软件逆向工程（CE204114）

孙聪

网络与信息安全学院

2020-08-31

教材与参考书

- 教材：软件逆向工程原理与实践，孙聪等，西安电子科技大学出版社，2018

- 随书代码:https://github.com/suncongxd/sw-re-book-code
- 课程资料:https://github.com/suncongxd/sw-re-course
- 参考书
  - Practical Reverse Engineering, Bruce Dang et al, Wiley. 2014
  - 逆向工程核心原理，李承远，人民邮电出版社，2014
  - Windows核心编程，Richter，机械工业出版社，2008
  - IA-32 Intel Architecture Software Developer’s Manual, Vol 1

成绩评定

- 期末考试：60%
- 上机作业（3次，8学时）：30%
  - 完成与汇编程序分析和编程、二进制文件分析、调试工具使用、DLL注入等相关的上机实验
  - 提交上机报告到助教邮箱
  - 如果上机作业有编程要求，应提交相应的源代码
  - 按照规定的deadline提交，超过deadline扣除该次作业所有分数
- 平时书面作业：10%
  - 提交书面作业文档到助教邮箱
  - 按照规定的deadline提交，超过deadline扣除该次作业所有分数

助教

- 潘建锋（sudo_rm@163.com）

课程内容

- 概述
- x86与x64体系结构
- 静态反汇编
- PE/ELF文件格式
- Windows系统编程基础
- DLL注入
- API钩取（线上）
- 调试工具与调试技术（线上）
- 混淆技术

课程内容

- 概述
- x86与x64体系结构
- 静态反汇编
- PE/ELF文件格式
- Windows系统编程基础
- DLL注入
- API钩取
- 调试工具与调试技术
- 混淆技术

提要

1. 逆向工程的概念和基本方法
2. 逆向工程的应用与合法性
3. 逆向分析Hello World程序

提要

什么是逆向工程

Reverse Engineering

定义1

the process of understanding a system

- system: 硬件设备、软件程序、协议、物理过程、化学过程、……
- 对于本课程：system=软件程序

定义2

从对象中提取知识或设计信息，并重建对象或基于对象信息的任意事物的过程

什么是软件逆向工程

定义

一种分析目标系统的过程，其目的是识别出系统的各个组件以及它们之间的关系，并在较高的抽象层次上以另一形式创建对系统的表示[1]，从而理解目标系统的结构和行为

两阶段过程

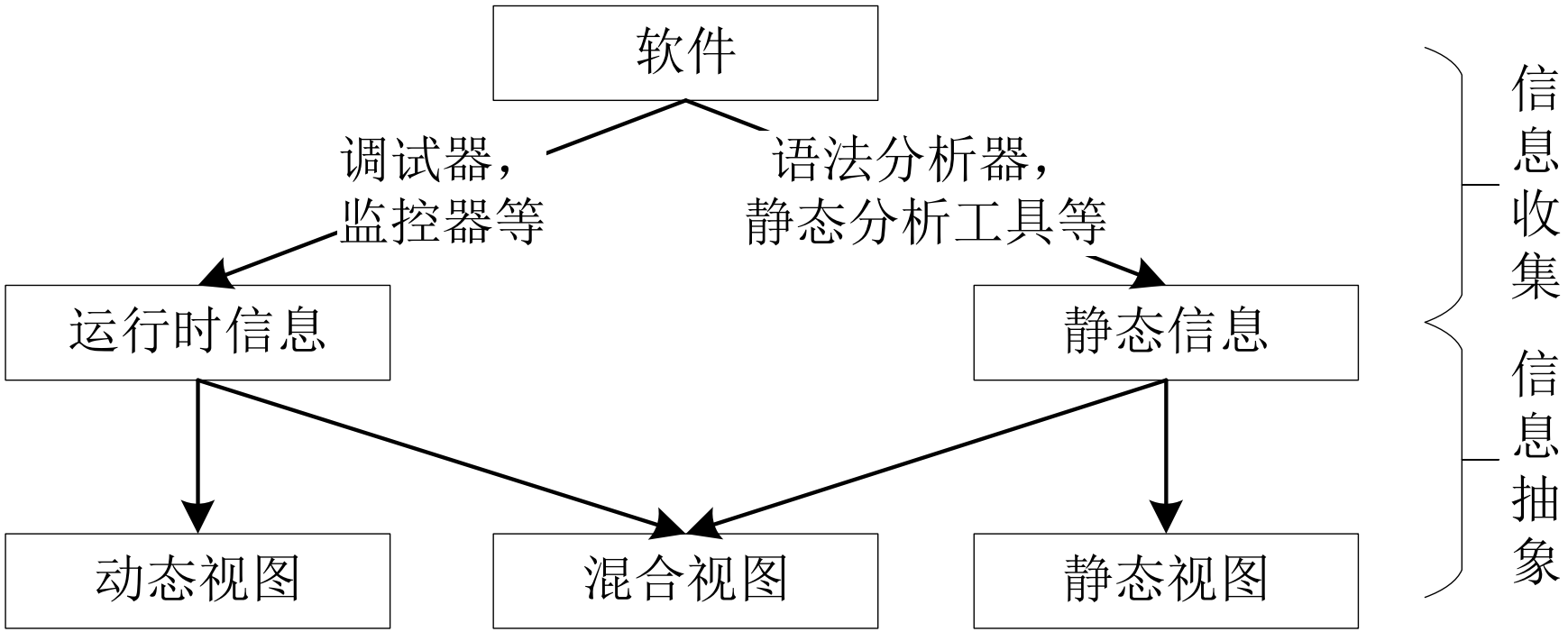
- 收集信息：parsers, debuggers, profilers, event recorders
- 抽象信息：构造可理解的高层次模型(静态或动态)

- Programmers have become part historian, part detective, and part clairvoyant[2].

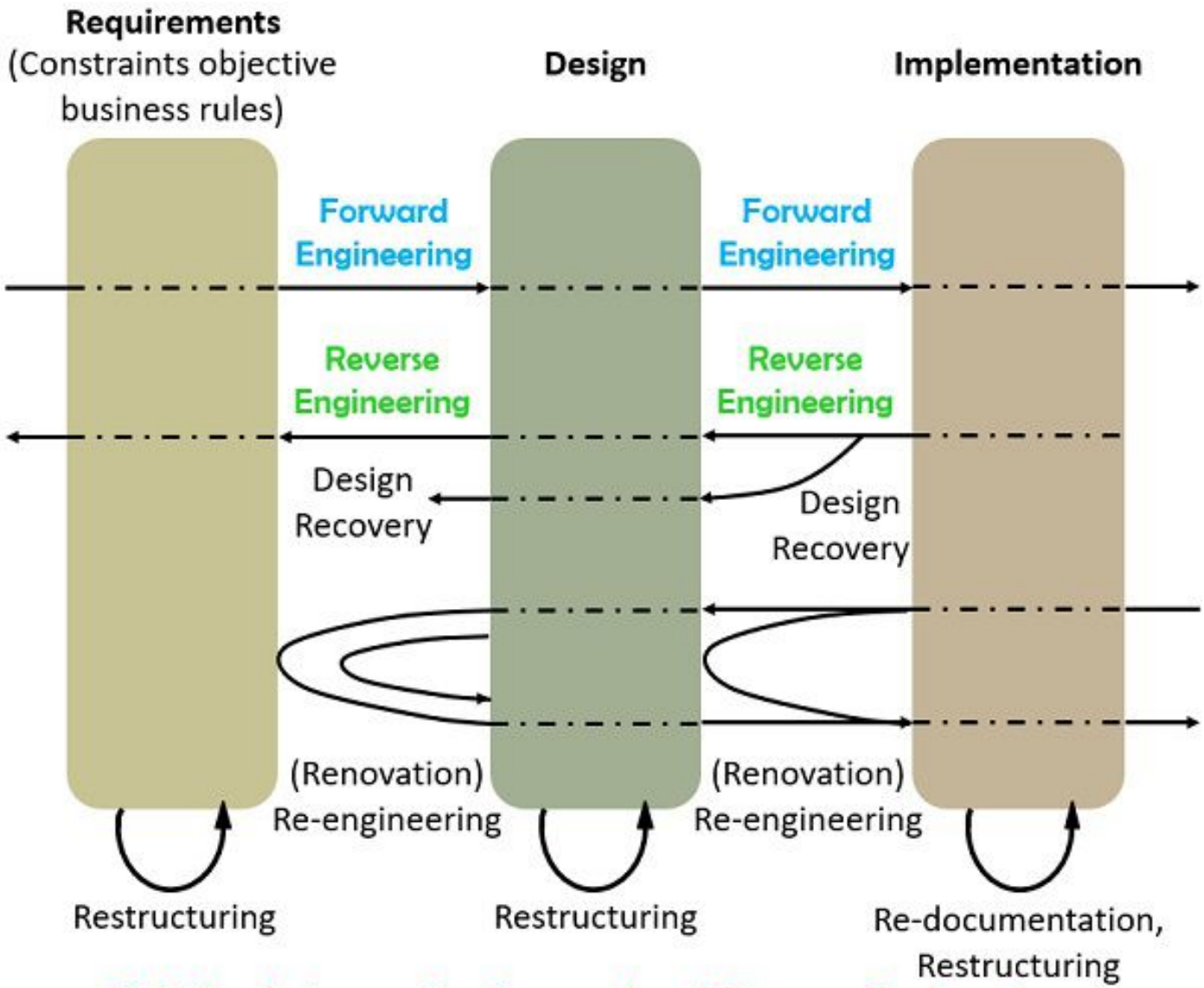
1 Chikofsky E J, Cross J H. Reverse engineering and design recovery: A taxonomy. IEEE software, 1990, 7(1): 13-17.

2 Corbi T A. Program Understanding: Challenge for the 1990s. IBM Systems Journal, 28(2):294-306, 1989.

什么是软件逆向工程

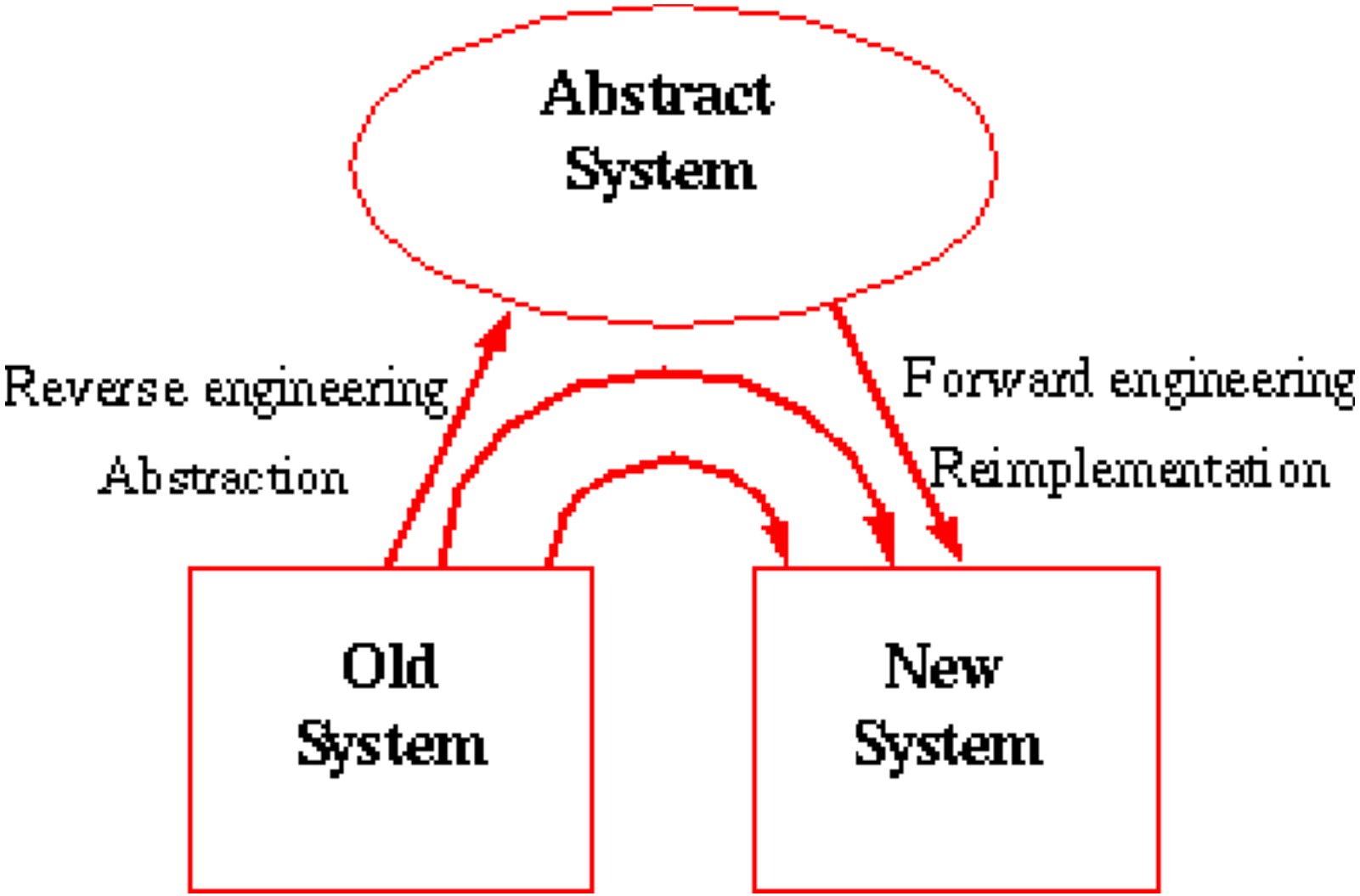

软件
调试器，
监控器等
语法分析器，
静态分析工具等
运行时信息
静态信息
动态视图
混合视图
静态视图
信息收集
信息抽象

软件逆向工程与正向工程



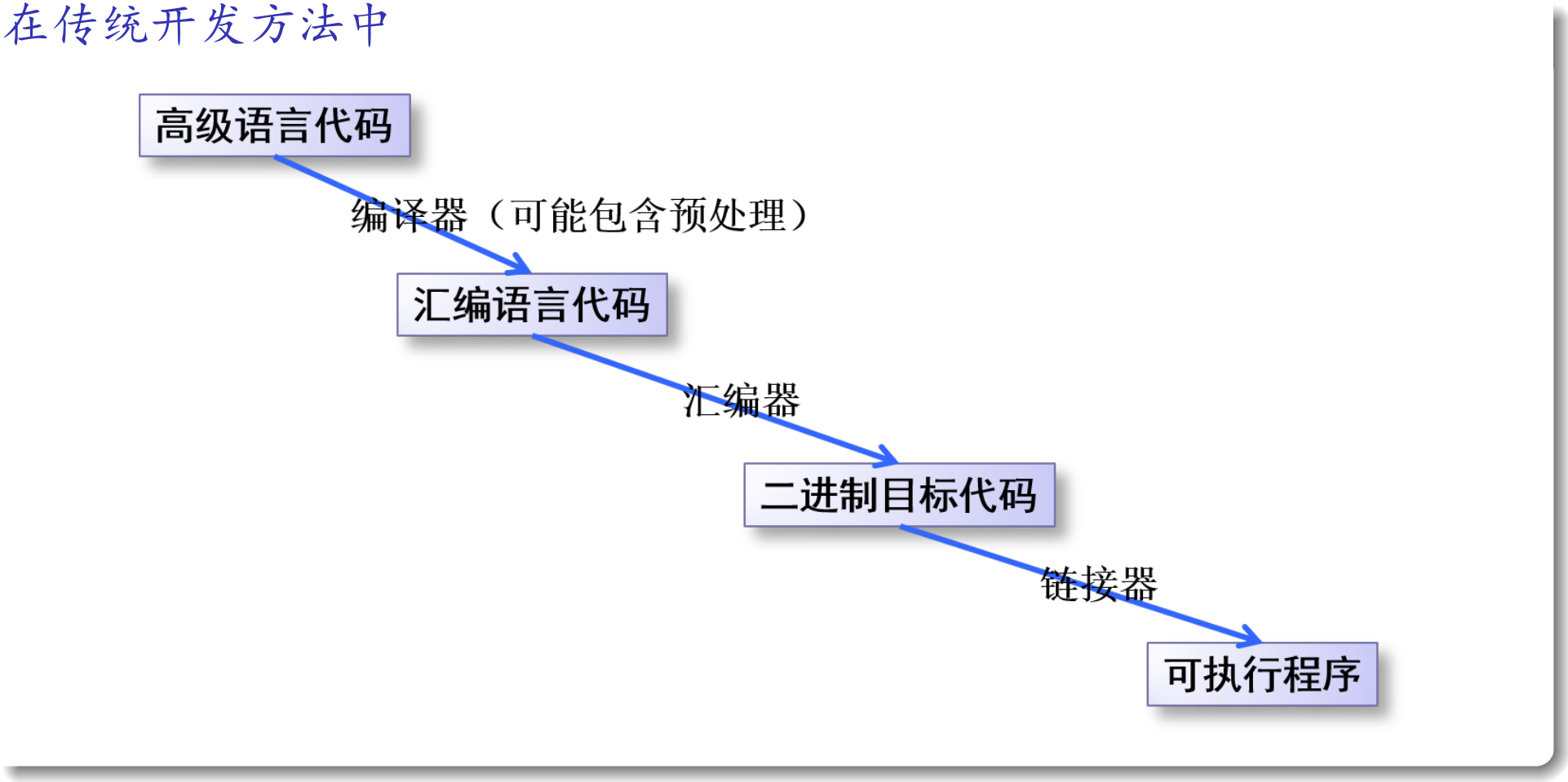
Relation between the Forward and Reverse Engineering

软件逆向工程与软件再工程



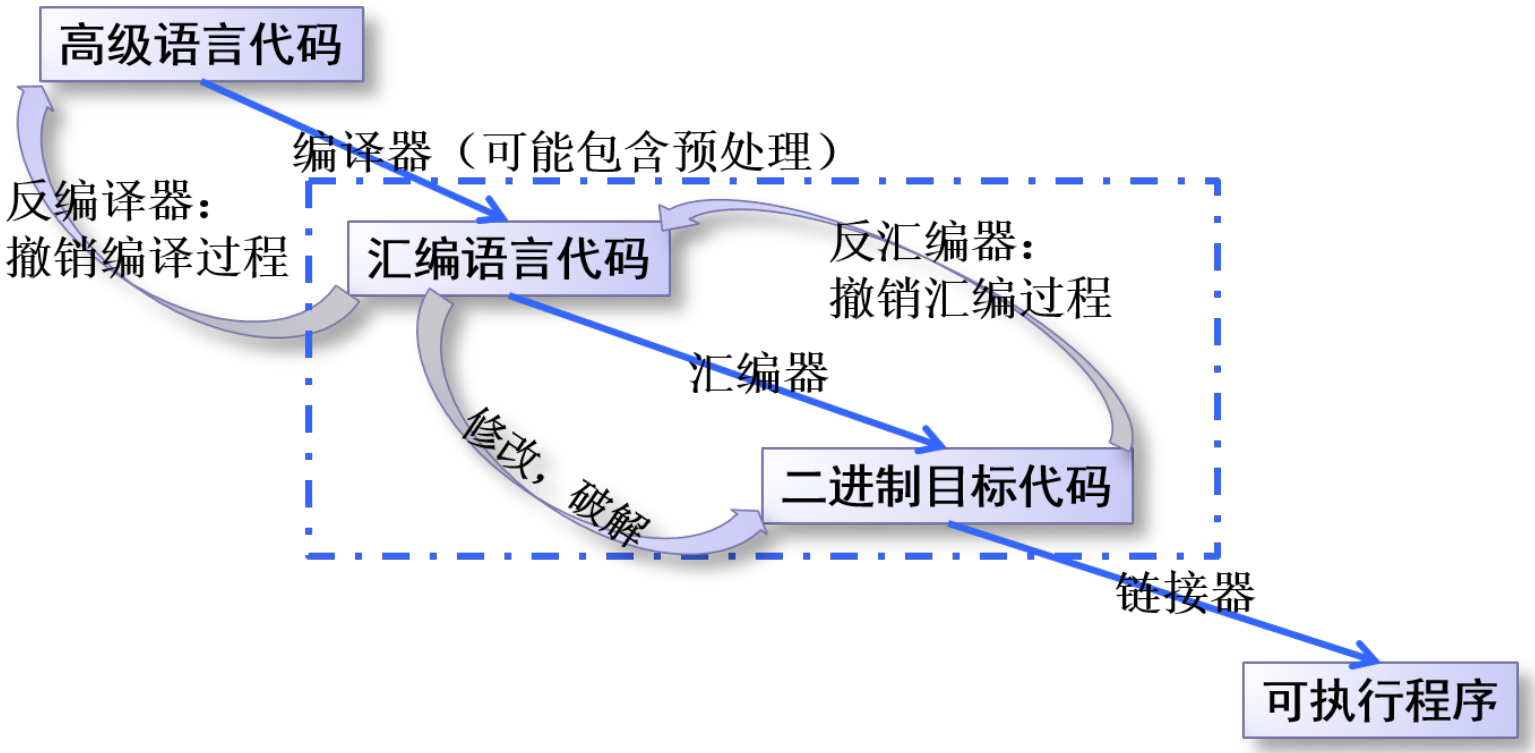
软件逆向工程的基本方法论

在传统开发方法中



软件逆向工程的基本方法论

逆向工程的主要活动：反汇编 + 分析 + 可能的修改(破解)



静态方法 vs. 动态方法

- 静态方法
  - 分析但不运行代码
  - 较动态方法更为安全
  - 反汇编器，如IDA Pro，objdump等
- 动态方法
  - 检查进程执行过程中寄存器、内存值的实时变化
  - 允许操作进程，通常应在虚拟机中运行
  - 调试器，如Windows下的WinDBG，Immunity，OllyDBG等，及Linux下的GDB

逆向过程的困难性

以反编译为例

- 编译过程造成信息损失
  - 机器语言中没有变量名、函数名，变量类型信息难以确定
- 编译属于多对多操作
  - 源程序可通过多种不同方式转换为汇编程序
  - 汇编程序也可通过多种不同方式转换为源程序
  - 对编译结果的反编译得到截然不同的源文件
- 反编译器依赖于编程语言和库
  - C语言的反编译器与Java反编译器不同
  - 反编译Windows二进制文件需要用支持Windows API的反编译器

提要

软件逆向工程的典型应用场景

- 恶意代码分析
  - 动态分析：沙箱中执行恶意程序，记录并报告恶意行为
  - 静态分析：通过程序代码理解程序行为（Program understanding/comprehension），自动/人工
- 闭源软件漏洞分析
  - 发现和分析漏洞：fuzzing或静态分析，静态分析以反汇编为基础
  - 开发破解程序（exploit）：结合使用反汇编器和调试器
- 闭源软件互操作性分析
  - 只能获得二进制程序而想开发与其互操作的软件
  - 依据已有程序，开发适用于其他硬件平台的程序(SW migration)，特别适用于支持新硬件的软件驱动程序
- 验证编译器的性能和准确性；调试器中生成汇编代码清单…

软件逆向工程的合法性

通常允许

- 以互操作为目的的逆向工程
- 当软件版权所有者无法进行软件错误修正时，通过逆向工程对软件错误进行修正和破解
- 在不违反专利权或商业秘密保护的前提下，通过逆向工程确定软件中不受版权保护的部分（如一些算法）

软件逆向工程的合法性

针对软件逆向工程的法规通常声明如下合法性限制

- 逆向工程人员为合法用户
- 逆向工程以互操作为目的，仅对实现互操作程序所必要的那部分程序进行逆向工程
- 需获取的“必要信息”不能从其他途径取得
- 通过逆向工程获得的信息不能用于实现互操作程序以外的目的，不能扩散给不必要的第三人
- 不能用于开发形式类似或有其他著作权侵权因素的程序
- 不得不合理地损害权利人的正当利益或妨碍计算机程序的正常使用

软件逆向工程的合法性

美国

- 对人工制品和过程的逆向工程权随该制品/过程的合法获得而被法律许可
- 对计算机软件的逆向工程受合同法约束
  - 由于大多数终端用户许可证(EULA)都禁止逆向工程，因而对合法取得的软件进行逆向工程也会因违反EULA而违反合同法

软件逆向工程的合法性

欧盟(1991 EC Copyright Directive on Legal Protection of Computer Programs)

- 对计算机程序进行非授权的复制、翻译、改编和转换构成对作者的侵权
- 对代码的复制和翻译如果对于实现该程序与其他程序的互操作而言不可避免，那么有权使用该软件的人可以复制和翻译该软件而应被视作无需获得权利人授权

案例: Sony Computer Entertainment Inc. v. Connectix Corp.
http://digital-law-online.info/cases/53PQ2D1705.htm

提要

HelloWorld程序

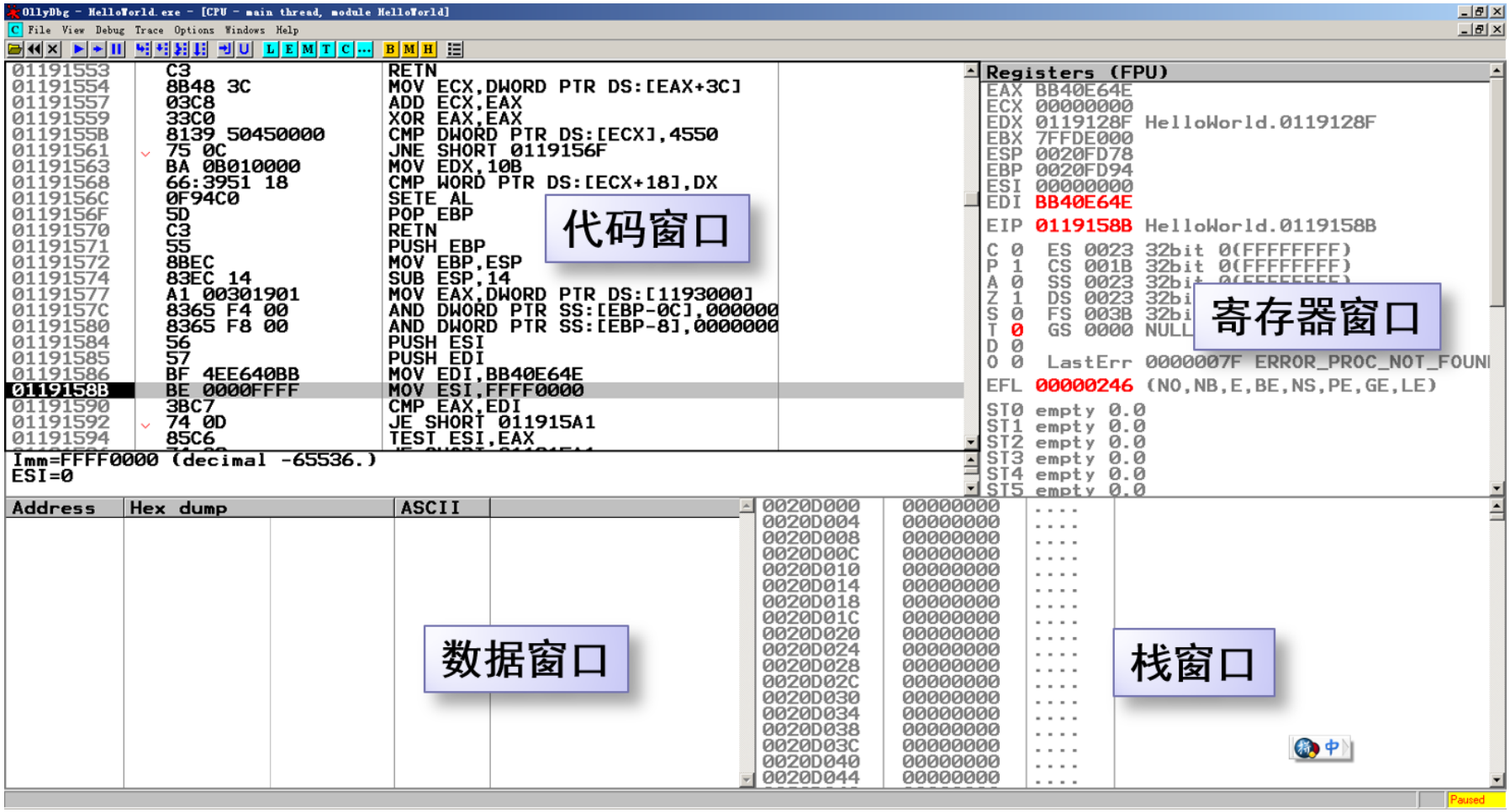
用VS编译以下程序(Release模式)，生成32位HelloWorld.exe

```
#include ”windows.h”
#include ”tchar.h”

int _tmain(int argc, TCHAR *argv[]){
    MessageBox(NULL,
        _T(”Hello World!”),
        _T(”www.xidian.edu.com”),
        MB_OK);
    return 0;
}
```

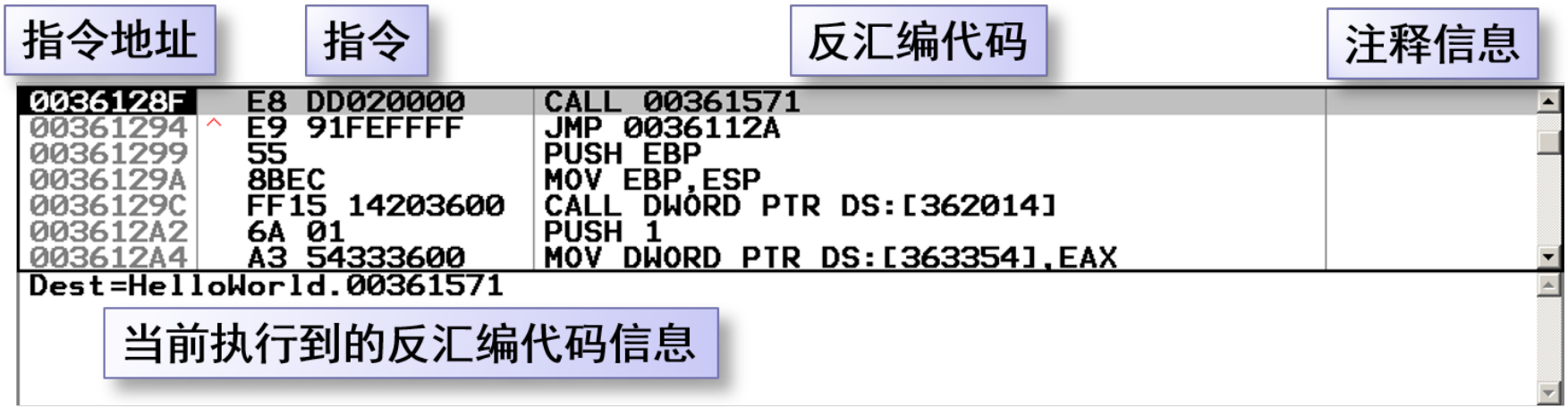
OllyDbg界面

- 将HelloWorld.exe装入OllyDbg

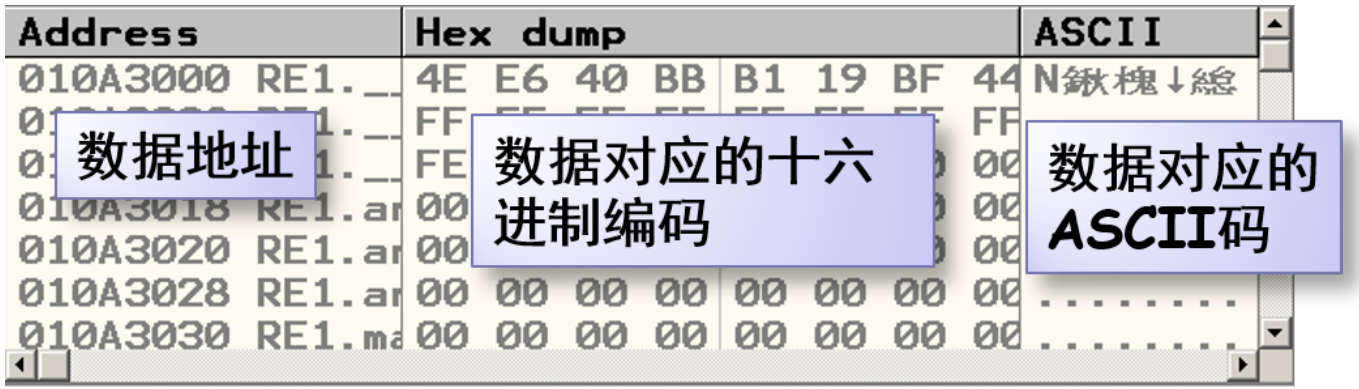


OllyDbg界面

- 代码窗口



- 数据窗口



OllyDbg界面

- 栈窗口

OllyDbg基本指令

快捷键	含义
F7	Step Into. 执行一句操作码，若遇调用命令CALL，则进入被调用函数代码内部
F8	Step Over. 执行一句操作码，若遇调用命令CALL，则直接执行函数，不进入函数内部
Ctrl+F2	重新开始调试(终止正在调试的进程后再次运行)
Ctrl+F9	运行到函数的RETN命令处，用于跳出函数
F2	设置断点
F9	程序运行到下一个断点处暂停
Alt+B	查看当前的断点列表
Alt+M	打开内存映射窗口(查看哪些库被加载到内存中,e.g.User32.dll)
Alt+F9	运行到用户代码处，用于快速跳出系统函数
Ctrl+G	光标移动到指定地址
Ctrl+E	打开编辑窗口，编辑已选内容
F4	让调试流运行到光标所在位置
;	添加注释
:	在指定地址添加特定标签

OllyDbg常用操作

- go to：光标移到目标地址（Ctrl+G），然后F4
- 列出程序中引用的所有字符串：汇编代码窗口右键 -> Search for -> All referenced text string
- 列出程序调用的API函数列表：汇编代码窗口右键 -> Search for -> All intermodular calls
- 保存对二进制的更改：选中更改的字符串，右键 -> Copy to executable file

OllyDbg常用操作

练习

- 在一条call指令处通过step into进入callee函数
- 通过step over跟踪main函数的返回过程(找到main函数的起始指令，找到main函数中的RETN 指令，在该RETN指令处设置断点，执行到该RETN指令，然后step over)

从指定的点开始调试

每次重新运行调试器，都从程序入口点起始，经常需要设置重要的点(地址)，并迅速跳转到该点进行调试。具体方法：

- 设置断点(F2)，运行到下一个断点(F9)，用Alt+B查看断点列表
- 运行到光标所在位置(F4)，如何提前将光标移到我们想要开始调试的位置？
  - 知道目标地址，光标移到目标地址(Ctrl+G)
  - 添加注释(;)，跳转到指定注释(右键点击代码区，Search for all user comments，双击要跳转到的注释)
  - 添加标签(:)，跳转到指定标签(Search for user-defined label，双击要跳转到的标签)

快速查找指定代码

- 代码执行法
  - 适用于代码量不大、功能明确的程序
  - 不断Step Over(F8)，直到通过预期的程序行为判断出特定代码片段正在执行
- 字符串检索法
  - 代码区Search for all referenced text string
- API检索法
  - 查找程序运行时调用的API列表：代码区Search for all intermodular calls
  - 查找被加载的DLL中提供的所有API：代码区Search for name in all modules

使用代码执行法找到main函数

```
0020112A    6A 0C           PUSH 0C
0020112C    68 18222000     PUSH OFFSET 00202218
00201131    E8 4A060000     CALL 00201780
00201136    33DB            XOR EBX,EBX
00201138    895D FC         MOV DWORD PTR SS:[EBP-4],EBX
0020113B    64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
00201141    8B50 04         MOV EDX,DWORD PTR DS:[EAX+4]
00201144    8BFB            MOV EDI,EBX
00201146    BE 68332000     MOV ESI,OFFSET 00203368
0020114B    8BCA            MOV ECX,EDX
0020114D    33C0            XOR EAX,EAX
0020114F    F0:0FB10E       LOCK CMPXCHG DWORD PTR DS:[ESI],
00201153    85C0            TEST EAX,EAX
00201155    74 0B           JE SHORT 00201162
00201157    3BC2            CMP EAX,EDX
00201159    75 F0           JNE SHORT 0020114B
0020115B    33F6            XOR ESI,ESI
0020115D    46              INC ESI
0020115E    8BFE            MOV EDI,ESI
00201160    EB 03           JMP SHORT 00201165
00201162    33F6            XOR ESI,ESI
00201164    46              INC ESI
00201165    3935 6C332000   CMP DWORD PTR DS:[20336C],ESI
0020116B    75 0A           JNE SHORT 00201177
0020116D    6A 1F           PUSH 1F
0020116F    E8 B0020000     CALL 00201424
00201174    59              POP ECX
00201175    EB 3A           JMP SHORT 002011B1
00201177    391D 6C332000   CMP DWORD PTR DS:[20336C],EBX
0020117D    75 2C           JNE SHORT 002011AB
0020117F    8935 6C332000   MOV DWORD PTR DS:[20336C],ESI
00201185    68 B8202000     PUSH OFFSET 002020B8
0020118A    68 A8202000     PUSH OFFSET 002020A8
0020118F    E8 D8050000     CALL 0020176C
00201194    59              POP ECX
00201195    59              POP ECX
00201196    85C0            TEST EAX,EAX
00201198    74 17           JE SHORT 002011B1
0020119A    C745 FC FEFFFFF MOV DWORD PTR SS:[EBP-4],-2
002011A1    B8 FF000000     MOV EAX,0FF
002011A6    E9 DE000000     JMP 00201289
002011AB    8935 1C302000   MOV DWORD PTR DS:[20301C],ESI
002011B1    3935 6C332000   CMP DWORD PTR DS:[20336C],ESI
002011B7    75 1B           JNE SHORT 002011D4
002011B9    68 A4202000     PUSH OFFSET 002020A4
```

启动函数（Stub Code），又称Startup Code。不同编译器根据自身特点添加的代码。

调试程序时，不需要仔细分析启动函数，但应能区分启动函数和用户代码

如何快速找到用户代码？

使用代码执行法找到main函数

```
0020112A    6A 0C           PUSH 0C
0020112C    68 18222000     PUSH OFFSET 00202218
00201131    E8 4A060000     CALL 00201780
00201136    33DB            XOR EBX,EBX
00201138    895D FC         MOV DWORD PTR SS:[EBP-4],EBX
0020113B    64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
00201141    8B50 04         MOV EDX,DWORD PTR DS:[EAX+4]
00201144    8BFB            MOV EDI,EBX
Dest=HelloWorld.00201780
```

由启动函数向下依次找CALL所调用的函数体中是否有main()

```
000B120E    8908            MOV DWORD PTR DS:[EAX],ECX
000B1210    FF35 28300B00   PUSH DWORD PTR DS:[0B3028]
000B1216    FF35 24300B00   PUSH DWORD PTR DS:[0B3024]
000B121C    FF35 20300B00   PUSH DWORD PTR DS:[0B3020]
000B1222    E8 D9FDFFFF     CALL 000B1000
000B1227    83C4 0C         ADD ESP,0C
000B122A    A3 18300B00     MOV DWORD PTR DS:[0B3018],EAX
000B122F    833D 30300B00 00 CMP DWORD PTR DS:[0B3030],0
Dest=HelloWorld.000B1000
```

当找到对000B1000的函数的调用时，观察该函数内部

```
000B1000    6A 00           PUSH 0
000B1002    68 00210B00     PUSH OFFSET 000B2100        ASCII "www.reversecore.com"
000B1007    68 14210B00     PUSH OFFSET 000B2114        ASCII "Hello World!"
000B100C    6A 00           PUSH 0
000B100E    FF15 94200B00   CALL DWORD PTR DS:[0B2094]
000B1014    33C0            XOR EAX,EAX
000B1016    C3              RETN
[000B2094]=USER32.76EDEA11
```

找到参数压栈操作

调用MessageBoxA()函数

由此可见000B1000函数为main()

为HelloWorld打补丁

将Hello World改为Hello Student

- 找到Hello World字符串所在内存

```
01221000  $ 6A 00      PUSH 0                    Style = MB_OK|MB_APPLMODAL
01221002  . 68 F4202201 PUSH OFFSET HelloWor.?    Title = "www.xidian.edu.com"
01221007  . 68 08212201 PUSH OFFSET HelloWor.?    Text = "Hello World!"
0122100C  . 6A 00       PUSH 0                    hOwner = NULL
0122100E  . FF15 A4202201 CALL DWORD PTR DS:[<&U  MessageBoxA
01221014  . 33C0        XOR EAX,EAX
01221016  . C3          RETN
01222108  OFFSET HelloWor.??_C@_0N@GCDOMLDM@Hello?5World?$CB?$AA@ (ASCII "Hello
```

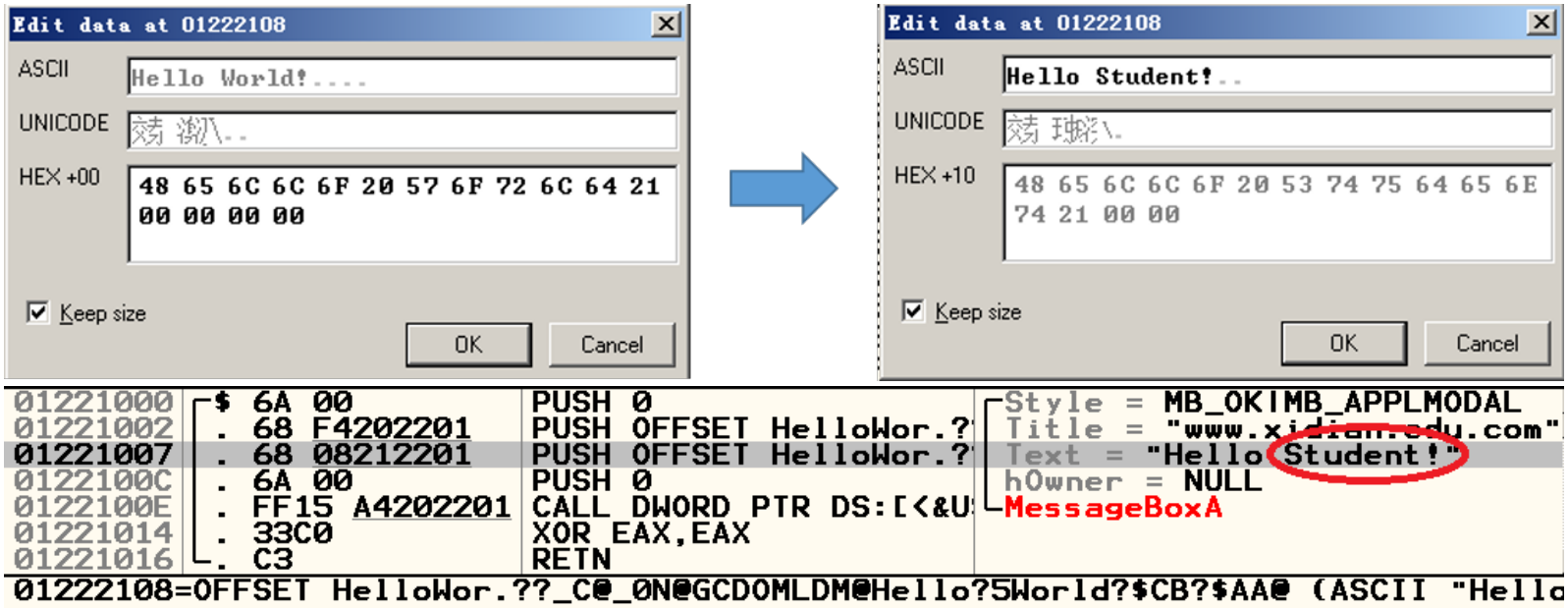
- 在数据窗口Ctrl+G，找到01222108

Address	Hex dump	ASCII
012220E8	60 0F 00 00 38 30 22 01 90 30 22 01 77 77 77 2E	'✲..80"⊖?"⊖www.
012220F8	78 69 64 69 61 6E 2E 65 64 75 2E 63 6F 6D 00 00	xidian.edu.com..
01222108	48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00 00 00 00	Hello World!....
01222118	48 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	H...............

为HelloWorld打补丁

将Hello World改为Hello Student

- 选中数据窗口的字符串，Ctrl+E打开编辑窗口进行编辑



- 保存对二进制的更改：选中更改的字符串，鼠标右键Copy to executable file

为HelloWorld打补丁

练习

- 尝试在内存其它区域新建字符串后传递给消息函数
- 如何调换“www.xidian.edu.cn”和“Hello Student!”的显示位置?

课后练习

- 按照老师在课堂上的逆向分析操作，在自己的电脑上对VC++编写的HelloWorld程序进行反汇编和逆向分析
- 1.6“思考与练习”：
  对以下C程序进行反汇编和逆向工程，得到能够输出“Reverse Eng”的.exe程序

```
#include <stdio.h>
int main(int argc, char* argv[]){
  printf(”Hello World\n”);
  return 0;
}
```

书面作业：阅读论文并撰写阅读报告

- Chikofsky E J, Cross J H. Reverse engineering and design recovery: A taxonomy. IEEE software, 1990, 7(1): 13-17.
- Canfora G, Di Penta M. New Frontiers of Reverse Engineering[C]// Future of Software Engineering (FOSE'07). 2007, 326-341.
- Canfora G, Di Penta M, Cerulo L. Achievements and challenges in software reverse engineering[J]. Communications of the ACM, 2011, 54(4): 142-151.
- Nelson M L. A Survey of Reverse Engineering and Program Comprehension, ODU CS 551-Software Engineering Survey, 1996.

- 4选1，写1篇1000字阅读报告，发送到助教邮箱，
  文档题目“[学号]姓名-paper1/2/3/4”
- Deadline: 2020/09/13, 23:59:59