

Краткие разборы задач Подмосковной олимпиады школьников по информатике, профиль “Информационная безопасность”

Расшифровка названий задач:

Название задачи (количество баллов) (категория / подкатегория)

Предполётные проверки (5) (misc/sanity)

Флаг был в инструкции участника. Её надо было внимательно прочитать.

Сетевые истории I (5) (misc/Networking)

В сети NET-A на схеме сети указаны подсети 172.16.40.1/X и 172.16.40.10/X.

Для того чтобы найти максимальное количество хостов нам в первую очередь надо найти максимальную общую маску подсети. Для этого нужно перевести IP адреса в двоичную форму:

$$\begin{aligned} 172.16.40.1 &= 10101100.00010000.00101000.00000001 \\ 172.16.40.10 &= 10101100.00010000.00101000.00001010 \end{aligned}$$

Как можно заметить, общая маска подсети заканчивается на бите №19:

$$\begin{aligned} 172.16.40.1 &= \mathbf{10101100.00010000.00101000.00000001} \\ 172.16.40.10 &= \mathbf{10101100.00010000.00101000.00001010} \end{aligned}$$

С такой маской количество хостов в сети будет максимальным, с учётом уже имеющихся хостов.

Ответ: 255.255.248.0

Сетевые истории II (5) (misc/Networking)

В этой задаче у нас есть две подзадачи:

1) Подсчитать количество камер в подсети NET-B можно путём нехитрой формулы $2^{(32-N)} - 2$, где N это количество бит в маске подсети. А ещё два компьютера мы вычитаем, так как у нас есть два особых адреса, которые не рекомендуется выдавать обычным хостам – это адрес сети и broadcast-адрес.

$$2^{(32-27)} - 2 = 2^5 - 2 = 32 - 2 = 30 \text{ камер можно разместить в этой подсети.}$$

2) Для подсчёта количества хранимой информации нам необходимо подсчитать количество возможных серверов в подсети NET-C. Делается это всё так же по формуле выше, но сначала надо подсчитать маску подсети по алгоритму, приведённому в задаче “Сетевые истории I”.

172.16.20.1	=	10101100.00010000.00010100.00000001
172.16.20.10	=	10101100.00010000.00010100.00001010
172.16.20.11	=	10101100.00010000.00010100.00001011
172.16.20.12	=	10101100.00010000.00010100.00001100

Общая максимальная маска /28, а значит максимальное количество компьютеров в сети: $2^{(32-28)} - 2 = 14$. Так как каждый компьютер хранит 64 ТБ информации, то общее количество хранимой информации равно $14 \cdot 64 = 896$ терабайт информации.

А ещё авторы ошиблись в схеме сети и указание маски подсети осталось у интерфейса пограничного маршрутизатора...

Салат I (5) (crypto)

Самый обычный ROT13 (шифр Цезаря со сдвигом на 13). Можно было решить, как и через кибершеф (cyberchef.informatics.ru), так и через написание скрипта на python-е.

Салат II (10) (crypto)

Уже не совсем обычный ROT, потому что он сделан по таблице ASCII. Тут нужно было писать скрипт на python-е для перебора ключей от 1 до 255 и проверяя расшифрованный текст на наличие формата флага " posh{ ".

Асимметричненько (10) (crypto)

Классическая задача на RSA, необходимо факторизовать параметр n, получить p и q, сгенерировать d и расшифровать сообщение.

Факторизация делается при помощи утилиты factor, которая идёт в составе coreutils, которые есть во многих Debian дистрибутивах по-умолчанию. В том числе и в Kali Linux.

Алгоритм генерации ключей и шифрования/расшифрования сообщений алгоритмом RSA можно найти в любом приличном учебнике/энциклопедии по вопросам основ информационной безопасности.

```

└─$ python3
Python 3.13.7 (main, Mar  3 2026, 12:19:54) [GCC 15.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> 0x69707fe901bcd58b32bf4a03
32631929761797973986424867331
>>> exit()
└─user@my-second-VM ~/Desktop
└─$ factor 32631929761797973986424867331
32631929761797973986424867331: 127099503829523 256743171913297

```

Китайская грамота (10) (reverse)

Реверс исходного кода. Правда исходники обфусцированы китайскими иероглифами. Нужно разобраться с обфускацией, затем алгоритмом и обратить "шифрование". Делается в редакторе кода (например, Visual Studio Code) путём аккуратной замены названий переменных на более читаемые и осмысленные.

Система безопасности 2000 (15) (reverse)

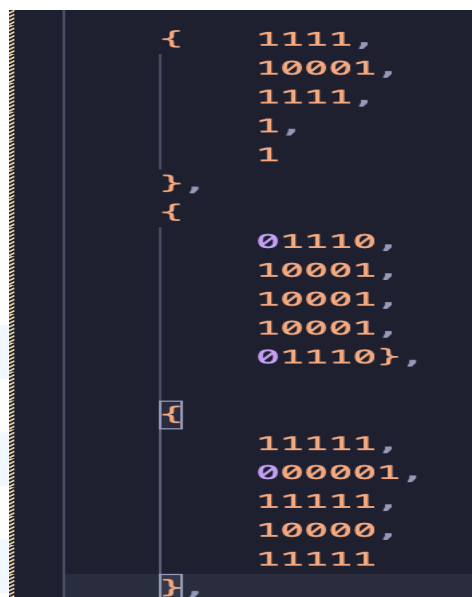
Вам дается бинарный исполняемый файл linux.
Данный файл можно запустить в консоли:

```
$> ./code
```

В программе вас просят ввести 28 матриц 5 на 5 символов из '#' и '-'. Необходимо для этого найти как в программе сверяется ввод с флагом и достать флаг.

Запускаем Ghidra, анализируем в ней наш бинарный файл. Находим, где считывается ввод и потом с чем он сверяется. Благо у нас есть переменная, по которой фиксируется правильность ввода флага и вследствие чего найти место проверки флага.

00102144	-#-#-	"-#-#-"	ds
0010214a	#-#-#	"#-#-#"	ds
00102150	-#-#-	"-#-#\n"	ds
00102160	-###-	"-###-"	ds
00102166	#---#	"#---#"	ds
0010216c	-###-	"-###-\n"	ds



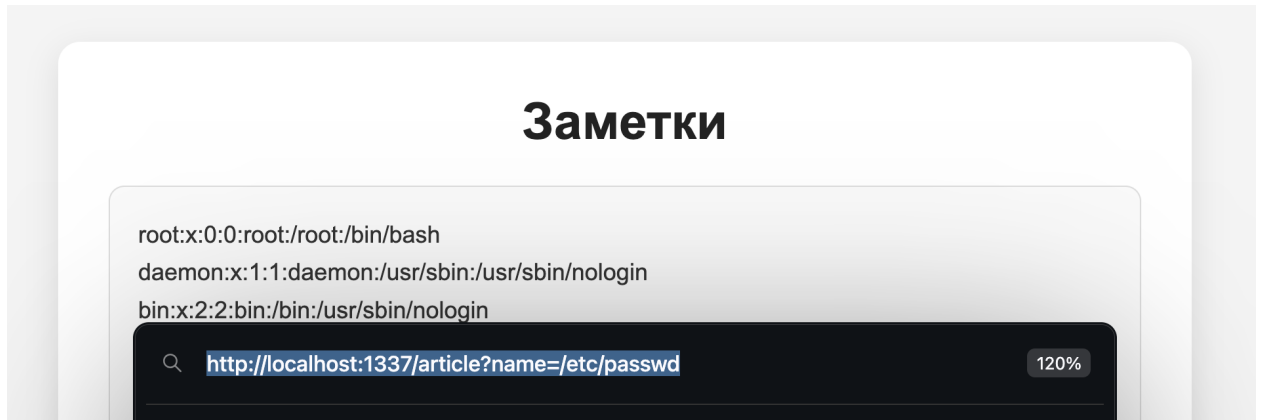
Тем самым получаем 28 матриц. Если внимательно на них посмотреть, то можно заметить, что в каждой матрице вписана какая-то буква. Остается перевести все матрицы в 28 символов, 'расшифровать' символы и составить флаг.

Рекомендуется делать это программными методами, например при помощи скрипта на языке python.

Забывчивый программист (5) (web)

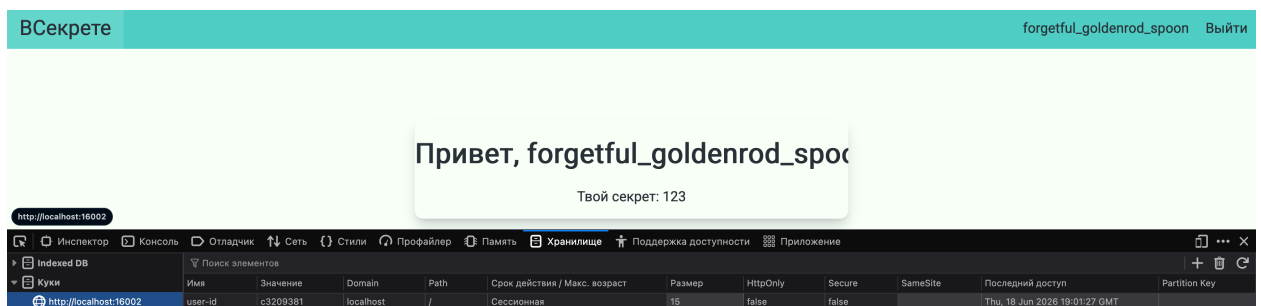
Довольно базовая и простая задача – нужно найти уязвимость LFD и через технику path traversal достать секрет за два шага. Флаг хранился сразу в трёх местах – ./flag.txt, /root/flag.txt и /etc/flag.txt

[Подробнее про уязвимость LFD и технику path traversal.](#)



Система для забывчивых программистов (15) (web)

Это задача на автоматизацию исследования данных на сайте. Можно заметить, что после регистрации пользователю выставляется кука в формате CRC32 от каких-то данных.



Тут нужно было провести небольшое исследование (например, при помощи кибершефа или скрипта на python-е) и узнать, что в куке просто хранится CRC32 от порядкового номера пользователя. Потом нужно написать скрипт, который генерирует запросы с перебором куки user_id и забирает с сайта секрет пользователя.

```
$ python3 ../solve/sploit.py | head -n 10
venomous_chocolate_screw 6Qdxt}71LQP9oUwRG{J9uvAo1ECi2RZs
attractive_ivory_bench WbZacoCAMxoRFLsC}ipCuLrcyMW_n*sU
deserted_aquamarinered_earrings EJS42JKJxIpBD{U2b4UZCvVV}Vu9}p_p
gentle_purple_user yNMe}*xdm7Uix7*iSzPtgDhhFfxC8tLr
polite_ivory_comb izsLTA5njgUi8Hdu9o6B5uE40pklETq
attractive_cyan_comb hVMM0_eloAvZIsPPaFlqDLG5gG}tex9F
funny_cyan_net JSSY7}ShxoMvGXfKMERJy49tduFMF*
flawless_cyan_sword v2zgaAluk9tLiQcRuv53VrynNng8fYov
smiling_crimson_earrings 4y7xggxTZZo-LynPajl0kKR9{Q9czDM
funny_brown_player exnbySRJVYRx*LNBDP}bK9wM4qfK-k{

$ python3 ../solve/sploit.py | grep "posh"
forgetful_crimson_monkey posh{test_flag}
```

ГЛАЗА ... (5) (forensics)

Нужно было посмотреть сгенерированный текстовый лог, понять что это лог HTTP сервера и найти попытку перебора пароля аккаунта админа.

Затем зайти на сайт, войти по паролю админа и забрать флаг на страничке контента по первому подходящему id (запрос к которому был сразу после успешной попытки входа злоумышленником).

Тут ничего нет (10) (forensics)

Классическая форензик-матрёшка. Нам дана картинка nothing.jpg.

В метаданных картинки есть подозрительная строчка с намёком на какой-то пароль. Флагом это не являлось, поэтому сдавать в систему было бессмысленно.

```
exiftool ../public/nothing.jpg
ExifTool Version Number      : 13.55
File Name                    : nothing.jpg
Directory                    : ../public
File Size                    : 54 kB
File Modification Date/Time   : 2026:06:06 23:24:24+03:00
File Access Date/Time        : 2026:06:06 23:36:13+03:00
File Inode Change Date/Time   : 2026:06:06 23:34:56+03:00
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               : None
Artist                        : Jonh Passwordowitch with password=s3cret
Y Cb Cr Positioning           : Centered
Image Width                   : 1280
Image Height                  : 1007
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                    : 1280x1007
Megapixels                    : 1.3
```

При этом утилита binwalk (или strings с некоторыми дополнительными действиями) подсвечивает нам, что за картинкой есть что-то ещё.

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	JPEG image, total size: 51676 bytes
51676	0xC9DC	ZIP archive, file count: 1, total size: 2484 bytes
54160	0xD390	ZIP archive, file count: 1, total size: 180 bytes

Analyzed 1 file for 85 file signatures (187 magic patterns) in 6.0 milliseconds

Достаём файлы при помощи binwalk -е и при попытке открыть один из архивов он спросит пароль. Вводим пароль из метаданных к картинке.

```
$ unzip n0thing.zip
Archive:  n0thing.zip
[n0thing.zip] nothing.base64 password: 🗝
```

А внутри – ещё более классическая матрёшка из кода base64! То есть внутри base64 кода будет ещё один base64 код и так далее... Рекомендуется раскодировать при помощи python или шефа, а не вручную.

```

$ cat nothing.base64
mBw0dQyWtULXVGwXsYd41YwZDRXrmxVU205V0lWBdNXa2M1vJFac2JETLhhmk0xVjBaS2Mv0k4XubGhoTwsFeFZtEdTmk15U2tWWWjHaG9UvLZ3VLZadGNFZFRNbEpJvM10bVZTSL
L2Mb67QxYwUldpPwmFKRTFVvwxSTM6RTFWa2QwYjFwDfNR2Fh1r2hhwWtaV00cxZF2XubZqVmtaMFVtceFNubJpGy0VwV2JURXdWak2XZE20c1dsagLSmhhZV14d4b2iYvNbv1V1Y
JabVLqQTFFSmWryZ2X0WVjZadEdMFZ3VjJKVYJZ2FpWRpYvBTaBZNscEhWjVv0bYJzXDRiMwV4VmtKWGjNR1VbZbTV0Sv1PqE9Jr3XkVhV4J1Z1XSKdJrmxh0uCRfQzSkd
jvclUyW0ZaaGec9GwAkjHvJD0JrF2h1v2zhxw0b21XwNRWGRVTVJWNvYtDGTWNBRIYUZSwJmGwwhZwmp2Hed0N1teFNiSE1pQ2x4w3b2EhXhVNVJ2TFTFW1lPwkt1R1pXU2ta
bFZ5wLpXa1p3YkdFeLFrBfDWM021TKVRKKUjYxYvVvBbZBTYkVwVvZteG9RMWRZv25KMGJHmFwakZHTKZaSGRhdfdiXB1Vj34U1d2tSkhhRLJXTU2wVfZGzSk5bF3zW1sU00x9rJZwa
kwoWTFVeFduSksXRXbW05xb1YxUlh0Vks5YkZweYFDUBMvPyv2xwVlZWcHjZKwG3WagCcmJGaFhTRUpJmN1sUXXhXbLZW1Vld0VFLVceFbAGRYZUC5a1XUKHmJvTzYtXG0SF
VuTLZha0pGvGtaVamVUXkhr2hPulhbd1ZSZDRJmR02JtoFSSJLhVSpV3KacvZJdG25aGmVLa2FvmtJenKEXYUjHtCFRwLVLJLVpYkZoa6FvNhWVEJrTKZkR1vsZGFSemx
PVfZad2VGNrKkREJQxYpGeVrSmNxbFpXbUkLXUKdaWRPU1dK1Rp6FJNHEJZVm10U1mXUkLa2V7YmtvaFteEtJrJp0G05WFZsCFZFUZYUuYwVfAdNaLZUvkda41NG
VnXBZ2F2XtL1HdGFGVrM1VwtoAa1jYt0NbhGHDuXk2v1x2OVVNNAwV0151LW1hR0ZVvMhX1L1WnD5bHBHGXGvJ2rSkpXbFZFeVnBfYmMwVYrFadG2FGWLV5a1psu
m1SeL1VLhWfKp1U5W05V14yZahZakZ2vJ5b1Cdsag1WnnB6VjYQkdVWmGxRjBacFvYhteHd1FV5ZhUkHGJGCFhZMhGLZvXaWfVrZJFWQ1JWJkxk2MxcEhRmhTLhCS1ZtMT
BVMI141VLhWdFpH1ZbXhhvYjFsc1Ph0Vd5bJhWBTaa2EemMvDjRmxhVl1dMNVZGX1R1ZLZU2kTLVjW4VjMwPhVZKc1RuL1BwbFvFVRG05S0WwR0kXZYLZ2vXadG9hMup0VW5
CV2JHaERuBxhhV1Z0VJsvK5wbkF3VLcvmMwXUxhXbk5UY1VaVLZteHdNMvPyv21GaLZrcDFXa1pPVjJfFeGNEVLdSRVpYxPgVmwQmULiR2hTYLhoWdXeg9RMVJHw5KYVJwCnWV
T2FTZxw5VLdsTmh5VEZ6V211b1Yj0tJ5VEUJhUkVakVqSkTL1J3UYZ0aGvYvwmQWwVdReVz2FdiEbpVms1YxULhSGRUvmmXwkvkVxR09pH0Wk21GSLBwVjFGZVZw
lPhR1dMwUPrVRLScXSMWxhJRhoUj0J1ZuVktNbfP0rTVCV1KtMv2vTJjK5FVwLpWiFZU2wP6G2NWTNRpWMTYrLhOnV2TDFhM1V4250aLJFSmhbwGR0TZAaGmZaFhBf
oUwBTaa1R5WxLhRcEYpYvXtK6LrKt6VrktJazXVZKwJZaV3cFLXV3R9JZtJfDeR2FdiVpVh1a4C05VNWNRkRzLUnmXwFpZXNUMXmKqM1YUv5SWJGcGhVMGXRU0ZkGdGNFNVdWmVmxNGiY
yXLS2A2RUYkdsVSLvsmFhR1pJkG0aFJsbDNkzWjVrVtJKR2NGcFPWNBvYkD4YwNt1KvMBgR7TJoB1zR1JctSMWRH0Z5YiVZlRkAw5V1ZKwGFGTlHekZ1Vj34V1U5YSLRbK5S
YLRGVFRWMLZ1V042UmX0L2EzQmFwMwMxYjFzeFDRLRia3BWWVRGd1JZCFX2bGqTaRk1Q1XmTWmVpHV2pkvZ2JHtJRua2R0ZDZASWfGahFSMh2QvM14a1UySxhIRmxqUJ1dSWFRWw
k1dG5XWUd091eChznBnLVm1K1WVUw1Ldha1p0W14a2NtVkdarTVXMYntKSLYUkP1Rk14U1hoalXhABVb2FHVJZ2F2T1RLNvAwFVZS0J1vMZV1RfZWkd0ldmwkTXRLZV2
xwaVbSgYpV3VhV7J0V1U1BWBjJUNX0wbk1LWZFRVbU2ZwTweLjYvNNbVbPjZUdov2JHULNAReZVjYfAaWGaFNIRam94V1RCYEWnN5bK5YYPXGvZkdtJbFY2U5S081WcdF
V3pYhPVY2FeNuZ2d1FfYXmUwXmGvZHVb2bTfYFKWZGwMfKMFdHVAa1J5b6dVbXhZTLzSLV1dYm1Y0Vj30MFZNMWYVfEYpXwtaafPwktR05GT1LZK23p3bXhTUA21
SFR3aFhR1J0VwXbk2b1fXvNRnVZLYkXZweLYvNtYUwZFUJWSY5XaZa1pHLZK1pGagHNMW3RJV1ZaTtYnNRua1ZYKkDSCFvRkZMKpYvW00U1BwDx2aXVkdKWGvIT1pWR
VozVjFaa1YXcERbXbRmp2MfZdGfXSMV5VU2taA1FNvNDbZbJHvKZ2wVdHdGp1RBNWU14a1R5wNRa1pLYvVKafZgZSMFYZ02aaZJ1WfHhXRXWV25V2NGW1hWfJYvM10d2
v5wNRnW5Y1TYSvN1LWdMwMk1IvGp5SVWYUj9aRFa1U2dGR2GafFNNMhRmXbNGEXvXhXbk5WkdcU1WpFNWV1LZ0TRVCK1Z1SmRZVXhBVJAc1dUSLZ1VFZ22vW05s1d1WvNwBGR
oYTFwb1ZXMRhMk50VmtkYViyGa9YUvJLWxac1pJTFNNVTE1VzKob2FSsPhbFpaYkRfMXZbFWlPkMvPyZEZWTLZuQXdhbFzQtLdF001WWLhibXhWV14d2R5wNRNRXqJ1N1R1Qx
WmthWmRiWJNwV2FWSMhZekp0ZVZKcLdsVnLSbHBQVmxPvGfWnXNbLJ3sUms1VFRWwDXRL1L5TLV0V01rWnpMmNhZvJaxSGFFUmFMW2h0vmpGa2MxcEdhRKSXUjN0sFyXZDBWMV4Y
k2oV2Z1WwA1pHvTFsmhE1VUN9U6bFV1W0911WpLXPRGFXEnw0uXN0TP0K

```