writeup

1. 检查题目保护，发现保护全开

```
wxy@ubuntu:/mnt/hgfs/Desktop/timu$ checksec pwn
[*] '/mnt/hgfs/Desktop/timu/pwn'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

2. 使用IDA pro对其进行逆向分析，发现漏洞出现在这里

```
1 void __cdecl Rule::edit_left(Rule *const this, size_t new_len)
2 {
3   void *v2; // rcx
4
5   if ( (signed int)new_len <= (signed int)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::size(this) )
6   {
7     v2 = (void *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(this);
8     read(0, v2, new_len);
9   }
10 }
```

可以看到参数new_len是size_t类型(即unsigned long long),而和字符串的长度进行比较时，使用的是signed int，这里就存在整数溢出，我们可以将new_len设置为0x100000000，从而绕过这个检查，然后read函数就可以进行堆溢出了

3. 漏洞利用

   可以看到Rule对象的布局如下：

```
00000000 Rule            struc ; (sizeof=0x38, align=0x8, copyof_381)
00000000                                       ; XREF: _Z11ReadGrammarv/r
00000000 left_           std::__cxx11::string ?
00000020 right_          std::vector<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char> >,st
00000038 Rule            ends
00000038
```

   最上面的是string，其下是vector。

   当文法的左部符号的长度比较小的时候(size < 16),该字符串其实是保存在string对象内部的，此时string中的指针指向string对象内部，如果此时进行溢出，一定会覆盖到vector对象，导致后面ShowGrammar的时候程序crash。

   这里我们需要输入一个较长的文法符号(size>16)，这时string内部会为字符串重新分配一块空间，这样就可以脱离这个对象布局，然后再进行溢出，这样就不会覆盖vector对象。

   此时该字符串保存的位置后面还有一个string对象，对该string对象内部的指针进行部分覆写，指向一个堆地址处，然后通过ShowGrammar进行泄露堆地址

   由于我们程序本身没有使用delete，所以堆中并没有main_arena的残留地址，这样无法泄露libc地址，我们需要人为创造出来。注意这里：

```
void __cdecl EditNonTerminal()
{
  __int64 v0; // rax
  std::vector<Rule,std::allocator<Rule> > *v1; // rax
  Rule *v2; // rbx
  std::vector<Rule,std::allocator<Rule> > *v3; // rax
  std::vector<Rule,std::allocator<Rule> >::iterator v4; // rax
  std::vector<Rule,std::allocator<Rule> > *v5; // rax
  __int64 v6; // rax
  size_t v7; // rbx
  Rule *v8; // rax
  __gnu_cxx::__normal_iterator<Rule*,std::vector<Rule,std::allocator<Rule> > > it; // [rsp+0h] [rbp-60h]
  size_t len; // [rsp+10h] [rbp-50h]
  std::__cxx11::string non_terminal; // [rsp+20h] [rbp-40h]
  unsigned __int64 v12; // [rsp+48h] [rbp-18h]

  v12 = __readfsqword(0x28u);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&non_terminal);
  v0 = std::operator<<<std::char_traits<char>>(&std::cout, "Non-Terminal:");
  std::ostream::operator<<(v0, &std::endl<char,std::char_traits<char>>);
  std::operator>><char,std::char_traits<char>,std::allocator<char>>(&std::cin, &non_terminal);
  v1 = Info::rules(&info);
  v2 = std::vector<Rule,std::allocator<Rule>>::end(v1)._M_current;
  v3 = Info::rules(&info);
  v4._M_current = std::vector<Rule,std::allocator<Rule>>::begin(v3)._M_current;
  it._M_current = std::find_if<__gnu_cxx::__normal_iterator<Rule *,std::vector<Rule,std::allocator<Rule>>>,EditNonTermina
                    v4,
                    (__gnu_cxx::__normal_iterator<Rule*,std::vector<Rule,std::allocator<Rule> > >)v2,
```

可以看到EditNonTerminal函数这里，会在栈中创建一个临时的string对象，我们可以输入一个长度>0x90的string，此时string会为其分配一个size>0x90的chunk，这样在函数返回的时候，string析构函数会将该chunk free掉，这样就可以得到一个main_arena的地址。

然后再利用堆溢出，修改string对象内部的指针，使其指向main_arena的位置，进行libc地址的泄露。

有了libc地址后，就可以写__malloc_hook。这里几个one_gadget都不能使用，需要使用realloc进行调整。

最后编写exp脚本，详情见"解题目录"