南開大學

汇编语言与逆向技术课程实验报告

实验八：REChallenge

学　　院　网络空间安全学院

专　　业　信息安全

学　　号　2313546

姓　　名　蒋枘言

班　　级　信息安全班

一、实验目的

1.熟悉反编译工具 Binary Ninja；

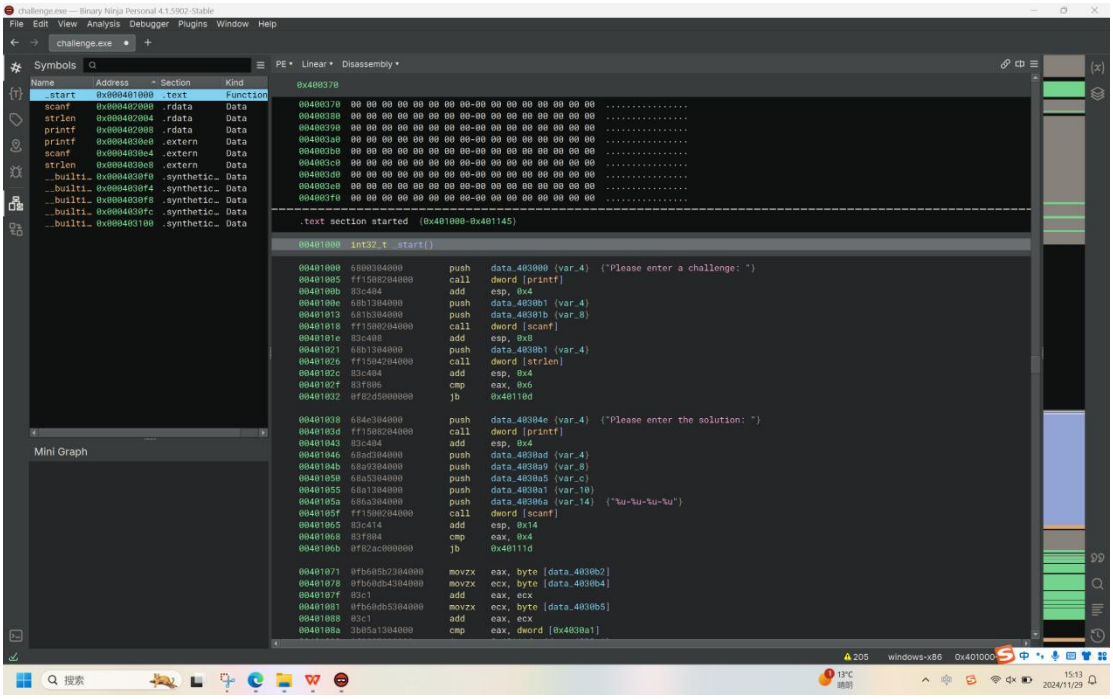
2.熟悉反汇编代码的逆向分析过程；

3.掌握反汇编语言中的数学计算、数据结构、条件判断、分支结构的识别和逆向分析。

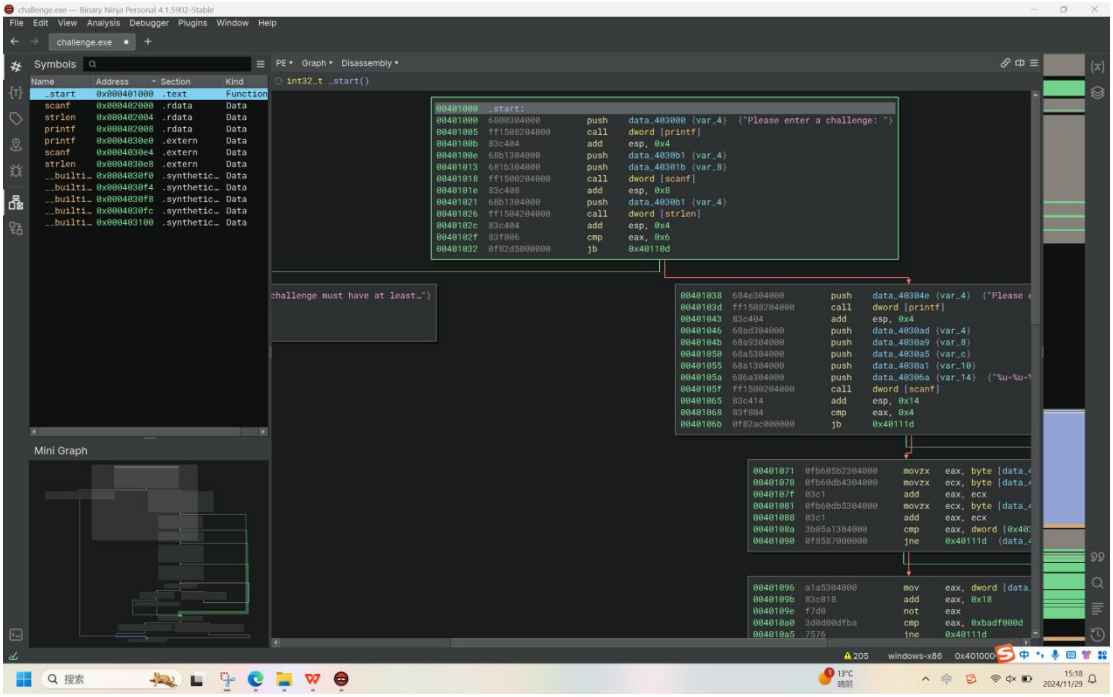
二、打开文件

使用 Binary Ninja 打开待分析的 exe 文件：challenge.exe。

反汇编代码如下图所示：



图形化视窗如下图所示：



三、逆向分析过程（代码逐行解释详见截图）

1.输入判断

```
00401000  _start:
00401000  // 这行代码将内存地址00403000（即data_403000）处的数据压入栈中。
00401000  // 注释表明这是一个字符串的地址，
00401000  // 且字符串内容为"Please enter a challenge: "。
00401000  6800304000          push    data_403000 {var_4}  {"Please enter a challenge: "}
00401005  // 调用printf函数，传递的参数是之前压入栈的字符串。
00401005  ff1508204000        call    dword [printf]
0040100b  // 这行代码清理栈，将栈指针esp向上移动4个字节，以移除之前push的字符串地址。
0040100b  83c404              add     esp, 0x4
0040100e  // 将另一个内存地址004030b1（即data_4030b1）处的数据压入栈中。
0040100e  // 这个地址可能指向一个用于接收用户输入的缓冲区。
0040100e  68b1304000          push    data_4030b1 {var_4}
00401013  // 将另一个内存地址0040301b（即data_40301b）处的数据压入栈中。
00401013  // 这个地址指向一个格式字符串，用于scanf函数。
00401013  681b304000          push    data_40301b {var_8}
00401018  // 调用scanf函数，根据之前压入栈的格式字符串和缓冲区地址来读取用户输入。
00401018  ff1500204000        call    dword [scanf]
0040101e  // 清理栈，将栈指针esp向上移动8个字节，以移除之前push的两个地址。
0040101e  83c408              add     esp, 0x8
00401021  // 再次将用户输入缓冲区的地址压入栈中，这次是为了传递给strlen函数。
00401021  68b1304000          push    data_4030b1 {var_4}
00401026  // 调用strlen函数，计算用户输入字符串的长度，并将结果存储在eax寄存器中。
00401026  ff1504204000        call    dword [strlen]
0040102c  // 清理栈，将栈指针esp向上移动4个字节，以移除之前push的用户输入缓冲区地址。
0040102c  83c404              add     esp, 0x4
0040102f  // 将eax寄存器中的值（即strlen函数的返回值）与6进行比较。
0040102f  83f806              cmp     eax, 0x6
00401032  // 如果eax中的值小于6（即字符串长度小于6），则跳转到地址0040110d。
00401032  0f82d5000000        jb      0x40110d
```

这是程序.text 节的入口点位置（00401000 处）。这一小部分的主要作用是提示用户输入一个字符串 challenge，并且对输入的字符串 challenge 进行判断。如果字符串 challenge 的长度小于 6，则跳转到 0040110d 位置。

2.输入的长度不符合要求

```
0040110d  681e304000          push    data_40301e {var_4}  {"The challenge must have at least…"}
00401112  ff1508204000        call    dword [printf]
00401118  83c404              add     esp, 0x4
0040111b  eb27                jmp     0x401144
```

0040110d 位置输出字符串“The challenge must have at least 6 characters”，随后跳转到位置 0401144。

3.程序结束

```
00401144  // 从当前函数返回到调用者。
00401144  c3                  retn    {__return_addr}
```

位置 0401144 是程序的结束位置。即如果字符串 challenge 的长度小于 6，在输出了字符串“The challenge must have at least 6 characters”之后，程序结束。

4.输入的长度符合要求

```
00401038  684e304000          push    data_40304e {var_4}  {"Please enter the solution: "}
0040103d  // 打印字符串"Please enter the solution: "。
0040103d  ff1508204000        call    dword [printf]
00401043  83c404              add     esp, 0x4
00401046  // 将4个地址压入栈中，用于存储更多用户输入数据。
00401046  68ad304000          push    data_4030ad {var_4}
0040104b  68a9304000          push    data_4030a9 {var_8}
00401050  68a5304000          push    data_4030a5 {var_c}
00401055  68a1304000          push    data_4030a1 {var_10}
```

```
0040105a  // 将地址0040306a压入栈中，这个地址指向字符串"%u-%u-%u-%u"，
0040105a  // 这是scanf函数用于格式化输入的格式字符串。
0040105a  // 在C语言中，格式控制符%u用于表示
0040105a  // 无符号十进制整数（unsigned int）的输出格式。
0040105a  686a304000          push    data_40306a {var_14}  {"%u-%u-%u-%u"}
0040105f  // 调用scanf函数，从标准输入读取四个无符号整数，
0040105f  // 并将它们存储在之前压入栈的地址指向的变量中。
0040105f  ff1500204000        call    dword [scanf]
00401065  // 将栈指针esp增加20（即十六进制数14），
00401065  // 以清理之前push操作压入栈的所有地址。
00401065  83c414              add     esp, 0x14
00401068  // 比较eax寄存器的值（scanf函数的返回值，表示成功读取的输入项数量）与4。
00401068  83f804              cmp     eax, 0x4
0040106b  // 如果eax小于4（即scanf没有成功读取4个输入项），则跳转到地址0040111d。
0040106b  0f82ac000000        jb      0x40111d
```

若输入的长度符合要求，首先会输出字符串“Please enter the solution: ”来引导用户输入答案。答案是一个形如“%u-%u-%u-%u”的字符串，“%u”是 C 语言的一个格式控制符，用于表示无符号十进制整数（unsigned int）的输出格式。

随后判断输入的个数是否为 4。若小于 4，则跳转到地址 0040111d。

5.输入的个数不符合要求

```
0040111d  // 将004030d1位置的数据与0作比较
0040111d  833dd130400000      cmp     dword [0x4030d1], 0x0
00401124  // 如果相等，则跳转至地址00401136处
00401124  7410                je      0x401136  {data_4030d1}
```

首先将 004030d1 位置的数据与 0 作比较。可以发现该位置的数据本来就被初始化为了 0，故相等。于是就会跳转到 00401136 位置。

```
00401136  6876304000          push    data_403076 {var_4}  {"Wrong :(\n\r"}
0040113b  ff1508204000        call    dword [printf]
00401141  83c404              add     esp, 0x4
```

随后输出字符串“Wrong :(”，代表答案错误。

按顺序执行 00401144，即**第 3 点**所讲的程序结束。

6.输入的个数符合要求——solution 第 1 个数判断

```
00401071  // 以下几行将输入的challenge的第2位、第4位、第5位相加，
00401071  // 结果与solution的第一个数比较。
00401071  0fb605b2304000      movzx   eax, byte [data_4030b2]
00401078  0fb60db4304000      movzx   ecx, byte [data_4030b4]
0040107f  03c1                add     eax, ecx
00401081  0fb60db5304000      movzx   ecx, byte [data_4030b5]
00401088  03c1                add     eax, ecx
0040108a  3b05a1304000        cmp     eax, dword [0x4030a1]
00401090  0f8587000000        jne     0x40111d  {data_4030a1}
```

将 challenge 的第 2、4、5 位相加，结果与 solution 的第 1 个数比较。若不相等，则跳转到地址 0040111d，后续和**第 5 点**相同。

7.输入的个数符合要求——solution 第 2 个数判断

```
00401096  a1a5304000          mov     eax, dword [data_4030a5]
0040109b  83c018              add     eax, 0x18
0040109e  // 对eax寄存器的值按位取反。
0040109e  f7d0                not     eax
004010a0  3d0d00dfba          cmp     eax, 0xbadf000d
004010a5  7576                jne     0x40111d
```

将 solution 的第 2 个数加上 24（即十六进制数 18），然后按位取反，与十六进制数 badf0000d 作比较。若不相等，则跳转到地址 0040111d，后续和**第 5 点**相同。

8.输入的个数符合要求——solution 第 3 个数判断

```
004010a7  a1a9304000          mov     eax, dword [data_4030a9]
004010ac  b9480c0000          mov     ecx, 0xc48
004010b1  // 将eax寄存器的符号扩展到edx:eax。
004010b1  // 即将edx设置为eax的符号位扩展到32位。
004010b1  // edx:eax表示一个64位的值，由edx和eax共同组成。
004010b1  99                  cdq
004010b2  // 用edx:eax中的值除以ecx中的值。
004010b2  // 商存储在eax中，余数存储在edx中。
004010b2  f7f1                div     ecx  {0xc48}
004010b4  8bf0                mov     esi, eax
004010b6  0fb605b1304000      movzx   eax, byte [data_4030b1]
004010bd  0fb60db3304000      movzx   ecx, byte [data_4030b3]
004010c4  // 将eax中的值乘以ecx中的值，结果存储在edx:eax中。
004010c4  // edx存储高位结果，eax存储低位结果。
004010c4  f7e1                mul     ecx
004010c6  3bc6                cmp     eax, esi
004010c8  7553                jne     0x40111d
```

将 solution 的第 3 个数除以十六进制数 c48，得到的商与 challenge 的第 1、3 位相乘的结果作比较。若不相等，则跳转到地址 0040111d，后续和**第 5 点**相同。

9.输入的个数符合要求——solution 第 4 个数判断

```
004010ca  68b1304000          push    data_4030b1 {var_4}
004010cf  ff1504204000        call    dword [strlen]
004010d5  83c404              add     esp, 0x4
004010d8  // 将eax寄存器的值移动到ecx寄存器中。
004010d8  // eax寄存器通常用于存储函数调用的返回值，
004010d8  // 这里是strlen函数的返回值（即字符串的长度）。
004010d8  8bc8                mov     ecx, eax
004010da  // 将eax寄存器的值减去其自身，用于将eax清零。
004010da  2bc0                sub     eax, eax  {0x0}
004010dc  // 使用xor（异或）操作将edx寄存器清零，因为异或自己总是得到0。
004010dc  33d2                xor     edx, edx  {0x0}
004010de  8b3dad304000        mov     edi, dword [data_4030ad]
004010e4  81f737130300        xor     edi, 0x31337
```

（第 10 点跳转位置）

```
004010ea  3bd1              cmp     edx, ecx
004010ec  730c              jae     0x4010fa
```

将 challenge 的长度与 edx 寄存器的值（被初始化为 0）作比较，若 edx 寄存器的值大于等于 challenge 的长度，则跳转至地址 004010fa。

10.edx 寄存器的值小于 challenge 的长度

```
004010ee  // 将edx+004030b1位置的值赋给寄存器ebx，
004010ee  // 并将高位清零。
004010ee  0fb69ab1304000    movzx   ebx, byte [edx+0x4030b1]
004010f5  03c3              add     eax, ebx
004010f7  // edx加1，然后接着循环。
004010f7  // 当edx的值增加到challenge的长度时就会跳出循环。
004010f7  42                inc     edx
004010f8  ebf0              jmp     0x4010ea
```

此部分是一个循环（循环到**第 9 点**中间），每次循环将 edx 加 1，循环结束后就将字符串 challenge 的每一位相加，值储存在 eax 中。

11.edx 寄存器的值大于等于 challenge 的长度

```
004010fa  83ef7b            sub     edi, 0x7b
004010fd  3bc7              cmp     eax, edi
004010ff  751c              jne     0x40111d
```

将 edi 寄存器的值减去十六进制数 7b 后与 eax 作比较。若不等，则跳转至 0040111d 位置，进入**第 5 点**。

```
00401101  c705d13040000100…  mov     dword [data_4030d1], 0x1
0040110b  eb10               jmp     0x40111d
```

若相等，则将 004030d1 位置的值修改成 1，再跳转至 0040111d 位置，进入**第 5 点**。

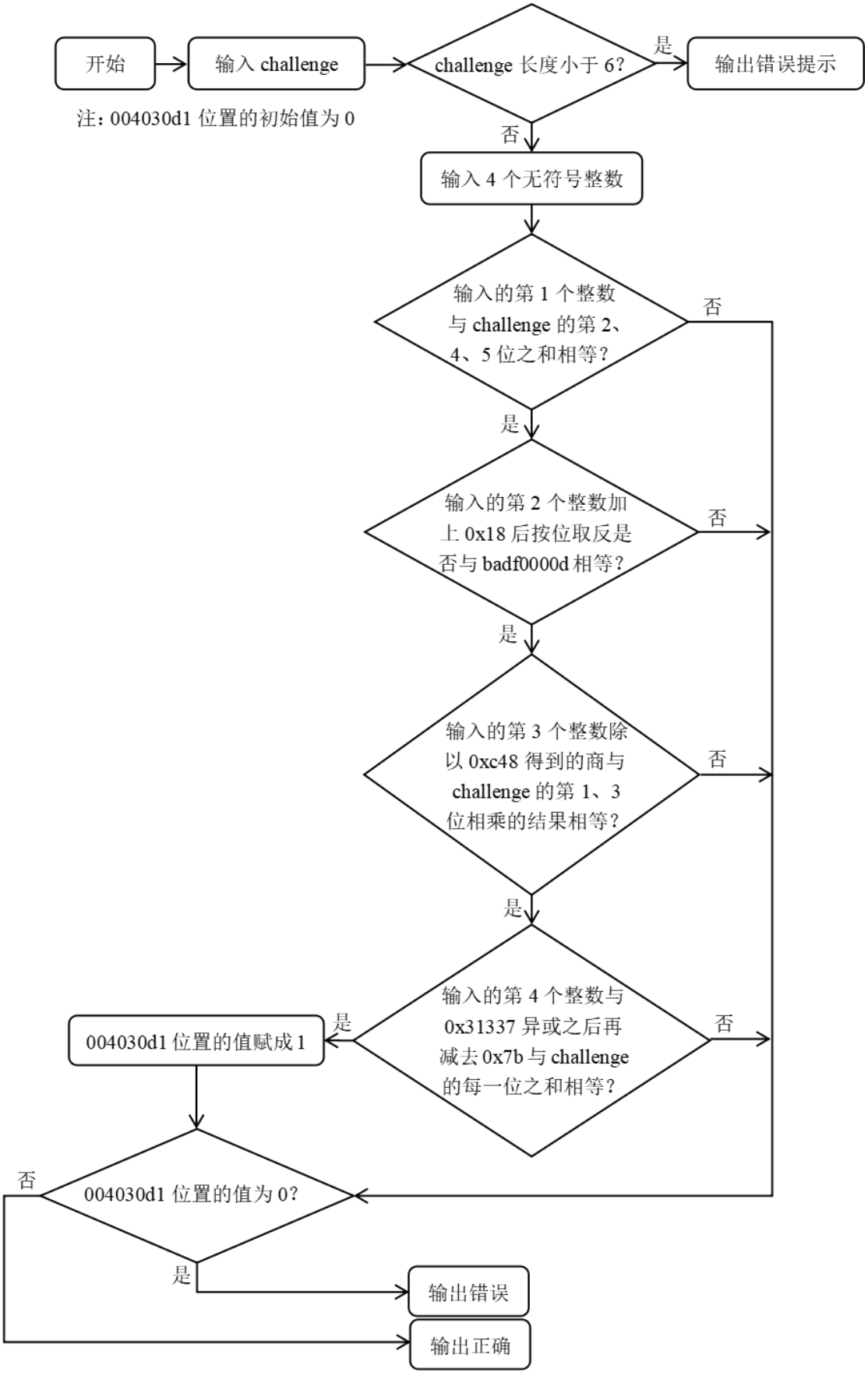
12.solution 正确

```
00401126  6881304000        push    data_403081 {var_4}  {"Congratulations, you made it!\n\r"}
0040112b  ff1508204000      call    dword [printf]
00401131  83c404            add     esp, 0x4
00401134  eb0e              jmp     0x401144
```

由于 004030d1 位置的值被修改成 1，不与 0 相等，所以跳转到 000401126 位置。输出“Congratulations, you made it!”，代表 solution 正确。随后跳转至 00401144 位置，进入**第 3 点**，程序结束。

四、逆向分析流程图

（见下页）



注：004030d1 位置的初始值为 0

五、结果验证

输入 challenge 为 114514，输入 solution 为 151-1159790554-8010912-201372，结果正确。