

前言

开始叠甲：逆向工程可能较前面几个模块较难，有一定的入门门槛，本次实验选取的内容在知识储备上理论仅需C语言知识，但逆向工程仍是一门较为辛苦需要耐心的技术，希望大家量力而行，玩的开心！！

本周实验内容（较第二周轻松）：老实的命令程序、flag 验证机(I) (II) 、错乱的程序、flag 生成器、简单计算器

- 老实的命令程序、flag 验证机(I) (II) 、错乱的程序、flag 生成器各占本次实验20%的分数
- 简单计算器 较前者困难，算作附加题(做不出来就别卷啦)

其中老实的命令程序、flag 验证机(I)会进行实操展示（送分），错乱的程序会有一定说明

实验要求：实验报告：给出详细的解题过程，本次实验中需至少包括两个部分 程序是在做什么、你是怎么获得flag的（最好附上截图和代码）

其他：

1. 即使没获取flag，给出二进制文件的内部逻辑也能获取大部分实验分数
2. 公平起见，助教不提供除本次实验工具之外的帮助，如果给予任何人题目上的提示将会统一告知
3. 本次实验大部分函数名已经展现了其主要功能，而且会提供足够多的提示
4. 分数与难度成反比（提倡玩的开心，而不是卷分数）











简介

逆向工程（Reverse Engineering，简称 RE ）一般指软件逆向工程，利用各种方式从一个可运行的目标软件中分析得到软件产品的源代码、设计原理、结构、算法、处理过程、运行方法、相关文档等，它可以帮助我们更好地理解软件和功能、漏洞和安全性，从而可以对其进行优化、改进和保护。

reverse一些其他用途：

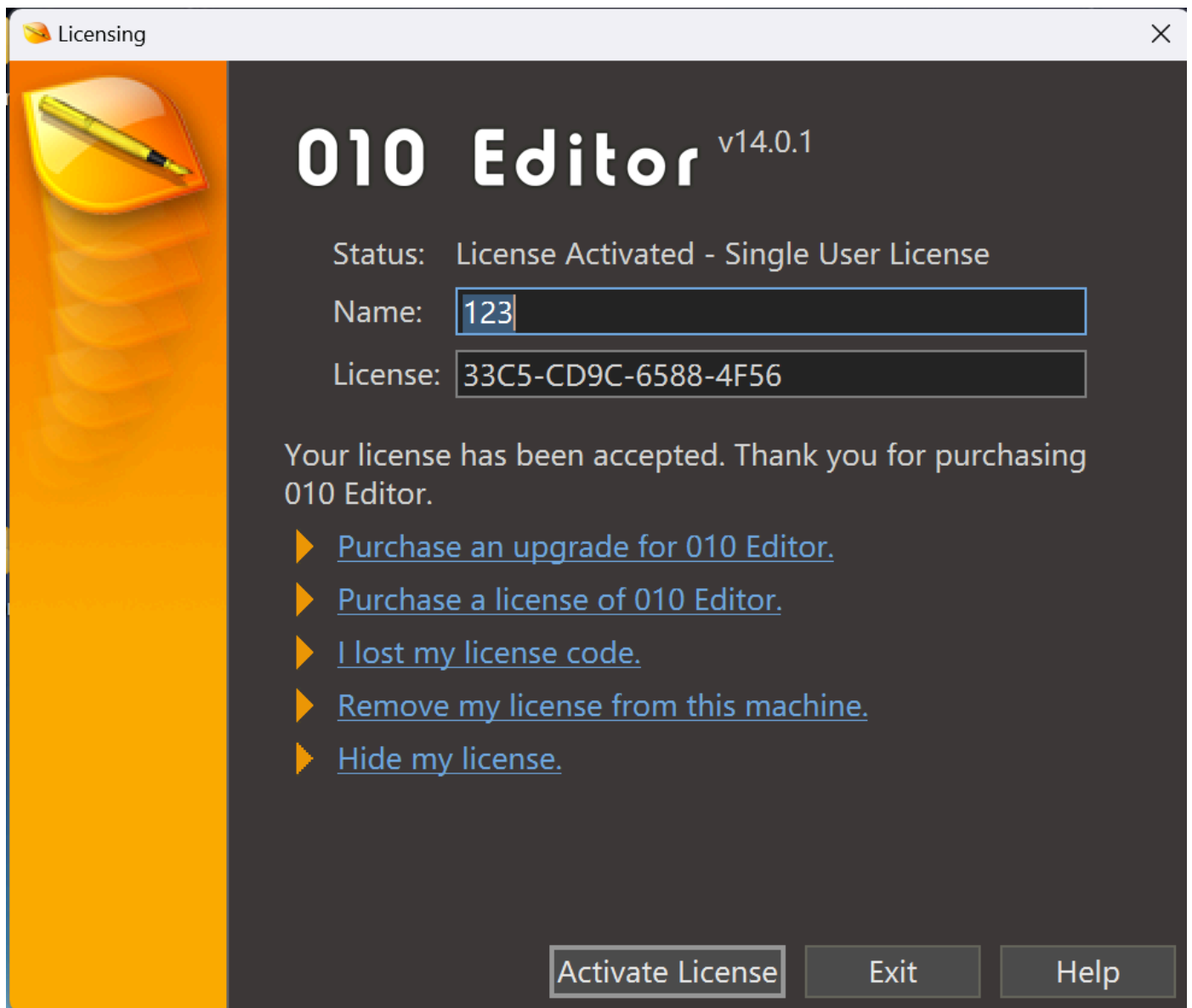
- **对软件进行一些自定义的修改**

杂交版pvz是用cheat engine在原作上进行dll注入实现的自定义玩法

 fonts	2024/5/3 9:48
 如果遇到问题请点我!!!	2024/5/3 9:48
 植物大战僵尸杂交版v.1.2	2024/5/3 9:54
 【看我】版本说明.txt	2024/4/22 21:35
 bass.dll	2024/4/22 21:27
 crash.txt	2024/5/5 22:31
 gdi42.dll	2024/4/22 21:27
 main.pak	2024/4/22 21:33
 PlantsVsZombies.exe	2024/4/22 21:27
 杂交启动器v1.2.EXE	2024/4/22 21:35

- **编写软件License的注册机**

由于为了离线使用、更好的性能和响应时间、简化架构等多种原因软件开发者可能会将许可证验证内置在二进制文件中，而不依赖网络验证，这就为逆向破解加密算法，制作自动生成可供使用的licence注册机提供了可能



手段

1. 静态分析 静态分析法是在不执行代码文件的情形下，对代码进行静态分析的一种方法。静态分析时并不执行代码，而是观察代码文件的外部特性，主要包括文件类型分析和静态反汇编、反编译。文件类型分析主要用于了解程序是用什么语言编写的或者是用什么编译器编译的，以及程序是否被加密处理过。在逆向过程中，主要是使用反汇编工具查看内部代码，分析代码结构。
2. 动态分析 动态分析法是在程序文件的执行过程中对代码进行动态分析的一种方法，其通过调试来分析代码，获取内存的状态等。在逆向过程中，通常使用调试器来分析程序的内部结构和实现原理。

可执行文件

逆向工程分析的对象是程序，即一个或多个可执行文件。下面简单介绍可执行文件的形成过程、常见可执行文件类型。

0x1 可执行文件的形成过程

可执行文件是计算机上的一种文件格式，它包含了程序代码、数据以及操作系统所需的其他信息，可以直接被操作系统加载和执行。下面是可执行文件的形成过程：

- **预处理 (Preprocessing) :**

在实际编译之前，源代码文件（通常为 .c, .cpp 等）首先经过预处理器处理。预处理器负责处理源代码中的预处理指令，这些指令包括宏定义（`#define`），条件编译（`#ifdef`, `#ifndef`, `#endif` 等）和文件包含指令（`#include`）。预处理结果是扩展后的源代码，其中包含了所有包含文件的内容，宏被展开，条件编译的代码被正确地保留或删除。

- **编译 (Compilation) :**

编译器将预处理后的源代码转换成汇编代码。这一步骤是将高级语言转换为低级的汇编语言，每条汇编指令对应着特定的机器码指令，但仍保留了标签和符号名称，这些对人类来说更易于理解。汇编代码是针对特定的处理器架构的，例如x86、ARM等。

- **汇编 (Assembly) :**

汇编器将汇编代码转换成机器码，生成所谓的目标文件（通常是 .o 或 .obj 文件）。这些目标文件包含了可以由计算机直接执行的二进制代码，但这些代码可能还需要链接其他代码（比如库函数）才能执行。

- **链接 (Linking) :**

链接器是最后阶段的工具，它将一个或多个目标文件与必要的库文件进行合并，生成最终的可执行文件（如 .exe 或无扩展的可执行文件）。在这个过程中，链接器解析所有未解决的符号引用，确保所有函数调用都指向正确的地址。链接可以是静态的或动态的。静态链接将所有必需的库函数复制到最终的可执行文件中，而动态链接记录所需库的引用，在程序运行时加载这些库。

- **加载和执行 (Loading and Execution) :**

一旦生成了可执行文件，操作系统负责加载它到内存中执行。加载器将可执行文件读入内存，进行必要的内存分配，并开始执行程序的入口点（通常是 main 函数）。对于动态链接的应用程序，加载器还负责解析和加载所有必要的动态链接库。这个过程涵盖了从编写源代码到最终执行可执行文件的完整路径。每个步骤都是为了确保代码能够正确地转换成机器能够理解和执行的格式。。

而在实际的环境中，由于需要考虑到生成的可执行文件的大小、可执行文件的运行性能、对信息的保护等原因，在每步过程中或多或少伴随着信息的丢失。

- **注释和格式化:**

源代码中包含的注释（例如 `// this is a comment` 或 `/* comment here */`）仅用于帮助人类理解代码的目的。在预处理阶段，这些注释会被移除，因为它们对程序执行没有任何影响。同样，代码的格式化，如空格、换行和缩进，也不会被编译到最终的机器码中。这些仅用于提高代码的可读性。

- **变量名和函数名:**

在源代码中使用的变量名和函数名在编译过程中会被替换为内存地址或寄存器。虽然这些名称对于程序员来说是有意义的，但对于机器来说，直接使用地址更为高效。在某些情况下，如果启用了调

试信息，这些名称可能会保留在符号表中，但在生产环境的最终产品中，这些名字通常不会被包含。

- **高级结构：**

- 高级编程语言支持的结构如循环（for, while）、条件判断（if-else）等在编译后转换为跳转和分支指令，这些更接近处理器的实际工作方式。编译器可能还会优化这些结构，如移除未执行的代码分支，循环展开等，控制流将更难还原。
- 对象和类在面向对象的语言中用于组织数据和功能，但在编译后，这些通常转化为简单的函数调用和数据结构。

- **类型信息：**

在机器码中，所有的操作只是针对内存地址和寄存器，而不关心这些地址和寄存器中的数据原本属于哪种高级类型

逆向则需要利用相关知识和经验，来还原其中的部分信息，进而还原全部或部分程序流程，从而实现分析者的各种目的。

0x2 不同格式的可执行文件

实际中，由于历史遗留问题 and 公司之间竞争等原因，上面介绍的每一步中产生的各种文件都会有多种文件格式。例如，Windows 系统使用的是 PE（Portable Executable）可执行文件，而 Linux 系统使用的是 ELF（Executable and Linkable Format）可执行文件。由于这两种可执行文件格式都是由 COFF（Common File Format）格式发展而来的，因此文件结构中的各种概念非常相似。

PE 文件由 DOS 头、PE 文件头、节表及各节数据组成；同时，如果需要引用外部的动态链接库，则有导入表；如果自己可以提供函数给其他程序来动态链接（常见于 DLL 文件），则有导出表。

我们这两次的实验仅涉及pe格式文件

静态分析

逆向工程中的静态分析是指在程序尚未运行的状态时进行逆向分析的行为。静态分析通过反汇编、反编译手段获得程序汇编代码或源代码，然后根据程序清单分析程序的流程，了解模块所完成的功能。

IDA

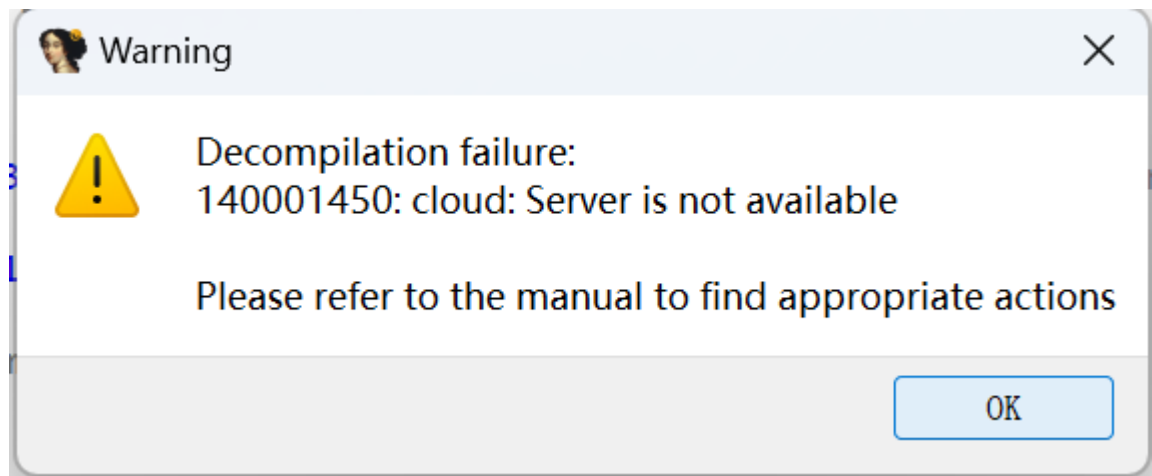
IDA（Interactive DisAssembler）是一款非常强大的逆向工程工具，广泛用于软件分析，尤其是在恶意软件分析和漏洞研究中。它由Hex-Rays SA开发，并提供了一系列的功能来帮助分析、调试和逆向工程复杂的软件程序。

本次实验：主要就是利用IDA在伪C代码的层面上进行一下简单的静态分析

环境配置

ida 官网获取免费的 idafree83_windows.exe

免费版的ida反编译需要联网进行，在网络或者是服务器问题时可能在F5时遇见



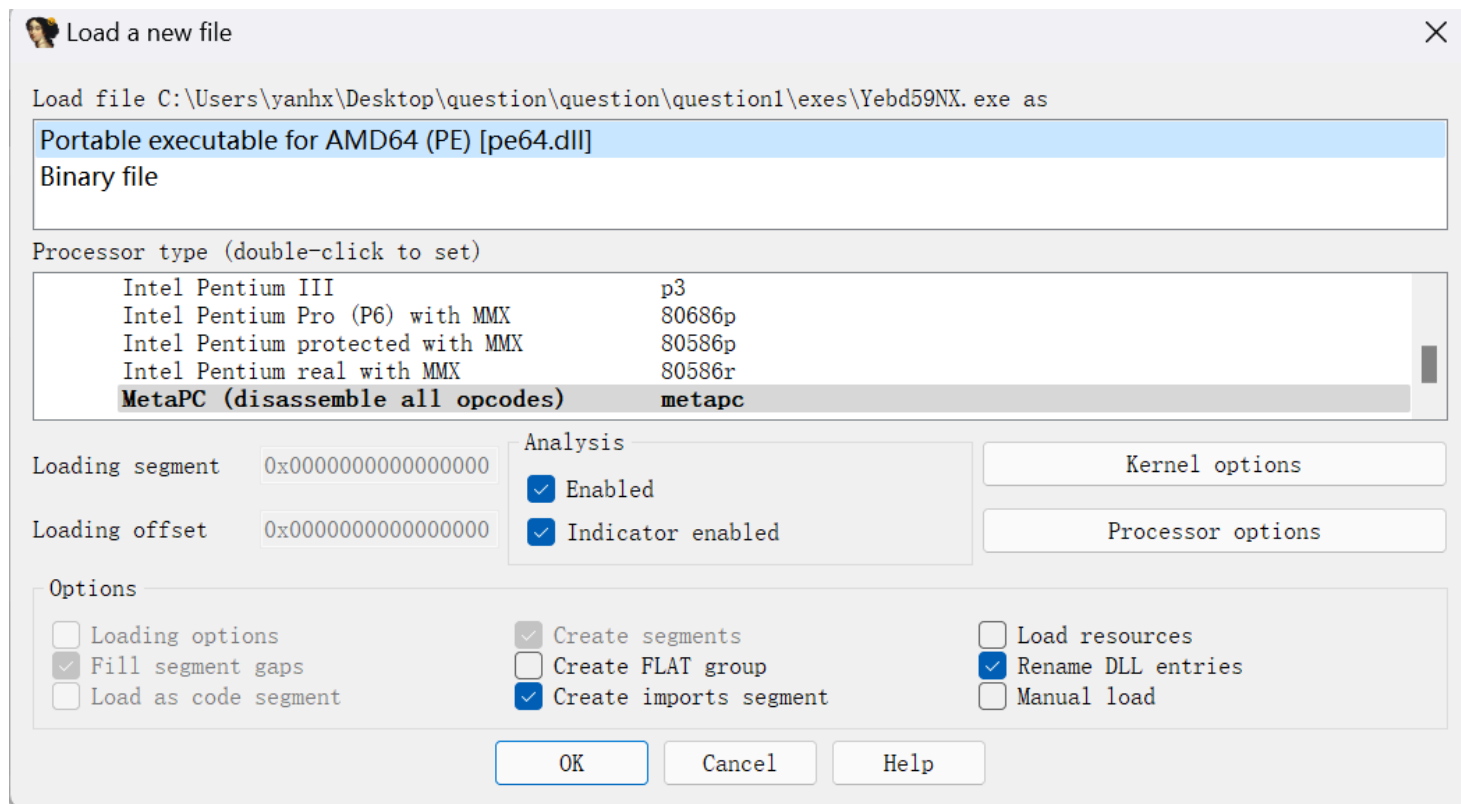
这是正常的情况，并非安装问题

注：本次课程内容（挑战）仅需ida免费版即可解决，并均已通过该版本测试

[看雪工具](#)获取破解版ida7.5(功能更全，反编译在本地进行，速度更快)

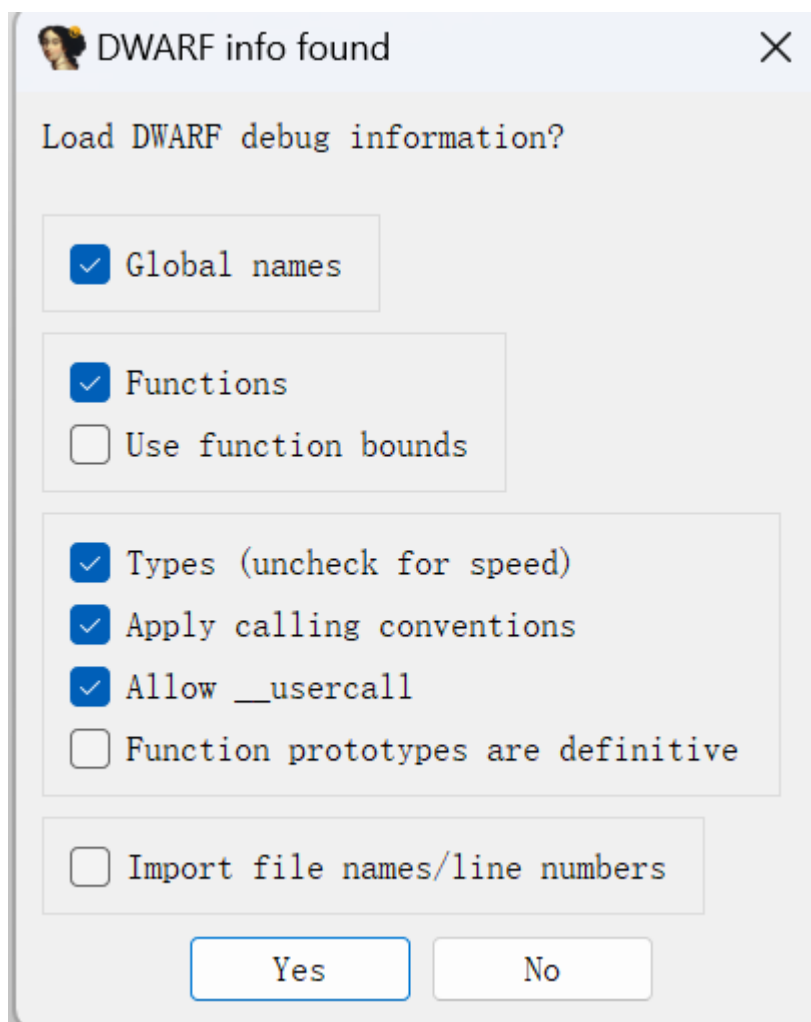
要点解析（在实操中会一一讲解）

load a new file



按照默认选项点击ok即可

load DWARF info



DWARF 是一种用于调试目的的标准文件格式，被广泛用于在不同的编程环境中存储和描述程序关于调试的信息。它是一种复杂的、高度组织化的格式，设计用来存储关于程序构造的详尽信息，这使得开发者和调试工具能够进行高效的源码级调试。

STRIP DWARF 调试信息可以通过使用 strip 命令或其他类似工具从二进制文件中移除。strip 命令是一个常用的工具，它可以减小可执行文件和对象文件的大小，通过删除不必要的调试信息和符号表信息。

一些直观展示

- 左侧的未strip的文件
- 右侧是strip后的文件

difference **文件校验和不一样**

010 Editor - C:\Users\yanhx\Desktop\question\question\question1\exes\Yebd59NX.exe

File Edit Search View Format Scripts Templates Debug Project Tools Window Help

Yebd59NX.exe x

0:0090 68 07 00 00 F0 00 26 00 0B 02 02 29 00 70 00 00 h...δ.&...).p...
0:00A0 00 A4 00 00 00 0C 00 00 F0 13 00 00 00 10 00 00 .@...δ...
0:00B0 00 00 00 40 01 00 00 00 00 10 00 00 00 02 00 00 ...@...
0:00C0 04 00 00 00 00 00 00 00 05 00 02 00 00 00 00 00
0:00D0 00 10 04 00 00 06 00 00 49 8F 04 00 03 00 60 01I...
0:00E0 00 00 20 00 00 00 00 00 00 10 00 00 00 00 00 00
0:00F0 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00
0:0100 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
0:0110 00 E0 00 00 F0 06 00 00 00 10 01 00 E8 04 00 00 .ä...δ...è...

Template Results - EXE.bt

Name	Value
MinorSubsystemVersion	2 CAh
Win32VersionValue	0 CCh
SizeOfImage	41000h D0h
SizeOfHeaders	600h D4h
Checksum	48F49h D8h
Subsystem	WINDOWS_GUI(3)

try.exe x

0090 00 00 00 00 F0 00 2E 02 0B 02 02 29 00 70 00 00δ.....).p...
00A0 00 A4 00 00 00 0C 00 00 F0 13 00 00 00 10 00 00 .@...δ...
00B0 00 00 00 40 01 00 00 00 00 10 00 00 00 02 00 00 ...@...
00C0 04 00 00 00 00 00 00 00 05 00 02 00 00 00 00 00
00D0 00 30 01 00 00 04 00 00 E1 5E 01 00 03 00 60 010.....á^...
00E0 00 00 20 00 00 00 00 00 00 10 00 00 00 00 00 00
00F0 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00
0100 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
0110 00 E0 00 00 F0 06 00 00 00 10 01 00 E8 04 00 00 .ä...δ...è...

Template Results - EXE.bt

Name	Value
MinorSubsystemVersion	2 CAh
Win32VersionValue	0 CCh
SizeOfImage	13000h D0h
SizeOfHeaders	400h D4h
Checksum	15EE1h D8h
Subsystem	WINDOWS_GUI(3)

Selected: 4 bytes (Range: 216 [D8h] to 219 [D8h])

Compare

Result	Address A	Size A	Address B
Only in A	AA00h	328B5h	
Only in A	340h	200h	
Difference	32Dh	1h	32Dh 1h
Difference	305h	1h	305h 1h
Difference	2DDh	1h	2DDh 1h
Difference	2B5h	1h	2B5h 1h
Difference	28Dh	1h	28Dh 1h
Difference	23Dh	1h	23Dh 1h
Difference	215h	1h	215h 1h
Difference	1EDh	1h	1EDh 1h
Difference	1C5h	1h	1C5h 1h
Difference	19Dh	1h	19Dh 1h
Difference	D1h	Ah	D1h Ah
Difference	86h	12h	86h 12h
Match	540h	A4C0h	340h A4
Match	32Eh	12h	32Eh 12h
Match	306h	27h	306h 27h
Match	2DEh	27h	2DEh 27h
Match	2B6h	27h	2B6h 27h
Match	28Eh	27h	28Eh 27h
Match	23Eh	4Fh	23Eh 4Fh
Match	216h	27h	216h 27h
Match	1EEh	27h	1EEh 27h
Match	1C6h	27h	1C6h 27h
Match	19Eh	27h	19Eh 27h
Match	DBh	C2h	DBh C2h
Match	98h	39h	98h 39h
Match	0h	86h	0h 86h

Output Find Results Find in Files Compare Histogram

Start: D8h Sel: 4 [4h] Size: 250,549 Hex ANSI LIT OVR

difference FileHeader里NumberOfSections的值不一样

未strip:20

strip后:11

010 Editor - C:\Users\yanhx\Desktop\question\question\question\Yebd59NX.exe

File Edit Search View Format Scripts Templates Debug Project Tools Window Help

Yebd59NX.exe x

0:0080 50 45 00 00 64 86 14 00 29 C2 C8 65 00 34 03 00 PE..d...ÄEe.4..

0:0090 68 07 00 00 F0 00 26 00 0B 02 02 29 00 70 00 00 h...ð.&...).p..

0:00A0 00 A4 00 00 00 0C 00 00 F0 13 00 00 00 00 00 00 .a...ð... ..

0:00B0 00 00 00 40 01 00 00 00 00 10 00 00 00 02 00 00 .@..... ..

0:00C0 04 00 00 00 00 00 00 00 05 00 02 00 00 00 00 00I.....

0:00D0 00 10 04 00 00 06 00 00 49 8F 04 00 03 00 60 01

0:00E0 00 00 20 00 00 00 00 00 00 10 00 00 00 00 00 00

0:00F0 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00

0:0100 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00

Template Results - EXE.bt

Name	Value
NumberOfSections	20 86h
TimeDateStamp	02/11/2024 12:48:41 88h
PointerToSymbolTable	209920 8Ch
NumberOfSymbols	1896 90h
SizeOfOptionalHeader	240 94h

try.exe x

0:0080 50 45 00 00 64 86 0B 00 04 4A 38 66 00 00 00 00 PE..d...J8f... ..

0:0090 00 00 00 00 F0 00 2E 02 0B 02 02 29 00 70 00 00ð.....).p..

0:00A0 00 A4 00 00 00 0C 00 00 F0 13 00 00 00 00 00 00 .a...ð... ..

0:00B0 00 00 00 40 01 00 00 00 00 10 00 00 00 02 00 00 .@..... ..

0:00C0 04 00 00 00 00 00 00 00 05 00 02 00 00 00 00 00

0:00D0 00 30 01 00 00 04 00 00 E1 5E 01 00 03 00 60 01 .0.....ä.....

0:00E0 00 00 20 00 00 00 00 00 00 10 00 00 00 00 00 00

0:00F0 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00

0:0100 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00

Template Results - EXE.bt

Name	Value
NumberOfSections	11 86h
TimeDateStamp	05/06/2024 03:09:56 88h
PointerToSymbolTable	0 8Ch
NumberOfSymbols	0 90h
SizeOfOptionalHeader	240 94h

Compare

Result	Address A	Size A	Address B
Only in A	AA00h	328B5h	
Only in A	340h	200h	
Difference	32Dh	1h	32Dh 1h
Difference	305h	1h	305h 1h
Difference	2DDh	1h	2DDh 1h
Difference	2B5h	1h	2B5h 1h
Difference	28Dh	1h	28Dh 1h
Difference	23Dh	1h	23Dh 1h
Difference	215h	1h	215h 1h
Difference	1EDh	1h	1EDh 1h
Difference	1C5h	1h	1C5h 1h
Difference	19Dh	1h	19Dh 1h
Difference	D1h	Ah	D1h Ah
Difference	86h	12h	86h 12h
Match	540h	A4C0h	340h A4
Match	32Eh	12h	32Eh 12h
Match	306h	27h	306h 27h
Match	2DEh	27h	2DEh 27h
Match	2B6h	27h	2B6h 27h
Match	28Eh	27h	28Eh 27h
Match	23Eh	4Fh	23Eh 4Fh
Match	216h	27h	216h 27h
Match	1EEh	27h	1EEh 27h
Match	1C6h	27h	1C6h 27h
Match	19Eh	27h	19Eh 27h
Match	DBh	C2h	DBh C2h
Match	98h	39h	98h 39h
Match	0h	86h	0h 86h

Output Find Results Find in Files Compare Histogram

only in A 未strip的文件结构体数组SectionHeaders[20]比SectionHeaders[11]大

Yebd59NX.exe x

0:0300 00 06 00 00 00 A2 00 00 00 00 00 00 00 00 00 00C.....

0:0310 00 00 00 00 40 00 00 00 2E 72 65 6C 6F 63 00 00@...reloc...

0:0320 84 00 00 00 00 20 01 00 00 02 00 00 00 A8 00 00

0:0330 00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 42@...B.....

0:0340 2F 34 00 00 00 00 00 00 20 06 00 00 00 30 01 00 /4.....0... ..

0:0350 00 08 00 00 00 AA 00 00 00 00 00 00 00 00 00 00

0:0360 00 00 00 00 40 00 00 42 2F 31 39 00 00 00 00 00@...B/19.....

0:0370 83 20 01 00 00 40 01 00 00 22 01 00 00 B2 00 00 f...@... ..

0:0380 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 42@...B.....

Template Results - EXE.bt

Name	Value
> SectionHeaders[11]	/4 340h
> SectionHeaders[12]	/19 368h
> SectionHeaders[13]	/31 390h
> SectionHeaders[14]	/45 3B8h
> SectionHeaders[15]	/57 3E0h
> SectionHeaders[16]	/70 408h

Compare

Result	Address A	Size A	Address B
Only in A	AA00h	328B5h	
Only in A	340h	200h	
Difference	32Dh	1h	32Dh 1h
Difference	305h	1h	305h 1h
Difference	2DDh	1h	2DDh 1h
Difference	2B5h	1h	2B5h 1h
Difference	28Dh	1h	28Dh 1h
Difference	23Dh	1h	23Dh 1h
Difference	215h	1h	215h 1h
Difference	1EDh	1h	1EDh 1h
Difference	1C5h	1h	1C5h 1h
Difference	19Dh	1h	19Dh 1h
Difference	D1h	Ah	D1h Ah
Difference	86h	12h	86h 12h
Match	540h	A4C0h	340h A4

未strip的文件有更多的Section

Yebd59NX.exe x

0:A9C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0:A9D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0:A9E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0:A9F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0:AA00 2C 00 00 00 02 00 00 00 00 00 08 00 00 00 00

0:AA10 00 10 00 40 01 00 00 00 24 04 00 00 00 00 00@...\$. ..

0:AA20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0:AA30 2C 00 00 00 02 00 97 26 00 00 08 00 00 00 00

0:AA40 A0 16 00 40 01 00 00 00 CF 00 00 00 00 00 00@...I.....

Template Results - EXE.bt

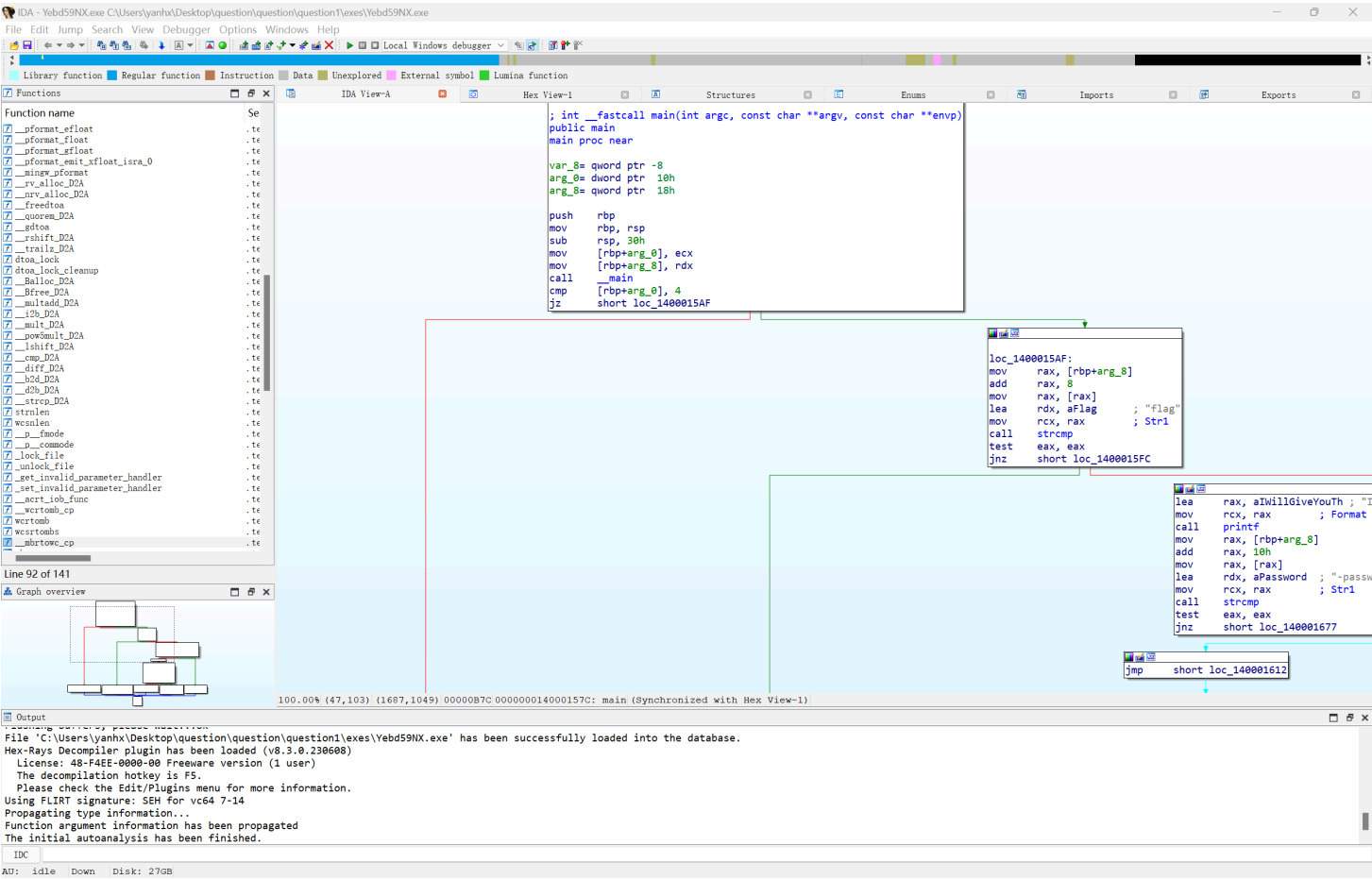
Name	Value
> Section[8]	.rsrc A200h
> Section[9]	.reloc A800h
> Section[10]	/4 AA00h
> Section[11]	/19 B200h
> Section[12]	/31 1D400h
> Section[13]	/45 20A00h

Compare

Result	Address A	Size A	Address B
Only in A	AA00h	328B5h	
Only in A	340h	200h	
Difference	32Dh	1h	32Dh 1h
Difference	305h	1h	305h 1h
Difference	2DDh	1h	2DDh 1h
Difference	2B5h	1h	2B5h 1h
Difference	28Dh	1h	28Dh 1h
Difference	23Dh	1h	23Dh 1h
Difference	215h	1h	215h 1h
Difference	1EDh	1h	1EDh 1h
Difference	1C5h	1h	1C5h 1h
Difference	19Dh	1h	19Dh 1h
Difference	D1h	Ah	D1h Ah
Difference	86h	12h	86h 12h
Match	540h	A4C0h	340h A4

图片未展示的difference 各个节区的偏移位置存在区别（这是由于未strip的文件有更多的节区，导致sectionheaders会更大一些）

ida界面



下面是界面的几个主要组成部分的详细介绍：

1. 功能窗口区域

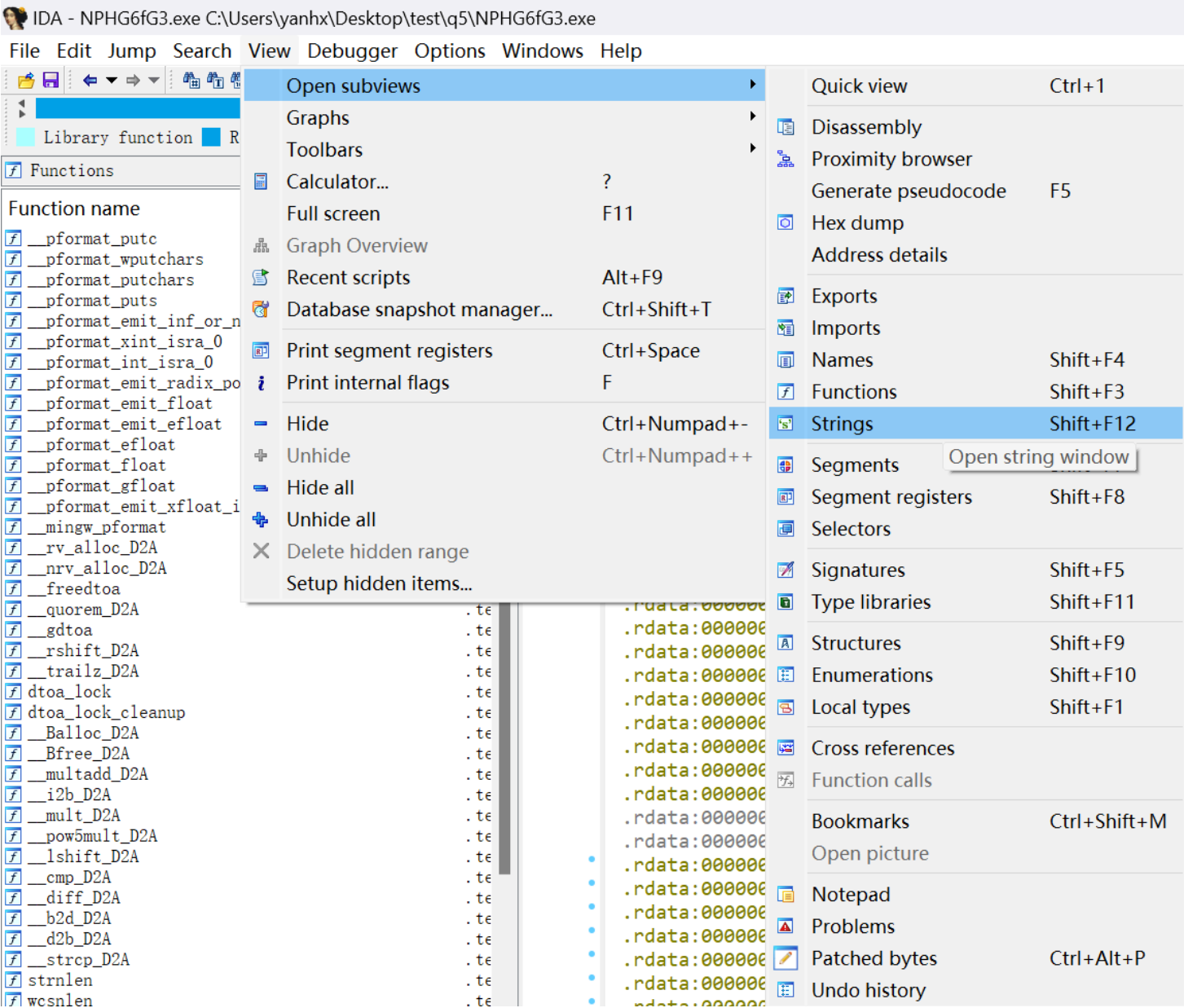
- 函数列表窗口（左上角）：列出了当前可执行文件中识别的所有函数。这些函数通常是从入口点分析得出的，包括程序自带的和静态链接库中的函数。点击任何一个函数，IDA会跳转到该函数的反汇编代码。
- 反汇编窗口（中间大窗口）：显示当前选定函数的反汇编代码。这里的代码使用汇编语言展示，并包括地址、操作码和操作数等信息。用户可以在此窗口中导航代码、添加注释和书签。
- 图形视图（左下角小窗口）：以图形形式显示函数的流程，帮助用户理解代码结构，如条件分支和循环。可以点击图中的任何部分，IDA会在反汇编窗口中对应跳转到该代码段。

2. 其他窗口和信息区域

- Hex View-1（未显示但提到）：以十六进制视图显示二进制数据。对于分析数据段和非执行代码区非常有用。
- 输出窗口（底部）：显示各种日志信息，如载入文件的状态、错误信息、以及其他调试信息。
- 菜单栏和工具栏：提供各种操作命令，如文件操作、视图切换、搜索功能、调试命令等。

字符串与字符串窗口

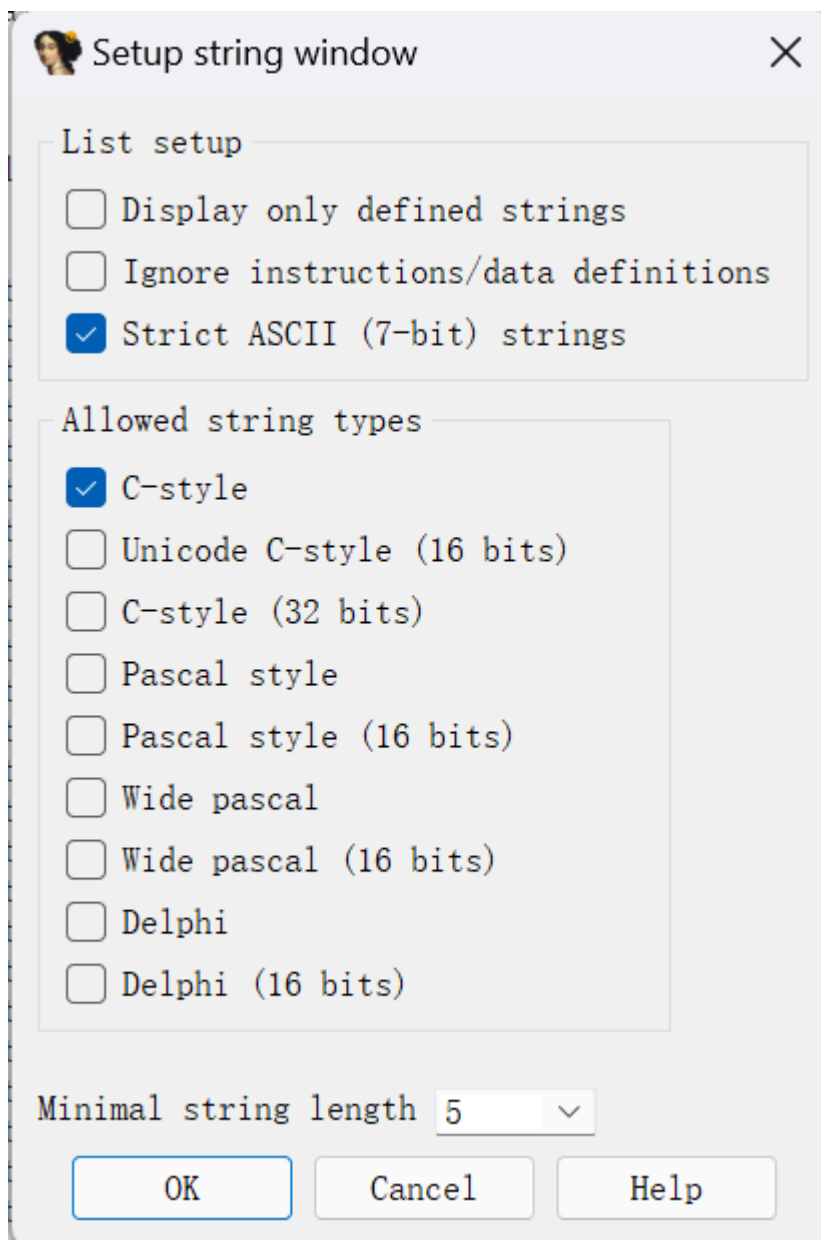
打开字符串窗口 view->open subviews->strings



显示如图

[illegible]

展示内容由你的设置决定 在string窗口右键->setup string



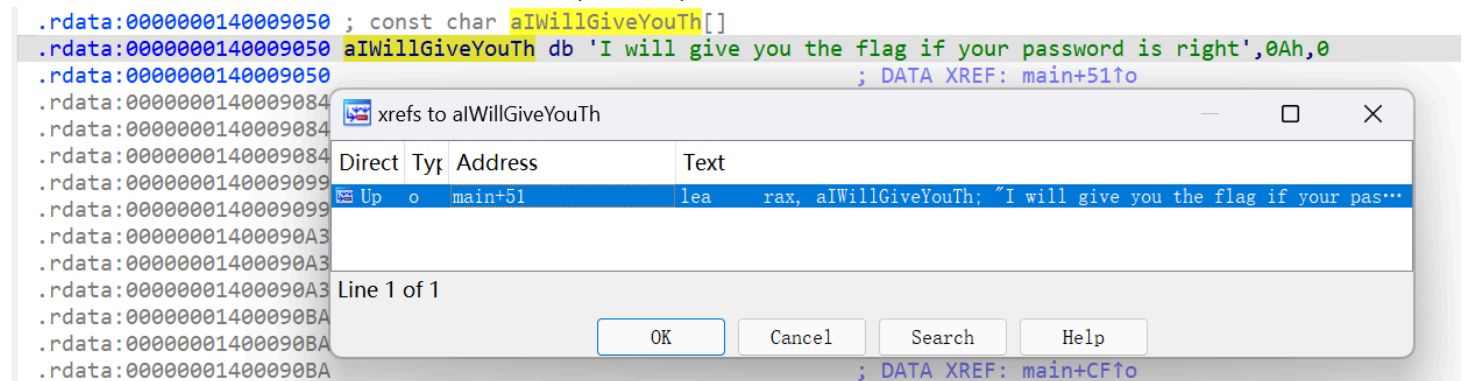
由于程序与用户用字符串进行了交互，那么通过字符串的使用位置就可以反推程序当前的功能，这是一种快速定位你感兴趣位置的办法。双击字符串，会转到反汇编窗口

```
.rdata:00000000140009048 ; const char aFlag[]
.rdata:00000000140009048 aFlag          db 'flag',0             ; DATA XREF: main+3Efo
.rdata:0000000014000904D                align 10h
.rdata:00000000140009050 ; const char aIWillGiveYouTh[]
.rdata:00000000140009050 aIWillGiveYouTh db 'I will give you the flag if your password is right',0Ah,0
.rdata:00000000140009050                ; DATA XREF: main+51fo
.rdata:00000000140009084 ; const char aWhatAreYouDoin[]
.rdata:00000000140009084 aWhatAreYouDoin db 'What are you doing?',0Ah,0
.rdata:00000000140009084                ; DATA XREF: main:loc_1400015FCfo
.rdata:00000000140009099 ; const char aPassword[]
.rdata:00000000140009099 aPassword      db '-password',0         ; DATA XREF: main+6Bfo
.rdata:000000001400090A3 ; const char aVerifyingPassw[]
.rdata:000000001400090A3 aVerifyingPassw db 'verifying password...',0Ah,0
.rdata:000000001400090A3                ; DATA XREF: main:loc_140001612fo
.rdata:000000001400090BA ; const char aHereYouAre[]
.rdata:000000001400090BA aHereYouAre   db 'Here you are!',0Ah,0
.rdata:000000001400090BA                ; DATA XREF: main+CFfo
```

通常字符串是分配在rdata节区

在Windows可执行文件（特别是PE格式，Portable Executable）中，.rdata区（或段）是一个重要的组成部分，用于存储只读数据。这个段的内容通常在程序运行时不会被修改，因此被操作系统映射为只读内存。

选中字符串符号（db前的变量名）点击x(快捷键)可以搜索引用调用



- Direction
 - **up**: 引用此处的地点在被引用的上侧
 - **down**: 引用此处的地点在被引用的下侧
- Type

The following types exist:

- **o** - offset, the address of the item is taken
- **r** - read access
- **w** - write access
- **t** - textual referenced (used for manually specified operands)
- **i** - informational (e.g. a derived class refers to its base class)
- **J** - far (intersegment) jump
- **j** - near (intrasegment) jump
- **P** - far (intersegment) call
- **p** - near (intrasegment) call
- **^** - ordinary flow
- **s** - xref from a structure
- **m** - xref from a structure member
- **k** - xref from a stack variable

(HELP 按键里有ida官方对该功能的使用说明)

双击即可跳转到引用该字符串的位置

反汇编窗口有关字符一些其他功能:

```

; const char unk_14000A04D[4]
unk_14000A04D db 66h ; f
               db 6Ch ; l
               db 61h ; a
               db 67h ; g
               db 0D5h
               db 0FDh
               db 0C8h
               db 0B7h
               db 0Ah
               db 0

```

对于已经有了类型的变量可以按U (undefine) 变为一个个未定义的字节

```

; const char aFlag_0[4]
aFlag_0 db 'flag'
         db 0D5h
         db 0FDh
         db 0C8h
         db 0B7h
         db 0Ah
         db 0

```

对于零散的字节可以按A(MakeStrlit)按照ascii的编码规则尽可能的恢复为字符串

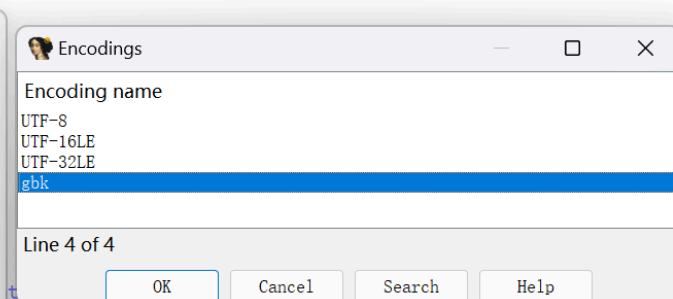
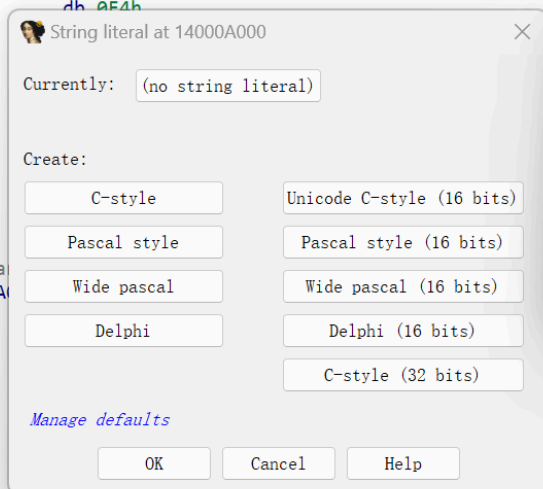
对于中文编码，ida不能自动识别此时可以

选中字符符号->Alt+A->Currently->ins(电脑上的快捷键)->输入gbk->再选择即可

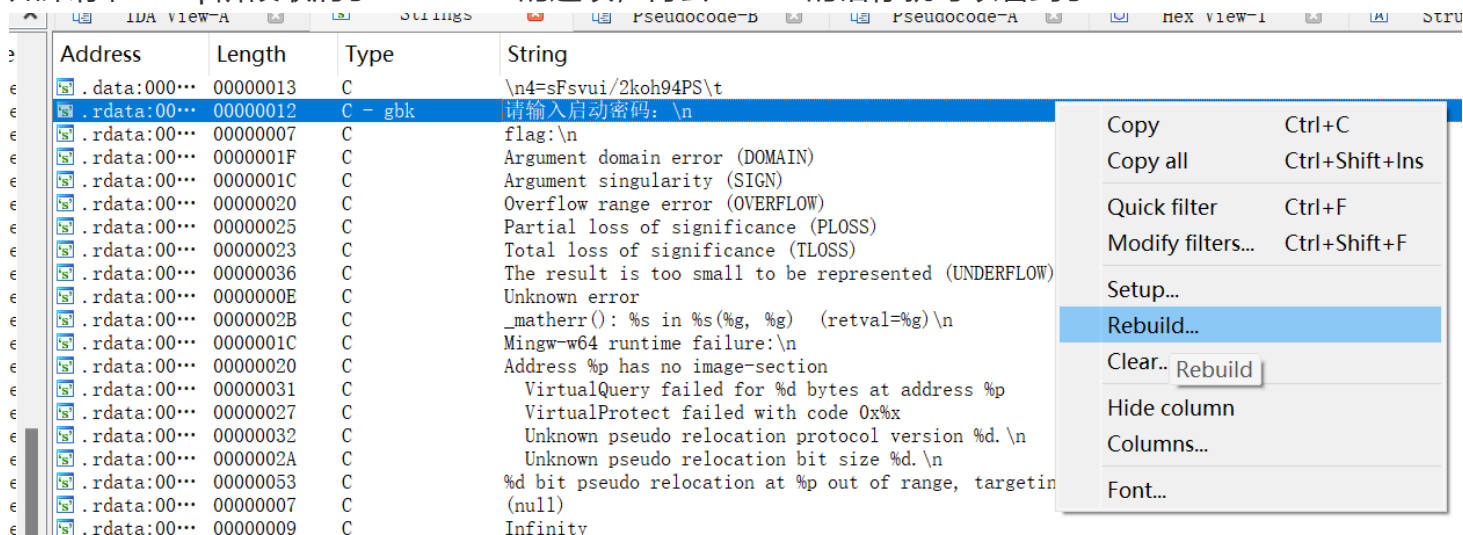
```

00A000 ;org 14000A000h
00A000 ; const char byte_14000A000
00A000 byte_14000A000 db 0C7h ; DATA XREF: Validator::get_input(void)+Cfo
00A001 db 0EBh
00A002 db 0CAh
00A003 db 0E4h
00A004 db 0C8h
00A005 db 0EBh
00A006 db 0C6h
00A007 db 0F4h
00A008
00A009
00A00A
00A00B
00A00C
00A00D
00A00E
00A00F
00A010
00A011
00A012 ; const char
00A012 byte_14000A000
00A012
00A012
00A013
00A014
00A015
00A016
00A017
00A018
00A019
00A01A db 0Ah
00A01B db 0
00A01C ; const char Control[2]

```



如果你在setup阶段取消了strict Acscii的选项，再去rebuild的话你就可以看到了



第二次实验部分实验存在gbk编码因此建议掌握以上方法

函数与类型修改

库函数

一些常见的函数

```
__acrt_iob_func(0);  
if ( fgets(Buffer, 50, v3) )
```

__acrt_iob_func是一个标准的C库函数，主要用于在Windows平台的MSVC（Microsoft Visual C++）环境中定位标准I/O对象。这个函数返回一个指向I/O对象的指针。

- 参数0表示返回stdin（标准输入）的文件指针
- 1表示stdout（标准输出）
- 2表示stderr（标准错误）

粉色的函数名表示该函数是从动态库中在运行时链接的

```
1 // attributes: thunk  
2 char *__cdecl fgets(char *Buffer, int MaxCount, FILE *Stream)  
3 {  
4     return __imp_fgets(Buffer, MaxCount, Stream);  
5 }
```

其数据地址会在.idata数据段，是一个导入数据段（存放有关导入函数的数据）由于是运行时链接所以此时数据是未初始化状态也是合理的


```

.idata:000000014000F2E8 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F2E8 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F2F0 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F2F0 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F2F8 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F2F8 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F300 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F300 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F308 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F308 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F310 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F310 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F318 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F318 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F320 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F320 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F328 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F328 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F330 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F330 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F338 ?? ?? ?? ?? ?? ?? ?? ??
.idata:000000014000F338 ?? ?? ?? ?? ?? ?? ?? ??

extrn __imp_exit:qword ; DATA XREF: EXIT
; char *(__cdecl *fgets)(char *Buffer, int MaxCount, FILE *Stream)
extrn __imp_fgets:qword ; DATA XREF: fgets
; int (*fprintf)(FILE *const Stream, const char *const Format, ...)
extrn __imp_fprintf:qword ; DATA XREF: fprintf
; int (__cdecl *fputc)(int Character, FILE *Stream)
extrn __imp_fputc:qword ; DATA XREF: fputc
; void (__cdecl *free)(void *Block)
extrn __imp_free:qword ; DATA XREF: free
; size_t (__cdecl *fwrite)(const void *Buffer, size_t ElementSize, size_t ElementCount, FILE *Stream)
extrn __imp_fwrite:qword ; DATA XREF: fwrite
; struct lconv *(__cdecl *localeconv)()
extrn __imp_localeconv:qword ; DATA XREF: localeconv
; void *(__cdecl *malloc)(size_t Size)
extrn __imp_malloc:qword ; DATA XREF: malloc
; void *(__cdecl *memcpy)(void *, const void *Src, size_t Size)
extrn __imp_memcpy:qword ; DATA XREF: memcpy
; void *(__cdecl *memset)(void *, int Val, size_t Size)
extrn __imp_memset:qword ; DATA XREF: memset
; _crt_signal_t (__cdecl *signal)(int Signal, _crt_signal_t Function)
extrn __imp_signal:qword ; DATA XREF: signal
; size_t (__cdecl *strncpy)(const char *Str, const char *Control)

```

当去除符号表时，一些标准库函数被抹去了名称，就需要通过一些经验与猜测来判断

函数窗口

点击函数窗口 ctrl+F->输入查询内容 会显示匹配到的函数名称

函数类型

点击函数名称按Y快捷键可以对函数声明进行修改，注意此时对函数名的修改是无效的，修改函数名或者是变量名是N键。由于函数返回值通常由rax存储并返回所以ida也通常会将返回值定为__int64型

```

while ( 1 )
{
    system("cls");
    sub_1400014A4(&unk_1400130FF);
    sub_1400014A4(&unk_140013115);
    sub_1400014A4(&unk_140013122);
    sub_1400014A4(&unk_14001312D);
    sub_1400014A4(&unk_140013136);
    sub_140001450(&unk_140013147, &v1);
    sub_1400014F8();
    if ( v1 == 3 )
        break;
    if ( v1 > 3 )
        goto LABEL_10;
    if ( v1 )
    {
        sub_1400014A4(&unk_140013147);
        system("cls");
    }
    else if ( v1 == 2 )
    {

```

Please enter a string

Please enter the type declaration __int64 __fastcall sub_1400014A4(_QWORD)

OK

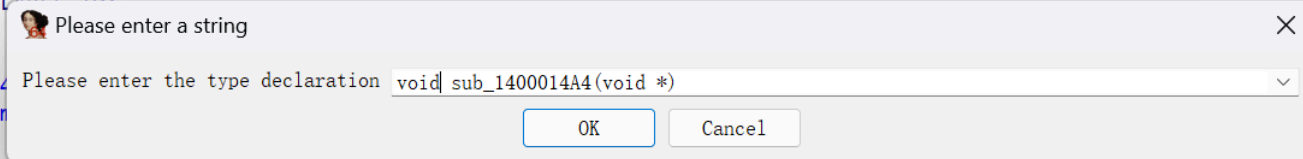
Cancel

该声明兼容一般的c声明如下亦可

```

system("cls");
sub_1400014A4(&unk_1400130FF);
sub_1400014A4(&unk_140013115);
sub_1400014A4(&unk_140013122);
sub_1400014A4(&unk_14001312D);
sub_1400014A4(&unk_140013136);
sub_140001450(&unk_140013147, &v1);
sub_1400014F8();
if ( v1 == 3 )
    break;
if ( v1 > 3 )
    goto L140013147;
if ( v1 < 3 )
{
    sub_1400014A4(&unk_140013147);
    system("cls");
}
else if ( v1 == 2 )
{
    sub_1400014A4(&unk_140013147);
    system("cls");
}
}

```



点击OK->返回后F5刷新

__fastcall 是一个函数调用约定,使用寄存器传参

还有很多具体细节可以查看官方手册, 本次实验不会用到

数据与类型修改

数据类型与函数类型的修改类似, 同样是Y键

为了更好的还原出数据的结构类型: 我们不得不补充一些在汇编层面上的知识

在ida反编译出来的函数的最开始一般是局部变量的声明

```

FILE *v3; // rax
char Buffer[8]; // [rsp+20h] [rbp-50h] BYREF
__int64 v6; // [rsp+28h] [rbp-48h]
__int64 v7; // [rsp+30h] [rbp-40h]
__int64 v8; // [rsp+38h] [rbp-38h]
__int64 v9; // [rsp+40h] [rbp-30h]
__int64 v10; // [rsp+48h] [rbp-28h]
char v11; // [rsp+50h] [rbp-20h]
int v12; // [rsp+54h] [rbp-1Ch]
int v13; // [rsp+58h] [rbp-18h]
int v14; // [rsp+5Ch] [rbp-14h]
int k; // [rsp+60h] [rbp-10h]
int i; // [rsp+64h] [rbp-Ch]
__int64 j; // [rsp+68h] [rbp-8h]

```

双击他可以进入ida为我们进行的栈分析, 由于函数的局部变量是分配在栈上的, 根据相对位置我们可以获取变量类型结构相关的一些信息。

```

-000000000000000051 db ? ; undefined
-000000000000000050 Buffer db 8 dup(?)
-000000000000000048 var_48 dq ?
-000000000000000040 var_40 dq ?
-000000000000000038 var_38 dq ?
-000000000000000030 var_30 dq ?
-000000000000000028 var_28 dq ?
-000000000000000020 var_20 db ?
-00000000000000001F db ? ; undefined
-00000000000000001E db ? ; undefined
-00000000000000001D db ? ; undefined
-00000000000000001C var_1C dd ?
-000000000000000018 var_18 dd ?
-000000000000000014 var_14 dd ?
-000000000000000010 var_10 dd ?
-00000000000000000C var_C dd ?
-000000000000000008 var_8 dq ?
+000000000000000000 s db 8 dup(?)
+000000000000000008 r db 8 dup(?)
+000000000000000010
+000000000000000010 ; end of stack variables

```

由于ida不能充分分析上下文语境，可能会造成如作为数组的连续变量被分为独立的一个一个的变量，我们由此可以参考着还原

如图buffer在fgets里明明输入了49个字符，这里的类型却声明为buffer[8]

当我们用Y进行类型更改的时候，声明为buffer[49],提示现在的类型比原来类型大，会摧毁别的变量，这里我们很有信心直接set the type(效果不理想的话可以按ctrl+z来返回)最后获得更好看的反编译

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     FILE *v3; // rax
4     char Buffer[8]; // [rsp+20h] [rbp-50h] BYREF
5     __int64 v6; // [rsp+28h] [rbp-48h]
6     __int64 v7; // [rsp+30h] [rbp-40h]
7     __int64 v8; // [rsp+38h] [rbp-38h]
8     __int64 v9; // [rsp+40h] [rbp-30h]
9     __int64 v10; // [rsp+48h] [rbp-28h]
10    char v11; // [rsp+50h] [rbp-20h]
11    int v12; // [rsp+54h] [rbp-1Ch]
12    int v13; // [rsp+58h] [rbp-18h]
13    int v14; // [rsp+5Ch] [rbp-14h]
14    int k; // [rsp+60h] [rbp-10h]
15    int i; // [rsp+64h] [rbp-Ch]
16    __int64 j; // [rsp+68h] [rbp-8h]
17
18    _main(argc, argv, envp);
19    *(_QWORD *)Buffer = 0i64;
20    v6 = 0i64;
21    v7 = 0i64;
22    v8 = 0i64;
23    v9 = 0i64;
24    v10 = 0i64;
25    v11 = 0;
26    printf("please input the flag\n");
27    v3 = __acrt_iob_func(0);
28    if ( fgets(Buffer, 49, v3) )
29    {
30        Buffer[strcspn(Buffer, "\n")] = 0;
31        v14 = strlen(Buffer);
32        if ( v14 == 35 )
33        {
34            j = 0i64;
35            v13 = 0;
36            v12 = 0;
37            for ( i = 0; i <= 34; ++i )
38            {
39                for ( j = big_num[i]; ; j >>= 7 )
40                {
41                    while ( 1 )
42                    {
43                        while ( 1 )
44                        {
45                            while ( 1 )
46                            {
47                                v13 = j & 1;

```



Please confirm



The new variable size is bigger than the old one.
It may overlap with other variables (and destroy them).
Did you mean to use 'char[49] *'?

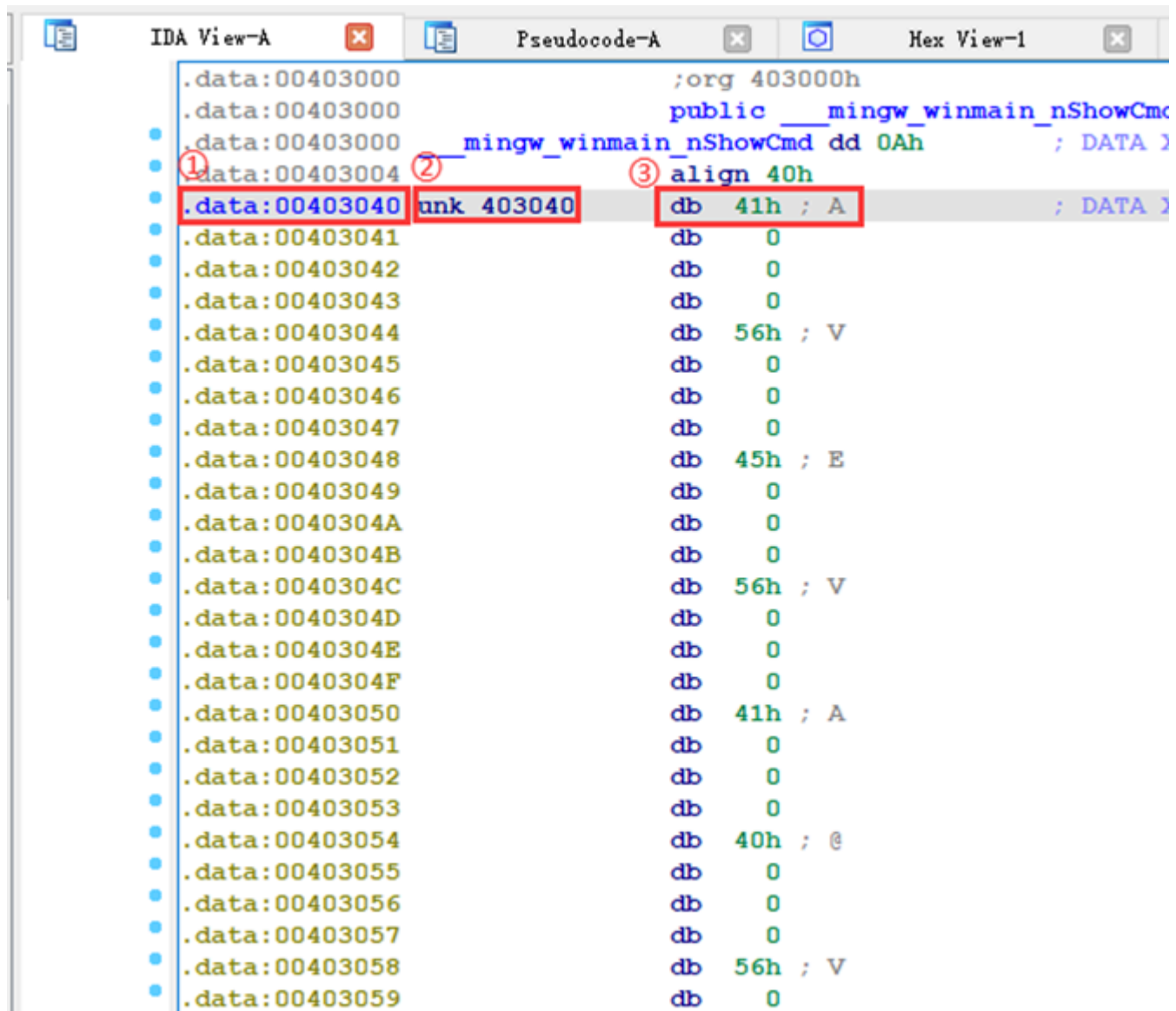
Convert to pointer

Set the type

Cancel

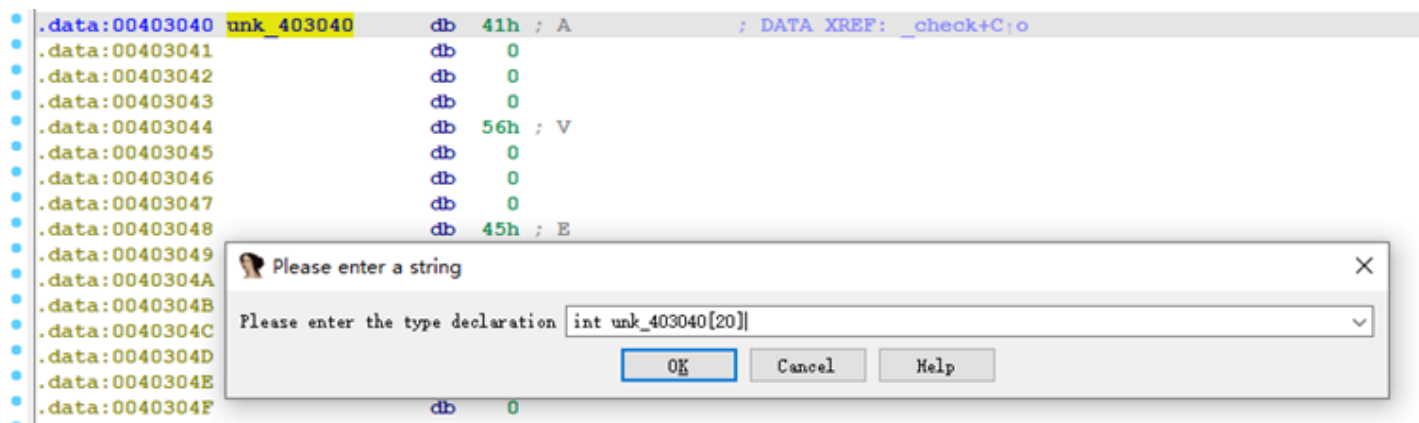
```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     FILE *v3; // rax
4     char Buffer[49]; // [rsp+20h] [rbp-50h] BYREF
5     int v6; // [rsp+54h] [rbp-1Ch]
6     int v7; // [rsp+58h] [rbp-18h]
7     int v8; // [rsp+5Ch] [rbp-14h]
8     int k; // [rsp+60h] [rbp-10h]
9     int i; // [rsp+64h] [rbp-Ch]
10    __int64 j; // [rsp+68h] [rbp-8h]
11
12    _main(argc, argv, envp);
13    memset(Buffer, 0, sizeof(Buffer));
14    printf("please input the flag\n");
15    v3 = __acrt_iob_func(0);
16    if ( fgets(Buffer, 49, v3) )
17    {
18        Buffer[strcspn(Buffer, "\n")] = 0;
19        v8 = strlen(Buffer);
20        if ( v8 == 35 )
21        {
22            j = 0i64;
23            v7 = 0;
24            v6 = 0;
25            for ( i = 0; i <= 34; ++i )
26            {
27                for ( j = big_num[i]; ; j >>= 7 )
28                {
29                    ...
30                }
31            }
32        }
33    }
34}
```

数据的导出

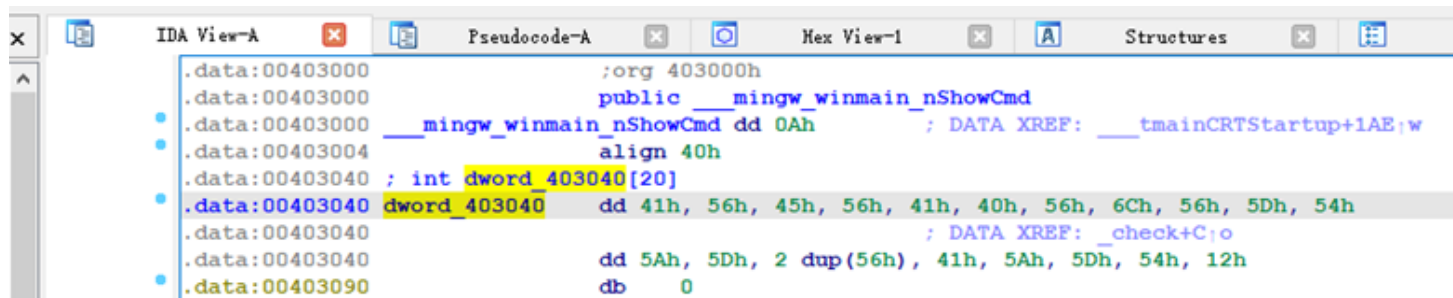


1. .data是指 data 段，后面的 16 进制数字是此行的地址；
2. unk_403040 是此处的变量名（其他位置没有名字是因为IDA没有识别到其他位置被使用）；
3. db 表明此位置是一个占用一字节的数值，相似的还有 dw（占用两字节），dd（占用四字节），dq（占用8字节）；41h 是这里的值，h 表示 16 进制；后面的 ; 是注释，A 是说明 0x41 转成 char 为 A。

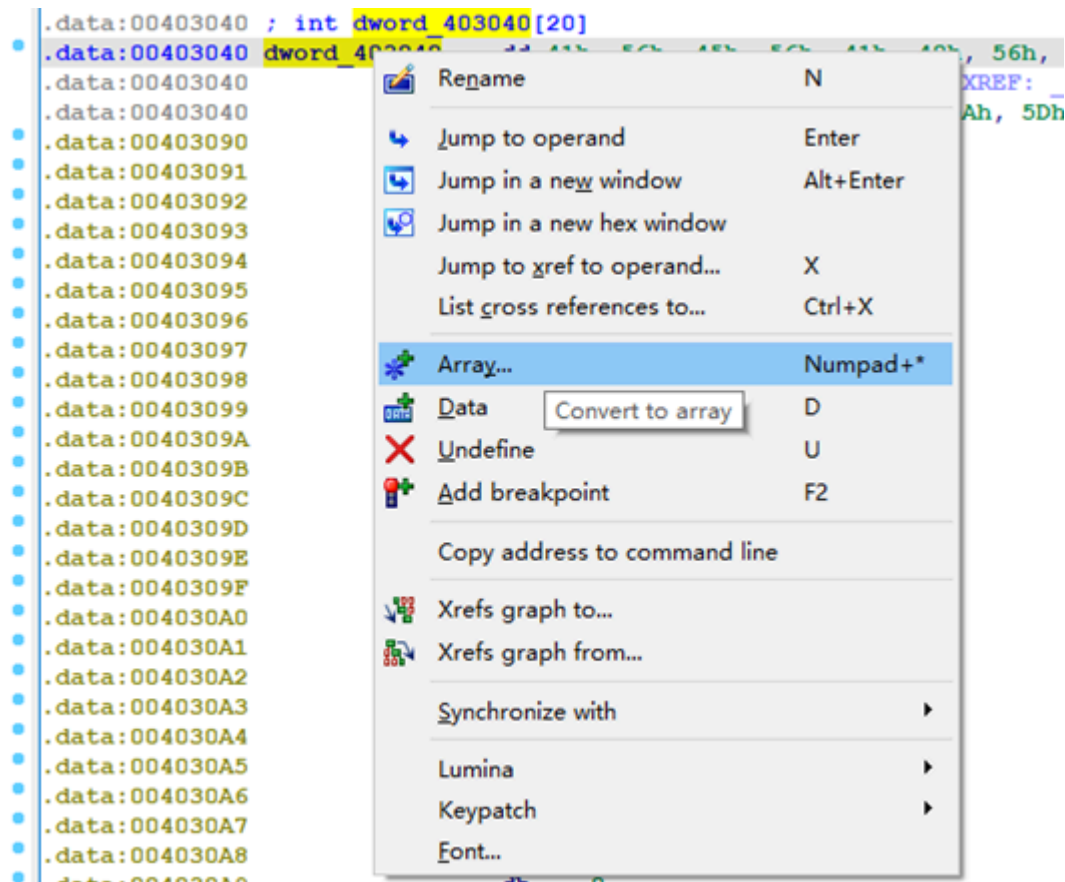
现在我们修改其类型为int[20],于是在上图所示界面，鼠标左键点击unk_403040，使其高亮，然后按 y（在反汇编窗口的数据处、伪代码窗口的变量处都能按y修改变量类型定义），弹出修改类型的窗口，在此窗口输入 C 语言的类型定义，int [20]或int unk_403040[20]，再 OK：



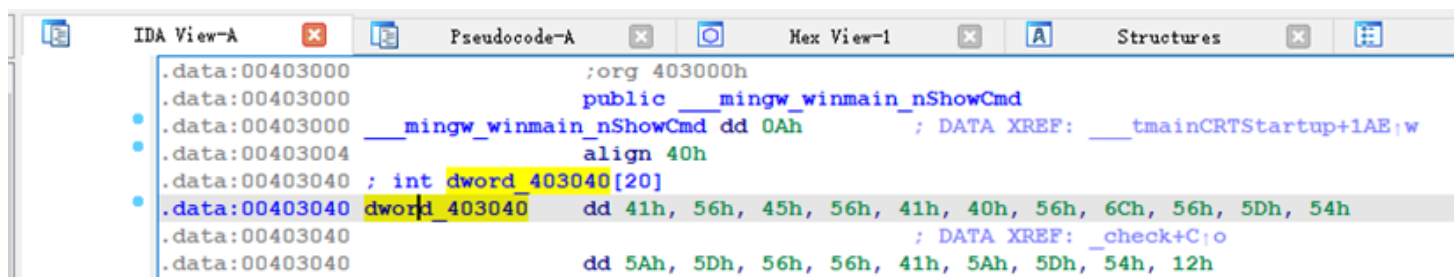
修改后的结果：



可以看到 IDA 能正确展示这些值了。中间有一个 2 dup(56h)，表示这里有两个连续的 0x56。要想将数据展示为熟悉的形式，在 dword_403040 处右键，Array...：

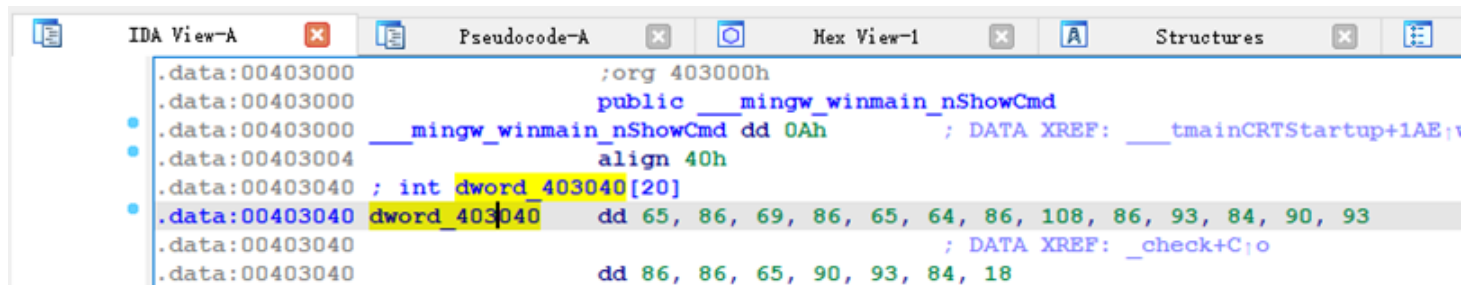


在弹出的窗口可以设置这个数组的属性、展示方式，各个选项可自行探索。去除左下角的 Use “dup” construct 即可去掉 dup：



```
.data:00403000      ;org 403000h
.data:00403000      public __mingw_winmain_nShowCmd
.data:00403000      __mingw_winmain_nShowCmd dd 0Ah          ; DATA XREF: __tmainCRTStartup+1AE;w
.data:00403004      align 40h
.data:00403040      ; int dword_403040[20]
.data:00403040      dword_403040 dd 41h, 56h, 45h, 56h, 41h, 40h, 56h, 6Ch, 56h, 5Dh, 54h
.data:00403040      ; DATA XREF: _check+C;0
.data:00403040      dd 5Ah, 5Dh, 56h, 56h, 41h, 5Ah, 5Dh, 54h, 12h
```

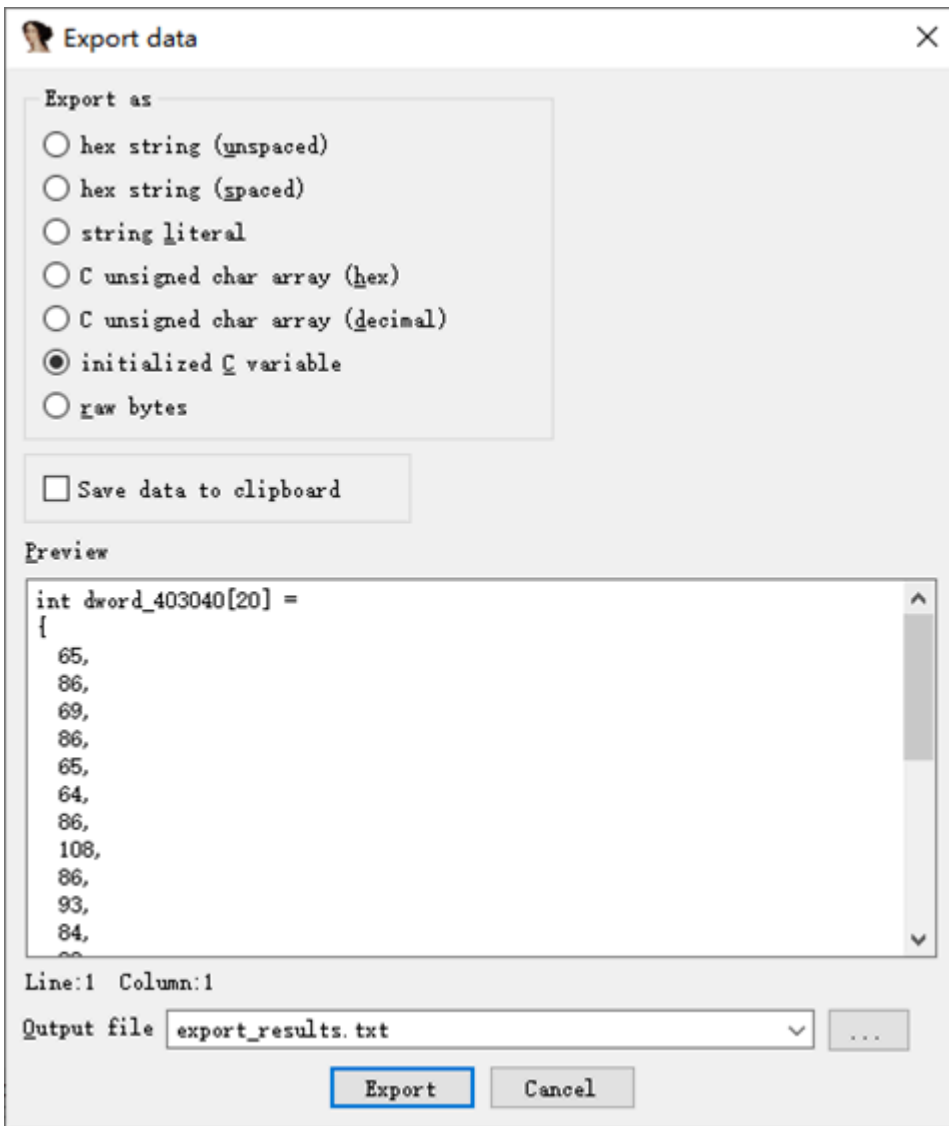
这种 16 进制的展示方式不是 C 语言的标准形式，看起来并不友好，可以按 h 切换 10/16 进制表示：



```
.data:00403000      ;org 403000h
.data:00403000      public __mingw_winmain_nShowCmd
.data:00403000      __mingw_winmain_nShowCmd dd 0Ah          ; DATA XREF: __tmainCRTStartup+1AE;w
.data:00403004      align 40h
.data:00403040      ; int dword_403040[20]
.data:00403040      dword_403040 dd 65, 86, 69, 86, 65, 64, 86, 108, 86, 93, 84, 90, 93
.data:00403040      ; DATA XREF: _check+C;0
.data:00403040      dd 86, 86, 65, 90, 93, 84, 18
```

这样看着就习惯多了。

如果要将数组的值导出，可以在上图中直接复制，更简单的方式是使用IDA提供的功能，再此处按下 shift+e（或者菜单栏的 Edit -> Export data），弹出的窗口选择 initialized C variable，这样下面的 Preview 框就有标准的 C 语言定义了。可以直接复制，也可以按 Export 导出到文件，写代码时就可以直接用了。



其他一些说明

- 键盘上"键可以展示或者消除类型转化的提示
- 键盘上'/'键可以输入注释
- ALT+M 可以设置书签
- CTRL+M 可以查看所有书签
- ida关闭时默认关闭会保存database，再次拖入文件的话可以重新读取

本次讲义由颜浩翔编写，参考杨锦东学长编写的逆向入门