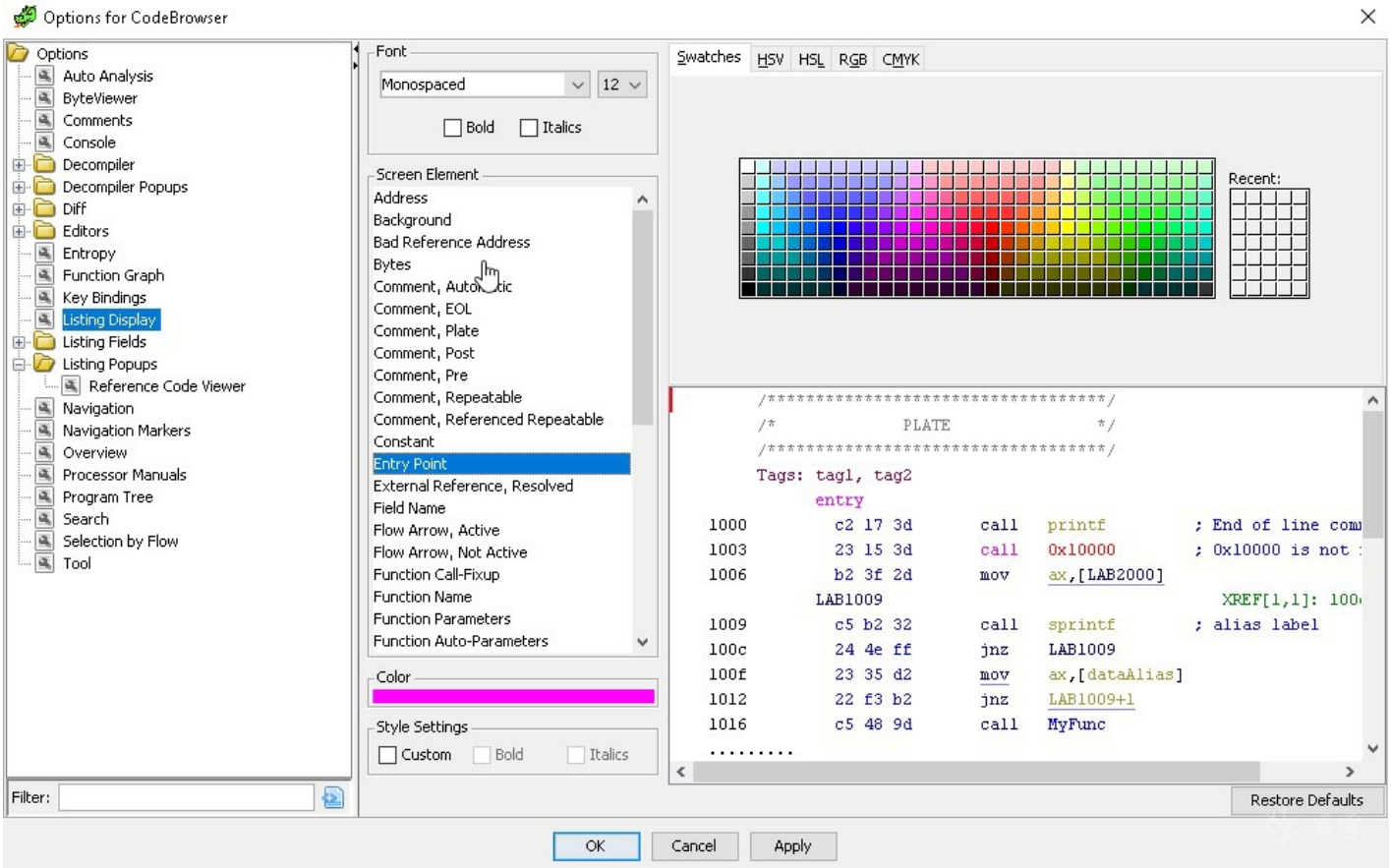
选项

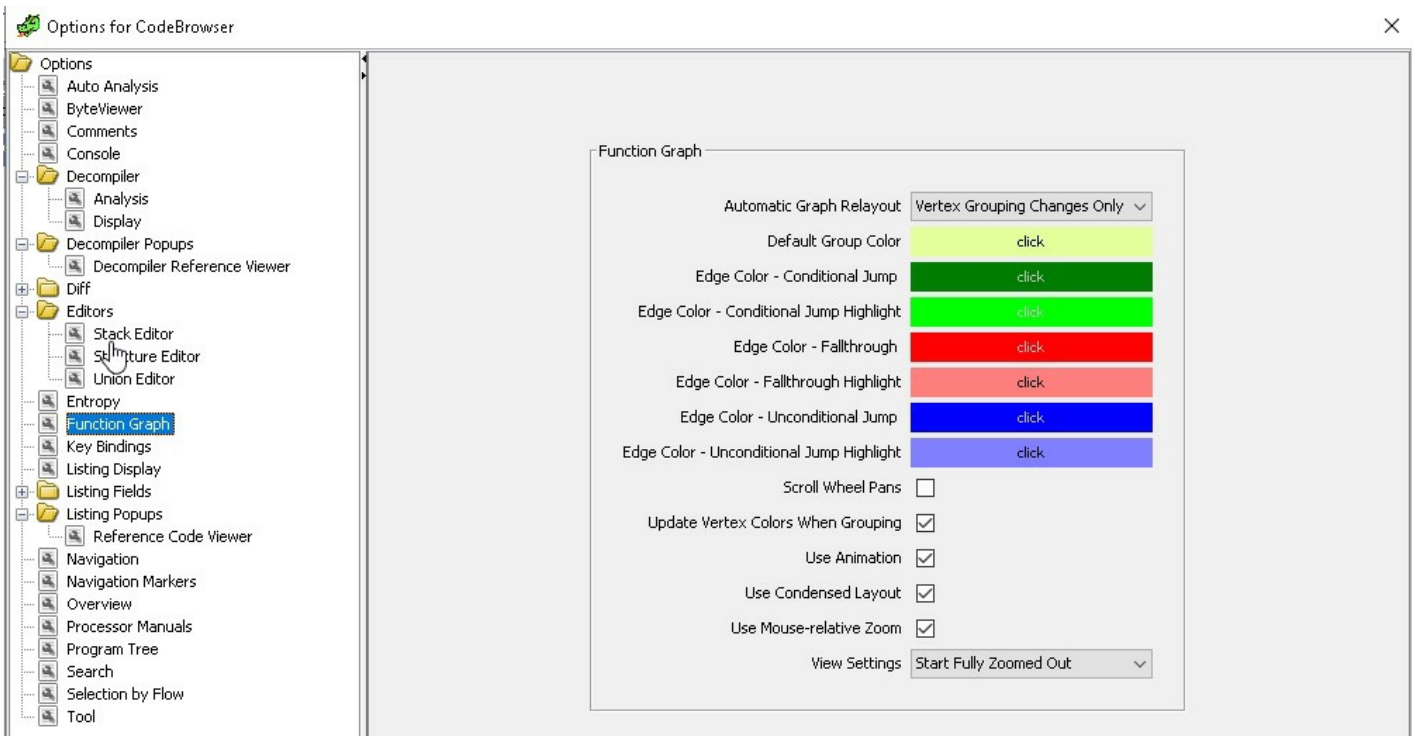
几乎任何东西都可以通过选项功能在Ghidra中配置：

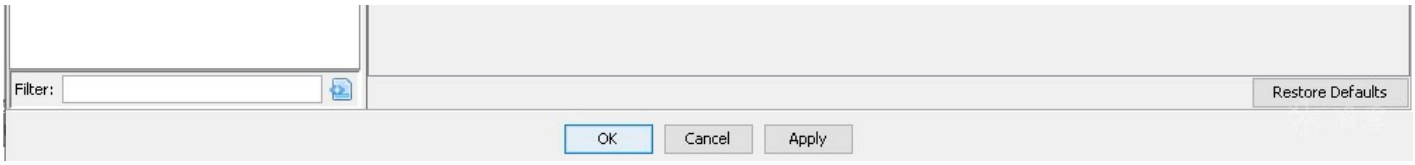





Overview
Processor Manuals
Program Tree
Search
Selection by Flow
Tool
Font
ABCabc
Highlight Cursor Line
Highlight Cursor Line Color
click
Non-Focus Cursor Color
click
Filter:
Restore Defaults
OK
Cancel
Apply

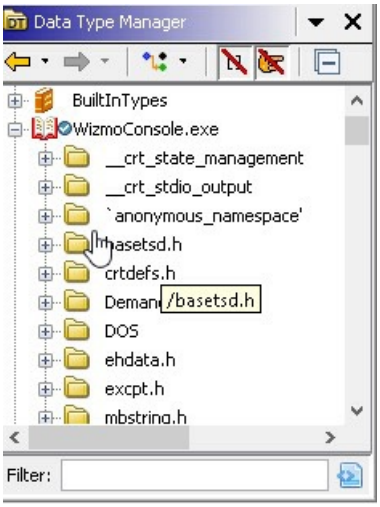

Options for CodeBrowser
Options
Auto Analysis
ByteViewer
Comments
Console
Decompiler
Decompiler Popups
Diff
Editors
Entropy
Function Graph
Key Bindings
Listing Display
Listing Fields
Listing Popups
Reference Code Viewer
Navigation
Navigation Markers
Overview
Processor Manuals
Program Tree
Search
Selection by Flow
Tool
Font
Monospaced
12
Bold
Italics
Screen Element
Address
Background
Bad Reference Address
Bytes
Comment, Automatic
Comment, EOL
Comment, Plate
Comment, Post
Comment, Pre
Comment, Repeatable
Comment, Referenced Repeatable
Constant
Entry Point
External Reference, Resolved
Field Name
Flow Arrow, Active
Flow Arrow, Not Active
Function Call-Fixup
Function Name
Function Parameters
Function Auto-Parameters
Color
Style Settings
Custom
Bold
Italics
Swatches
HSV
HSL
RGB
CMYK
Recent:
/************************************/
/* PLATE */
/************************************/
Tags: tag1, tag2
entry
1000 c2 17 3d call printf ; End of line com
1003 23 15 3d call 0x10000 ; 0x10000 is not
1006 b2 3f 2d mov ax,[LAB2000]
LAB1009 XREF[1,1]: 100
1009 c5 b2 32 call sprintf ; alias label
100c 24 4e ff jnz LAB1009
100f 23 35 d2 mov ax,[dataAlias]
1012 22 f3 b2 jnz LAB1009+1
1016 c5 48 9d call MyFunc
.....
Filter:
Restore Defaults
OK
Cancel
Apply

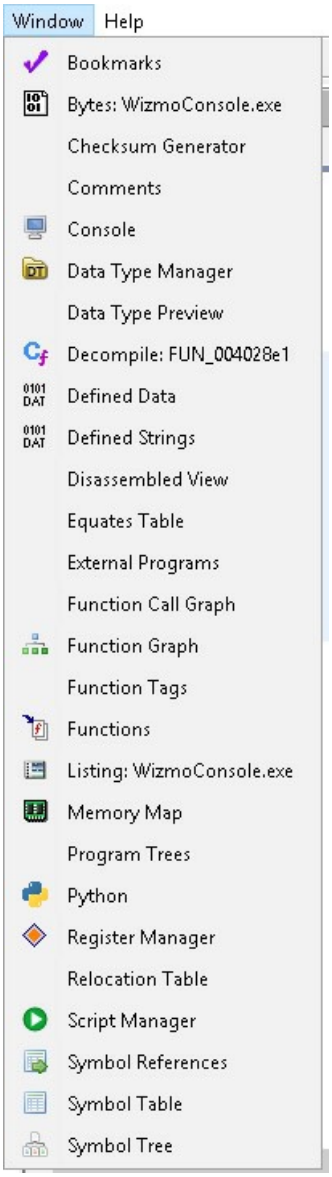

Options for CodeBrowser
Options
Auto Analysis
ByteViewer
Comments
Console
Decompiler
Analysis
Display
Decompiler Popups
Decompiler Reference Viewer
Diff
Editors
Stack Editor
Structure Editor
Union Editor
Entropy
Function Graph
Key Bindings
Listing Display
Listing Fields
Listing Popups
Reference Code Viewer
Navigation
Navigation Markers
Overview
Processor Manuals
Program Tree
Search
Selection by Flow
Tool
Function Graph
Automatic Graph Relayout
Vertex Grouping Changes Only
Default Group Color
click
Edge Color - Conditional Jump
click
Edge Color - Conditional Jump Highlight
click
Edge Color - Fallthrough
click
Edge Color - Fallthrough Highlight
click
Edge Color - Unconditional Jump
click
Edge Color - Unconditional Jump Highlight
click
Scroll Wheel Pans
Update Vertex Colors When Grouping
Use Animation
Use Condensed Layout
Use Mouse-relative Zoom
View Settings
Start Fully Zoomed Out

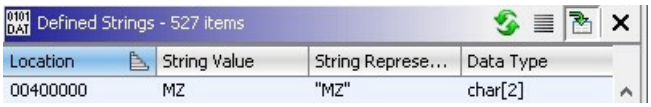


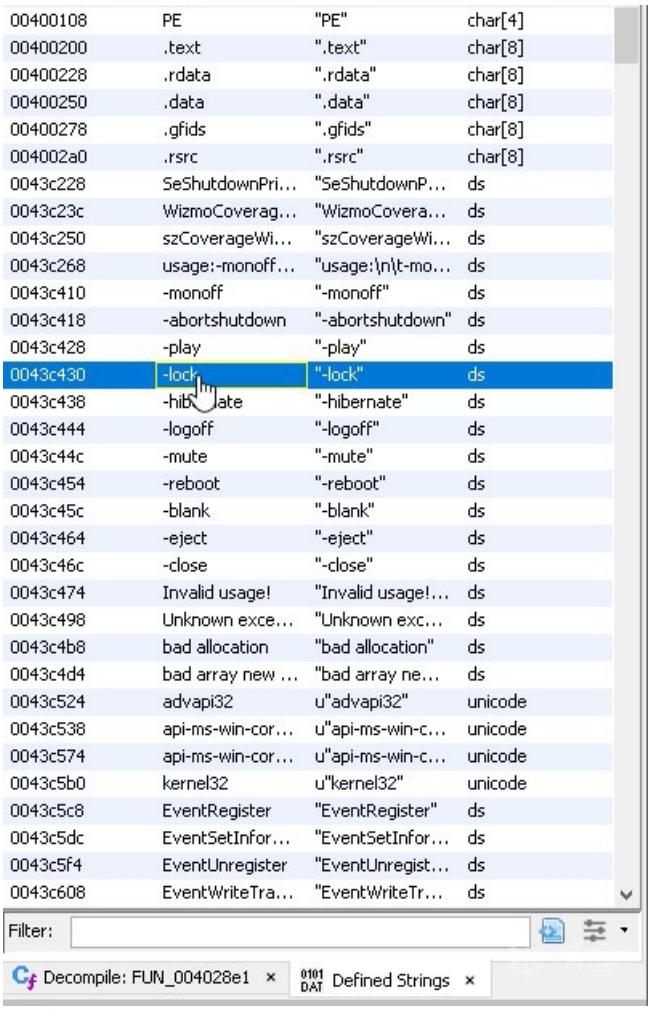
其他截图

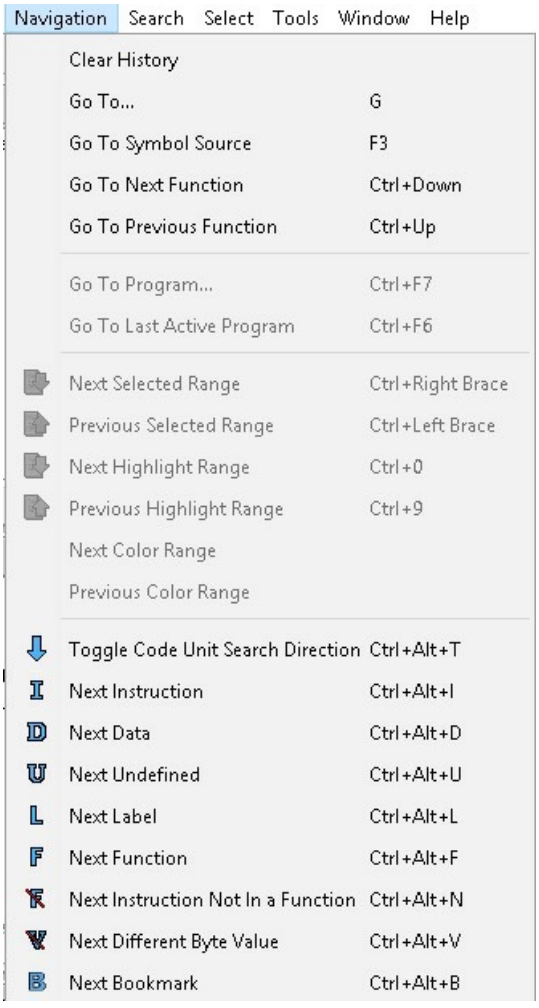
以下是Ghidra的一些其他截图：

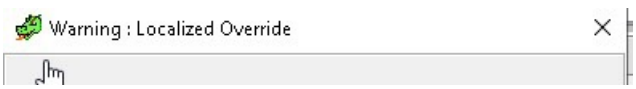


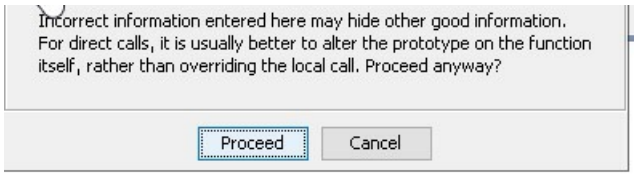


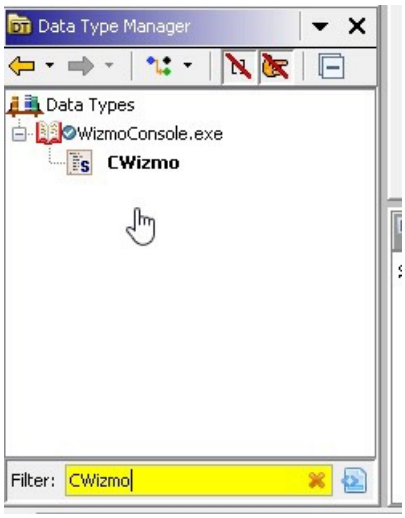


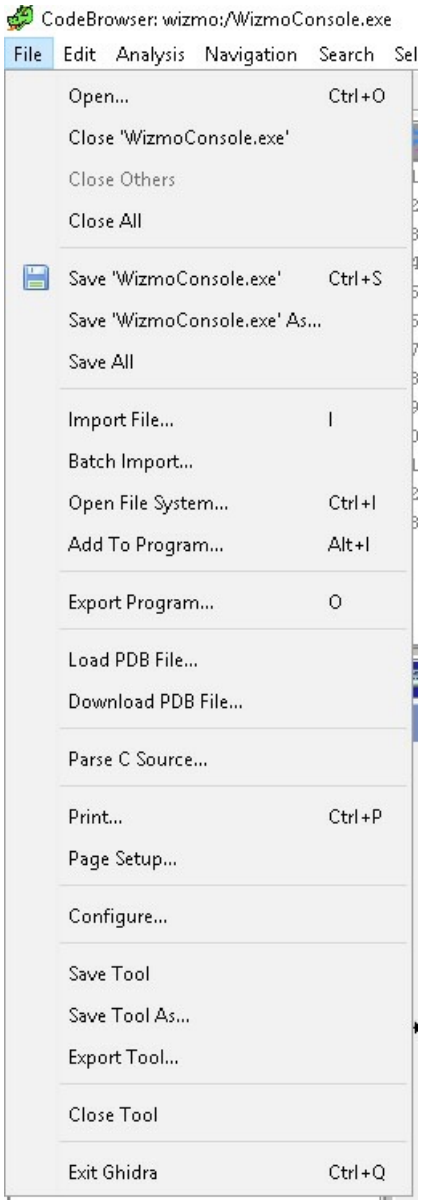

00400108 PE "PE" char[4]
00400200 .text ".text" char[8]
00400228 .rdata ".rdata" char[8]
00400250 .data ".data" char[8]
00400278 .gfids ".gfids" char[8]
004002a0 .rsrc ".rsrc" char[8]
0043c228 SeShutdownPri... "SeShutdownP... ds
0043c23c WizmoCoverag... "WizmoCovera... ds
0043c250 szCoverageWi... "szCoverageWi... ds
0043c268 usage:-monoff... "usage:\n\t-mo... ds
0043c410 -monoff "-monoff" ds
0043c418 -abortshutdown "-abortshutdown" ds
0043c428 -play "-play" ds
0043c430 -lock "-lock" ds
0043c438 -hibernate "-hibernate" ds
0043c444 -logoff "-logoff" ds
0043c44c -mute "-mute" ds
0043c454 -reboot "-reboot" ds
0043c45c -blank "-blank" ds
0043c464 -eject "-eject" ds
0043c46c -close "-close" ds
0043c474 Invalid usage! "Invalid usage!... ds
0043c498 Unknown exce... "Unknown exc... ds
0043c4b8 bad allocation "bad allocation" ds
0043c4d4 bad array new ... "bad array ne... ds
0043c524 advapi32 u"advapi32" unicode
0043c538 api-ms-win-cor... u"api-ms-win-c... unicode
0043c574 api-ms-win-cor... u"api-ms-win-c... unicode
0043c5b0 kernel32 u"kernel32" unicode
0043c5c8 EventRegister "EventRegister" ds
0043c5dc EventSetInfor... "EventSetInfor... ds
0043c5f4 EventUnregister "EventUnregist... ds
0043c608 EventWriteTra... "EventWriteTr... ds
Filter:
Decompile: FUN_004028e1
Defined Strings


Navigation Search Select Tools Window Help
Clear History
Go To... G
Go To Symbol Source F3
Go To Next Function Ctrl+Down
Go To Previous Function Ctrl+Up
Go To Program... Ctrl+F7
Go To Last Active Program Ctrl+F6
Next Selected Range Ctrl+Right Brace
Previous Selected Range Ctrl+Left Brace
Next Highlight Range Ctrl+0
Previous Highlight Range Ctrl+9
Next Color Range
Previous Color Range
Toggle Code Unit Search Direction Ctrl+Alt+T
Next Instruction Ctrl+Alt+I
Next Data Ctrl+Alt+D
Next Undefined Ctrl+Alt+U
Next Label Ctrl+Alt+L
Next Function Ctrl+Alt+F
Next Instruction Not In a Function Ctrl+Alt+N
Next Different Byte Value Ctrl+Alt+V
Next Bookmark Ctrl+Alt+B

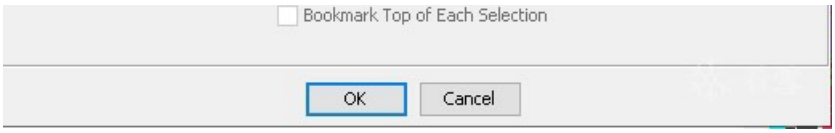

Warning : Localized Override

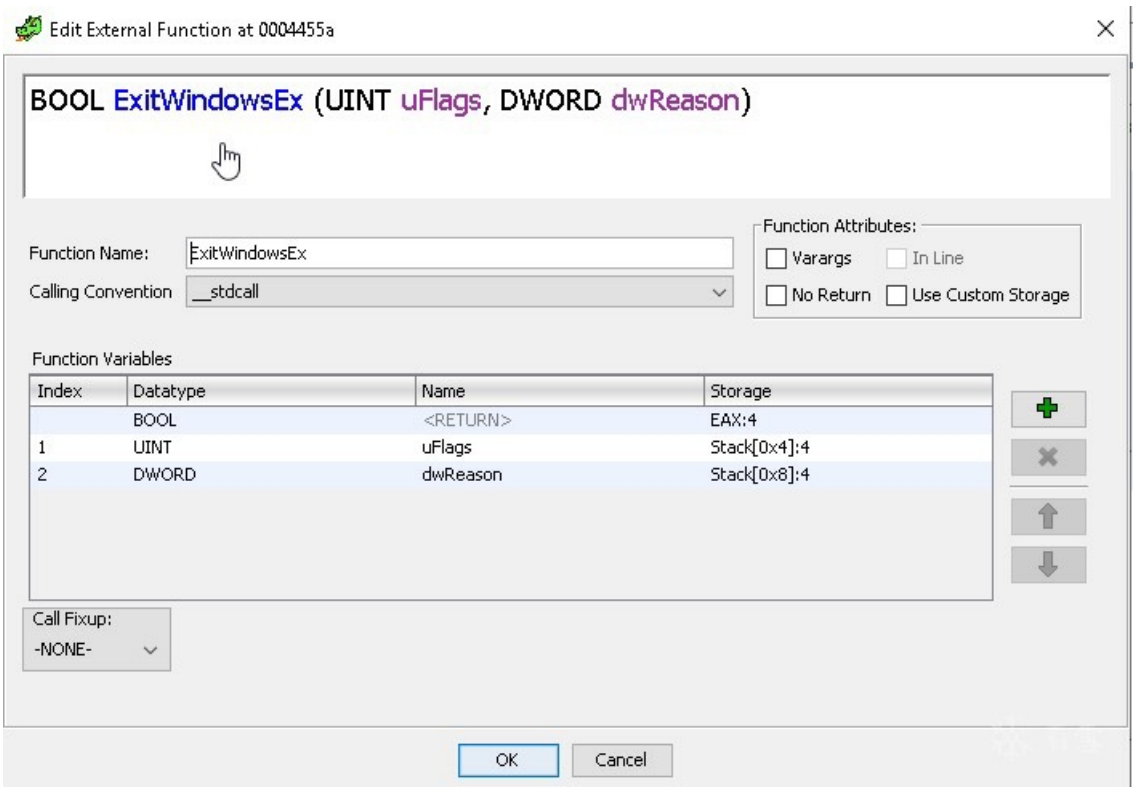

Incorrect information entered here may hide other good information.
For direct calls, it is usually better to alter the prototype on the function itself, rather than overriding the local call. Proceed anyway?
Proceed
Cancel

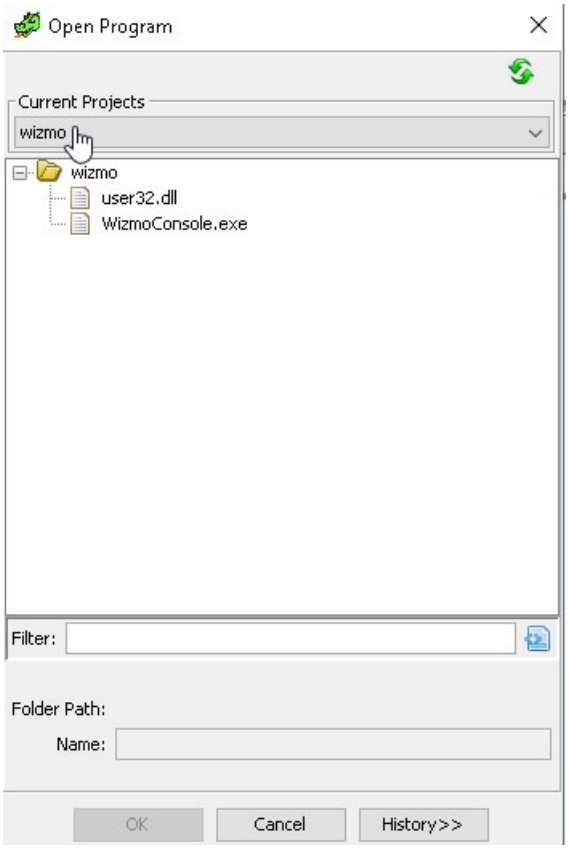

Data Type Manager
Data Types
WizmoConsole.exe
CWizmo
Filter: CWizmo

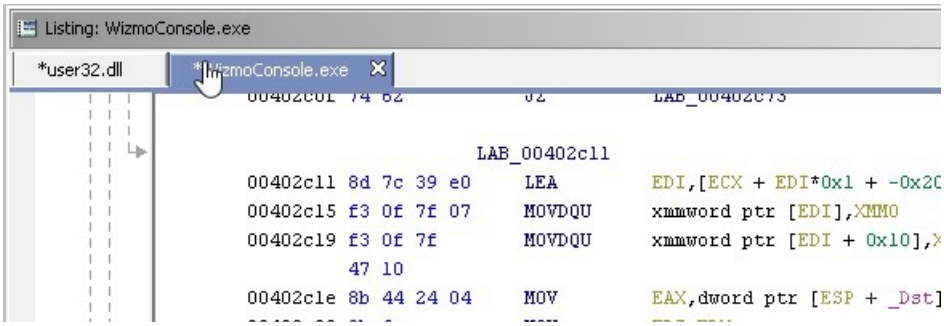

CodeBrowser: wizmo:/WizmoConsole.exe
File Edit Analysis Navigation Search Sel
Open... Ctrl+O
Close 'WizmoConsole.exe'
Close Others
Close All
Save 'WizmoConsole.exe' Ctrl+S
Save 'WizmoConsole.exe' As...
Save All
Import File... I
Batch Import...
Open File System... Ctrl+I
Add To Program... Alt+I
Export Program... O
Load PDB File...
Download PDB File...
Parse C Source...
Print... Ctrl+P
Page Setup...
Configure...
Save Tool
Save Tool As...
Export Tool...
Close Tool
Exit Ghidra Ctrl+Q

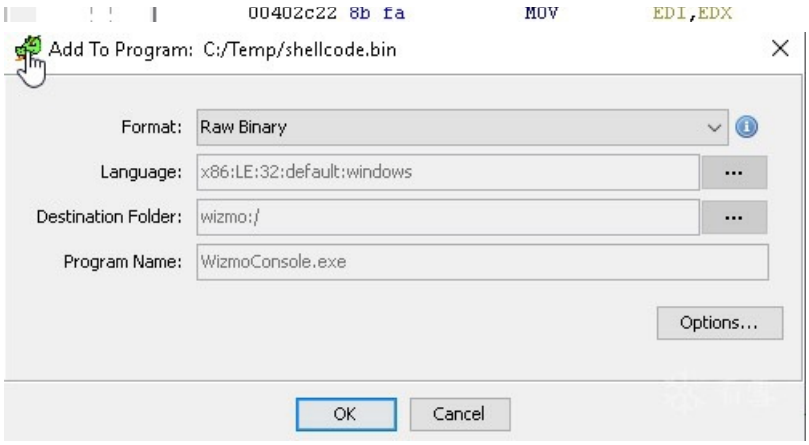

Note Bookmark
Address: 00401010
Category:
Description: Constructor

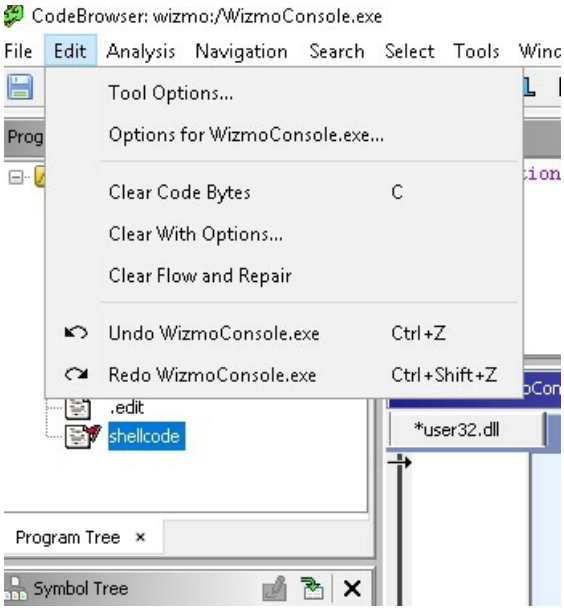

Bookmark Top of Each Selection
OK
Cancel

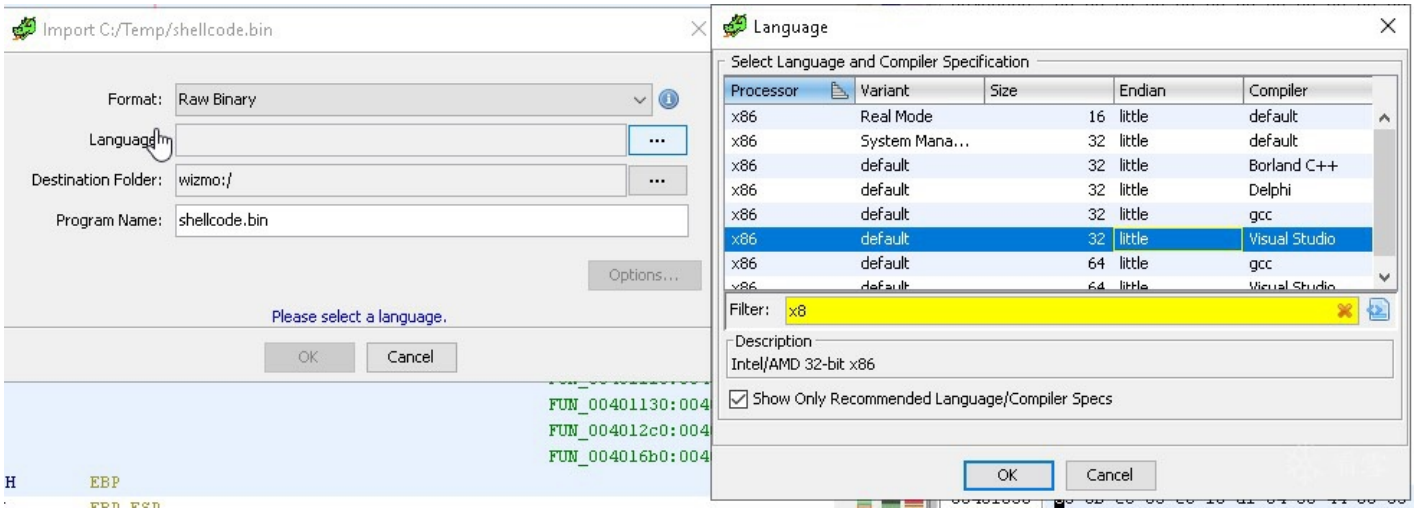

Edit External Function at 0004455a
BOOL ExitWindowsEx (UINT uFlags, DWORD dwReason)
Function Name: ExitWindowsEx
Calling Convention __stdcall
Function Attributes:
Varargs
In Line
No Return
Use Custom Storage
Function Variables
Index Datatype Name Storage
BOOL <RETURN> EAX:4
1 UINT uFlags Stack[0x4]:4
2 DWORD dwReason Stack[0x8]:4
Call Fixup:
-NONE-
OK
Cancel

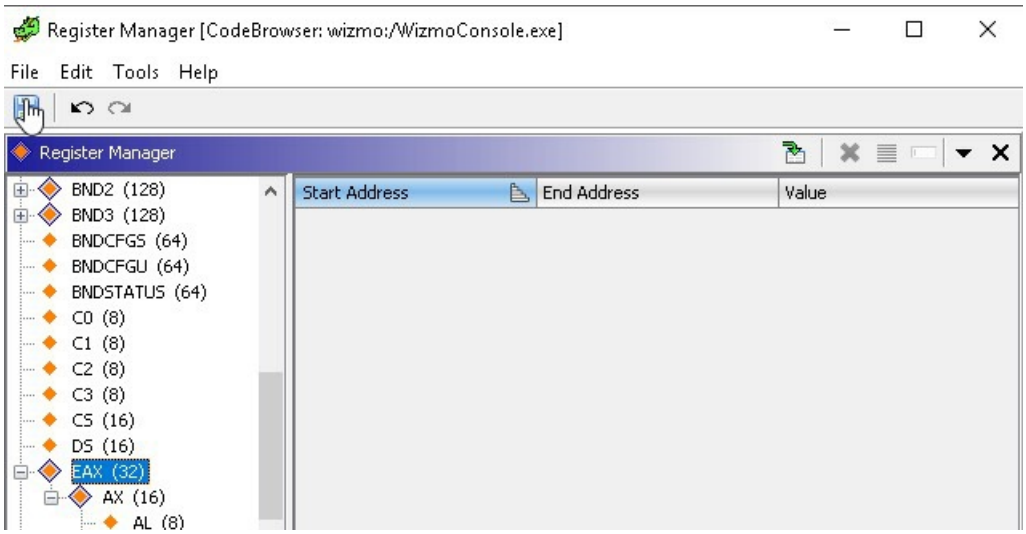

Open Program
Current Projects
wizmo
wizmo
user32.dll
WizmoConsole.exe
Filter:
Folder Path:
Name:
OK
Cancel
History>>


Listing: WizmoConsole.exe
*user32.dll
*WizmoConsole.exe
LAB_00402c11
00402c11 8d 7c 39 e0 LEA EDI,[ECX + EDI*0x1 + -0x2
00402c15 f3 0f 7f 07 MOVDQU xmmword ptr [EDI],XMM0
00402c19 f3 0f 7f MOVDQU xmmword ptr [EDI + 0x10],
47 10
00402c1e 8b 44 24 04 MOV EAX,dword ptr [ESP + _Dst]


00402c22 8b fa MOV EDI,EDX
Add To Program: C:/Temp/shellcode.bin
Format: Raw Binary
Language: x86:LE:32:default:windows
Destination Folder: wizmo:/
Program Name: WizmoConsole.exe
Options...
OK
Cancel


CodeBrowser: wizmo:/WizmoConsole.exe
File Edit Analysis Navigation Search Select Tools Winc
Tool Options...
Options for WizmoConsole.exe...
Clear Code Bytes C
Clear With Options...
Clear Flow and Repair
Undo WizmoConsole.exe Ctrl+Z
Redo WizmoConsole.exe Ctrl+Shift+Z
.edit
shellcode
*user32.dll
Program Tree
Symbol Tree


Import C:/Temp/shellcode.bin
Format: Raw Binary
Language:
Destination Folder: wizmo:/
Program Name: shellcode.bin
Options...
Please select a language.
OK
Cancel
Language
Select Language and Compiler Specification
Processor Variant Size Endian Compiler
x86 Real Mode 16 little default
x86 System Mana... 32 little default
x86 default 32 little Borland C++
x86 default 32 little Delphi
x86 default 32 little gcc
x86 default 32 little Visual Studio
x86 default 64 little gcc
Filter: x8
Description
Intel/AMD 32-bit x86
Show Only Recommended Language/Compiler Specs
OK
Cancel
FUN_00401130:004
FUN_004012c0:004
FUN_004016b0:004
EBP


Register Manager [CodeBrowser: wizmo:/WizmoConsole.exe]
File Edit Tools Help
Register Manager
BND2 (128)
BND3 (128)
BNDCFGS (64)
BNDCFGU (64)
BNDSTATUS (64)
C0 (8)
C1 (8)
C2 (8)
C3 (8)
CS (16)
DS (16)
EAX (32)
AX (16)
AL (8)
Start Address
End Address
Value

结论

在使用Ghidra的UI几个小时之后，我发现它很有用且功能强大，但这还不足以让我从IDA Pro切换到Ghidra：

- 我已经使用IDA Pro超过22年了。抛弃这种经验并开始学习新工具并不容易。
  - 由于与Hex-Rays合作并为IDA的许多功能做出了贡献，我非常了解它的SDK和内部结构，而我对Ghidra一无所知。
  - 如果我想学习Ghidra的API，我也可以做到。但是，目前还不存在商业理由。
- 调试器：IDA有很多调试器
  - 它们是我认为的IDA Pro中的最佳功能。我很难离开IDA的调试器。
- 客户支持：世界上最好的客户支持
  - 多年来，Hex-Rays的客户支持已经宠坏了我。您不能指望任何其他公司具有相同水平的响应能力和专业性。是的，即使是亚马逊的客户服务也比不上Hex-Ray的。
- IDA是用C++编写的
  - IDA，至少在Windows平台上，感觉比Ghidra更整洁、更快
- 更高程度的互动性
  - 从我与Ghidra的小交互中可以看出，IDA仍然有许多交互功能和方法来修改反汇编列表和Hex-Rays反编译器输出。
- IDA具有高度可编程性
  - 是的，Ghidra具有可编程性。
  - 但在我看来，IDA仍然打败了Ghidra：可以用C++，Python，JavaScript，OCaml编写插件/处理器模块/文件加载器。
- IDA支持更多处理器模块和文件加载器（文件格式）。如果您执行processor_modules * file_loaders的乘法运算，IDA支持1200多个不同的文件输入！
  最后，我个人不会使用Ghidra，因为它还没有像IDA或其反编译器那样强大。当Ghidra开源并被社区采用时，我们将看到哪个软件逆向工程（SRE）工具仍然是国王：Binary Ninja，radare，IDA Pro，Hopper等？