Defcon CTF 18 Qual pp300 题目详解 casperkid@insight-labs.org

先来到主函数里 在用 IDA 逆向分析的时候可以养成一个好习惯
根据分析函数的大概功能对函数名进行重新命名 这样方便以后快速识别目前处于什么位置

```
int __cdecl sub_8048D34()
{
  int v1; // ST1C_4@1

  v1 = sub_8048F7A(word_804E404);
  sub_804915E("fcfl");
  sub_80490D0(v1, (int)sub_804C18B);
  return 0;
}
```

像这个地方我认为的主模块了 我就可以把 sub_8048D4()重命名为 pp300_main

```
int __cdecl pp300_main()
{
  int v1; // ST1C_4@1

  v1 = sub_8048F7A(word_804E404);
  sub_804915E("fcfl");
  sub_80490D0(v1, (int)sub_804C18B);
  return 0;
}
```

现在先来分析第一个未知函数 sub_8048F7A

```
int __cdecl sub_8048F7A(uint16_t a1)
{
  int optval; // [sp+2Ch] [bp-ACh]@1
  __int16 s; // [sp+BCh] [bp-1Ch]@1
  uint16_t v4; // [sp+BEh] [bp-1Ah]@1
  int fd; // [sp+CCh] [bp-Ch]@3

  optval = 1;
  memset(&s, 0, 0x10u);
  s = 2;
  v4 = htons(a1);
  if ( signal(17, handler) == (__sighandler_t)-1 )
    err(-1, "Unable to set SIGCHLD handler");
  fd = socket(2, 1, 0);
  if ( fd == -1 )
    err(-1, "Unable to create socket");
  if ( setsockopt(fd, 1, 2, &optval, 4u) == -1 )
    err(-1, "Unable to set reuse");
  if ( bind(fd, (const struct sockaddr *)&s, 0x10u) == -1 )
    err(-1, "Unable to bind socket");
  if ( listen(fd, 20) == -1 )
    err(-1, "Unable to listen on socket");
  return fd;
}
```

对网络编程熟悉的少年可以很快就反应过来 这是一个初始化 socket 的函数
那传的参数不出意外应该是端口号

看到 main 部分

```
int __cdecl pp300_main()
{
  int v1; // ST1C_4@1

  v1 = sub_8048F7A(word_804E404);
  sub_804915E("fcfl");
  sub_80490D0(v1, (int)sub_804C18B);
  return 0;
}
```

传参是 word_804E404 点过去可以看到是 16 进制的值 15B3h

```
.data:0804E404 word_804E404     dw 15B3h
```

用计算器可以快速算出 转换成十进制便是 5555
我们再修改一下函数名 便成了

```
int __cdecl pp300_main()
{
  int v1; // ST1C_4@1

  v1 = Bind_socket(port);                       // port = 5555
  sub_804915E("fcfl");
  sub_80490D0(v1, (int)sub_804C18B);
  return 0;
}
```

现在继续分析第二个函数 sub_804915E
进去后发现会检测当前 user 是不是叫 fcfl
getpwnam()
函数功能：获取用户登录相关信息

```
int __cdecl sub_804915E(const char *name)
{
  struct passwd *v2; // [sp+1Ch] [bp-Ch]@1

  v2 = getpwnam(name);
  if ( !v2 )
    err(-1, "Failed to find user %s\n", name);
  if ( sub_80491BE(v2) == -1 )
    err(-1, "drop_privs failed!\n");
  return 0;
}
```

如果获取用户登陆相关失败就报错

```
casperkid@casperkid-desktop:~/桌面/Defcon/pp300$ ./PwtentPwnables\(300\).bin
PwtentPwnables(300).bin: drop_privs failed!
: Operation not permitted
casperkid@casperkid-desktop:~/桌面/Defcon/pp300$
```

所以要新建一个用户名为 fcfl 并用其登陆后 再启动程序就 OK 了

```
fcfl@casperkid-desktop:~/defcon$ pwd
/home/fcfl/defcon
fcfl@casperkid-desktop:~/defcon$ ./PwtentPwnables300
```

现在就可以正常运行了

我们继续看到之前那里 这时可以再重命名第二个函数名

```
int __cdecl pp300_main()
{
  int v1; // ST1C_4@1

  v1 = Bind_socket(port);
  checkuser("fcfl");
  sub_80490D0(v1, (int)sub_804C18B);
  return 0;
}
```

目前为止这个程序先是会绑定 5555 端口

然后 checkuser 查看当前用户是否为 fcfl

```
^C
fcfl@casperkid-desktop:~/defcon$ ./PwtentPwnables300
```

fcfl@casperkid-desktop: ~

文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

```
$ bash
fcfl@casperkid-desktop:~$ nc -vv 127.0.0.1 5555
Connection to 127.0.0.1 5555 port [tcp/*] succeeded!


fantasy chicken farmin league


menu
 c) create account
 l) login
 q) quit
```

接着分析下一个函数 sub_80490D0

```
int __cdecl pp300_main()
{
  int v1; // ST1C_4@1

  v1 = Bind_socket(port);
  checkuser("fcfl");
  sub_80490D0(v1, (int)sub_804C18B);
  return 0;
}
```

这个函数重命名为 child_process 它用来创建子进程

这个函数是在 Defcon pp 题里很经典的一个题目模型 这样每次都产生子进程

即使子进程异常崩溃了 也不会影响主进程

在实际竞赛时 当选手进行远程 exploit 时 如果 shellcode 不正确 会造成子进程崩溃掉
但也不会影响主进程 可以再反复远程连接

```
void __cdecl child_process(int fd, int (__cdecl *a2)(_DWORD))
{
  socklen_t addr_len; // [sp+1Ch] [bp-2Ch]@2
  struct sockaddr addr; // [sp+20h] [bp-28h]@2
  int v4; // [sp+30h] [bp-18h]@1
  int v5; // [sp+34h] [bp-14h]@2
  int v6; // [sp+38h] [bp-10h]@3
  int status; // [sp+3Ch] [bp-Ch]@5

  v4 = 1;
  while ( v4 )
  {
    addr_len = 16;
    v5 = accept(fd, &addr, &addr_len);
    if ( v5 != -1 )
    {
      v6 = fork();
      if ( v6 != -1 )
      {
        if ( !v6 )
        {
          close(fd);
          status = a2(v5);
          close(v5);
          exit(status);
        }
        close(v5);
      }
    }
  child process:0
```

现在的函数名情况便是

```
int __cdecl pp300_main()
{
  int v1; // ST1C_4@1

  v1 = Bind_socket(port);
  checkuser("fcfl");
  child_process(v1, (int)pp300_function);
  return 0;
}
```

pp300_function 便是整个题目正式的功能函数
我们将要进行逆向分析的也是这个部分
等会继续跟进看 pp300_function

现在我们先根据选项新建一个账号大概看看这个程序在做啥
我先随便创建了一个账号 用户名 Casper / 密码 com333
然后再 login 进去

```
menu
 c) create account
 l) login
 q) quit
c
1: c
enter new username: casper
enter new info: I Love YOu
enter new office: haha
enter new pass: com333


fantasy chicken farmin league


menu
 c) create account
 l) login
 q) quit
l
1: l
enter username: casper
enter password: com333
lcgged in!


fantasy chicken farmin league


menu    (casper)
 L) logout
 b) buy chickens
 i) incinerate money
 s) sell eggs
 p) display my info
 u) update my info
 q) quit
```

继续逆向分析 pp300_function 这个函数里
在 while (1) 下面有个函数 (现在已经被我重命名为 connect_success 了)
跟进去就可以看到我们在上图看到的登陆后的返回信息了

```
while ( 1 )
{
  sub_804C09F(userlist);
  connect_success(fd);
  *( DWORD *)count = recv cl
```

这里的字符串和登陆成功后的字符串是一样的

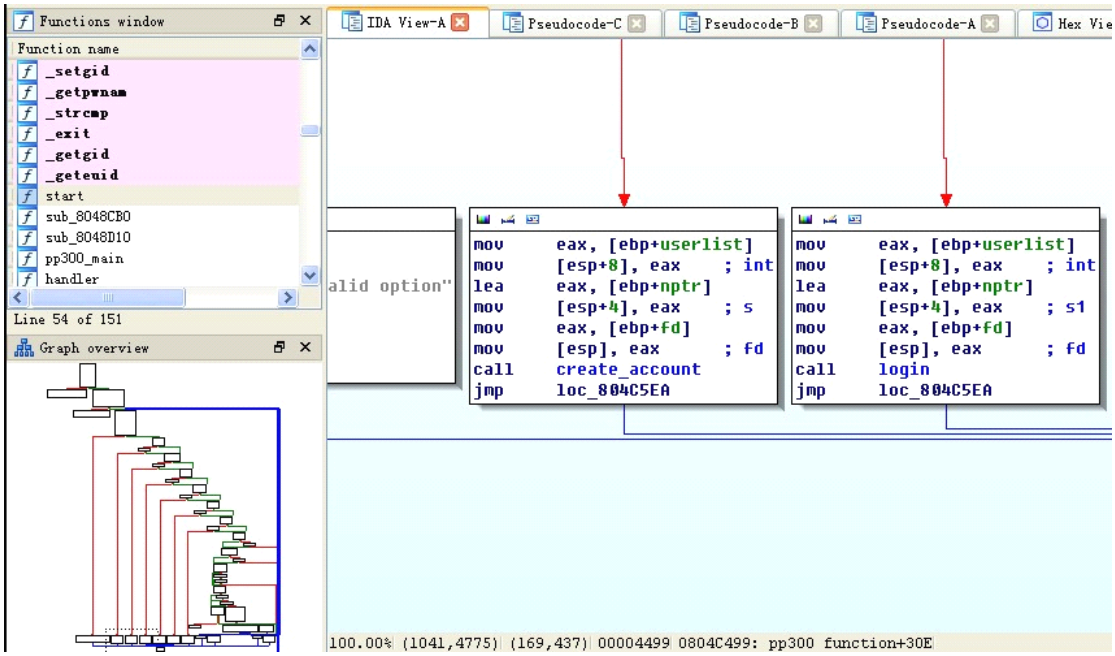
```
int __cdecl connect_success(int fd)
{
  char v2; // [sp+8h] [bp-10h]@1
  char v3; // [sp+8h] [bp-10h]@2

  print(fd, "\n\nfantasy chicken farmin league\n", 0);
  if ( login_success )
  {
    send_message(fd, "\n\nmenu    (%s)\n", (unsigned int)&s1);
    send_message(fd, " L) logout\n", v3);
    send_message(fd, " b) buy chickens\n", v3);
    send_message(fd, " i) incinerate money\n", v3);
    send_message(fd, " s) sell eggs\n", v3);
    send_message(fd, " p) display my info\n", v3);
    send_message(fd, " u) update my info\n", v3);
    if ( login_as_admin )
      send_message(fd, " P) print userlist\n", v3);
  }
  else
  {
    send_message(fd, "\n\nmenu\n", v2);
    send_message(fd, " c) create account\n", v3);
    send_message(fd, " l) login\n", v3);
  }
  return send_message(fd, " q) quit \n", v3);
}
```

但注意观察 里面有个 P) 的选项在我登陆后是没有的
所以我们可以大胆地猜测一下 这是个管理员在能看到的选项
但是怎样才能办到像管理员登陆呢?
我们观察下 login_success 的地址 和 login_as_admin 的地址
是不是挨得很近啊~ 有木有啊~ 亲~ 甚至可以大胆地猜测我们可以溢出 覆盖过去

```
.bss:0804E419                   align 4
.bss:0804E41C dword_804E41C     dd ?
.bss:0804E41C
.bss:0804E420 login_as_admin    dd ?
.bss:0804E420
.bss:0804E424 login_success     dd ?
.bss:0804E424
.bss:0804E428 ; char s1
.bss:0804E428 s1                db ?
```

逆向 pp300_function 函数时从图示关系可以看出左下角的每个红色标记对应一个选项
比如 c) 就是对应 create_account 在图中我一个根据每个选项的对应重命名了函数名



再测其他功能 p) 显示我的信息 嘿嘿可以看到很有趣的东西
<node> 89dcb20 这不是地址么
下面还有 perm: 1 next:0 prev: 89dc878
可以想到什么？对!~ 这是个链表 分配堆地址衔接出来的链表

```
 p) display my info
 u) update my info
 q) quit
p
1: p
<node> 89dcb20
  chickens: 0
  eggs:     0
  monies:   1000
  id:       0
  username: casper
  info:     I Love YOu
  office:   haha
  password: c0f14a54d35edb5555d1e10c9bc46e80

  perm:     1
  next:     0
  prev:     89dc878
```

下面的图是整个程序的流程伪代码 图是高精度转换的 大家可以放大看
大家可以很容易就观察出我们应该如何走才能进行 exploit

```
void pp300_function()
{
    open("/home/fcfl/user.db");
    while ( 1 )
    {
        get_choice_from_user(choice);
        switch choice
        {
            case c:                 // c - create account
                create_account();
                break;
            case l:                 // l - login
                login();
                //----------------------------------
                switch after_choice
                {
                    case L:         // L - logout
                        Logout();
                        break;
                    case b:         // b - buy chickens
                        buy_chickens();
                        break;
                    case i:         // i - incinerate money
                        incinerate_money();
                        break;
                    case s:         // s - sell eggs
                        sell_eggs();
                        break;
                    case p:         // p - display my info
                        display_my_info();
                        break;
                    case u:         // u - update my info
                        update_my_info();
                        break;
                    case q:         // q - quit
                        quit();
                        break;

                    login_as_admin();
                    //----------------------------------
                    case P:         // P - print userlist
                        print_userlist();
                        break;
                    case 6:         // 6 - read key from server
                        read_key_from_server("/home/fcfl/key");
                        break;
                    //----------------------------------

                    default:
                    print("that's not a valid option");
                    break;
                }
                //----------------------------------

                break;
            case q:                 // q - quit
                quit();
                break;
            default:
                print("that's not a valid option");
                break;
        }
    }
}
```

我们要想办法获得管理员权限并登陆后 选择隐藏选项 6 我们就可以获得这关的 key

然后我们再回过头来想想 要获得管理员权限就要逆向分析一下登陆部分

```
int __cdecl checklogin(int fd, char *a2, const char *s1, const char *s
{
  char dest; // [sp+17h] [bp-31h]@3
  char v6; // [sp+37h] [bp-11h]@3
  int v7; // [sp+38h] [bp-10h]@1
  char *src; // [sp+3Ch] [bp-Ch]@1

  v7 = 1;
  src = a2;
  while ( *((_DWORD *)src + 19) )
  {
    src = (char *)*((_DWORD *)src + 19);
    if ( !strcmp(s1, src) )
    {
      hash_password(s, &dest);
      v6 = 0;
      if ( !strncmp(&dest, src + 20, 0x20u) )
      {
        send_message(fd, "lcgged in!\n", (_BYTE)src + 20);
        login_success = 1;
        strncpy(&::s1, src, 0x14u);
        if ( *((_DWORD *)src + 14) > 0x1F3u )
          login_as_admin = 1;
        return 0;
      }
    }
  }
  return v7;
```

事实上用 IDA 的 F5 看时常容易看错

```
:0804A911                jnz     short loc_804A975
:0804A913                mov     eax, [ebp+userlist]
:0804A916                add     eax, 14h
:0804A919                mov     [esp+8], eax    ; char
:0804A91D                mov     dword ptr [esp+4], offset aLcggedIn ; "lcgged in!\r
:0804A925                mov     eax, [ebp+fd]
:0804A928                mov     [esp], eax      ; fd
:0804A92B                call    send_message
:0804A930                mov     ds:login_success, 1
:0804A93A                mov     eax, [ebp+userlist]
:0804A93D                mov     dword ptr [esp+8], 14h ; n
:0804A945                mov     [esp+4], eax    ; src
:0804A949                mov     dword ptr [esp], offset s1 ; dest
:0804A950                call    _strncpy
:0804A955                mov     eax, [ebp+userlist]
:0804A958                mov     eax, [eax+38h]
:0804A95B                cmp     eax, 1F3h
:0804A960                jbe     short loc_804A96C
:0804A962                mov     ds:login_as_admin, 1
:0804A96C
```

真正看汇编代码的时候 才很明显地看出

是用 userlist+0x38(userlist->perm)去跟 0x1f3h 比较

如果值比 0x1f3h 更大 则赋予用户 admin 权限

```
login()
{
    if (checklogin(username, password) == 1)
    {
        login_as_user = 1;
        if (user->perm > 0x1f3h)
            login_as_admin = 1;
    }
}
```

现在就要观察从什么地方进行溢出才能覆盖到那个 perm 的位置

通常对于溢出漏洞挖掘 自然少不了观察常见的不安全函数名
例如 strcpy() strncpy() 诸如此类的
我们在 update_user_info() 函数里可以找到 strcpy()

```
:ext:0804B343                   add     eax, [ebp+nptr]
:ext:0804B346                   mov     byte ptr [eax], 0
:ext:0804B349                   mov     edx, [ebp+nptr]
:ext:0804B34C                   mov     eax, [ebp+dest]
:ext:0804B34F                   add     eax, 86h
:ext:0804B354                   mov     [esp+4], edx    ; src
:ext:0804B358                   mov     [esp], eax      ; dest
:ext:0804B35B                   call    _strcpy         ; Vul - can be exploited
:ext:0804B360
```

结合 print_userlist() 里的数据结构

```
int __cdecl sub_804A6C0(int fd, int a2)
{
  send_message(fd, "<node> %x\n", a2);
  send_message(fd, "  chickens: %u \n", *(_DWORD *)(a2 + 60));
  send_message(fd, "  eggs:     %u \n", *(_DWORD *)(a2 + 68));
  send_message(fd, "  monies:   %u \n", *(_DWORD *)(a2 + 64));
  send_message(fd, "  id:       %u \n", *(_DWORD *)(a2 + 72));
  send_message(fd, "  username: %s \n", a2);
  send_message(fd, "  info:     %s \n", a2 + 84);
  send_message(fd, "  office:   %s \n", a2 - 122);
  send_message(fd, "  password: %s \n\n", a2 + 20);
  send_message(fd, "  perm:     %u \n", *(_DWORD *)(a2 + 56));
  send_message(fd, "  next:     %x \n", *(_DWORD *)(a2 + 76));
  return send_message(fd, "  prev:     %x \n", *(_DWORD *)(a2 + 80));
}
```

还有 create_account() 里的数据结构 其实还包括了 update_user_info()里的数据结构

```
void *__cdecl write_userinfo_into_db(int a1, int a2,
{
  void *result; // eax@1
  void *v12; // [sp+1Ch] [bp-Ch]@1

  result = malloc(0x9Cu);
  v12 = result;
  if ( result )
  {
    strcpy((char *)result, src);
    strcpy((char *)v12 + 84, a4);
    strcpy((char *)v12 + 134, office);
    strcpy((char *)v12 + 20, a6);
    *((_DWORD *)v12 + 18) = a7;
    *((_DWORD *)v12 + 14) = a8;
    *((_DWORD *)v12 + 16) = a9;
    *((_DWORD *)v12 + 15) = a10;
    *((_DWORD *)v12 + 17) = a11;
    *((_DWORD *)v12 + 19) = *(_DWORD *)(a2 + 76);
    *((_DWORD *)v12 + 20) = a2;
    *(_DWORD *)(a2 + 76) = v12;
    result = 0;
  }
  return result;
}
```

可以逆向分析出 userlist 的数据结构如下

```
--------------- 0 (node address)
|   username   |
--------------- 20
|   password   |
--------------- 53
|    ????      |
--------------- 56
|    perm      | <===== IsAdmin
--------------- 60
|chicken count|
--------------- 64
|   monies     |
--------------- 68
|  eggs count  |
--------------- 72
|     uid      |
--------------- 76
|    next      | =====> next node address
--------------- 84
|    info      |
--------------- 134
|   office     |
---------------
```

现在我们要开始观察两个节点之间如何来精确溢出覆盖数据了
先随便创建两个用户 casper 和 alice

```
<node> 89dcb20
  chickens: 0
  eggs:     0
  monies:   1000
  id:       0
  username: casper
  info:     123
  office:   AAAAAAAAAAAAAAAAAAAA12
  password: c0f14a54d35edb5555d1e10c9bc46e80

  perm:     1
  next:     0
  prev:     89dc878
```

```
<node> 89dcc18
  chickens: 0
  eggs:     0
  monies:   1000
  id:       0
  username: alice
  info:     dwqe
  office:   BBBBBBBBBBBBBBBBBBBBdw
  password: c0f14a54d35edb5555d1e10c9bc46e80

  perm:     1
  next:     89dcb20
  prev:     89dc878
```

把两个节点放在一起观察的话 内存分布如下图

```
--------------- 0 (node address) 0x89dcb20
|   username   | (casper)
--------------- 20
|   password   | hash(com333)
--------------- 53
|    ????      |
--------------- 56
|    perm      | (1)
--------------- 60
|chicken count|
--------------- 64
|   monies     |
--------------- 68
|  eggs count  |
--------------- 72
|     uid      |
--------------- 76
|    next      | (0)
--------------- 84
|    info      | (info1)
--------------- 134              0x89dcba6
|   office     | (office1)
---------------
|   .......    |
--------------- 0 (node address) 0x89dcc18
|   username   | (Alice)
--------------- 20
|   password   | hash(com333)
--------------- 53
|    ????      |
--------------- 56
|    perm      | (1)
--------------- 60
```

<node alice> -> <note casper> -> head
0x89dcc18　　0x89dcb20
那么来计算下 casper->office 到 alice 的距离
0x89dcc18 - (0x89dcb20 + 134) = 0x72 = 114
那么 casper->office 的内容至少要 114 个 byte 才能够到 alice 的 node
再继续推理 当 casper->office 的内容要开始覆盖 alice 的数据时
首先要满足 alice->password 必须是符合规范的 hash
这里可以使用一个已知 passwordhash 去覆盖
覆盖格式就是 114*A + username + hash(password) + perm = 114 + 20 + 36 + 2(perm) = 172
这里注意 这里一共要用 casper->office 覆盖 3 次
(聪明的少年 自己思考下看能知道原因不 为什么要覆盖 3 次才行呢？)

第一次覆盖到 perm(>0x1f3h)
114*A+20*B+36*C+perm(00) 这里的 00 是 ASC 码的 00
所以实际对应的是 0x3030 比 0x1f3h 的值更大
第二次覆盖到 password_hash(c0f14a54d35edb5555d1e10c9bc46e80)
114*A+20*B+c0f14a54d35edb5555d1e10c9bc46e80
第三次覆盖到 username(Alice)
114*A+Alice

```
--------------- 0 (node address) 0x89dcb20
|   username  | (casper)
--------------- 20
|   password  | hash(com333)
--------------- 53
|    ????     |
--------------- 56
|    perm     | (1)
--------------- 60
|chicken count|
--------------- 64
|   monies    |
--------------- 68
|  eggs count |
--------------- 72
|     uid     |
--------------- 76
|    next     | (0)
--------------- 84
|    info     | (info1)
--------------- 134                0x89dcba6
|   office    | (114个A) <============== 第1次覆盖   第2次覆盖   第3次覆盖
---------------
|   .......   |
--------------- 0 (node address) 0x89dcc18
|   username  | (Alice)  <============== 第1次覆盖   第2次覆盖   第3次覆盖
--------------- 20
|   password  | hash(com333)  <======== 第1次覆盖   第2次覆盖
--------------- 53
|    ????     |          <============== 第1次覆盖
--------------- 56
|    perm     | (0x3030)对应ascii 00 <= 第1次覆盖
--------------- 60
```

三次覆盖完后再用 Alice 的账号去登陆 再选择隐藏选项 6
便获得了 key "UsermyPower-CasperKid"